Josyula R. Rao
Berk Sunar (Eds.)

LNCS 3659

Cryptographic Hardware and Embedded Systems – CHES 2005

7th International Workshop
Edinburgh, UK, August/September 2005
Proceedings

Springer

Lecture Notes in Computer Science 3659

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Josyula R. Rao Berk Sunar (Eds.)

Cryptographic Hardware and Embedded Systems – CHES 2005

7th International Workshop
Edinburgh, UK, August 29 – September 1, 2005
Proceedings

Springer

Volume Editors

Josyula R. Rao
IBM T.J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532, USA
E-mail: jrrao@us.ibm.com

Berk Sunar
Worcester Polytechnical Institute
Department of Electrical and Computer Engineering
100 Institute Road, Worcester, MA 01609, USA
E-mail: sunar@wpi.edu

Library of Congress Control Number: 2005931119

CR Subject Classification (1998): E.3, C.2, C.3, B.7, G.2.1, D.4.6, K.6.5, F.2.1, J.2

ISSN 0302-9743
ISBN-10 3-540-28474-5 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-28474-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11545262 06/3142 5 4 3 2 1 0

Preface

These are the proceedings of the 7th Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005) held in Edinburgh, Scotland from August 29 to September 1, 2005. The CHES workshop has been sponsored by the International Association for Cryptologic Research (IACR) for the last two years.

We received a total of 108 paper submissions for CHES 2005. The double-blind review process involved a 27-member program committee and a large number of external sub-referees. The review process concluded with a two week discussion process which resulted in 32 papers being selected for presentation. We are grateful to the program committee members and the external sub-referees for carrying out such an enormous task. Unfortunately, there were many strong papers that could not be included in the program due to a lack of space. We would like to thank all our colleagues who submitted papers to CHES 2005.

In addition to regular presentations, there were three excellent invited talks given by Ross Anderson (University of Cambridge) on "What Identity Systems Can and Cannot Do", by Thomas Wille (Philips Semiconductors Inc) on "Security of Identification Products: How to Manage", and by Jim Ward (Trusted Computing Group and IBM) on "Trusted Computing in Embedded Systems". It also included a rump session, chaired by Christof Paar, featuring informal talks on recent results.

The focus of CHES 2005 was similar to that of the earlier CHES workshops with the addition of a few new topics of emerging interest among which were smart card attacks and architectures, tamper resistance on the chip and board level, true and pseudo random number generators, special-purpose hardware for cryptanalysis, embedded security, cryptography for pervasive computing (e.g., RFID, sensor networks), device identification, non-classical cryptographic technologies, and side channel cryptanalysis. Special attention was paid to trusted computing platforms.

Special compliments go out to Colin D. Walter, the general chair and local organizer of CHES 2005, who brought the workshop to the beautiful historic town of Edinburgh, Scotland making it as much of a cultural event as a stimulating technical gathering. Christof Paar held the publicity Chair of CHES and was helpful at all stages of the organization. We would like to thank our corporate sponsors Cryptography Research Inc., escrypt GmbH, Gemplus, IBM, and RSA Security, who made it possible to have a lively event with their generous contributions. We would like to thank our dedicated webmaster Jens-Peter Kaps for maintaining the CHES website and review system even when he was travelling. Finally, we would like to thank the CHES steering committee members for giving us the honor of being part of such an influential conference.

August 2005 Josyula R. Rao and Berk Sunar

7th Workshop on Cryptographic Hardware and Embedded Systems

August 29 – September 1, 2005, Edinburgh, Scotland
http://www.chesworkshop.org/

Organizing Committee

Colin D. Walter (General Chair) ..... Comodo Research Lab, UK
Christof Paar (Publicity Chair) ..... Ruhr-Universität Bochum, Germany
Josyula R. Rao (Program Co-chair) .. IBM T.J. Watson Research Center, USA
Berk Sunar (Program Co-chair) ..... Worcester Polytechnic Institute, USA

Program Committee

Ross Anderson ..... Cambridge University, UK
Mohammed Benaissa ..... The University of Sheffield, UK
Suresh Chari ..... IBM T.J. Watson Research Center, USA
Kris Gaj ..... George Mason University, USA
Louis Goubin ..... Université de Versailles-St-Quentin-en-Yvelines, France
Jorge Guajardo ..... Infineon Technologies, Germany
Çetin Kaya Koç ..... Oregon State University, USA
Peter Kornerup ..... University of Southern Denmark, Denmark
Pil Joong Lee ..... Postech, South Korea
David Naccache ..... Gemplus, France and
Royal Holloway, University of London, UK
Elisabeth Oswald ..... Graz University of Technology, Austria
Christof Paar ..... Ruhr-Universität Bochum, Germany
Daniel Page ..... University of Bristol, UK
Bart Preneel ..... Katholieke Universiteit Leuven, Belgium
Pankaj Rohatgi ..... IBM T.J. Watson Research Center, USA
Ahmad Sadeghi ..... Ruhr-University Bochum, Germany
Kouichi Sakurai ..... Kyushu University, Japan
David Samyde ..... FemtoNano, France
Erkay Savaş ..... Sabanci University, Turkey
Werner Schindler ..... Bundesamt für Sicherheit
in der Informationstechnik, Germany
Jean-Pierre Seifert ..... Intel, USA
Nigel Smart ..... University of Bristol, UK
Francois-Xavier Standaert ..... Université Catholique de Louvain, Belgium

Tsuyoshi Takagi ..... Future University, Hakodate, Japan
Elena Trichina ..... Spansion, USA
Ingrid Verbauwhede .. ESAT/COSIC Division, Katholieke Universiteit, Leuven
Colin Walter ..... Comodo Research Lab, UK

Steering Committee

Marc Joye ..... Gemplus, Card Security Group, France
Burt Kaliski ..... RSA Laboratories, USA
Çetin Kaya Koç ..... Oregon State University, USA
Christof Paar ..... Ruhr-Universität Bochum, Germany
Jean-Jacques Quisquater ..... Université Catholique de Louvain, Belguim
Josyula R. Rao ..... IBM T.J. Watson Research Center, USA
Berk Sunar ..... Worcester Polytechnic Institute, USA
Colin D. Walter ..... Comodo Research Lab, UK

External Referees

Onur Acıiçmez
Dakshi Agrawal
Mehdi-Laurent Akkar
Roberto Avanzi
Murat Aydos
Yoo Jin Baek
Lelia Barlow
Lejla Batina
Chevallier-Mames Benoit
Guido Bertoni
Régis Bevan
Mike Bond
Eric Brier
Julien Brouchier
Christophe De Cannière
Dario Carluccio
Laurent Caussou
Juyoung Cha
Herve Chabanne
Nam Su Chang
Kookrae Cho
Mathieu Ciet
Jolyon Clulow
Jean-Sbastien Coron
Nicolas Courtois
Colin van Dyke
Serdar S. Erdem
Martin Feldhofer
Patrick Felke
Wieland Fischer
Jacques J.A. Fournier
Patrick George
Christophe Giraud
Robert Granger
Johann Großschädl
Adnan Gutub
Ghaith Hammouri
Dong Guk Han
Helena Handschuh
Oliver Hauck
Alireza Hodjat
Tetsuya Izu
Mark Jung
Charanjit Jutla
Deniz Karakoyunlu
Paul Karger
Manabu Katagi
Alisher Anatolyevich Kholmatov
Tae Hyun Kim
Minho Kim
Shinsaku Kiyomoto
François Koeune
Sandeep Kumar
Klaus Kursawe
Soonhak Kwon
Gerard Lai
Joe Lano
Peter Leadbitter
Hyang-Sook Lee
Jung Wook Lee
Kerstin Lemke
HuiYun Li
Marco Macchetti
François Macé
Stefan Mangard
Marian Margraf
Nele Mentens
Atsuko Miyaji
Christophe Mourtel
Elke de Mulder
Robert Mullins

Michael Neve
Richard Noad
Francis Olivier
Gerardo Orlando
Siddika Berna Ors
Pascal Paillier
Fabrice Pautot
Matthew Parker
Eric Peeters
Jan Pelzl
Gilles Piret
Thomas Popp
Axel Poschmann
Christine Priplata
Kumar Ranganathan
Nalini Ratha
Arash Reyhani-Masoleh
Gaël Rouvroy
Andy Rupp
Reiner Sailer
Junichiro Saito
Ryuichi Sakai
Yasuyuki Sakai
Kazuo Sakiyama
Gökay Saldamlı
Hisayoshi Sato
Akashi Satoh
Daniel Schepers
Jörg Schwenk
Kai Schramm
Jong Hoon Shin
Jamshid Shokrollahi
Nicolas Sklavos
Sergei Skorobogatov
Colin Stahlke
Martijn Stam
Makoto Sugita
Katsuyuki Takashima
Stefan Tillich
Michael Tunstall
Shigenori Uchuyama
Guy Vandenbosch
Ihor Vasylstov
Frederik Vercauteren
Karine Villegas
Camille Vuillaume
Andre Weimerskirch
Claire Whelan
Christopher Wolf
Johannes Wolkerstorfer
Thomas Wollinger
Yeon Hyeong Yang
Jeong Il Yoon
Young Tae Youn

Previous CHES Workshop Proceedings

CHES 1999: Çetin K. Koç and Christof Paar (Editors). *Cryptographic Hardware and Embedded Systems*, vol. 1717 of *Lecture Notes in Computer Science*, Springer-Verlag, 1999.

CHES 2000: Çetin K. Koç and Christof Paar (Editors). *Cryptographic Hardware and Embedded Systems – CHES 2000*, vol. 1965 of *Lecture Notes in Computer Science*, Springer-Verlag, 2000.

CHES 2001: Çetin K. Koç, David Naccache, and Christof Paar (Editors). *Cryptographic Hardware and Embedded Systems – CHES 2001*, vol. 2162 of *Lecture Notes in Computer Science*, Springer-Verlag, 2001.

CHES 2002: Burton S. Kaliski, Çetin K. Koç, and Christof Paar (Editors). *Cryptographic Hardware and Embedded Systems – CHES 2002*, vol. 2523 of *Lecture Notes in Computer Science*, Springer-Verlag, 2002.

CHES 2003: Colin D. Walter, Çetin K. Koç, and Christof Paar (Editors). *Cryptographic Hardware and Embedded Systems – CHES 2003*, vol. 2779 of *Lecture Notes in Computer Science*, Springer-Verlag, 2003.

CHES 2004: Marc Joye and Jean-Jacques Quisquater (Editors). *Cryptographic Hardware and Embedded Systems – CHES 2004*, vol. 3156 of *Lecture Notes in Computer Science*, Springer-Verlag, 2004.

Table of Contents

Side Channels I

Arithmetic for Cryptanalysis

Low Resources

Special Purpose Hardware

Hardware Attacks and Countermeasures I

Arithmetic for Cryptography

Side Channel II (EM)

Side Channel III

Trusted Computing

Hardware Attacks and Countermeasures II

Hardware Attacks and Countermeasures III

Efficient Hardware I

Efficient Hardware II

Resistance of Randomized Projective Coordinates Against Power Analysis

William Dupuy and Sébastien Kunz-Jacques

DCSSI Crypto Lab,
51, bd de Latour-Maubourg, 75700 PARIS 07 SP
william.dupuy@laposte.net
kunzjacq@yahoo.fr

Abstract. Embedded devices implementing cryptographic services are the result of a trade-off between cost, performance and security. Aside from flaws in the protocols and the algorithms used, one of the most serious threats against secret data stored in such devices is Side Channel Analysis.

Implementing Public Key Cryptography in low-profile devices such as smart cards is particularly challenging given the computational complexity of the operations involved. In the area of elliptic curve cryptography, some choices of curves and coefficient fields are known to speed up computations, like scalar multiplication. From a theoretical standpoint, the use of optimized structures does not seem to weaken the cryptosystems which use them. Therefore several standardization bodies, such as the NIST, recommend such choices of parameters. However, the study of their impact on practical security of implementations may have been underestimated.

In this paper, we present a new chosen-ciphertext Side-Channel Attack on scalar multiplication that applies when optimized parameters, like NIST curves, are used together with some classical anti-SPA and anti-DPA techniques. For a typical exponent size, the attack allows to recover a secret exponent by performing only a few hundred adaptive power measurements.

1 Introduction

The use of elliptic curves for cryptographic purposes was proposed by Miller [10] in 1985 and Koblitz [8] in 1987. Since then, it became an essential part of public key cryptography. In particular, many cryptosystems rely on the intractability of the discrete logarithm problem (DLP) on elliptic curves. The main advantage of this problem is that it is believed to be harder to solve than other number-theoretic problems. As a consequence, for a similar security level, it is possible to use smaller objects than with integer factorization for example. This property is especially attractive for embedded systems, where storage requirements and computation times are critical.

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 1–14, 2005.

Cryptosystems relying on DLP on elliptic curves use the *scalar multiplication* operation in some large elliptic curve group $(G, +)$

$$P \in G \to kP \tag{1}$$

where k is a secret data. Because of DLP hardness, it is believed to be infeasible to compute k from the knowledge of one or several pairs (P, kP).

In a situation where no reasonable attack on a cryptographic algorithm is known, Kocher first observed in 1996 [9] that the measurement of the algorithm computation time could still reveal secret information. This paved the way to Side Channel Attacks that take advantage of the measurement of physical signals emitted by a cryptographic device during a computation to gain access to secret data.

Since then, several examples of Side Channel Attacks led to various countermeasures being developed. Concerning scalar multiplication in EC groups, the use of scalar multiplication algorithms with a regular computation flow like *double-and-add always* or *Montgomery Ladder* is an answer to Simple Power Analysis (SPA), while randomized projective coordinates, first proposed by [4], are used to counter Differential Power Analysis (DPA).

In this paper, we present a new side-channel attack against scalar multiplication implementing these countermeasures, when the EC group used is chosen among the NIST [12], ANSI [1] or SEC [13] recommended curves. It is a Goubin-style attack [6] that uses distinguished points whose presence can be detected along the computation by an observation of power traces despite the randomization countermeasure. It leverages the particular shape of the underlying coefficient fields.

The paper is organized as follows. We first briefly review some facts about elliptic curves in section 2. Then section 3 presents some classical Side Channel Attacks and common countermeasures to prevent them. Finally, sections 4 and 5 present the details of our attack.

2 Elliptic Curves

2.1 Elliptic Curve Equation

Let $\mathbb{K}$ be a finite field of characteristic p. Over this field, we set the equation (E)

$$y^2 + a_1xy = x^3 + a_2x^2 + a_4x + a_6$$

The elliptic curve (C) associated to (E) is the set of all points of $\mathbb{K}^2$ satisfying (E), together with a particular point $\mathcal{O}$ called *point at infinity*. $\mathbb{K}$ is the *coefficient field* of the curve.

Up to an affine change of variables, if $p = 2$, we can set $a_1 = 1$ and $a_4 = 0$. The equation can then be rewritten $y^2 + xy = x^3 + a_2x^2 + a_6$. If $p \geq 3$, we can set $a_1 = a_2 = 0$ and then (E) becomes $y^2 = x^3 + a_4x + a_6$.

Together with an addition law, this set forms a commutative group. We do not describe the group law here since it does not play any role in the attack we present.

2.2 Affine and Projective Representation

A point on a curve of equation (E) is a solution of (E). Therefore the simplest representation of a point on a curve of equation (E) is the corresponding solution of (E) in $\mathbb{K}^2$. This is the *affine* representation.

Nevertheless, other representations can be preferred. We are mainly interested in *projective coordinates*. Given $P = (x, y)$ in affine coordinates, its representation in projective coordinates is $P = (xZ, yZ, Z)$ for any $Z \in \mathbb{K}^*$. If a finite solution of (E) is represented by (α, β, γ), then $\gamma \neq 0$. The point at infinity $\mathcal{O}$ is represented by $(0, \beta, 0)$ for any $\beta \neq 0$.

The projective representation is not unique. In fact, for some finite solution (x, y) of (E) with $x \neq 0$ and $y \neq 0$, any of the three projective coordinates can take an arbitrary value in $\mathbb{K}^*$. This observation is the basis of the randomized projective coordinates countermeasure, which we will describe in section 3.2. Projective representation is also used to increase the efficiency of point addition computations since for example, it allows to compute the group law without having to perform modular inversion in the coefficient field.

2.3 Recommended Coefficient Fields for NIST Elliptic Curves

Curves recommended by standardization bodies such as NIST, ANSI, or SEC are usually defined over $\mathbb{F}_p$, or $\mathbb{F}_2[x]/(P)$ where P a primitive polynomial. We focus on NIST recommended curves from now on. Other standardized curves present similar properties as the ones of the NIST.

Curves Defined on Binary Fields. The coefficient field is here of the form $\mathbb{F}_2[x]/(P)$. The primitive polynomials standardized by the NIST are:

$$\begin{aligned} P_{233}(x) &= x^{233} + x^{74} + 1 \\ P_{283}(x) &= x^{283} + x^{12} + x^7 + x^5 + 1 \\ P_{409}(x) &= x^{409} + x^{87} + 1 \\ P_{571}(x) &= x^{571} + x^{10} + x^5 + x^2 + 1 \end{aligned}$$

We can notice that these polynomials are very sparse. This has to do with hardware efficiency.

Curves Defined on Prime Order Fields. For these curves, the coefficient field is $\mathbb{F}_p$, with p among

$$\begin{aligned} p_{192} &= 2^{192} - 2^{64} - 1 \\ p_{224} &= 2^{224} - 2^{96} + 1 \\ p_{256} &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \\ p_{384} &= 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1 \\ p_{521} &= 2^{521} - 1 \end{aligned}$$

As in the binary case, the sparse form of these primes simplifies and speeds up operations in the coefficient field.

2.4 Scalar Multiplication

As already mentioned, cryptosystems relying on discrete logarithm on elliptic curves make a consistent use of *scalar multiplication*. Given a public point P on the elliptic curve, and a secret scalar k, this operation consists in computing kP.

Let us write $k = \sum_{i=0}^{n-1} k_i 2^i$. The most basic algorithm that computes kP given P and a "black-box" implementation of the group law is the following:

```
[double-and-add from MSB to LSB]
INPUT: P in C
R=0
for i from n-1 to 0
     R <- 2R
     if k_i=1
          R <- R+P
end for;
RETURN R
```

3 Side Channel Attack and Common Countermeasures

3.1 Classes of Attacks

SPA: Simple Power Analysis applies when the sequence of operations performed during some computation depends on a secret value. When the operations used are sufficiently complex, they can be easily detected by physical measures and the sequence of operations performed can be retrieved.

For instance, in a *double-and-add* algorithm, an addition is performed only if the corresponding bit of k is set to 1. Assuming that doubling and adding have noticeably different power consumption signatures, one observation of a power consumption curve can be enough to extract the secret exponent value.

DPA: Differential Power Analysis was introduced by [3] on DES implementations, but it applies to public-key cryptography as well.

For DPA to work, some intermediate value v manipulated by a cryptographic device must depend on known input and output values and on a few secret bits. The power consumption of some operation manipulating v is measured for several input values. To each value k of the secret bits involved corresponds a partition of the input and output messages into subsets leading to the same value for v. A guess for k can be checked as follows: if the value of k is correct, averaging the power consumption inside these subsets should yield noticeably different results among subsets. If it is wrong, results should be roughly identical no matter the subset chosen.

Goubin-Style Attacks. L. Goubin [6] first noticed that some properties of intermediate values may be invariant under randomization. For example, if a coordinate of some projective point representation is zero, it remains equal to zero whatever the randomization applied. If such a remarkable property can be detected, an attack can be built as follows: input values are chosen so that a remarkable value appears during the computation only if some hypothesis about a secret is correct. The measure then allows the attacker to test his hypothesis.

The attack we present follows this framework.

3.2 Countermeasures

Many countermeasures have been developed to make the attacks presented in section 3.1 impractical. Most widely used ones are presented here.

Regularization of the Instructions Flow. For an algorithm to be protected against SPA, its instruction flow must not depend on secret values. *Double-and-add always*, or *Montgomery ladder* [11] are examples of such algorithms:

```
[Double-and-add always from MSB to LSB]
INPUT: P in C
R[0]=0
for i from n-1 to 0
      R[0]<- 2R[0]
      R[1]<- R[0]+P
      R[0]<- R[k_i]
end for;
RETURN R[0]

[Montgomery ladder]
INPUT: P in C
R[0]=0;R[1]=P
for i from n-1 to 0
      R[1-k_i]<- R[k_i]+R[1-k_i]
      R[k_i]<- 2R[k_i]
end for;
RETURN R[0]
```

Randomization of Data Representation. is targeted at DPA. If the representation of temporary values is randomized, an intermediate value does not depend only on inputs and key bits, but also on some random data out of control of the attacker. Consequently, aggregating measures is no longer possible.

In the case of scalar multiplication, expressing a point in randomized projective (or Jacobian) form, as suggested by Coron [4], is a common instantiation of this countermeasure.

Randomization of the computation flow In order to prevent Goubin-style attacks, randomness can be introduced in the sequence of operations performed. Here are two examples of techniques applied to the scalar multiplication $P \to kP$:

- **Point blinding** The hardware computes $k(P + R)$ and kR, for some random point R, separately [4] or together using a trick due to Shamir [7].
- **Random exponent** If q is the order of the underlying group, then $qP = 0$. Therefore if $(k + rq)P$ is computed instead of kP for some random value r, the final result is unchanged, but the binary representation of the secret key is scrambled by the addition of rq all along the computation.

4 The Attack: Theory

4.1 Assumptions on the Target Device

We aim at retrieving the n-bit secret scalar k stored in a cryptographic device performing scalar multiplication $P \to kP$ for any point P of our choice, on an elliptic curve whose coefficient field is defined by a sparse polynomial for the binary field case or a "sparse" prime for the prime field case (see 2.3).

An element e in the coefficient field can always be written $e = \sum_{i=0}^{n-1} e_i u^i$ with $u = 2$ if $\mathbb{K} = \mathbb{F}_p$, and $u = x$ if $\mathbb{K} = \mathbb{F}_{2^n} = \mathbb{F}_2[x]$. Since we will observe Hamming weights during the attack, we assume that our target crypto device represents e in the standard way by the binary string $\{e_i\}$.

The secret scalar, on the other hand, is an object of $\mathbb{Z}/q\mathbb{Z}$ where q is the number of elements of the chosen elliptic curve group. We will write

$$k = \sum_{i=0}^{n-1} k_i 2^i$$

The attack we propose applies to implementations having the following properties:

- Points are represented with randomized projective coordinates.
- No randomization of the computation flow is performed.

We focus on *double-and-add always* from the MSB to the LSB or on the Montgomery ladder. However, the particular choice of the scalar multiplication algorithm used is irrelevant, and we target more generally algorithms that perform one computation step per exponent bit. We suppose that in step j the point $K_j P$ is manipulated, with

$$K_j = \sum_{i=n-1-j}^{n-1} k_i 2^{i-(n-1-j)}$$

On the measurement side, we assume we have access to the Hamming weights of the values manipulated, up to some noise.

4.2 Overview of the Attack

Suppose that some special point P_0 can be distinguished from a random point, for example by power analysis. Since we assumed in section 4.1 that on input P and during step j, the multiplication algorithm manipulates

$$K_j.P = \left(\sum_{i=n-1-j}^{n-1} k_i 2^{i-(n-1-j)} \right) P$$

asking for the computation of $k.(1/K_jP_0)$ makes P_0 appear at the j-th step of computation. Because $K_j = 2K_{j-1}+k_{n-1-j}$, assuming K_{j-1} is known, the value of the next unknown bit k_{n-1-j} can be recovered as follows:

> Assume that $k_{n-1-j} = 0$ and that consequently $K_j = 2K_{j-1}$. Observe the computation of $k\,(1/K_j)P_0$. If P_0 is detected at step j, the hypothesis on bit k_{n-1-j} was correct. Otherwise, $k_{n-1-j} = 1$ and $K_j = 2K_{j-1} + 1$.

The above applies for $j = 0$ as well with $K_{-1} = 0$.

For each bit, several computations might be performed to improve the reliability of the guess of k_{n-j}. Then, by iterating this algorithm, the whole secret k can be extracted.

4.3 Using Hamming Weights to Build a Distinguishable Point

We choose a point of the form

$$P_0 = (u^\lambda, y)$$

in affine coordinates, with λ as small as possible. Its representation in projective form is $P_0 : (X = u^\lambda Z, Y = yZ, Z)$ for some random $Z \in \mathbb{K}^*$.

For each value of λ we can expect that there is a point with abscissa u^λ with probability $1/2$: in $\mathbb{F}_p$, this is the case if and only if $2^{3\lambda}+a2^\lambda+b$ is a square, while in $\mathbb{F}_{2^n}$, it depends on whether the polynomial $p(y) = y^2 + x^\lambda y + x^{3\lambda} + ax^{2\lambda} + b$ has roots. For all NIST curves, λ can be chosen ≤ 5.

Detecting the Distinguishable Point. Because of the form of common coefficient fields such as NIST fields, we show in sections 4.4 and 4.5 that for a random Z, $X = u^\lambda Z$ is close to Z rotated by λ bits on the left ($Z <<< \lambda$), therefore

$$U = \mathsf{Ham}(X) - \mathsf{Ham}(Z)$$

is small. At the opposite, for a random point where coordinates are uncorrelated, U has mean 0 and variance $\mathsf{V}(U) = 2(n/4) = n/2$. Therefore,

> We measure U to discriminate P_0 from a random point.

As usual, increasing the number of experiments decreases the error probability; several scalar multiplications lead to as many observations of U as necessary. Statistical tests can then be performed as described in section 5.1 to make a decision according to the observations.

Now, let us estimate the Hamming distance between $u^\lambda Z$ and $(Z <<< \lambda)$ on both fields types.

4.4 Binary Fields

Let $P(x) = 1+x^n+\sum_{i=1}^{I} x^{m_i}$ with $1 \leq m_i < m_{i+1} < n$ be a primitive polynomial over $\mathbb{Z}_2[X]$ of degree n. Let $e = n-\deg(P(x)-x^n) = m_I$. We assume that $e > \lambda$; this is true for NIST curves which satisfy $e > n/2$. More generally, multiplication optimization in $\mathbb{F}_{2^n} = \mathbb{F}_2[x]/(P)$ commands to choose e large.

Let $Z \in \mathbb{F}_{2^n}$. Remember that elements of $\mathbb{F}_{2^n}$ are represented in the usual polynomial base. For $\lambda < e$, set $Z = Z_1 + x^{n-\lambda} Z_2$ with $\deg(Z_1) < n - \lambda$ and $\deg(Z_2) < \lambda$: $(Z <<< \lambda) = x^\lambda Z_1 + Z_2$.

$x^\lambda Z$ and $(Z <<< \lambda) \bmod P$ are related by:

$$\begin{aligned} x^\lambda Z &\equiv x^\lambda Z_1 \oplus x^n Z_2 \\ &\equiv x^\lambda Z_1 \oplus (x^n - P) Z_2 \\ &\equiv (Z <<< \lambda) \oplus Z_2 \oplus (x^n - P) Z_2 \\ &\equiv (Z <<< \lambda) \oplus \sum_{i=1}^{I} x^{m_i} Z_2 \end{aligned}$$

Since $\lambda < e$, the above result is the reduced expression of the difference mod P. Each term $x^{m_i} Z_2$ is a λ-bit pattern that can affect at most a λ-bit window of the difference. Therefore at most $I\lambda$ bits differ from Z and $x^\lambda Q$.

Under the assumption that the λ-bit windows do not overlap, the exact computation of the probability law of $\mathsf{Ham}(x^\lambda Q) - \mathsf{Ham}(Q)$ can be carried out; this is useful to improve the attack (see section 5.1, Neyman-Pearson). The computation is performed in appendix A.1. The non-overlapping assumption is satisfied for the NIST curves P_{233} and P_{409}.

4.5 Prime Fields

We work here in $\mathbb{F}_p$ with p is prime. This case is more complex than the binary case because of the carry propagations that occur while adding values mod p.

Let e be the the greatest integer such that $2^n - 1 - p < 2^{n-e}$. For all NIST curves, $e \geq 32$. Distinguished points for curves on prime fields satisfy $\lambda \leq 3$: thus we always have $e - \lambda \geq 29$.

Let $Z \in \mathbb{F}_p$, $Z = Z_1 + 2^{n-\lambda} Z_2$, with $Z_1 < 2^{n-\lambda}$ and $Z_2 < 2^\lambda$.

$(Z <<< \lambda) = 2^\lambda Z_1 + Z_2$ and

$$\begin{aligned} 2^\lambda Z &\equiv 2^\lambda Z_1 + 2^n Z_2 \,[p] \\ &\equiv 2^\lambda Z_1 + (2^n - p) Z_2 \,[p] \\ &\equiv (Z <<< \lambda) + \Delta \,[p] \quad \text{with } \Delta = (2^n - 1 - p) Z_2 \end{aligned}$$

Since $Z_2 < 2^\lambda$ and $2^n - 1 - p \leq 2^{n-e}$, $\Delta < 2^{n-(e-\lambda)}$. Since $p \geq 2^n - 2^{n-e}$, with probability around $1 - 2^{e-\lambda} \geq 1 - 2^{-29}$, $(Z <<< \lambda) + \Delta$, viewed as an integer, is reduced mod p (i.e. it lies in the interval $[0, p-1]$). We can thus forget reduction mod p and study the effect of adding Δ to $(Z <<< \lambda)$ in $\mathbb{Z}$.

Sparse primes like NIST primes satisfy relations of the form $2^n - 1 - p = \sum_{i=1}^{I} \varepsilon_i \, 2^{m_i}$, with I small and $\varepsilon_i = \pm 1$ (see section 2.3; in the NIST case, $I \leq 3$). Therefore $\Delta = \sum_{i=1}^{I} \varepsilon_i \, 2^{m_i} \, Z_2$. Δ is composed of I λ-bit blocks; we now assume as in the binary case that these blocks do not overlap, and this hypothesis is fulfilled for all NIST curves mod p.

On average, carries beyond λ-bit blocks of multiples of Z_2 ("block carries") do not change $U = \mathsf{Ham}(Z <<< \lambda) - \mathsf{Ham}(2^\lambda Z)$, and have a small influence on $\mathsf{V}(U)$ as shown in appendix A.2. Since inside each block the Hamming weight is not changed on average, $\mathsf{E}(U) = 0$ as in the binary case. Excluding the block carries, at most $I\lambda$ bits differ between $2^\lambda Z$ and Z.

5 The Attack: Practice

5.1 Statistical Tests

During the course of the attack, we target some specific bit k_{n-j} manipulated during step $j+1$. We compute m times $k.(1/(2K_j)P_0)$ and collect m measures U_i, $1 \leq i \leq m$, of U. We must then choose a guess for k_{n-j} depending on $S = (U_1, \ldots, U_m)$. Let $\mathcal{D}_h$ be the law of U if $k_{n-j} = h$, $\mathsf{P}_{\mathcal{D}_0}(U = k) = p_{k,0}$ and $\mathsf{P}_{\mathcal{D}_1}(U = k) = p_{k,1}$.

Neyman-Pearson Test. It is well known from the Neyman-Pearson lemma that the test that has the smallest error probability if both hypothesis on k_{n-j} are equally likely, consists in computing the probability of the sample S observed according to both hypotheses, and to select the hypothesis $k_{n-j} = h$ for which the probability of the sample is the highest; this is the hypothesis that explains best the observed value. Knowing the $p_{k,h}$, one can compute the probability of S under hypothesis h through

$$\mathsf{P}_{\mathcal{D}_h}(S) = \mathsf{P}_h = p_{U_1,h} p_{U_2,h} \ldots p_{U_m,h} \tag{2}$$

Test Based on a Variance Estimator. While the Neyman-Pearson test on S is optimal, it requires the exact knowledge of $\mathcal{D}_0$ and $\mathcal{D}_1$. A slightly less efficient, but simpler test consists in estimating the variance of S. If $k_{n-j} = 0$, $\mathsf{V}(U) = V_0 = (I\lambda)/2$ (binary case) or $(I(\lambda+1))/2$ (prime case), whereas if $k_{n-j} = 1$, $\mathsf{V}(U) = V_1 = n/2$.

After m experiments, $\mathsf{V}(U)$ is estimated by $\overline{V} = \frac{1}{m}\sum_{i=1}^{m} U_i^2$. The probability P_h that $\overline{V}$ takes some specific value under $\mathcal{D}_h$ is then computed by approximating both laws $\mathcal{D}_0$ and $\mathcal{D}_1$ by normal laws[1]: the law of $\overline{V}$ under $\mathcal{D}_h$ is approximated by V_h/m times a χ_2 with m degrees of freedom. The Neyman-Pearson decision rule is then used *on* $\overline{V}$: $k_{n-j} = 0$ is decided if and only if $\mathsf{P}_0 > \mathsf{P}_1$.

Necessary Number of Experiments. The error probability of the Neyman-Pearson decision rule on some function f of the observation S *for one experiment* depends on the statistical distance between $f(\mathcal{D}_0)$ and $f(\mathcal{D}_1)$

$$\sum_k |\mathsf{P}_{\mathcal{D}_0}(f(S) = k) - \mathsf{P}_{\mathcal{D}_1}(f(S) = k)|$$

and similarly, on several experiments, the distance between $f(\mathcal{D}_0) \times \ldots f(\mathcal{D}_0)$ and $f(\mathcal{D}_1) \times \ldots \times f(\mathcal{D}_1)$ could be computed. However, this is not practical. Some approximations exist, like the Kullback-Leibler distance, or the Square Euclidean Imbalance (see [2] or [5]). Very roughly, they state that for a constant error rate the number of experiments depends on the distributions like $\left(\sum_k [\mathsf{P}_{\mathcal{D}_0}(f(S) = k) - \mathsf{P}_{\mathcal{D}_1}(f(S) = k)]^2\right)^{-1}$.

Practically, we prefer adaptive strategies that estimate on the fly the error probability.

Adaptive Strategies. If m measures are performed, resulting in some observation S of probability P_h under $\mathcal{D}_h$, the probability that hypothesis h actually holds is

$$\mathsf{P}(h = 0|S) = \frac{\mathsf{P}_0}{\mathsf{P}_0 + \mathsf{P}_1} \quad \text{and} \quad \mathsf{P}(h = 1|S) = \frac{\mathsf{P}_1}{\mathsf{P}_0 + \mathsf{P}_1}$$

During a series of m experiments, m *being a fixed value*, the probability ratio $\mathsf{P}(h = 0|S)/\mathsf{P}(h = 1|S) = \mathsf{P}_0/\mathsf{P}_1$ indicates the confidence in the decision made. In the experiments we perform, some threshold $\delta > 1$ is set. We perform more experiments as long as $1/\delta < \mathsf{P}_0/\mathsf{P}_1 < \delta$. If $\mathsf{P}_0 > \delta\mathsf{P}_1$ we decide $h = 0$, and if $\mathsf{P}_1 > \delta\mathsf{P}_0$ we decide $h = 1$. Since the number of experiments is computed adaptively, experiments are no longer independent and for example (2) is not strictly true anymore. However we assume that the confidence estimation $\mathsf{P}_0/\mathsf{P}_1$ is still meaningful.

Recovering the Whole Key. Even if the error probability for each bit guess is small, since we are dealing with large secret values (at least 192 bits), the probability that at least one error occurs during the attack is high. Additionally, after one error at step j, since next experiments rely on the value of K_j, subsequent tests will fail to detect P_0 and with high probability, the next guessed bits will be equal to 1.

Of course, one way to overcome this problem is to have a very low error probability per bit. However, more subtle approaches can be devised: for example, if

[1] this is justified by the central limit theorem for $\mathcal{D}_1$; for $\mathcal{D}_0$, this can be considered as an heuristic hypothesis.

a long run of ones is guessed, one can attempt to restart from the computation step where the run begins.

5.2 Experimental Results

We simulated a Montgomery Ladder using randomized projective coordinates on the various NIST curves. We used the most basic variance estimator, with no backtracking in case of long runs of ones. We looked for the number of measurements required to guess the whole secret scalar with a success probability of 90%. No noise was added to the measurements, unlike in a real setting.

The number of measurements that had to be performed in order to reach a confidence level of 90% does not grow linearly in the size of the scalar. In fact, it depends on $I\lambda$; this is to be expected because of the expression of $\mathsf{V}(U)$ under the hypothesis $k_{n-1-j} = 0$.

Current results are summarized in table 1 below.

Table 1. Experiments Required for a 90% Confidence Level

Curve	Total number of experiments	Experiments per bit	λ	$I\lambda$
p_{192}	1117	6	2	2
p_{224}	2347	10	6	6
p_{256}	2729	11	4	12
p_{384}	2519	7	1	3
p_{521}	1305	3	n.a.	0
B_{233}	482	2	1	1
B_{283}	1854	7	5	15
B_{409}	789	2	1	1
B_{571}	2219	4	5	15

6 Conclusion

In this paper, we presented a new chosen-ciphertext Side-Channel Attack on elliptic curve scalar multiplication. It does not apply to any elliptic curve, but rather to curves whose coefficient fields are chosen to enable efficient implementations on resource-constrained hardware; unfortunately, this kind of hardware is precisely the target of choice for Side-Channel Attacks.

The attack is able to defeat some widely used countermeasures like anti-SPA scalar multiplication algorithms and projective coordinate randomization. It is stopped by more complex defenses like point blinding and scalar randomization; these countermeasures do not however come for free in hardware. The attack might also be prevented if the cryptanalyst cannot have full control over the scalar multiplication input.

Practically, basic simulations show that the attack is able to recover a secret scalar with a success rate of 90% on any NIST curve using no more than 11 power measurements per bit guessed, using a very simple statistical test. This

lead us to think that it is a practical threat that should be taken into account by implementors.

References

1. ANSI X9.62-1998. Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), 1998.
2. T. Baigneres, P. Junod, and S. Vaudenay. How Far Can We Go Beyond Linear Cryptanalysis? In *Advances in Cryptology – ASIACRYPT' 04*, volume 3329 of *LNCS*, pages 432–450. Springer-Verlag, 2004.
3. C.Kocher, J.Jaffe, and B.Jun. Differential Power Analysis. In *Advances in Cryptology – CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer-Verlag, 1999.
4. J.-S. Coron. Resistance Against Differential Analysis for Elliptic Curve Cryptography. In *Advances in Cryptology – CHES'99*, volume 1717 of *LNCS*, pages 292–302. Springer-Verlag, 1999.
5. W. Feller. *An Introduction To Probability Theory and Its Applications*. Wiley Series In Probability And Mathematical Statistics. John Wiley & Sons, 1968.
6. L. Goubin. A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In *Advances in Cryptology – PKC'03*, volume 2567 of *LNCS*, pages 199–210. Springer-Verlag, 2003.
7. K. Itoh, T. Izu, and M. Takenaka. Efficient Countermeasures Against Power Analysis for Elliptic Curve Cryptosystems. In *CARDIS*, pages 99–114, 2004.
8. N. Koblitz. Elliptic Curve Cryptosystems. In *Mathematics of Computation*, volume 48, pages 203–209. Springer-Verlag, 1987.
9. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer-Verlag, 1996.
10. V. Miller. Use of Elliptic Curve in Cryptography. In *Advances in Cryptology – CRYPTO' 85*, volume 218 of *LNCS*, pages 417–426. Springer-Verlag, 1985.
11. P. Montgomery. *Speeding the Pollard and Elliptic Curves Methods of Factorization*, volume 44. Math. Comp, 1985.
12. NIST. Recommended Elliptic Curves for Federal Government Use, 2000.
13. Standards for Efficient Cryptography Group/ Certicom Research. SEC 2: Recommended Elliptic Curve Cryptography Domain Parameters, Version 1.0, 2000. `http://www.secg.org`.

A Computation of the Probability Law of U

A.1 Binary Case

In that section, we compute the exact probability law of the Hamming weight difference between $x^\lambda Z$ and Z $<<<$ λ, under the assumption that the λ-bit windows do not overlap.

We use the same notations as in 4.4: $\mathbb{F}_{2^n} = \mathbb{F}_2[X]/(P)$ and Z is a random uniform value in F_{2^n}. For some $\lambda < n - k_I$,

$$U = \mathsf{Ham}(Z) - \mathsf{Ham}(x^\lambda Z)$$

is the random variable whose law we want to compute. We saw in section 4.4 that if $Z = Z_1 + x^{n-\lambda} Z_2$ with $\deg(Z_1) < n - \lambda$ and $\deg(Z_2) < \lambda$, then

$$x^\lambda Z \equiv (Z <<< \lambda) \oplus \sum_{i=1}^{I} x^{k_i}.Z_2$$

Set $(Z <<< \lambda) = \sum_{i=0}^{n-1} z_i x^i$. Then $Z_2 = \sum_{j=0}^{\lambda-1} z_j\, x^j$. Let U_j be the contribution of the j-th bit of Z_2, z_j, to U. Under the non-overlapping condition, $U = U_0 + \ldots + U_{\lambda-1}$ and

$$\begin{aligned} U_j &= \mathsf{Ham}(Z <<< \lambda) - \mathsf{Ham}\left((Z <<< \lambda) \oplus z_j \sum_{i=1}^{I} x^{k_i+j}\right) \\ &= z_j \sum_{i=1}^{I} (2z_{k_i+j} - 1) = z_j \left(2 \sum_{i=1}^{I} z_{k_i+j} - 2I\right) \end{aligned}$$

and for each i, j, z_j and z_{k_i+j} are independent because $k_i \neq 0$. If W is a binomial random variable $\mathcal{B}(I, 1/2)$,

$$\begin{aligned} \mathsf{P}(U_j = k) &= \frac{1}{2}\,\mathsf{P}(2W - I = k) \quad \text{if } k \neq 0 \\ \mathsf{P}(U_j = 0) &= \frac{1}{2} + \frac{1}{2}\,\mathsf{P}(2W = I) \end{aligned}$$

In particular, $\mathsf{E}(U_j) = 0$, and $\mathsf{V}(U_j) = I/2$. Now $U_0, \ldots, U_{\lambda-1}$ depend on different bits of Z and are therefore independent: the law of U is simply the law of the sum of λ independent "copies" of U_0, for example. In order to implement a Neyman-Pearson test on outcomes of U, its law can therefore be derived by computing the λ-th convolution power of the law of U_0. In order to perform variance tests, we only need $\mathsf{E}(U) = 0$, and $\mathsf{V}(U) = I\lambda/2$.

A.2 Large Prime Case

In the prime field case, we want to approximate the law of $U = \mathsf{Ham}(Z) - \mathsf{Ham}(2^\lambda Z)$, where $0 \leq Z < p$ is random and the Hamming weight is computed on reduced representations mod p.

In section 4.5, we proved that the law of U is very close to the law of

$$U' = \mathsf{Ham}(Z') - \mathsf{Ham}(Z' + \Delta)$$

with Z' a random value in $[0, 2^n - 1]$,

$$\Delta = (2^n - p - 1)(Z' \mod 2^\lambda) = \sum_{i=1}^{I} \varepsilon_i\, 2^{m_i}\, (Z' \mod 2^\lambda)$$

and $\varepsilon_i = \pm 1$. Set $Z' \mod 2^\lambda = Z'_2$. λ copies of Z'_2 are added or subtracted at I different λ-bit windows in Z. For the prime numbers we consider, $m_{i+1} - m_i \gg \lambda$ and we will therefore assume that these windows do not overlap, and even more, that no carry can propagate from one window to the other. We will handle separately bit differences occurring inside these windows and bit differences outside them, caused by carries overflowing the windows. The first category of bit differences will be enumerated by a random value U'_i, and the second one by U'_o: $U' = U'_o + U'_i$. We will assume that U'_i and U'_o are independent.

The contribution c_i of each λ-bit window to U'_i is a random binomial value satisfying $c_i/2 - 1 \sim \mathcal{B}(\lambda, 1/2)$, and these contributions are independent because they involve independent bits of Z' (and although they both involve Z'_2). Therefore $U'_i/2 - 1 \sim \mathcal{B}(I\lambda, 1/2)$.

Let us focus on the contribution c_o of a term $2^{m_i} Z'_2$ to U'_o, corresponding to a case $\varepsilon_i = 1$. With probability $1/2$, no carry occurs and $c_o = 0$. If a carry occurs, c_o contributes to U'_o in the following way:

Contribution	1	0	-1	$-i$
Probability	1/4	1/8	1/16	$2^{-(i+2)}$

For example, in the second case of the above table, two bits 01 in Z' are changed by the carry into 10.

In fact, $c_0 = b(1 - Z)$ where b is a Bernoulli variable that is equal to one if and only if a carry occurs, $\mathsf{P}(Z = i) = 2^{-i+1}$ for $i \geq 0$, and b and Z are independent. With the help of this expression, one can check that $\mathsf{E}(c_o) = 0$ and $\mathsf{V}(c_o) = 1/2$. We would have obtained the same result for $\varepsilon_i = -1$, although c_0 would be changed into $-c_0$.

Finally, in the simplified model corresponding to the assumptions we made, $\mathsf{E}(U) = 0$ and $\mathsf{V}(U) = (I(\lambda + 1))/2$. Also note that the modeling above can be used to compute the probability law of U.

Templates as Master Keys

Dakshi Agrawal[1], Josyula R. Rao[1], Pankaj Rohatgi[1], and Kai Schramm[2]

[1] IBM Watson Research Center, P.O. Box 704
Yorktown Heights, NY 10598, USA
{agrawal, jrrao, rohatgi}@us.ibm.com
[2] Communication Security Group, Ruhr-Universität Bochum
Universitaetsstrasse 150, 44780 Bochum, Germany
schramm@crypto.ruhr-uni-bochum.de

Abstract. We introduce two new attacks: the *single-bit template attack* and the *template-enhanced DPA attack*. The single-bit template attack can be used very effectively to classify even *single* bits in a single side channel sample with a high probability of correctness. The template-enhanced DPA attack, combines traditional DPA with single-bit template attacks to show that if an adversary has access to a test card with even a slightly biased RNG, then he/she can break *protected* cryptographic implementations on a target card even if they have perfect RNGs. In support of our claim, we report results from experiments on breaking two implementations of DES and AES *protected by the masking countermeasure* running on smartcards of different manufacturers.

In light of these results, the threat of template attacks, generally viewed as intrinsically difficult to mount, needs to be reconsidered.

1 Introduction

Several side channel cryptanalytic techniques, such as those based on measuring timing, power consumption and electromagnetic emanations have been used effectively to launch a wide range of attacks such as simple power analysis (SPA), differential power analysis (DPA), higher order DPA, template attacks and multi-channel attacks [Koc96, KJJ99, AARR02, CRR02, ARR03] against a wide variety of cryptographic devices. While countermeasures, even provably secure ones, have been developed for some attacks such as DPA, the perceived difficulty (in terms of the work effort required by an adversary) of launching other attacks has led developers to discount their feasibility.

This is particularly true for template attacks. For instance, the very high successful classification results that can often be achieved with the analysis of a single side channel sample, make template attacks the ideal choice to attack ciphers, such as stream ciphers, which use ephemeral keys. However, until now all published works [CRR02, RO04] used template attacks to classify the state of a byte, e.g., a key byte in RC4. This makes the process of creating templates quite tedious since 256 templates need to be created for each byte. Further, templates for the full attack cannot be precomputed as the templates for a subsequent key

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 15–29, 2005.

byte need to be created for each likely hypothesis for the earlier key bytes, i.e., the template building process can only be guided by partial attack results. In this paper, we show that this apparent difficulty is not intrinsic and present two new attack techniques to surmount it.

1.1 Contributions

Our first contribution is the *single-bit template attack* technique. For a given bit, this attack *uses DPA to build templates*. It relies on our experimental observation that templates can be built from peaks observed in a DPA attack and these templates can predict the value of a *single DPA-targeted bit* in a *single side channel sample* with high probability. Thus, even though the specific computation yielding the single sample uses byte sized variables, the template can predict a single bit from those variables.

This technique immediately yields attacks where an adversary precomputes a large number of single-bit templates using several different DPA attacks on a test device and uses these precomputed templates and their classification probabilities to attack a single sample from an identical target device. These templates provide the best guess for each of the DPA-targeted bits and the template classification probabilities can be used to guide a weighted brute force search for the key. With enough precomputed templates, the entropy of the key is reduced substantially making the weighted brute force search practical. For example, in an experiment on a DES implementation, just attacking the 32-bits of S-box output in the first round, reduced the key entropy by over 16 bits. Clearly, by building templates, for DPA attacks carried on other variables in other rounds, the key entropy could be further reduced.

Reflecting further on the single-bit template attack, it should be evident, that knowledge of a single-bit template is comparable to having some partial knowledge about the key used in a card. Possession of several such single-bit templates is akin to having a master key that can be used to break any of a collection of cards from the same mask. This is true even for cards that are protected by DPA countermeasures such as secret sharing and random masking [GP99, CJR+99, AG01], if single-bit templates for the bits being processed in such cards can be built.

The second major contribution of this paper is to introduce *template-enhanced DPA* attacks which can be used to attack DPA protected cards under some assumptions. The problem with such cards is that single-bit templates (as described earlier) cannot be built, since in principle, the DPA protection renders DPA (the first step in building single-bit templates) infeasible. However, in practice, this is not a limitation, as there are multiple ways to get hold of a test card with a (slightly) biased RNG. For example, an adversary in collusion with the designers, testers and maintainers of card software may have hooks to add code to disable specific RNG registers on their own test cards while changes to deployed cards in the field may be much more tightly controlled and impossible for an adversary. Some production cards may fail the RNG tests at fabrication time and may be discarded only to be picked up by an adversary.

In our experience, we have sometimes encountered even production cards with slight RNG bias (to the tune of 3–4%). Therefore, if cards are not tested or tested to wide tolerance limits, then it is highly likely that several cards in the field may have slightly biased RNGs. As a last resort, an attacker could mount an intrusive attack to disable the RNG on his own test card.

Given a test card with a (slightly) biased RNG, an adversary can successfully perform multiple DPA attacks on the test card to build single-bit templates. The DPA peaks in these attacks would occur at locations where the masked value of the predicted variable bit (such as an S-box output bit) occur, since the masking is imperfect. Single-bit templates built using these DPA peaks would then be able to classify corresponding bits of the masked variables used in any card, including cards that have a perfect RNG. The *template-enhanced DPA* attack works by setting the DPA selector function to be the XOR of the standard DPA selector function (e.g., an S-box output bit for a key hypothesis) and the classification obtained by the single-bit templates (such as the masked S-box output bit). Depending on the effectiveness of the template classification, this DPA selector function will have high correlation with the mask bit being used. Thus for the right key hypothesis, this attack will show DPA peaks at locations where the random mask is being used.

We demonstrate such a single-bit template attack for two DPA protected implementations: a protected DES implementation on a 6805 based smartcard and a protected AES implementation on an AVR architecture. We also report a surprising result that indicates that in practice, the bias of the RNG in the test card has little relevance to the effectiveness of the *template-enhanced DPA* attack. The RNG bias only affects the effort required to build single-bit templates. The classification error with single-bit templates built using a slightly biased RNG is not significantly worse than the classification error using templates built using a completely broken (fixed at 0) RNG.

The paper is organized as follows: In Section 2, we introduce single-bit template attacks. In Section 3, we introduce the template-enhanced DPA attacks and show how it can be uses to attack two smartcards of different architecture[1], which run protected implementations of DES and AES.

2 Single-Bit Template Attack

We extend earlier work on template attacks [CRR02, RO04] that focused on classifying a byte in a computation, e.g., a byte of key used in RC4, by showing how template attacks can be applied to classify single bits in a computation from a single side channel sample.

A template attack begins by selecting variables occurring in the computation for which templates would be built. Furthermore, it requires a selection of significant points for each of the selected variables that are included in the corresponding template. Having a good selection criterion for significant points

[1] Smartcard A is an ST19 based on the 6805 architecture and smartcard B is an Atmel ATmega163 based on the AVR architecture.

is critical to the success of template attacks and this problem has been well studied: ideally, the significant points should have high variance with respect to the particular variable of interest. For example, Bohy et al. [BNSQ03] suggest Principal Component Analysis (PCA) while Rechberger et al. [RO04] suggest a simpler and computationally less expensive approach that resembles classical DPA. For the single-bit template attack, we let DPA attacks guide the selection of both the bits in the computation for which templates are built and significant points included in these templates. Templates are built for the bits for which a DPA attack is successful and the significant points included in a template are the points with the top N highest DPA-peaks.

We illustrate the attack by means of an example. Consider an unprotected implementation of DES on smartcard A. Consider the 32 s-box output bits of the DES computation in round one. For the unprotected DES implementation, one can easily perform DPA for each of the 32 output bits. Correspondingly, we built a pair of templates for each output bit corresponding to the bit being equal to 0 and 1 respectively. In order to build these templates, we performed a DPA of each output bit using the improved DPA metric described in [ARR03] which results in a higher signal-to-noise ratio (SNR) than the standard DPA. The improved metric is computed by using the following formula:

$$M_{H_i} = \frac{\left(\mu_{H_i} - \mu_{H_v}\right)^2}{\frac{\sigma^2_{H_v,0}}{N_0} + \frac{\sigma^2_{H_v,1}}{N_1}} - \ln\left(\frac{\frac{\sigma^2_{H_i,0}}{N_0} + \frac{\sigma^2_{H_i,1}}{N_1}}{\frac{\sigma^2_{H_v,0}}{N_0} + \frac{\sigma^2_{H_v,1}}{N_1}}\right) \tag{1}$$

where μ_H is the difference of sample means of signals in the 0-bin and the 1-bin respectively for a hypothesis H. Similarly, $\sigma^2_{H,0}$ and $\sigma^2_{H,1}$ are the sample variances of the signals in the 0-bin and the 1-bin respectively for a hypothesis H. H_i denotes a hypothesis where a subkey is assumed to be i, and H_v is a special hypothesis (null hypothesis) where signals are partitioned in the 0-bin and the 1-bin randomly.

Figure 1 displays the improved metric of s-box 1, bit 0. The figure reveals several points in time that clearly correlate with the selected s-box output bit. In

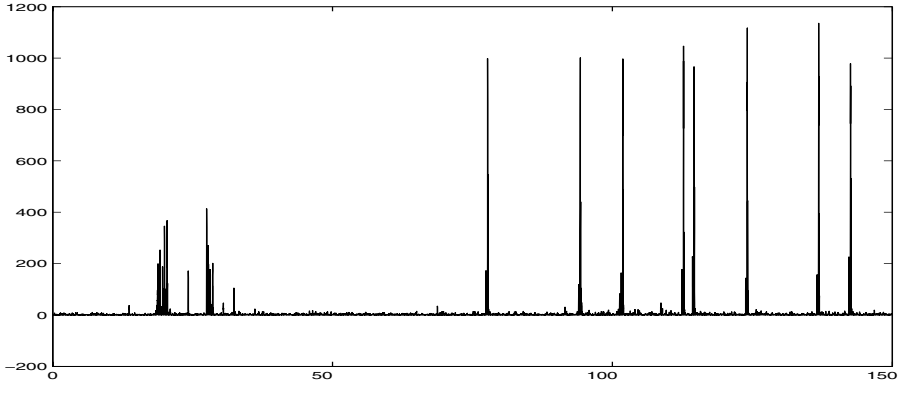


Fig. 1. Improved DPA metric of s-box 1, bit 0 of the test device. Time in μs.

Table 1. *s*-box output bit classification success rates and entropy loss

	s-box 1	**s-box 2**	**s-box 3**	**s-box 4**	**s-box 5**	**s-box 6**	**s-box 7**	**s-box 8**
bit 0	1.00	0.91	0.88	0.93	0.77	0.72	0.80	0.84
bit 1	0.98	0.88	0.92	0.94	1.00	0.92	0.97	0.77
bit 2	0.75	0.89	0.99	0.92	0.95	0.83	0.90	0.79
bit 3	0.90	0.91	0.72	0.85	0.83	0.86	1.00	0.89
entropy loss	2.57	2.10	2.13	2.30	2.28	1.50	2.61	1.35

our experiments, we chose the 50 highest peaks from this DPA metric to select significant points and built a pair of templates for these points for each *s*-box output bit using a single set of 1400 side channel samples.

To estimate classification success rate, we classified the state of the 32 *s*-box output bits using a single set of another 100 random side channel samples measured from the same device. The classification success rates $\eta_{S_i b_j}$ for the i-th s-box and j-th bit, $1 \leq i \leq 8$ and $0 \leq j \leq 3$, together with the corresponding entropy loss are shown in Table 1. The classification success rates ranged from 0.72 to 1.00; in the worst case s-box 3, bit 3 and s-box 6, bit 0 were predicted correctly for only 72 of the 100 samples. From these results, the probability that the entire 32-bit output of all s-boxes is classified correctly is $\prod_{i=1}^{8}\prod_{j=0}^{3}\eta_{S_i b_j} = 0.0154$ which although small is still 66-million times higher than a random guess.

These results can also be viewed in terms of entropy loss. For a particular bit, if the classification success rate is p, then its corresponding entropy loss is given by $1 + (1-p)\log_2(1-p) + p\log_2(p)$. To compute the entropy loss for multiple bits we can add the individual losses (this corresponds to the worst case where classification of different bits is independent). From this formula, we can see that 16.8-bits of entropy has been lost from the 48-bits of the DES key used in the first round (out of a maximum possible loss is 32-bits if the classification was perfect). The loss of entropy of the keyspace can be translated into reduced expected computational cost of a guided exhaustive search through the entire keyspace that examines more likely keys earlier than the less likely keys.

For DES implementations, the attack can be improved substantially. Templates can be built not just for round 1, *s*-box output bits but also for other bits such as the data bits fed to the second round. These templates will further narrow down the possibilities for the 48 key bits used in the first round. In addition, templates can be built for the corresponding DPA attacks on the last two rounds of DES (which utilize another 48-bit size subset of the key) and so on. Depending on the implementation, single-bit templates can also be built directly for the key bits that are likely to be highly effective since the same key bits show up in multiple locations in a round and across multiple rounds.

To summarize, single-bit template attacks are capable of classifying a single bit in a single side channel sample with high probability even though the influence of a single bit on the side channel signal is generally very little at

a particular instance of time, and is superimposed by several sources of noise including that from other adjacent bits. Cryptographic algorithms with high contamination properties [CRR02], such as DES, are ideally suited for single-bit classification. Multiple precomputed single-bit templates can lead to practical guided keyspace search algorithms using only a single sample from the target device. Moreover, single-bit attacks when combined with other attacks can result in much more devastating attacks as we show in the next section.

3 Attacking the Masking Countermeasure: Template-Enhanced DPA

The proposed attack consists of two steps: a *profiling phase* and a *hypothesis testing phase*. In the profiling phase, the adversary, who is in possession of a test card with a biased RNG, builds templates, and in the *hypothesis testing phase*, the adversary uses these prebuilt templates to mount a DPA-like attack on a target card which is identical to the test card, but has a perfect RNG.

3.1 Profiling Phase

We assume that the adversary has a test card with a biased RNG that produces 0 bits with some biased probability $\nu \neq 0.5$, and that the adversary only faces masking countermeasures such as the duplication method [GP99][2][3]. A masking countermeasure generally blinds all intermediate key-dependent variables with randomly generated masks. The original values of the intermediate variables can be recovered from their blinded values by applying the inverse mask. Non-linear functions such as the s-boxes in DES and AES cannot be dealt with this way; they are typically handled by creating masked tables in RAM. While the unmasked s-box output $s(x \oplus k)$ never occurs as a run-time variable during the execution of the algorithm, both the masked output $s(x \oplus k) \oplus m$ and the mask m do occur and thus leak in the side channel sample.

As an illustration, consider the upper plots of Figures 2 and 3 that show DPA attacks on two test cards, one with a protected DES implementation, and another with a protected AES implementation. The target of both attacks was the bit 0 of s-box 1 in round one. The differential samples were obtained by switching off the RNGs of the test cards ($\nu = 1$). Both plots show peaks at points in time when the masked s-box output bit leaks. Note that the differential trace from the AES implementation contains less peaks compared to the DES implementation due to the lower contamination properties of AES.

The first step in the profiling stage is to perform exhaustive DPA attacks on the test card using as many samples as possible. In a card with a biased RNG, where the mask is not perfectly random, such an attack will succeed since the

[2] We make this simplifying assumption just for the sake of exposition, the attacks would work if bad RNG has different biases for different bits in a random byte.

[3] We assume other countermeasures, such as the desynchronization of side channel samples due to random wait states etc., have been removed using signal processing.

DPA prediction of an algorithmic bit (e.g., s-box output bit $s(x \oplus k)$) would be correlated with the masked value of that bit. A successful DPA attack will give us the subkey k (in fact we will get all the subkeys) and also reveal the points of time t^* when targeted masked algorithmic bit (e.g., masked s-box output bit $s(x \oplus k) \oplus m$) leaks.

The second step of the profiling phase is to create single-bit templates based on each of the DPA attacks. For each DPA attack, the adversary builds a pair of templates for the masked bit being 0 and 1 by using the collected samples at the points where the DPA peaks appear. It may seem that building the template pairs will require that the adversary knows which of the N collected samples have the masked bit 0 and which have the masked bit 1. This is *not possible* in general, unless the RNG is completely broken in a known way (e.g., fixed at 0). Instead the adversary blindly assumes that the bit is exactly the same as the DPA prediction and builds the templates anyway.

Clearly, if the RNG is not fixed at 0, but has a probability ν of outputing a 0 bit, the templates built by an adversary have significant errors. For example when $\nu > 0.5$, then the 0-bit template will be built using roughly $\nu * N/2$ samples that are actually 0 samples and roughly $(1 - \nu) * N/2$ samples that are actually 1's. When $\nu < 0.5$, then the templates are inverted: the 0 template is built using more 1 samples than 0 samples. Such templates are equally useful since they will *consistently* predict the bit incorrectly with high probability. When $\nu = 0.5$, DPA will not work and the templates as described here cannot be built.

We will show later in the paper that even though significant errors are introduced in the templates when the RNG is very slightly biased, i.e., when ν is close to 0.5, *if enough signals are used to build these templates*, then the performance of *the template-enhanced DPA attack is not significantly impacted*—the attack works almost as well as an attack using perfect templates ($\nu = 1$).

3.2 Hypothesis Testing Phase

Once the adversary has built templates to classify masked s-box output bits in DES or AES using a test device with imperfect RNG, he/she is given a target device to attack that is identical to the test device, except for the fact that its RNG is perfect.

The adversary can make a hypothesis regarding the secret key k used in the target device, and for a particular side channel sample, use the key hypothesis to predict the unmasked output bit $s(x \oplus k)$. Furthermore, the adversary can use template classification to predict the masked output bit $s(x \oplus k) \oplus m$. These two together can be used to predict the mask bit m itself[4]

$$m = \underbrace{[s(x \oplus k)]}_{\texttt{prediction}} \quad \oplus \quad \underbrace{[s(x \oplus k) \oplus m]}_{\texttt{template classification}} \tag{2}$$

Since the mask bit m is an intermediate variable in the algorithm, it will leak at some instances of time in the side channel sample. The idea is to perform

[4] We assume that boolean masking is used.

a DPA-like attack on the prediction of m according to the equation above. If the hypothesized value of k is correct, peaks will show up in the corresponding differential trace at points in time when the mask bit m leaks.

The number of samples required to perform this attack depends on two main factors: the number of samples required to perform a DPA attack based on a perfect prediction for m and the template classification error probability ϵ. The first factor is a function of the leakage properties of m in the smart-card, while the second factor is dependent on the quality of single-bit templates. A higher value of ϵ results in worse SNR of the differential sample since classification errors make the predicted and actual values of the mask m less correlated. To estimate the impact of classification error ϵ, we modified the SNR model proposed by Messerges et al. in [MDS99] to account for the additional noise caused by the misclassification (details are given in the Appendix). Table 2 shows the impact of ϵ on the proposed template-enhanced DPA attack. This table assumes that a certain SNR ratio is obtained using 100 side-channel samples with perfect classification and computes how many side-channel samples would be needed to achieve the same SNR with different values of ϵ. Given the classification results obtained for single-bit templates in the earlier section, where all error probabilities were less than 0.3 and many were under 0.1, it would be reasonable to assume that the template-enhanced DPA attacks would be a factor of 1.5 to 6 more expensive (in terms of the number of required samples) than the regular DPA attacks.

Table 2. Number of measurements N required to achieve a constant SNR in a template-enhanced DPA attack for different template classification errors ϵ

ϵ	0.00	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.48	0.49
N	100	123	156	204	278	400	625	1,111	2,500	10,000	62,500	250,000

3.3 Results

We performed the proposed template-enhanced DPA attack on two smardcards: a protected DES implementation on the smartcard A and a protected AES implementation on the smartcard B. For each smartcard, in the profiling phase, the templates were built with the RNG turned off ($\nu = 1$). In the hypothesis testing phase, traces were obtained with the RNG on and working perfectly ($\nu = 0.5$). For the smartcard A, the lower plot in Figure 2 shows the differential trace of the template-enhanced DPA attack on the hypothesized mask bit m. A similar differential trace for the smartcard B is shown in the lower plot of Figure 3. Both plots contain distinct peaks even though the masking protection was fully functional. For completeness, Figure 4 shows a template-enhanced DPA trace for a false key hypothesis for smartcard B, which shows no peaks.

If the RNG in the test card during the profiling phase is just slightly biased instead of being broken, then the templates obtained from the test card would

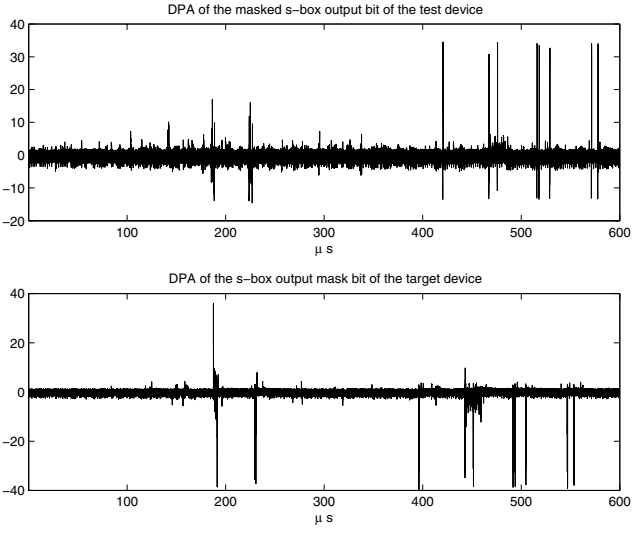


Fig. 2. Smartcard A: DPA of the masked s-box output bit using the test device and DPA of the mask bit using the target device

have significant cross-contamination. One may conjecture that as a result, the probability of error ϵ would be higher as the bias in RNG becomes smaller. However, this is not the case—in the appendix, we prove the following counter-intuitive result.

Theorem 1. *If the noise covariance matrix of side channel traces is the same for two values of a mask bit and enough traces are available from a test card with a biased RNG* ($0.5 < \nu < 1.0$)*, then the templates prepared from such traces give the same probability of error as the templates obtained from a test-device with broken RNG* ($\nu = 1$)*.*

In our experiments, we found that the noise covariance matrices of side channel traces for different values of mask bit are nearly the same. For the actual covariance matrices obtained in one of our experiments, we performed a Monte Carlo simulation of how well the signal classification works when templates are built using different numbers of samples from the test card with different RNG biases. In this simulations, the samples were generated by sampling from the noise probability distributions and the RNG bias was simulated by randomly misclassifying samples into the bins used to build templates. We also performed an actual experiment where 1000 samples were obtained from the test card and templates were build for different RNG biases (again simulated by putting samples randomly in incorrect bins). The results of these experiments are shown in Figure 5. Three plots are derived from the Monte Carlo simulation involving 1000, 10,000, and 100,000 traces from a simulated test card with biased RNG to build templates. These three plots show that as the number of traces from the test card increases, the probability of classification error becomes insensitive to

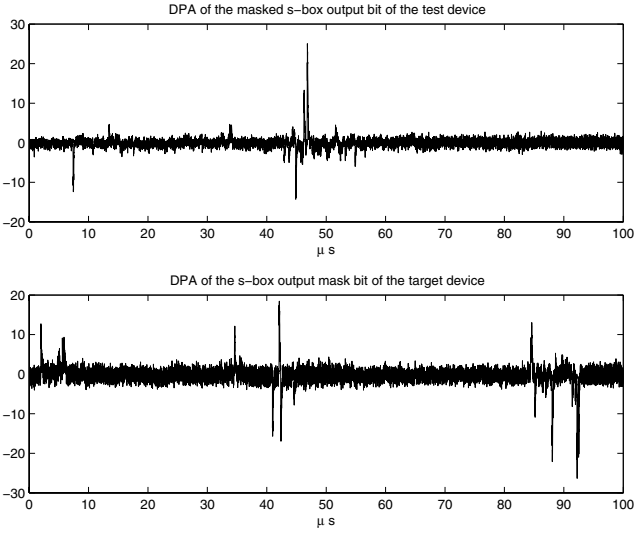


Fig. 3. Smartcard B: DPA of the masked s-box output bit using the test device and DPA of the mask bit using the target device

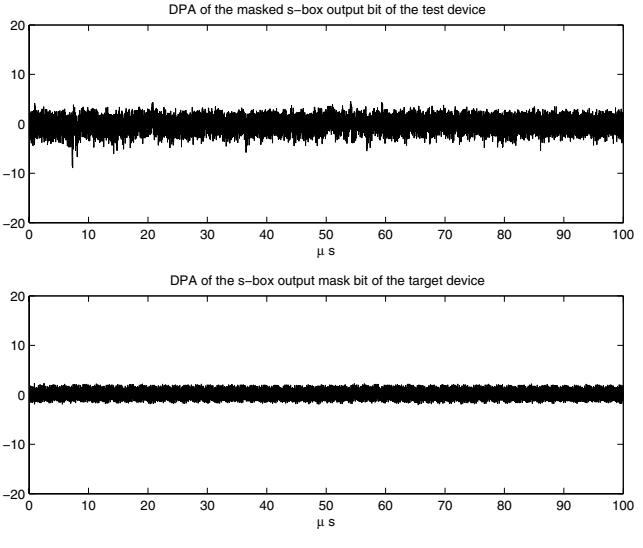


Fig. 4. Smartcard B: DPA of the masked s-box output bit using the test device and DPA of the mask bit using the target device (both with wrong hypothesis)

the RNG bias. The fourth plot is the experimental using 1000 samples from a test card to build templates. The experimental curve is in excellent agreement with our analytical results.

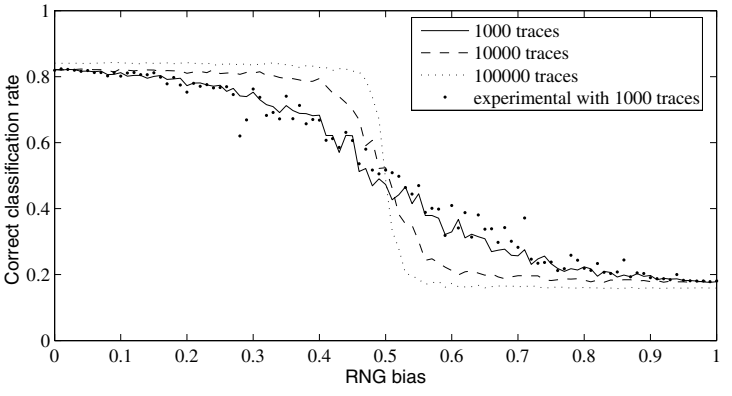


Fig. 5. Probability of correct classification versus RNG bias.

In summary, even with a test card with very small RNG bias, it is possible to mount template-enhanced DPA attacks; the only effect of a small bias is that many more samples are needed to build templates that are as good as template built from a card with completely broken RNG.

Acknowledgments: We would like to thank Helmut Scherzer for providing us with a protected DES implementation with a switchable RNG on smartcard A and Andreas Krügersen for the AES implementation with switchable RNG on smartcard B.

References

[AARR02] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM Side – Channel(s). In B.S. Kaliski, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2002*, volume 2535, pages 29–45. Springer-Verlag, 2002.

[AG01] M.-L. Akkar and C. Giraud. An Implementation of DES and AES Secure against Some Attacks. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001*, volume LNCS 2162, pages 309–318. Springer-Verlag, 2001.

[ARR03] D. Agrawal, J. R. Rao, and P. Rohatgi. Multi-channel Attacks. In C. D. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2003*, volume 2779, pages 2–16. Springer-Verlag, 2003.

[BNSQ03] L. Bohy, M. Neve, D. Samyde, and J.-J. Quisquater. Principal and Independent Component Analysis for Crypto-systems with Hardware Unmasked Units. In *e-Smart 2003*, 2003.

[CJR+99] S. Chari, C. S. Jutla, J. R. Rao, , and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *Advances in Cryptology — CRYPTO '99*, volume LNCS 1666, pages 398 – 412. Springer-Verlag, August 1999.

[CRR02] S. Chari, J.R. Rao, and P. Rohatgi. Template Attacks. In B.S. Kaliski, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2002*, volume LNCS 2523, pages 13–28. Springer-Verlag, 2002.

[GP99] L. Goubin and J. Patarin. DES and Differential Power Analysis: the Duplication Method. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 1999*, volume LNCS 1717, pages 158–172. Springer-Verlag, 1999.

[KJJ99] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis: Leaking Secrets. In *Advances in Cryptology — CRYPTO '99*, volume LNCS 1666, pages 388–397. Springer-Verlag, 1999.

[Koc96] P. Kocher. Timing Attacks on Implementations of Diffie-Hellmann, RSA, DSS, and Other Systems. In *Advances in Cryptology — CRYPTO '96*, volume LNCS 1109, pages 104–113. Springer-Verlag, 1996.

[MDS99] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Investigations of Power Analysis Attacks on Smartcards. In *USENIX Workshop on Smartcard Technology*, pages 151–162, 1999.

[MDS02] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Examining Smart-Card Security under the Threat of Power Analysis Attacks. *IEEE Transactions On Computers*, 51(4):1–12, April 2002.

[RO04] C. Rechberger and E. Oswald. Practical Template Attacks. In *Workshop on Information Security Applications — WISA 2004*, August 2004.

[Tre68] H. L. Van Trees. *Detection, Estimation, and Modulation Theory, Part I.* John Wiley & Sons. New York, 1968.

A Sensitivity of Probability of Success on Bias

Let H_0 and H_1 denote two hypotheses corresponding to the target bit being equal to 0 and 1 respectively. Let p_{H_0} and p_{H_1} model the distribution of captured side-channel emanations under H_0 and H_1, respectively. Assume that p_{H_0} and p_{H_1} are multivariate Gaussian distributions [CRR02, ARR03] with means $\mathbf{m}_0$ and $\mathbf{m}_1$, and variances $\mathbf{\Sigma}_0$ and $\mathbf{\Sigma}_1$, respectively.

Let α be the mixing factor, that is, the samples collected for H_0 are distributed according to the Gaussian mixture distribution $(1-\alpha)p_{H_0} + \alpha p_{H_1}$, and the samples collected for H_1 are distributed according to the Gaussian mixture distribution $(1-\alpha)p_{H_1} + \alpha p_{H_0}$. As a result of mixing, the mean and covariance of samples collected for H_0 is given by

$$\widetilde{\mathbf{m}}_0 = \int \mathbf{s}\Big((1-\alpha)p_{H_0}(\mathbf{s}) + \alpha p_{H_1}(\mathbf{s})\Big)d\mathbf{s} = (1-\alpha)\mathbf{m}_0 + \alpha\mathbf{m}_1 \tag{3}$$

$$\begin{aligned}\widetilde{\mathbf{\Sigma}}_0 &= \int (\mathbf{s}-\widetilde{\mathbf{m}}_0)(\mathbf{s}-\widetilde{\mathbf{m}}_0)'\Big((1-\alpha)p_{H_0}(\mathbf{s}) + \alpha p_{H_1}(\mathbf{s})\Big)d\mathbf{s} \\ &= (1-\alpha)\mathbf{\Sigma}_0 + \alpha\mathbf{\Sigma}_1 + \alpha(1-\alpha)\Delta\mathbf{m}\Delta\mathbf{m}'\end{aligned} \tag{4}$$

where A' denotes the transpose of the matrix A. We note that derivation of (4) requires tedious but straight-forward algebraic manipulations. Similarly, the mean and covariance of samples collected for H_1 is given by

$$\widetilde{\mathbf{m}}_1 = (1-\alpha)\mathbf{m}_1 + \alpha\mathbf{m}_0 \tag{5}$$

$$\widetilde{\mathbf{\Sigma}}_1 = (1-\alpha)\mathbf{\Sigma}_1 + \alpha\mathbf{\Sigma}_0 + \alpha(1-\alpha)\Delta\mathbf{m}\Delta\mathbf{m}' \tag{6}$$

During the hypothesis testing phase, an adversary would use distorted templates based on (3)–(6) to classify the target bit from a captured side-channel emanation $\mathbf{s}$. Specifically, the decision criterion is given by

$$(\mathbf{s}-\widetilde{\mathbf{m}}_0)'\widetilde{\mathbf{\Sigma}}_0^{-1}(\mathbf{s}-\widetilde{\mathbf{m}}_0) - (\mathbf{s}-\widetilde{\mathbf{m}}_1)'\widetilde{\mathbf{\Sigma}}_1^{-1}(\mathbf{s}-\widetilde{\mathbf{m}}_1) > \log(|\widetilde{\mathbf{\Sigma}}_1|) - \log(|\widetilde{\mathbf{\Sigma}}_0|) \tag{7}$$

where a decision is made in favor of H_1 if the above inequality is true, and in favor of H_0 otherwise.

By assuming $\mathbf{\Sigma}_0 = \mathbf{\Sigma}_1 = \mathbf{\Sigma}$[5], (7) can be reduced to the following [Tre68]

$$(\widetilde{\mathbf{m}}_1 - \widetilde{\mathbf{m}}_0)'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{s} > \frac{1}{2}\Big(\widetilde{\mathbf{m}}_1'\widetilde{\mathbf{\Sigma}}^{-1}\widetilde{\mathbf{m}}_1 - \widetilde{\mathbf{m}}_0'\widetilde{\mathbf{\Sigma}}^{-1}\widetilde{\mathbf{m}}_0\Big) \tag{8}$$

By using (3) and (5) along with the symmetry of inverses of covariance matrices to cancel common terms, we can further simplify (8) to

$$\Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{s} > \frac{1}{2}\Big(\mathbf{m}_1'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{m}_1 - \mathbf{m}_0'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{m}_0\Big) \tag{9}$$

Note that $\Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{s}$ is a linear combination of Gaussian variables. As a result, under the hypothesis H_0, $\Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{s}$ is Gaussian distributed with the following mean and variance

$$E[\Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{s}] = \Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{m}_0 \tag{10}$$

$$V[\Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{s}] = \Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{\Sigma}\widetilde{\mathbf{\Sigma}}^{-1}\Delta\mathbf{m} \tag{11}$$

Let $Q(x), x \geqslant 0$ denote the probability of a Gaussian random variable with mean 0 and variance 1 being larger than x. Under the hypothesis H_0 (and by symmetry, under the hypothesis H_1), the probability of error incurred by using the distorted templates is given by

$$P(\text{error}) = Q\Big(\frac{|\frac{1}{2}\big(\mathbf{m}_1'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{m}_1 - \mathbf{m}_0'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{m}_0\big) - \Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{m}_0|}{\sqrt{\Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{\Sigma}\widetilde{\mathbf{\Sigma}}^{-1}\Delta\mathbf{m}}}\Big) \tag{12}$$

We can express the numerator of $Q(\cdot)$ in the above equation solely in terms of $\Delta\mathbf{m}$ by realizing that $\mathbf{m}_1'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{m}_0$ is one dimensional and therefore it equals to its transpose $\mathbf{m}_0'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{m}_1$.

$$\begin{aligned}
&\frac{1}{2}\big(\mathbf{m}_1'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{m}_1 - \mathbf{m}_0'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{m}_0\big) - \Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{m}_0 \\
&\qquad = \frac{1}{2}\mathbf{m}_1'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{m}_1 + \frac{1}{2}\mathbf{m}_0'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{m}_0 - \frac{1}{2}\mathbf{m}_1'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{m}_0 - \frac{1}{2}\mathbf{m}_0'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{m}_1 \\
&\qquad = \frac{1}{2}\mathbf{m}_1'\widetilde{\mathbf{\Sigma}}^{-1}\Delta\mathbf{m} - \frac{1}{2}\mathbf{m}_0'\widetilde{\mathbf{\Sigma}}^{-1}\Delta\mathbf{m} \\
&\qquad = \frac{1}{2}\Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\Delta\mathbf{m}
\end{aligned}$$

[5] In our experiments, this assumption holds well.

Thus, probability of error can be expressed as

$$P(\text{error}) = Q\Big(\frac{\frac{1}{2}|\Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\Delta\mathbf{m}|}{\sqrt{\Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{\Sigma}\widetilde{\mathbf{\Sigma}}^{-1}\Delta\mathbf{m}}}\Big) \tag{13}$$

Our task is to prove that the argument of $Q(\cdot)$ in the above equation is independent of α, and therefore, the probability of error in hypothesis testing phase is independent of the RNG bias. Our strategy is to factorize the numerator and denominator of the argument of $Q(\cdot)$ in (13), and show that factors involving α cancel each other out. The first step towards this factorization is to obtain an expression for $\widetilde{\mathbf{\Sigma}}^{-1}$ in terms of $\mathbf{\Sigma}^{-1}$ by using the matrix inversion lemma. The matrix inversion lemma states that for arbitrary matrices A, U, C, and V, with the only restriction that inverses of A and C exist and the product UCV and the sum $A + UCV$ are well-defined, the following holds true

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \tag{14}$$

Substituting $A = \mathbf{\Sigma}, U = \alpha(1-\alpha)\Delta\mathbf{m}, C = 1$, and $V = \Delta\mathbf{m}'$, we obtain

$$\begin{aligned}\widetilde{\mathbf{\Sigma}}^{-1} &= (\mathbf{\Sigma} + \alpha(1-\alpha)\Delta\mathbf{m}\cdot 1\cdot\Delta\mathbf{m}')^{-1} \\ &= \mathbf{\Sigma}^{-1} - \alpha(1-\alpha)\mathbf{\Sigma}^{-1}\Delta\mathbf{m}\Big(1 + \alpha(1-\alpha)\Delta\mathbf{m}'\mathbf{\Sigma}^{-1}\Delta\mathbf{m})\Big)\Delta\mathbf{m}'\mathbf{\Sigma}^{-1}\end{aligned}$$

Let $\beta = \Delta\mathbf{m}'\mathbf{\Sigma}^{-1}\Delta\mathbf{m}$. Since β is a one dimensional quantity, it can be factored out to obtain

$$\widetilde{\mathbf{\Sigma}}^{-1} = \mathbf{\Sigma}^{-1} - \alpha(1-\alpha)(1 + \alpha(1-\alpha)\beta)\mathbf{\Sigma}^{-1}\Delta\mathbf{m}\Delta\mathbf{m}'\mathbf{\Sigma}^{-1} \tag{15}$$

Now we are ready to factor the numerator.

$$\begin{aligned}\Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\Delta\mathbf{m} &= \Delta\mathbf{m}'\mathbf{\Sigma}^{-1}\Delta\mathbf{m} - \alpha(1-\alpha)(1+\alpha(1-\alpha)\beta)\Delta\mathbf{m}'\mathbf{\Sigma}^{-1}\Delta\mathbf{m}\Delta\mathbf{m}'\mathbf{\Sigma}^{-1}\Delta\mathbf{m} \\ &= \beta(1 - \alpha\beta(1-\alpha)(1+\alpha(1-\alpha)\beta))\end{aligned} \tag{16}$$

Similarly, to factorize the denominator, we perform the following steps.

$$\begin{aligned}&\Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\mathbf{\Sigma}\widetilde{\mathbf{\Sigma}}^{-1}\Delta\mathbf{m} \\ &\quad= \Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\Big(I - \alpha(1-\alpha)(1+\alpha(1-\alpha)\beta)\Delta\mathbf{m}\Delta\mathbf{m}'\mathbf{\Sigma}^{-1}\Big)\Delta\mathbf{m} \\ &\quad= (\Delta\mathbf{m}'\widetilde{\mathbf{\Sigma}}^{-1}\Delta\mathbf{m})\Big(1 - \alpha\beta(1-\alpha)\big(1+\alpha(1-\alpha)\beta\big)\Big) \\ &\quad= \beta\Big(1 - \alpha\beta(1-\alpha)\big(1+\alpha(1-\alpha)\beta\big)\Big)^2\end{aligned} \tag{17}$$

Using (16) and (17), the numerator and denominator of (13) can be simplified to give the following expression for probability of error

$$P(\text{error}) = Q\Big(\frac{1}{2}\sqrt{\beta}\Big) \tag{18}$$

Note that since $\mathbf{\Sigma}^{-1}$ is a positive definite matrix, $\beta > 0$. Furthermore, β only depends on the statistics of emanations under H_0 and H_1. In particular, it does not depend on α.

B Impact of Classification Error on SNR

Let ν be the probability of correct classification in a DPA attack for a bit X. If $\nu \neq 1$, then the erroneous classification of the bit X can be interpreted as an additional noise source in the differential trace. If the bit X leaks at times t^* and δ denotes the average difference in amplitude of two l-bit wide operands separated by the Hamming distance one, the expected values of the zero-bit and one-bit partition are[6]:

$$E[p_i(t^*)|X = 0] = a + \frac{l-1}{2} \cdot \delta + (1 - \nu) \cdot \delta \tag{19}$$

$$E[p_i(t^*)|X = 1] = a + \frac{l-1}{2} \cdot \delta + \nu \cdot \delta \tag{20}$$

where a denotes some operand independent voltage offset and the term $\frac{l-1}{2} \cdot \delta$ denotes the average algorithmic noise caused by the remaining $l - 1$ bits of the operand. The differential trace $\Delta(t^*)$ can then be given as

$$\Delta(t = t^*) = E[p_i(t^*)|X = 1] - E[p_i(t^*)|X = 0] = (2 \cdot \nu - 1) \cdot \delta \tag{21}$$

whereas $\Delta(t \neq t^*)$ approximates zero. A possible expression of the SNR of a differential trace in DPA attacks was given by Messerges et al. in [MDS99]. We enhance their SNR description with the additional noise factor $(2 \cdot \nu - 1)$ caused by the misclassification, which yields

$$SNR = \frac{(2 \cdot \nu - 1) \cdot \delta \cdot \sqrt{N}}{\sqrt{8 \cdot \sigma^2 + \delta^2 \cdot (\alpha \cdot l + l - 1)}} \tag{22}$$

where N denotes the number of measured side channel traces, α denotes the percentage of algorithmic noise[7] at times $t \neq t^*$ and σ^2 denotes the variance of non-algorithmic time-invariant noise contained in a single trace. Let us assume that in case of perfect classification at $\nu = 1$, an adversary would have to measure $N = 100$ traces to obtain a differential trace with an acceptable SNR. From the above formula, it follows that for an arbitrary error ν the adversary will need $(\frac{10}{2\nu-1})^2$ samples to obtain the same SNR. Table 2 provides the number of traces needed for different values of $\epsilon = 1 - \nu$, using this formula.

[6] For simplicity we assume that the power signal is linear proportional to the Hamming weight of the leaked operand x, i.e. $p_i(t) = a + \delta \cdot HW(x)$.

[7] according to [MDS02] α can be often neglected.

A Stochastic Model for Differential Side Channel Cryptanalysis

Werner Schindler[1], Kerstin Lemke[2,⋆], and Christof Paar[2]

[1] Bundesamt für Sicherheit in der Informationstechnik (BSI),
Godesberger Allee 185-189, 53175 Bonn, Germany
Werner.Schindler@bsi.bund.de
[2] Horst Görtz Institute for IT Security,
Ruhr University Bochum, 44780 Bochum, Germany
{lemke, cpaar}@crypto.rub.de

Abstract. This contribution presents a new approach to optimize the efficiency of differential side channel cryptanalysis against block ciphers by advanced stochastic methods. We approximate the real leakage function within a suitable vector subspace. Under appropriate conditions profiling requires only one test key. For the key extraction we present a 'minimum principle' that solely uses deterministic data dependencies and the 'maximum likelihood principle' that additionally incorporates the characterization of the noise revealed during profiling. The theoretical predictions are accompanied and confirmed by experiments. We demonstrate that the adaptation of probability densities is clearly advantageous regarding the correlation method, especially, if multiple leakage signals at different times can be jointly evaluated. Though our efficiency at key extraction is limited by template attacks profiling is much more efficient which is highly relevant if the designer of a cryptosystem is bounded by the number of measurements in the profiling step.

Keywords: Differential Side Channel Cryptanalysis, Stochastic Model, Minimum Principle, Maximum Likelihood Principle, Power Analysis, DPA, Electromagnetic Analysis, DEMA, Template Attack.

1 Introduction

Side channel cryptanalysis exploits physical information that is leaked during the computation of a cryptographic device. The most powerful leakage consists of instantaneous physical signals which are direct responses on the internal processing. These instantaneous observables can be obtained by measuring the power dissipation or the electromagnetic emanation of the cryptographic device as a function of time. Power analysis, which was first introduced in [9] and electromagnetic analysis ([8]) are based on the dependency of the side channel information on the value of intermediate data, which is in turn caused by the physical implementation.

⋆ Supported by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT, the European Network of Excellence in Cryptology.

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 30–46, 2005.

Advanced stochastic methods have turned out to be efficient tools to optimize pure timing and combined timing and power attacks. Using such methods, the efficiency of some known attacks could be increased considerably (up to a factor of fifty), some attacks could be generalized and new attacks were conceived ([12,13,14]). The understanding of the source of an attack and its true risk potential is important for a designer of a cryptographic system for implementing effective and reliable countermeasures that prevent also privileged attacks.

This contribution gives a thorough stochastic approach to optimize the efficiency of differential side channel analysis applied against block ciphers. In our work, the quantification of side channel leakage is done in a chosen vector subspace. Under suitable conditions it requires only measurements under one test key, and even this test key need not be known. Our approach aims to achieve the efficiency of the template attacks in the key extraction phase but requires far less measurements in the profiling phase, e.g., in case of AES we guess that savings in the order of up to one hundred are feasible. This is surely interesting for designers of cryptosystems in order to assess the susceptibility of their implementations towards attacks. The mathematical model is supported by an experimental analysis of an AES implementation on an 8-bit microcontroller. Further, we show how our model can be generalized to comprehend both masking countermeasures as well as the usage of multiple physical channels.

1.1 Related Work

Differential side channel cryptanalysis identifies the correct key value by statistical methods for hypothesis testing. Differential Power Analysis (DPA) ([9]) turned out to be a very powerful technique against unknown implementations. The single measurements are partitioned accordingly to the result of a selection function that depends both on known data and on key hypotheses. [9] suggested to just use the difference of means for the two sets of single measurements. Improved statistics are the student's T-Test and the correlation method which are given in [2]. Additional guidelines for testing the susceptibility of an implementation are presented in [3].

Other contributions assume that the adversary is more powerful, e.g, that the adversary is able to load key data into the cryptographic device. Profiling as a preparation step of power analysis was first described by [6]. Probably the most sophisticated strategy is a template based attack ([4]) which aims to optimize Simple Power Analysis (SPA) and requires a precise characterization of the noise. Moreover, physical information can be captured simultaneously by different measurement set-ups, e.g., by measuring the EM emanation and the power consumption in parallel ([1]).

2 The Mathematical Model

In this section we introduce a new mathematical model for differential side channel attacks against block ciphers. We investigate this model (Subsect. 2.1)and

exploit these insights to derive optimal decision strategies (Subsects. 2.2 and 2.3). The success probability (or equivalently, the risk potential) and the efficiency of our approach are considered.

We assume that the adversary (e.g., the designer) measures physical observables at time t in order to guess a subkey $k \in \{0,1\}^s$. The letter $x \in \{0,1\}^p$ denotes a known part of the plaintext or the ciphertext, respectively. We view a measurement at time t as a realization of the random variable

$$I_t(x,k) = h_t(x,k) + R_t. \tag{1}$$

The first summand $h_t(x,k)$ quantifies the deterministic part of the measurement as far it depends on x and k. The term R_t denotes a random variable that does not depend on x and k. Without loss of generality we may assume that $E(R_t) = 0$ since otherwise we could replace $h_t(x,k)$ and R_t by $h_t(x,k) + E(R_t)$ and $R_t - E(R_t)$, respectively. We point out that (1) does not cover masking techniques. A generalization of (1) and the main results in Subsects. 2.2 and 2.3, however, is straight-forward (cf. Subsect. 2.4). From now on we assume that the plaintext is known by the adversary but our results can be directly transferred to 'known-ciphertext' attacks.

Example 1. In Sect. 3 an AES implementation targeting one S-Box is analyzed. Then t is an instant, e.g., during the first round and $x, k \in \{0,1\}^8$.

2.1 Fundamental Theorems

The central goal of Subsect. 2.2 is to estimate the distribution of the random vector $(I_{t_1}(x,k), \ldots, I_{t_m}(x,k))$ where $t_1 < \cdots < t_m$ are different instants that are part of the side-channel measurements We work out important facts that will be used in the next subsection.

Definition 1. *As usual* $\|\cdot\|\colon \mathbb{R}^n \to \mathbb{R}$ *denotes the Euclidean norm, that is* $\|(z_1, z_2, ..., z_n)\|^2 = \sum_{j=1}^{n} z_j^2$. *In this work, terms* $\boldsymbol{b}^T$ *and* A^T *stand for the transpose of the vector* $\boldsymbol{b}$ *and the matrix* A*, respectively. The term* $\widetilde{f}$ *denotes an estimator of a value* f*. Random variables are denoted with capital letters while their realizations, i.e. values assumed by these random variables, are denoted with the respective small letters.*

Mathematical Model. The random variables R_t, X and K (resp. R_t, $X_1, X_2, \ldots, X_N$, and K) are defined over the same probability space $(W, \mathcal{W}, P)$, where W is a sample space, $\mathcal{W}$ a σ-algebra consisting of subsets of W and P a probability measure on $\mathcal{W}$. More precisely, $R_t\colon W \to \mathbb{R}$; $X, X_1, \ldots, X_N\colon W \to \{0,1\}^p$ (random plaintext parts) and $K\colon W \to \{0,1\}^s$ (random subkey). By assumption, the random variables R_t, X and K (resp. R_t, $X_1, X_2, \ldots, X_N$, and K) are independent. For the sake of readability in (2), for instance, we suppress the subscript $_{X,R_t,K=k}$ as this should be obvious.

Theorem 1. *Let $k \in \{0,1\}^s$ denote the correct subkey. Then the following assertions are valid:*

(i) The minimum

$$\min_{h'\colon \{0,1\}^p \times \{0,1\}^s \to \mathbb{R}} E\left(\left(I_t(X,k) - h'(X,k)\right)^2\right) \tag{2}$$

is attained at $h' = h_t$. If $\mathrm{Prob}(X = x) > 0$ for all $x \in \{0,1\}^p$ (e.g., if X is equidistributed on $\{0,1\}^p$) the minimum is exclusively attained for $h' = h_t$.
(ii) Let $t_1 < t_2 \cdots < t_m$. Then the minimum

$$\min_{h'_1,\ldots,h'_m\colon \{0,1\}^p \times \{0,1\}^s \to \mathbb{R}} E\left(\| \left(I_{t_1}(X,k) - h'_1(X,k), \ldots, I_{t_m}(X,k) - h'_m(X,k)\right) \|^2\right) \tag{3}$$

is attained at $(h'_1, \ldots, h'_m) = (h_{t_1}, \ldots, h_{t_m})$.
(iii) For each $x \in \{0,1\}^p$ we have $h_t(x,k) = E_{X=x}\left(I_t(X,k)\right)$.

Proof. Clearly, $I_t(X,k) - h'(X,k) = \Delta h(X,k) + R_t$ with $\Delta h = h_t - h'$. Squaring both sides and evaluating their expectations yields

$$E\left(\left(I_t(X,k) - h'(X,k)\right)^2\right) = E\left(\Delta h(X,k)^2\right) + E\left(R_t^2\right) \geq E\left(R_t^2\right)$$

since $E(R_t) = 0$, and since $\Delta h_t(X,k)$ and R_t are independent by assumption. If $\mathrm{Prob}(X = x) > 0$ for all $x \in \{0,1\}^p$ then $E(\Delta h(X,k)^2) > 0$ for $h' \neq h_t$ which completes the proof of (i). Similarly,

$$\begin{aligned} &E\left(\| \left(I_{t_1}(X,k) - h'_1(X,k), \ldots, I_{t_m}(X,k) - h'_m(X,k)\right) \|^2\right) \\ &= \sum_{j=1}^{m} E\left(\left(\Delta h(X,k) + R_{t_j}\right)^2\right) \geq \sum_{j=1}^{m} E\left(R_{t_j}^2\right), \end{aligned}$$

which verifies (ii), while (iii) follows immediately from (1).

Note that Theorem 1 (ii) says that we may determine the unknown functions $h_{t_1}, \ldots, h_{t_m}$ separately although we are interested in the joint distribution of $(I_{t_1}(X,k), \ldots, I_{t_m}(X,k))$. Principally, the 2^{p+s} unknown function values $h_t(x,k)$ could be estimated separately using Theorem 1(iii). Though satisfactory from a theoretical point of view this approach is impractical.

Considering the concrete implementation a designer (resp., an adversary) should be able to determine a (small) subset $\mathcal{F}_t \subset \mathcal{F} := \{h'\colon \{0,1\}^p \times \{0,1\}^s \to \mathbb{R}\}$ that either contains the searched function h_t itself or at least a function h_t^* that is sufficiently 'close' (to be made precise below) to h_t. For simplicity we restrict our attention to the case $\mathcal{F}_t = \mathcal{F}_{u;t}$, where this set of functions is a real vector subspace that is spanned by u known functions $g_{jt}\colon \{0,1\}^p \times \{0,1\}^s \to \mathbb{R}$. More precisely,

$$\mathcal{F}_{u;t} := \{h'\colon \{0,1\}^p \times \{0,1\}^s \to \mathbb{R} \mid \sum_{\mathrm{j}=0}^{\mathrm{u}-1} \beta'_{\mathrm{j}} \mathrm{g}_{\mathrm{jt}} \text{ with } \beta'_{\mathrm{j}} \in \mathbb{R}\} \tag{4}$$

We may assume that the functions g_{jt} are linearly independent so that $\mathcal{F}_{u;t}$ is isomorphic to $\mathbb{R}^u$. In particular, the minimum on the right-hand side of (6) always exists. Theorem 2 will turn out to be crucial for the following. In the following h_t^* will always denote an element in $\mathcal{F}_{u;t}$ where (6) and (7) attain their minimum.

Theorem 2. *As in Theorem 1 let $k \in \{0,1\}^s$ denote the correct subkey.*
(i) For each $h' \in \mathcal{F}_{u;t}$ we have

$$\begin{aligned} &E\left(\left(I_t(X,k) - h'(X,k)\right)^2\right) - E\left(\left(I_t(X,k) - h_t(X,k)\right)^2\right) \\ &= E_X\left(\left(h_t(X,k) - h'(X,k)\right)^2\right) \geq 0 \end{aligned} \tag{5}$$

where $E_X(\cdot)$ denotes the expectation with respect to the random variable X, i.e. the right-hand term equals $\sum_{x\in\{0,1\}^p} \mathrm{Prob}(X = x)\left(h_t(x,k) - h'(x,k)\right)^2$.

$$(ii) \quad E_X\left(\left(h_t(X,k) - h_t^*(X,k)\right)^2\right) = \min_{h'\in\mathcal{F}_{u;t}} E_X\left(\left(h_t(X,k) - h'(X,k)\right)^2\right) \tag{6}$$

implies

$$E\left(\left(I_t(X,k) - h_t^*(X,k)\right)^2\right) = \min_{h'\in\mathcal{F}_{u;t}} E\left(\left(I_t(X,k) - h'(X,k)\right)^2\right). \tag{7}$$

(iii) Let $t_1 < t_2 \cdots < t_m$. If $h'_j \in \mathcal{F}_{t_j}$ for all $j \leq m$ then

$$\begin{aligned} &E\left(\| \left(I_{t_1}(X,k) - h'_1(X,k), \ldots, I_{t_m}(X,k) - h'_m(X,k)\right) \|^2\right) \\ &= E\left(\| \left(I_{t_1}(X,k) - h_{t_1}(X,k), \ldots, I_{t_m}(X,k) - h_{t_m}(X,k)\right) \|^2\right) + \\ &\sum_{j=1}^{m} E_X\left(\left(h_{t_j}(X,k) - h'_j(X,k)\right)^2\right). \end{aligned} \tag{8}$$

Proof. Assertion (i) can be shown similarly as Theorem 1(i) while (ii) and (iii) are immediate consequences from (i).

Remark 1.

(i) If X is equidistributed on $\{0,1\}^p$ and if we interpret $h_t(\cdot,k)$ and $h'(\cdot,k)$ as 2^p-dimensional vectors the L^2-distance $\sqrt{E_X((h_t(X,k) - h'(X,k))^2)}$ between $h_t(\cdot,k)$ and $h'_t(\cdot,k)$ equals (apart from a constant) the Euclidean distance, and $h_t^*(\cdot,k)$ is the orthogonal projection of $h_t(\cdot,k)$ onto $\mathcal{F}_{u;t}$.
(ii) It is natural to select the function $h_t^* \in \mathcal{F}_{u;t}$ that is 'closest' to h_t, i.e. that minimizes $E_X((h_t(X,k) - h'(X;k))^2)$ on $\mathcal{F}_{u;t}$. Theorem 2 says that h_t^* can alternatively be characterized by another mimimum property (7), and that the approximators $\widetilde{h}_{t_1}^*, \ldots, \widetilde{h}_{t_m}^*$ may be determined separately. Theorem 3 below provides a concrete formula to estimate the unknown coefficients $\beta_{0,t}^*, \ldots, \beta_{u-1,t}^*$ of h_t^* with respect to the base $g_{0,t}, \ldots, g_{u-1,t}$.

(iii) An appropriate choice of the functions $g_{0,t}, \ldots, g_{u-1,t}$, i.e. of $\mathcal{F}_{u;t}$, is essential for the success rate of the attack. Of course, the vector subspace $\mathcal{F}_{u;t}$ should have a small L^2-distance to the unknown function h_t. An appropriate choice may require some insight in the qualitative behaviour of the side channel observables. Clearly, $\mathcal{F}_{u_1,t} \subseteq \mathcal{F}_{u_2,t}$ implies that $h^*_{u_2,t}$ is at least as good $h^*_{u_1,t}$ but the number of measurements in the profiling phase increases with the dimension of $\mathcal{F}_{u;t}$.

Definition 2. *Let V denote an arbitrary set and let $\phi\colon \{0,1\}^p \times \{0,1\}^s \to V$ be a mapping for which the images $\phi(\{0,1\}^p \times k') \subseteq V$ are equal for all subkeys $k' \in \{0,1\}^s$. We say that the function h_t has Property (EIS) ('equal images under different subkeys') if $h_t = \overline{h}_t \circ \phi$ for a suitable mapping $\overline{h}_t\colon V \to \mathbb{R}$, i.e. $h_t(x,k)$ can be expressed as a function of $\phi(x,k)$.*

Example 2. $p = s$, $\phi(x,k) := x \odot k$ where $\odot$ denotes any group operation on $\{0,1\}^p =: V$ (e.g. '$\oplus$').

Lemma 1. *Assume that $h_t(\cdot,\cdot)$ has property (EIS). Then for any pair $(x',k') \in \{0,1\}^p \times \{0,1\}^s$ there exists an element $x'' \in \{0,1\}^p$ with $h_t(x',k') = h_t(x'',k)$.*

Proof. By assumption, $\phi(\{0,1\}^p, k) = \phi(\{0,1\}^p, k')$. Consequently, there exists an $x'' \in \{0,1\}^p$ with $\phi(x'',k) = \phi(x',k')$ and hence $h_t(x'',k) = h_t(x',k')$.

If considerations on the fundamental properties of the physical observables suggest that $h_t(\cdot,\cdot)$ meets (at least approximately) the invariance property (EIS) it is reasonable to select functions g_{jt} that allow representations of the form $g_{jt} = \overline{g}_{jt} \circ \phi$ with $\overline{g}_{jt}\colon V \to \mathbb{R}$. Then

$$h^*_t = \overline{h}^*_t \circ \phi \text{ with } \overline{h}^*_t(y) := \sum_{j=0}^{u-1} \beta_{jt} \overline{g}_{jt}(y) \tag{9}$$

(see Sect. 3.1). As an important consequence it is fully sufficient to determine $\widetilde{h}^*_t(\cdot,k) \in \mathcal{F}_{u;t}$ for any single subkey $k \in \{0,1\}^s$, which is an enourmous advantage over a pure template attack which requires 2^{p+s} templates. An advanced template attack that exploits Lemma 1 requires 2^p templates. If possible, we recommend to select plaintexts from a uniform distribution so that deviations $|h_t(x,k) - h^*_t(x,k)|$ count equally to the L^2-distance for all (x,k). Whether the invariance assumption (EIS) is really justified for $h_t(\cdot,\cdot)$ may be checked by a second profiling with another subkey.

2.2 The Profiling Phase

In this subsection we explain how to determine approximators of $h_t(\cdot,\cdot)$, or more precisely, of $h^*_t(\cdot,\cdot)$ and the distribution of the noise vector $(R_{t_1}, \ldots, R_{t_m})$. We interpret the 'relevant parts' $x_1, x_2, \ldots x_{N_1}$ (i.e. input for the function h_t) of known plaintexts as realization of independent random variables $X_1, X_2, \ldots, X_{N_1}$ that are distributed as X. The Law of Large Numbers implies

$$\frac{1}{N_1}\sum_{j=1}^{N_1}\left(i_t(x_j,k)-h'(x_j,k)\right)^2 \overset{N_1\to\infty}{\longrightarrow} E\left(\left(I_t(X,k)-h'(X,k)\right)^2\right) \tag{10}$$

with probability 1 for any $h'\colon \{0,1\}^p \times \{0,1\}^s \to \mathbb{R}$. Here $i_t(x_j,k)$ denotes the measurement at time t for curve j which has the plaintext part $x_j \in \{0,1\}^p$.

Theorem 3. *(Estimation of h_t) Again, let k denote the correct subkey. For any $h' := \sum_{j=0}^{u-1}\beta'_j g_{jt} \in \mathcal{F}_{u;t}$ we have*

$$\sum_{j=1}^{N_1}\left(i_t(x_j,k)-h'(x_j,k)\right)^2 = \|\boldsymbol{i}_t - A\boldsymbol{b}\|^2 \tag{11}$$

where $A = (a_{ij})_{1\le i\le N_1;0\le j<u}$ is a real-valued $(N_1 \times u)$-matrix, $\boldsymbol{b} \in \mathbb{R}^u$ and $\boldsymbol{i} \in \mathbb{R}^{N_1}$. More precisely, $a_{ij} := g_j(x_i,k)$, $\boldsymbol{b} := (\beta'_0,\ldots,\beta'_{u-1})^T$ and $\boldsymbol{i}_t := (i_t(x_1,k),\ldots,i_t(x_{N_1},k))^T$. Any solution $\boldsymbol{b}^* = (b_0^*,\ldots,b_{u-1}^*)^T$ of

$$A^T A\boldsymbol{b} = A^T\boldsymbol{i}_t \tag{12}$$

minimizes the right-hand side of (11). If the $(u\times u)$-matrix A^TA is regular then

$$\boldsymbol{b}^* = (A^TA)^{-1}A^T\boldsymbol{i}_t. \tag{13}$$

Due to (10) we use the approximator $\widetilde{h}_t^*(x,k) = \sum_{j=0}^{u-1}\beta_{jt}^* g_{jt}(x,k)$ with $\beta_{jt}^* := b_j^*$.

Proof. Equation (11) is obvious whereas (12) is well-known (cf. [7], Subsect. 6.2.1 with $X = A$, $Y = \boldsymbol{i_t}$ and $B = \boldsymbol{b}$; least square estimator) whereas the final assertions are obvious.

Remark 2. We already know that if h_t has the property (EIS) the profiling need only be done for one subkey k. We point out that the adversary need not even know this subkey. In fact, for a given measurement vector $\boldsymbol{i_t}$ the adversary applies Theorem 3 to all possible subkeys $k' \in \{0,1\}^s$ and computes the respective coefficient vectors $\boldsymbol{b}^{*'}$. If $k' \neq k$ Theorem 3 indeed determines an optimal function $\widetilde{h}_t^{*'} \in \mathcal{F}'_{u,t}$ which is spanned by the functions $g'_{jt}(x,k) := g_{jt}(x,k) + (g_{jt}(x,k') - g_{jt}(x,k))$ in place of the g_{jt} while the measurement vector $\boldsymbol{i_t}$ implicitly depends on the (unknown) correct subkey k. Hence it is very likely that $\mathcal{F}'_{u,t}$ has a larger L^2-distance to h_t than $\mathcal{F}_{u;t}$ and, consequently $\|\boldsymbol{i}_t - A\boldsymbol{b}^*\|^2 < \|\boldsymbol{i}_t - A\boldsymbol{b}^{*'}\|^2$ for all instances t. The adversary just adds these squared norms for each admissible subkey over several instants t, and decides for that subkey for which this sum is minimal (see Sect. 3.1 for an experimental verification). In fact, the determination of k is a by-product of the profiling phase which costs no additional measurements. At least principally, this observation could also be used for a direct attack without profiling, which yet requires a sufficient number of measurements.

Definition 3. *$\boldsymbol{R_t}$ denotes the random vector $(R_{t_1},\ldots,R_{t_m})$ in the following. Similarly, we use the abbreviations $\boldsymbol{I_t}(x,k)$, $\boldsymbol{i_t}(x_j,k)$, $\boldsymbol{h_t}(x,k)$ and $\boldsymbol{h_t^*}(x,k)$, where $\boldsymbol{t}$ stands for $(t_1,\ldots,t_m)$.*

After having determined the approximators $\widetilde{h}^*_{t_1}, \ldots, \widetilde{h}^*_{t_m}$ the adversary uses a second set that consists of N_2 measurement curves to estimate the distribution of the m-dimensional random vector $\boldsymbol{R_t} = \boldsymbol{I_t}(X,k) - \boldsymbol{h_t}(X,k)$. We point out that in general the components $R_{t_1}, \ldots, R_{t_m}$ of $\boldsymbol{R_t}$ are not independent, and unlike the functions h_{t_j} they hence cannot be guessed separately. In the most general case the adversary interpolates the N_2 vectors $\{\boldsymbol{i_t}(x_j,k) - \widetilde{\boldsymbol{h}}^*_{\boldsymbol{t}}(x_j,k) \mid j \le N_2\}$ by a smooth probability density f_0. In the experimental part of this paper we assume that the random vector $\boldsymbol{R_t}$ is jointly normally distributed with covariance matrix $C = (c_{ij})_{1 \le i,j \le m}$, i.e. $c_{ij} := E(R_{t_i}R_{t_j}) - E(R_{t_i})E(R_{t_j}) = E(R_{t_i}R_{t_j})$ since $E(R_{t_i}) = E(R_{t_j}) = 0$. If the covariance matrix C is regular the random vector $\boldsymbol{R_t}$ has the m-dimensional density $f_0 := f_C$ with

$$f_C \colon \mathbb{R}^{\mathrm{m}} \to \mathbb{R} \qquad \mathrm{f_C}(\boldsymbol{z}) = \frac{1}{\sqrt{(2\pi)^{\mathrm{m}} \det \mathrm{C}}} \mathrm{e}^{-\frac{1}{2}\boldsymbol{z}^{\mathrm{T}}\mathrm{C}^{-1}\boldsymbol{z}} \tag{14}$$

(cf. [7], for instance). Note that the adversary merely has to estimate the components c_{ij} for $i \le j$ since the covariance matrix is symmetric.

2.3 The Key Extraction Phase

By our mathematical model $\boldsymbol{I_t}(x,k) - \boldsymbol{h_t}(x,k) = \boldsymbol{R_t}$ for all $(x,k) \in \{0,1\}^p \times \{0,1\}^s$, and $E(R_{t_j}) = 0$ for each $j \le m$. If $\boldsymbol{R_t}$ has the density $f_0 \colon R^m \to [0,\infty)$ (e.g., $f_0 = f_C$ for a suitable covariance matrix C), and if k° denotes the (unknown) correct subkey of the attacked device then for each $x \in \{0,1\}^p$ we have

$$\boldsymbol{I_t}(x,k^\circ) \text{ has density } f_{k^\circ} \text{ with } f_{k^\circ}(\boldsymbol{z}) := f_0(\boldsymbol{z} - \boldsymbol{h_t}(x,k^\circ)). \tag{15}$$

After having observed N_3 measurement curves (with known parts $x_1, \ldots, x_{N_3}$) the adversary evaluates the product

$$\alpha(x_1, \ldots, x_{N_3}; k) := \prod_{j=1}^{N_3} \widetilde{f}_k\left(\boldsymbol{i_t}(x_j,k^\circ)\right) = \prod_{j=1}^{N_3} \widetilde{f}_0\left(\boldsymbol{i_t}(x_j,k^\circ) - \widetilde{\boldsymbol{h}}^*_{\boldsymbol{t}}(x_j,k)\right) \tag{16}$$

for all subkeys $k \in \{0,1\}^s$ where $\widetilde{f}_0$ denotes the approximation of the exact density f_0 that the adversary has determined in the second step of the profiling phase. Note that $\boldsymbol{i_t}(x_j,k^\circ)$ are observables that depend implicitly on the correct subkey k°. Note further that

$$\widetilde{f}_k(\boldsymbol{z}) = \widetilde{f}_0\left(\boldsymbol{z} - \widetilde{\boldsymbol{h}}^*_{\boldsymbol{t}}(x,k')\right) = \widetilde{f}_{k^\circ}\left(\boldsymbol{z} + (\boldsymbol{h_t}(x,k^\circ) - \widetilde{\boldsymbol{h}}^*_{\boldsymbol{t}}(x,k'))\right). \tag{17}$$

If the profiling phase has been successful $\boldsymbol{h_t}(x,k^\circ) - \widetilde{\boldsymbol{h}}^*_{\boldsymbol{t}}(x,k') \approx \widetilde{\boldsymbol{h}}^*_{\boldsymbol{t}}(x,k^\circ) - \widetilde{\boldsymbol{h}}^*_{\boldsymbol{t}}(x,k') \approx \boldsymbol{h_t}(x,k^\circ) - \boldsymbol{h_t}(x,k')$ and $\widetilde{f}_0 \approx f_0$. The adversary decides for k' if the term $\alpha(x_1, \ldots, x_{N_3}; k')$ is maximal (maximum likelihood principle).

We point out that the correct subkey k° also fulfils a minimum property:

$$\min_{k' \in \{0,1\}^s} E\left(\|\boldsymbol{I_t}(X,k^\circ) - \boldsymbol{h_t}(X,k')\|^2\right) = E\left(\|\boldsymbol{I_t}(X,k^\circ) - \boldsymbol{h_t}(X,k^\circ)\|^2\right). \tag{18}$$

The situation is similar to Theorem 1 where the correct function $\boldsymbol{h_t}(X, \cdot)$ attains a minimum for the given (correct) subkey. Equation (18) can be verified as Theorem 1. In fact, the left-hand terms in (18) equal $\sum_{j=1}^{m}(E_X(h_{t_j}(x, k^\circ) - h_{t_j}(x, k))^2) + E(R_{t_j}^2))$. As an alternative to the maximum likelihood approach described above the adversary may decide for that subkey $k' \in \{0,1\}^s$ that minimizes

$$\frac{1}{N_2}\sum_{j=1}^{N_2} \|\boldsymbol{i_t}(x_j, k^\circ) - \widetilde{\boldsymbol{h}}_{\boldsymbol{t}}^*(x_j, k')\|^2 \tag{19}$$

This key extraction is less efficient than the maximum likelihood approach as it (explicitly) only considers the deterministic part $\boldsymbol{h}_t$. On the other hand it saves the second part of the profiling phase which may be costly for large m (cf. Sect. 3).

To perform the overall attack the adversary subsequently applies (16) or (19) to obtain the ranking of the candidates for all subkeys. Assuming that one plaintext-ciphertext pair is known, 'candidate vectors' consisting of probable subkey candidates can be checked.

Template attacks aim at h_t itself whereas our approach estimates h_t^*. Hence the key extraction efficiency of the template attacks gives an upper bound for our approach. However, if the vector subspace $\mathcal{F}_{u;t}$ has been chosen appropriately this efficiency gap should be small, especially due to the presence of noise.

We point out that the designer may estimate the risk potential against template attacks by a stochastic simulation. If $\mathcal{F}_{u;t}$ was chosen suitably the $\widetilde{f}_{k'}$ should be close to the true densities $f_{k'}$ and in particular of similar shape. In the simulation the designer yet assumes that the estimated densities $\widetilde{f}_{k'}$ were exact, which corresponds to a template attack with large sample size.

If the attacked device processes several subkeys simultaneously, the efficiency of the overall attack can be further increased by applying a two-step stochastic sieving process, viewing the key extraction process as a sequence of statistical decision problems. The interested reader is referred to [14], Sect. 4 (see also [13], Sect. 7) where such a sieving algorithm was introduced for a timing attack on a weak AES implementation. This sieving process is applicable to hardware-based cryptographic implementations since all subkeys are processed in parallel, but it is not detailed in this contribution.

2.4 Generalizations of Our Model

Our model in equation (1) is not appropriate if the device under test applies algorithmic masking mechanisms that use (pseudo-)random numbers. However, (1) allows a straight-forward generalization. We merely have to replace $h_t\colon \{0,1\}^p \times \{0,1\}^s \to \mathbb{R}$ by $h_{b,t}\colon \{0,1\}^p \times \{0,1\}^v \times \{0,1\}^s \to \mathbb{R}$ where the second argument denotes the random number that is used for masking. Analogously to (3) the minimum

$$\min_{\boldsymbol{h}'_{b,\boldsymbol{t}}\colon \{0,1\}^p \times \{0,1\}^v \times \{0,1\}^s \to \mathbb{R}^m} E\left(\|\boldsymbol{I_t}(X, Y, k) - \boldsymbol{h}'_{b,\boldsymbol{t}}(X, Y, k)\|^2\right) \tag{20}$$

is attained at $\boldsymbol{h}_{b,\boldsymbol{t}}$ where Y denotes a random variable (independent of X and $\boldsymbol{R}_{\boldsymbol{t}}$) that models the random numbers used for masking. Under the reasonable assumption that the designer has access to these random numbers the profiling works analogously as in Subsect. 2.2, yielding a density $\widetilde{f}_{b;0}\colon \mathbb{R}^{\mathrm{m}} \to \mathbb{R}$. In Definition 2 the function ϕ is simply replaced by $\phi_b\colon \{0,1\}^p \times \{0,1\}^v \times \{0,1\}^s \to V$. Of course, in the key extraction phase knowledge of the masking random numbers $y_1, \ldots, y_{N_3}$ cannot be assumed. The designer, resp. the adversary, hence decides for the subkey k' that maximizes the product

$$\alpha_b(x_1, \ldots, x_{N_3}; k) := \prod_{j=1}^{N_3} \sum_{y' \in \{0,1\}^v} \mathrm{Prob}(y_j = y') \widetilde{f}_0\Big(\boldsymbol{i}_{\boldsymbol{t}}(x_j, y, k^\circ) - \widetilde{\boldsymbol{h}}^*_{b,\boldsymbol{t}}(x_j, y', k)\Big) \quad (21)$$

among all $k \in \{0,1\}^s$ (cf. (16)). The mixture of densities on the right-hand side expresses the fact that the true density also depends on the unknown random numbers $y_1, \ldots, y_{N_3}$. If these random numbers are unbiased and independent then $\mathrm{Prob}(Y_j = y') = 2^{-v}$ for all $j \leq N_3$ and $y' \in \{0,1\}^v$. Due to lack of space we skip a formal proof of (21). The generalized model can be used for high-order differential side-channel attacks. One possible goal is to quantify the efficiency of particular masking techniques.

Reference [1] considers the case where signals from several side-channels can be measured simultaneously. Our model can also be generalized to this situation in a natural way: We just have to replace the scalar function $h_t(x,k)$, or more generally $h_{b,t}(x,y,k)$, by the q-dimensional vector $h_{[q],b,t}(x,y,k) := (h_{1,b,t}(x,y,k), \ldots, h_{q,b,t}(x,y,k))$ where $h_{n,b,t}(x,y,k)$ quantifies the deterministic part of the n^{th} side-channel. Similarly, instead of I_t and R_t we consider q-dimensional random vectors $I_{[q],b,t}$ and $R_{[q],b,t}$ for each instant. The correct vector-valued function $\boldsymbol{h}_{[q],b,\boldsymbol{t}}$ minimizes

$$E\left(\sum_{j=1}^{m}\sum_{n=1}^{q}(I_{n,b,t_j}(X,Y,k) - h'_{n,b,t_j}(X,Y,k))^2\right) \quad (22)$$

among all $\boldsymbol{h}'_{[q],b,\boldsymbol{t}}\colon \{0,1\}^p \times \{0,1\}^v \times \{0,1\}^s \to (\mathbb{R}^{\mathrm{q}})^m$.

3 Experimental Analysis

An AES implementation on an 8-bit ATM163 microcontroller was developed for the experimental evaluation of the efficiency achieved by our new decision strategies. The AES was implemented in Assembly language and does not include any countermeasures. The side channel information was gained by measuring the instantaneous current consumption in the ground line. Four measurement series were recorded using 2000 single measurements with a different fixed AES key $\boldsymbol{k} = \{k_1, ..., k_{16}\}$ in each series. The random input data $\boldsymbol{x} = \{x_1, ..., x_{16}\}$ were chosen independently from a uniform distribution. It is $x_l \in \{0,1\}^8$ and $k_l \in \{0,1\}^8$ with $l \in \{1, ..., 16\}$.

The following list summarizes the steps in the profiling (Steps 1 to 4) and key extraction phase (Steps 5 to 7). Note that for the minimum principle Step 4 is skipped ($N_2 = 0$) and Step 6 is applied at key extraction. Instead of Step 6 the maximum likelihood principle uses Step 7.

1. Perform $N_1 + N_2$ measurements using a static key $\boldsymbol{k}$ and known data $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$.
2. With regards to the attacked device select for each instant t the functions $g_{i,t}(\cdot,\cdot)$ that span the vector subspace $\mathcal{F}_{u;t}$.
3. Choose a selection function that combines k_l and x_l and apply Theorem 3 to a subset of N_1 measurements to obtain the estimators $\widetilde{h}_t^*(\cdot,\cdot)$. (Optionally: Repeat Steps 1 to 3 for another test key $\boldsymbol{k}_2$ and compare the results in order to verify the assumption (EIS).)
4. Choose instants $t_1 < \cdots < t_m$. Use the complementary subset of N_2 measurements to obtain the density $\widetilde{f}_0 \colon \mathbb{R}^m \to \mathbb{R}$. (maximum likelihood principle only)
5. Perform N_3 measurements using the target device with the unknown static key $\boldsymbol{k}^\circ$ and known data $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$.
6. Choose instants $t_1 < \cdots < t_m$ and apply (18) and (19) to guess the correct subkey k_l° of the attacked device. (minimum principle only)
7. Apply (16) to guess the correct subkey k_l° of the attacked device. (maximum likelihood principle only)

For comparison, even when exploiting (EIS) template attacks require $2^8 \cdot N_2$ single measurements for an AES implementation.

3.1 The Profiling Phase: Estimation of h_t^*

For profiling we chose the selection function $S(\phi(x,k))$ for the AES S-Box S with $\phi(x,k) = x \oplus k$ where we suppress the byte-number indicating index l of plaintext and subkey. For the vector subspaces we tested different choices, that are evaluated regarding their efficiency in Section 3.3. The chosen vector subspace is applied to the overall time frame, i.e., we do not use a combination of several vector subspaces at different instants.

In this Section, profiling is presented in more detail for the nine-dimensional bit-wise coefficient model, referenced as vector subspace $F_9 = \mathcal{F}_{9;t}$ for all instants t. According to equation (9) with $u = 9$, Theorem 3 and Lemma 1 the deterministic side channel contribution $h_t(\phi(x,k))$ is approximated by

$$\widetilde{h}_t^*(\phi(x,k)) = b_{0t} + \sum_{i=1}^{8} b_{it} \cdot g_i(\phi(x,k)) \tag{23}$$

wherein $g_i(\phi(x,k)) \in \{0,1\}$ is the i-th bit of $S(\phi(x,k))$. The coefficient b_{0t} gives the expectation value of the non-data dependent signal part and the coefficients b_{it} with $i \neq 0$ are the bitwise data dependent signal portions. Though the internal processing of the implementation is deterministic, the measurands are not: noise is an important contribution to the physical signal. The coefficients b_{it}

are revealed by solving an overdetermined system of N_1 linear equations (see Theorem 3).

The experimental results show that the resulting coefficients b_{it} differ in amplitude, so that the use of the Hamming weight model can not be of high quality. The coefficients b_{it} were computed on all four measurement series independently. As it can be exemplary seen in Fig. 1 the deviations of coefficients revealed at the four series are relatively small. As the four series were done with different AES keys, these experimental results confirm the assumptions of Lemma 1 saying that it is justified to perform the profiling of $h_t^*(\cdot, k)\colon \{0,1\}^p \to \mathbb{R}$ for only one subkey $k \in \{0,1\}^s$.

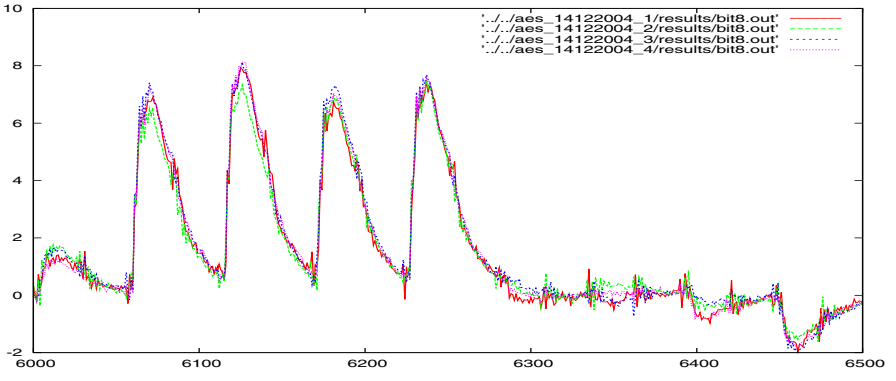


Fig. 1. Coefficient $b_{8,t}$ for all four measurement series as a function of time t. The signals of bit no. 8 (least significant bit) turned out to be the most significant ones. It is $N_1 = 2000$.

Profiling Without Knowing the Key. In case that the subkey k is unknown the estimation of h_t^* may be performed for all possible key values $k' \in \{0,1\}^8$ (cf. Remark 2 in Sect. 2.2). It was experimentally confirmed that the term $\|(i_t(x,k) - \widetilde{h}_t^{*'}(x,k'))^2\|$ indeed was minimal for the correct subkey k. By analyzing the relevant time frame of 6500 instants the difference between the first and the second candidate was 1.9 times larger than the difference between the second and the last candidate. However, we note that the usage of the correlation method [2] to determine k needs less computational efforts.

3.2 The Profiling Phase: Estimation of the Noise

The characterization of the noise was done independently of the estimation of the coefficients b_{it}. Concretely, as preparation step for the maximum likelihood principle we used $N_1 = 1000$ for the extraction of the coefficients b_{it}. The computations of the covariance matrix $C = (c_{ij})_{1 \le i,j \le m}$ for sets of m points were done with $N_2 = 1000$ and $N_2 = 5000$. For the case $N_2 = 5000$ we combined three measurement series, except for the one that is used for the key extraction later on.

3.3 The Key Extraction Phase: Minimum Principle

For the minimum principle given by equations (18) and (19) the estimation of h_t^* is needed, but not the estimation of the noise contribution. If not stated otherwise, only one measurement series served for the profiling step ($N_1 = 2000$) and the key extraction is applied at another series.

First, a suitable choice of m points in time t has to be found[1]. We used $\|b\| = \|(b_{1,t}, b_{2,t}, ..., b_{8,t})\|$ as the measure for our decision. Concretely, we chose the threshold $\tau = 30$ in the following selections for F_9.

S_1: By selecting all instants with $\|b\| \geq \tau$ we obtained seven different signals[2] and the number of instants was $m = 147$. For each signal, most instants are in series.
S_2: At each signal with $\|b\| \geq \tau$ we took the time yielding the maximum value of $\|b\|$. Here, we obtained 7 different instants.
S_3: We chose only one point in time yielding the maximum value of $\|b\|$.
S_4: We chose points that fulfill $\|b\| \geq \tau > var_t$ with $var_t := empVar(i_t(x_j, k) : j \leq N_1)$ denoting the empirical variance. Here, we obtained $m = 100$ different positions in time, but only at five different signals.
S_5: We chose points that fulfill $\|b\| \geq \tau > var_t$ yielding the same result as selection S_4 and we add additionally all points in time that fulfill $\|b\| > \tau$ at the remaining two signals. Altogether, we obtained $m = 120$.
S_6: For each of the seven signals with $\|b\| \geq \tau$ we chose three points by visual inspection, so that the instants chosen are spread over one signal. For the selection S_6 it is $m = 21$.

The minimum value of equation (19) is computed for all subkeys $k' \in \{0,1\}^8$. In this contribution we assess the efficiency by the average number of single measurements needed to achieve a certain success rate using a given number N_3 of single measurements taken from the same measurement set. The success rate (SR) was tested by ten thousand random choices of N_3 single measurements from one series. It can be seen in Table 1 that 10 single measurements yield already a success rate of about 75 % and beyond 30 single measurements the success rate can be above 99.9 %. The best results were gained at the selections S_5 and S_6.

Choice of Vector Subspaces. Different vector spaces are evaluated regarding their efficiency. The choice of high-dimensional vector spaces, e.g, by including all terms of $g_i(\phi(x,k))g_{i'}(\phi(x,k))$ $(i \neq i')$ (see (9) and (23)) did not lead to great improvements. We observed only weak contributions of second-order coefficients that even vanish at many combinations. We present results for

$F_2 = \mathcal{F}_{2;t}$ for all t: the Hamming weight model ($u = 2$),
$F_5 = \mathcal{F}_{5;t}$ for all t: a set of four bit-wise coefficients ($u = 5$) (these are the most significant bit-wise coefficients of F_9),

[1] Note, that we do not consider the covariance of the noise at the chosen points in this approach for key extraction.
[2] We assign all instants that occur during one instruction cycle to one signal.

Table 1. Success Rate (SR) that the correct subkey value is the best candidate as result of (18) and (19) by using N_3 randomly chosen measurements for the analysis at the set of instants S_1 to S_6. The vector space used was F_9

N_3	SR for S_1	SR for S_2	SR for S_3	SR for S_4	SR for S_5	SR for S_6
2	5.57 %	5.64 %	1.06 %	3.31 %	6.35 %	6.36 %
3	12.06 %	11.14 %	1.65 %	7.49 %	13.21 %	13.57 %
5	29.14 %	28.47 %	3.00 %	21.43 %	32.81 %	33.40 %
7	50.39 %	48.20 %	4.39 %	39.41 %	54.23 %	53.88 %
10	75.29 %	73.45 %	8.29 %	65.45 %	78.97 %	78.69 %
15	94.27 %	92.92 %	14.68 %	89.22 %	95.77 %	95.15 %
20	98.57 %	98.31 %	22.26 %	97.59 %	99.17 %	98.82 %
30	99.92 %	99.89 %	39.34 %	99.85 %	99.97 %	99.95 %

Table 2. Success Rate (SR) that the correct key value is the best candidate as result of (18) and (19) by using N_3 randomly chosen measurements in different vector spaces

N_3	SR for F_2 ($\tau = 1$)	SR for F_5 ($\tau = 8$)	SR for F_{10} ($\tau = 30$)	SR for F_{16} ($\tau = 70$)
2	2.59 %	4.22 %	5.18 %	4.81 %
3	4.75 %	9.03 %	11.27 %	9.73 %
5	11.63 %	21.97 %	27.28 %	23.69 %
7	21.66 %	37.61 %	47.66 %	41.04 %
10	37.77 %	62.22 %	72.94 %	65.05 %
15	62.46 %	86.36 %	93.57 %	88.69 %
20	80.36 %	95.71 %	98.41 %	96.17 %
30	96.23 %	99.74 %	99.88 %	99.81 %

$F_{10} =$ $\mathcal{F}_{10;t}$ for all t: a set of the bit-wise coefficient model and one carefully chosen second-order coefficient ($u = 10$), and
$F_{16} =$ $\mathcal{F}_{16;t}$ for all t: the bit-wise coefficient model and seven consecutive second order coefficients ($u = 16$).

For Table 2 the time instants are chosen in the same way as described for F_9 with S_1 at the beginning of Section 3.3 and the thresholds τ are indicated. High-dimensional vector spaces require more measurement curves than low-dimensional ones: There is a trade-off between the number of measurements used during profiling and the dimension of a suitable vector space. In our case, F_9 (see Table 1 and 2) seems to be a good choice though there is some space left for optimization, e.g., by using $N_1 = 5000$, $N_3 = 10$, and $\tau = 10$ the success rate of F_{10} was 80.19% and superseded the corresponding result for F_9 (77.31%). Another optimization would be to select only contributing functions $g_{i,t}(\cdot, \cdot)$ for the chosen vector subspace at the relevant instants.

Comparison with the Correlation Method. Herein, the efficiency gain of the minimum principle is compared with the correlation method of [2] on base of the same pool of measurement data. The correlation method checks for the max-

Table 3. Success Rate (SR) obtained for the correlation method using the 8-bit Hamming weight and the least significant bit (lsb-Bit) as the selection function. The last column shows the SR if the weighted estimated coefficients b_{it} using F_9 are used for the correlation.

N_3	SR (Hamming weight)	SR (lsb-Bit)	SR (estimated b_{it})
5	0.82 %	0.51 %	1.12 %
7	1.31 %	0.84 %	2.37 %
10	2.74 %	1.17 %	4.60 %
15	6.04 %	2.11 %	9.33 %
20	9.70 %	3.55 %	16.67 %
30	19.67 %	6.54 %	31.99 %
50	41.27 %	16.53 %	62.84 %
100	82.85 %	45.22 %	96.13 %

imum correlation peak obtained and it does not evaluate joined sets of multiple instants.

The success rate obtained with the correlation method is illustrated in Table 3 and can be compared with selection S_3 in Table 1 which was restricted to the same instant. In comparison, the correlation method yields worse success rates than the minimum principle. By taking, e.g., $N_3 = 10$ the minimum principle yields an improvement by a factor of 3.0 regarding the Hamming weight prediction and by a factor of 7.1 regarding the best result of one bit prediction of the correlation method. Even, if the estimated coefficients b_{it} of the minimum principle are known an improvement by a factor of 1.8 is achieved. (Note that the relative factor depends on N_3.) As the minimum principle uses the adaptation of probability densities it is advantageous if compared to the correlation method that exploits the linear relationship. Moreover, we point out that the success rate of the minimum principle increases greatly, if multiple signals are jointly evaluated.

3.4 The Key Extraction Phase: Maximum Likelihood Principle

For the maximum likelihood principle as described in Section 2.3 and equation (16) both the estimation of h_t^* and the estimation of the noise is needed. The profiling was done as described in the corresponding parts of Section 3.1 and 3.2.

The m-dimensional random vector $\boldsymbol{Z} = (I_{t_1}(X,k) - \widetilde{h}^*_{t_1}(X,k), \ldots, I_{t_m}(X,k) - \widetilde{h}^*_{t_m}(X,k))$ is assumed to be jointly normally distributed with covariance matrix C. The strategy is to decide for the key hypothesis k' that maximizes equation (16) for the multivariate Gaussian distribution using N_3 measurements which is equivalent to find the minimum of the expression $\sum_{i=1}^{N_3} \boldsymbol{z_i^T} C^{-1} \boldsymbol{z_i}$.

The analysis was done by using the vector subspace F_9 with the selections S_2 and S_6 defined at the beginning of Section 3.3. Note, that for the single instant selection S_3 the maximum likelihood principle reduces to the minimum principle.

Again, the success rate (SR) was computed using ten thousand random choices of one measurement series. As shown in Table 4, based on $N_2 = 1000$

Table 4. Success Rate (SR) that the correct key value is the best candidate as result of equation (16) by using N_3 randomly chosen single measurements for the analysis. All results are based on F_9 with $N_1 = 1000$. If not explicitly stated it is $N_2 = 1000$.

N_3	SR for S_2	SR for S_6	SR for S_2 (N_2=5000)	SR for S_6 (N_2=5000)
2	6.06 %	4.73 %	7.39 %	6.55 %
3	13.93 %	10.45 %	17.06 %	16.00 %
5	36.30 %	28.04 %	43.70 %	41.43 %
7	61.12 %	51.48 %	70.51 %	68.34 %
10	84.33 %	78.26 %	91.08 %	90.17 %
15	97.97 %	95.86 %	99.14 %	99.25 %
20	99.85 %	99.49 %	99.97 %	99.96 %
30	99.99 %	>99.99 %	>99.99%	>99.99 %

a significant improvement was achieved for the selection S_2 regarding Table 1, but not for the selection S_6. This decrease by using the maximum likelihood principle if $N_3 < 15$ and $N_2 = 1000$ for S_6 can be explained by our limited profiling process: the estimation error at the profiling of a 7×7 covariance matrix is significantly lower than the error committed for a 21×21 matrix on the base of $N_2 = 1000$. This assessment is confirmed by the corresponding columns in Table 4 for $N_2 = 5000$. Both the success rates for S_2 and S_6 were further enhanced. As result, a high value for N_2 can be crucial for the maximum likelihood principle, especially if high dimensions are used for the covariance matrix.

The maximum likelihood method needs typically twice the number of measurements during profiling. Therefore, even though key extraction is less efficient under certain circumstances the 'minimum principle' might be preferred. Given 15 measurements, it can be read out from Table 4 that the maximum probability to find the correct key value is 99.25 %. The resulting probability to decide for the correct AES key is $(0.9925)^{16} = 0.8865$.

The number N_3 of measurements can be further reduced if it is tolerated that the correct key value is 'only' among the first best candidates as result of differential side channel cryptanalyis and a plaintext-ciphertext pair is available. E.g., if the correct key value is among the first four subkey candidates with high probability, up to 2^{32} tries remain to localize the correct key value. In case of S_2 and $N_3 = 10$ the corresponding success rate that the correct subkey is at least at the fourth position of the subkey ranking is 97.58 %, if $N_2 = 1000$, and 99.42 %, if $N_2 = 5000$.

4 Conclusion

This contribution proposes a new mathematical approach to optimize the efficiency of differential side channel cryptanalysis by stochastic methods. The quantification of side channel leakage is done in a chosen vector space and does not even (necessarily) require knowledge of one test key. For the key extraction we present a 'minimum principle' that solely uses deterministic data dependencies and the 'maximum likelihood principle' that additionally incorporates the char-

acterization of the noise revealed during profiling. We have shown how our model can be generalized to comprehend both masking countermeasures as well as the usage of multiple physical channels. The theoretical predictions derived from our mathematical model are accompanied and confirmed by experiments. We conclude that the adaptation of probability densities by our methods is clearly advantageous regarding the correlation method, especially, if multiple leakage signals at different instants can be jointly evaluated. Though our efficiency at key extraction is limited by template attacks profiling is much more efficient.

References

1. D. Agrawal, J.R. Rao, P. Rohatgi: Multi-Channel Attacks. In: C.D. Walter, Ç.K. Koç, C. Paar (eds.): Cryptographic Hardware and Embedded Systems — CHES 2003, Springer, LNCS 2779, Berlin 2003, 2–16.
2. M. Aigner, E. Oswald: Power Analysis Tutorial, Technical Report, TU Graz.
3. J.-S. Coron, P. Kocher, D. Naccache: Statistics and Secret Leakage. In: Y. Frankel (ed.): Financial Cryptography (FC 2000), Springer,157–173. LNCS 1962, Berlin 2001.
4. S. Chari, J.R. Rao, P. Rohatgi: Template Attacks. In: B.S. Kaliski Jr., Ç.K. Koç, C. Paar (eds.): Cryptographic Hardware and Embedded Systems — CHES 2002, Springer, LNCS 2523, Berlin 2003, 13–28.
5. J.-S. Coron, P. Kocher, D. Naccache: Statistics and Secret Leakage. In: Y. Frankel (ed.): Finacial Cryptography - FC 2000, Springer, LNCS 1962, Berlin 2001, 157–173.
6. P.N. Fahn, P.K. Pearson: IPA: A New Class of Power Attacks. In: Ç.K. Koç and C. Paar: Cryptographic Hardware and Embedded Systems - CHES 1999, Springer, Lecture Notes in Computer Science 1717, Berlin 1999, 173–186.
7. Fang, K.-T. and Zhang, Y.-T.: Generalized Multivariate Analysis, Berlin, Springer 1990.
8. K. Gandolfi, C. Mourtel, F. Olivier: Electromagnetic Analysis: Concrete Results. In: Ç Koç, D. Naccache, C. Paar (eds.): Cryptographic Hardware and Embedded Systems - CHES 2001, Springer, LNCS 2162, Berlin 2001, 251–261.
9. P.C. Kocher, J. Jaffe, B. Jun: Differential Power Analysis. In: M. Wiener (ed.): Advances in Cryptology – CRYPTO '99, Springer, LNCS 1666, Berlin 1999, 388–397.
10. K. Lemke, K. Schramm, C. Paar: DPA on n-Bit Sized Boolean and Arithmetic Operations and Its Application to IDEA, RC6, and the HMAC-Construction. In: M. Joye and J.-J. Quisquater (eds.): Cryptographic Hardware and Embedded Systems - CHES 2004, Springer, LNCS 3156, Berlin 2004, 205–219.
11. W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery: Numerical Recipes in C — The Art of Scientific Computing. Second Edition, Cambridge University Press, 1992.
12. W. Schindler: A Timing Attack against RSA with the Chinese Remainder Theorem. In: Ç.K. Koç, C. Paar (eds.): Cryptographic Hardware and Embedded Systems — CHES 2000, Springer, LNCS 1965, Berlin 2000, 110–125.
13. W. Schindler: On the Optimization of Side-Channel Attacks by Advanced Stochastic Methods. In: S. Vaudenay (ed.): Public Key Cryptography — PKC 2005, Springer, LNCS 3386, Berlin 2005, 85–103.
14. W. Schindler, F. Koeune, J.-J. Quisquater: Improving Divide and Conquer Attacks Against Cryptosystems by Better Error Detection / Correction Strategies. In: B. Honary (ed.): Cryptography and Coding — IMA 2001, Springer, LNCS 2260, Berlin 2001, 245–267.

A New Baby-Step Giant-Step Algorithm and Some Applications to Cryptanalysis

Jean Sébastien Coron[1], David Lefranc[2], and Guillaume Poupard[3]

[1] Université du Luxembourg, Luxembourg
coron@clipper.ens.fr
[2] France Télécom, 42 rue des Coutures, F-14066 Caen, France
david.lefranc@francetelecom.com
[3] DCSSI Crypto Lab, 51 Boulevard de Latour-Maubourg, 75700 Paris 07 SP, France
Guillaume.Poupard@m4x.org

Abstract. We describe a new variant of the well known Baby-Step Giant-Step algorithm in the case of some discrete logarithms with a special structure. More precisely, we focus on discrete logarithms equal to products in groups of unknown order. As an example of application, we show that this new algorithm enables to cryptanalyse a variant of the GPS scheme proposed by Girault and Lefranc at CHES 2004 conference in which the private key is equal to the product of two sub-private keys of low Hamming weight. We also describe a second attack based on a known variant of the Baby-Step Giant-Step algorithm using the low Hamming weight of the sub-private keys.

Keywords: Baby-Step Giant-Step algorithm, discrete logarithm, GPS scheme, binary trees, low Hamming weight.

1 Introduction

In 1976, public key cryptography was introduced by Diffie and Hellman [2]. In their seminal paper, the authors originally explained how to use a mathematical assumption, namely *the discrete logarithm problem*, to obtain a key establishment protocol.

Since this first result, many identification schemes using the discrete logarithm problem have been proposed [4,8,15] and the security of this problem has been extensively studied (see [9] for a survey). Two major results are the *Baby-Step Giant-Step algorithm* due to Shanks [1] and the *rho method* due to Pollard [12]. These algorithms, used to recover discrete logarithms, are now the references to provide lower security bounds for the size of discrete logarithms since they apply as a generic method in any mathematical structure.

One of the main advantages of discrete-logarithm-based identification or signature schemes is that, when used with precomputations, they generally require only few computations for the prover or the signer so that such schemes are well designed for an integration in low cost chips. For example, in the well known Schnorr identification scheme [15], a prover using precomputations can be authenticated at the cost of one modular multiplication and one modular addition.

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 47–60, 2005.

On the opposite, in a RSA-based [14] identification schemes [3,6], the prover usually has to compute at least one modular exponentiation.

Another attractive discrete-logarithm-based identification scheme is the GPS scheme introduced by Girault [4] and proved secure by Poupard and Stern [13]. Like the Schnorr scheme, if used with precomputations, the GPS scheme is very efficient since, during the execution of the protocol, the prover has only to compute one addition and one multiplication without any modular reduction. With the appearance of new technologies, like RFID tags or more generally very low cost chips, in which even a multiplication may be too difficult to compute, embedding cryptographic protocols in such devices is now becoming a new challenge. One solution may be the use of discrete-logarithm-based schemes since they already provide efficient solutions for low cost chips. However, to better the integration of such schemes in very low cost devices, improvements are generally required and two different approaches can be distinguished.

The first one consists in designing new schemes with new computation requirements. For instance, Okamoto, Tada and Miyaji [11] proposed a new identification scheme based on the discrete logarithm problem in which a prover using precomputations has only to compute one modular reduction and one addition. However, Stern and Stern [17] proved this scheme to be insecure with the suggested parameter sizes. In addition to this cryptanalysis, they also proposed a variant of the GPS identification scheme, based on a new operation called *dove-tailing*, which is more efficient than the classical GPS scheme. Finally, Okamoto, Katsuno and Okamoto [10] suggested another variant of the GPS identification scheme in which the original multiplication can be replaced by additions of several private keys.

The second approach consists in using existing schemes but with specific parameters. A classical example may be the use of a low Hamming weight (number of non zero bits in the binary representation) discrete logarithm to decrease the computation cost of the associated exponentiation. However, Stinson [18] proposed some Baby-Step Giant-Step variants for such discrete logarithms. At CHES 2004 conference, Girault and Lefranc [5] proposed some variants of the GPS identification scheme, well designed for an integration in RFID tags. One of these variants is based on the use of specific discrete logarithms equal to products of low Hamming weight numbers. For lack of security guarantees on such private keys, the authors recalled the state-of-the-art on the different Baby-Step Giant-Step algorithms and then checked that it was not efficiently applicable to their new type of private keys. In this paper, we present a new variant of Baby-Step Giant-Step algorithm to attack such private keys. We believe that this variant is also of independent interest.

This paper is organized as follows. After recalling some useful variants of the Baby-Step Giant-Step algorithm in Section 2, we describe in Section 3 our new algorithm for discrete logarithms equal to products of sub-private keys in groups of unknown order. In Section 4, we briefly recall the GPS scheme and the Girault-Lefranc private keys. Then we present two attacks on such private keys:

the first one is an application of our new algorithm and the second one makes use of a known variant of the Baby-Step Giant-Step algorithm.

2 Baby-Step Giant-Step Algorithms

In this section, we recall Shanks' *Baby-Step Giant-Step algorithm* [1] and some useful variants. We first specify the notations we use.

Notations. Let $G = \langle g \rangle$ be a finite cyclic abelian group generated by the element g and written multiplicatively. Let n be the order of G. As a consequence, we have $G = \{g^i;\ i \in [\![0, n-1]\!]\}$. For any value v in G, *the discrete logarithm* of v in base g, denoted $\log_g v$, is the unique non-negative integer x less than n such that $v = g^x$.

The *discrete logarithm problem* is to compute $\log_g v$ given g and v.

Let ℓ denote the value $\lceil \log_2 n \rceil$. Then, the binary representation of $x = \log_g v$ requires at most ℓ bits such that we can write

$$x = \sum_{i=0}^{\ell-1} x_i 2^i,$$

where $x_i \in \{0, 1\}$ for $0 \leq i \leq \ell - 1$. The *Hamming weight* of an integer x, denoted $wt(x)$, is equal to the number of 1's in its binary representation.

Let $t < \ell$ be a positive integer. The *Hamming weight t discrete logarithm problem* is to compute $\log_g v$ given g and v with the extra information $wt(\log_g v) = t$.

2.1 The Classical Baby-Step Giant-Step Algorithm

One of the most famous and generic algorithms dealing with the discrete logarithm problem is the so-called Baby-Step Giant-Step algorithm. Introduced by Shanks [1], it is a time-memory trade-off with time complexity $\mathcal{O}(\sqrt{n})$ group multiplications.

The algorithm works as follows. Let $m = \lceil n^{1/2} \rceil$. For any given value $v \in \langle g \rangle$, $x = \log_g v$ is less than n so it can be written as $a + b \times m$ with a and b strictly less than m. From the equality $v = g^{\log_g v} = g^{a+b\times m}$, we obtain that $v \times g^{-bm} = g^a$ for some values a and b less than m. Thus, in the two following lists

$$\left(1, g, g^2, \ldots, g^{m-2}, g^{m-1}\right)$$

and

$$\left(v, vg^{-m}, vg^{-2m}, \ldots, vg^{-(m-2)m}, vg^{-(m-1)m}\right),$$

there exists at most two collisions, i.e two couples $(g^{a_0}, vg^{-b_0 m})$ and $(g^{a_1}, vg^{-b_1 m})$ such that $g^{a_0} = vg^{-b_0 m}$ and $g^{a_1} = vg^{-b_1 m}$. The value x is obtained using the couple with the smallest b_i and $x = a_i + b_i m$. The time complexity of this algorithm mainly relies on the computation of the lists both of which contains m elements. Thus, the time complexity of this algorithm is $\mathcal{O}(\lceil n^{1/2} \rceil)$ group

multiplications. However, in this generic algorithm, the space requirement is also $\mathcal{O}\big(\lceil n^{1/2} \rceil\big)$.

In order to decrease such a large space requirement, Pollard [12] proposed two randomized variants of this algorithm, known as *rho* and *lambda* methods. The generic idea is two find a linear equation over $\log_g v$. The space requirement is then very small and the expected running time of these variants is still $\mathcal{O}\big(\lceil n^{1/2} \rceil\big)$ group multiplications.

2.2 Low Hamming Weight Discrete Logarithms

In 1999, Stinson described some variants [18] of the Baby-Step Giant-Step algorithm in the case of the Hamming weight t discrete logarithm problem, i.e the computation of discrete logarithms for which the Hamming weight is known to be t.

Without loss of generality, let us assume that $\ell = \lceil \log_2 n \rceil$ is even (otherwise we consider $\ell + 1$). This algorithm relies on the concept of *splitting system*.

Definition 1 (Splitting system). *Let t and ℓ be such that $0 < t < \ell$. A (ℓ,t)-splitting system is a pair $(X,\mathcal{B})$ that satisfies:*

- *$|X| = \ell$, and $\mathcal{B}$ is a set of subsets of X, each subset having $\ell/2$ elements.*
- *$\forall\ Y \subset X$ such that $|Y| = t$, $\exists\ B \in \mathcal{B}$ such that $|B \cap Y| = t/2$.*

For example, let t and ℓ be two even integers such that $0 < t < \ell$. Let $X = [\![0, \ell - 1]\!]$ and let $\mathcal{B} = \{B_i\ ;\ 0 \leq i \leq \ell/2 - 1\}$ where for all $0 \leq i \leq \ell/2 - 1$, $B_i = \{i + j \bmod \ell\ ;\ 0 \leq j \leq \ell/2 - 1\}$. The pair $(X,\mathcal{B})$ is a (ℓ,t)-splitting system.

Thus, let $v \in \langle g \rangle$ such that $wt\big(\log_g v\big) = t$ (assumed to be even). We now use the above splitting system in the algorithm. Any element of $\langle g \rangle$ is now identified to the set of the positions of the non zero bits involved in the binary representation of its discrete logarithm. Thus, v is identified to a subset $Y \subset X$ of t elements. The goal of the algorithm is to find a decomposition of Y into two subsets of $t/2$ elements using the splitting system.

For all B_i in $\mathcal{B}$
- For all $Y_i^j \subset B_i$ of $t/2$ elements, identify the corresponding value A_i^j in G. Let $\mathcal{L}_1$ be the list of pairs (Y_i^j, A_i^j).
- Consider the set $W_i = [\![0, \ell - 1]\!] \setminus B_i$.
- For all $W_i^k \subset W_i$ of $t/2$ elements, identify the corresponding value B_i^k in G. Let $\mathcal{L}_2$ be the list of pairs $(W_i^k, v \times (B_i^k)^{-1})$.
- If two values $A_i^{j_0}$ in $\mathcal{L}_1$ and $v \times (B_i^{k_0})^{-1}$ in $\mathcal{L}_2$ meet for one given set $Y_i^{j_0}$ and one given set $W_i^{k_0}$, then output the element of $\langle g \rangle$ identified to $Y_i^{j_0} \cup W_i^{k_0}$; otherwise go on the loop over B_i.

The time complexity of Stinson's algorithm is $\mathcal{O}(\ell \binom{\ell/2}{t/2})$ group exponentiations[1] and the space requirement is $\mathcal{O}\big(\binom{\ell/2}{t/2}\big)$.

[1] This algorithm can be turned into a randomized Las Vegas variant the time complexity of which is $\mathcal{O}(\sqrt{t}\binom{\ell/2}{t/2})$.

A detailed analysis of the number of group multiplications required to find the result in the worst case is $\ell/2 \times (t-1)\binom{\ell/2}{t/2}$ group multiplications.

2.3 Discrete Logarithms as Products in Groups of Known Order

This second variant of the Baby-Step Giant-Step algorithm focuses on discrete logarithms equal to products of integers. More precisely, it addresses the problem of computing the value $x = \log_g v$ for a given $v \in \langle g \rangle$, whenever $x = x_1 \times x_2 \bmod n$ with $x_1 \in X_1$, $x_2 \in X_2$. We denote by $|X_1|$ and $|X_2|$ the respective cardinalities of X_1 and X_2.

This variant is described in a technical report of Hoffstein and Silverman [7] and works as follows. From the equality $v = g^x = g^{x_1 \times x_2 \bmod n}$, we immediately obtain that

$$v^{x_2^{-1} \bmod n} = g^{x_1}.$$

As a consequence, in the two following sets $\left\{v^{j^{-1} \bmod n}; j \in X_1\right\}$ and $\left\{g^i; i \in X_2\right\}$, there exists at least one collision, i.e a same value obtained for one $v^{j_0^{-1} \bmod n}$ and one g^{i_0} such that x is equal to $j_0 \times i_0 \bmod n$.

The time complexity is $\mathcal{O}\big(|X_1| + |X_2|\big)$ group exponentiations since it mainly relies on the construction of two sets containing respectively $|X_1|$ and $|X_2|$ elements. In terms of number of group multiplications, without any assumption on the structure of the sets X_1 and X_2, the time complexity is $\mathcal{O}\big((|X_1| + |X_2|)\log_2 n\big)$ but this bound can be decreased for some specific choices of those sets. Finally, the smallest set must be stored so that the space complexity is $\mathcal{O}\big(\min(|X_1|, |X_2|)\big)$.

3 Discrete Logarithms as Products in Groups of Unknown Order

We now present a new variant of the Baby-Step Giant-Step algorithm that can be used to compute discrete logarithms equal to a product in a group of unknown order. As an application of this method, we propose a cryptanalysis in the next section.

3.1 Preliminaries

In this section, we consider a finite cyclic abelian group $G = \langle g \rangle$, written multiplicatively, of unknown order n. Let X_1 and X_2 be two sets of integers, the problem we address is to compute the discrete logarithm $x = \log_g v$ for a given value $v \in G$, whenever $x = x_1 \times x_2 \bmod n$ with $x_1 \in X_1$, $x_2 \in X_2$. We denote by $|X_1|$ and $|X_2|$ the respective cardinalities of the two sets X_1 and X_2.

The variant of the Baby-Step Giant-Step suggested by Hoffstein and Silverman and recalled in Section 2.3 cannot be applied, since it requires modular inversion modulo the unknown order n of the group. Our new variant enables us to overcome this problem.

3.2 Overview of the Full Algorithm

We first present the general method used in the algorithm. As for the second variant of the baby-Step Giant-Step algorithm presented in Section 2.3, we first consider the general equation in G

$$v^{x_1^{-1} \bmod n} = g^{x_2}. \tag{1}$$

As explained above, considering directly this equation is no longer interesting since computing inverses modulo the unknown order is not feasible.

However, we can use a trick that has for example already been used by Shoup [16]. We denote

$$\pi_1 = \prod_{i \in X_1} i,$$

and we raise Equation 1 to the power π_1, so that we obtain $v^{\pi_1 x_1^{-1} \bmod n} = g^{\pi_1 x_2}$ which can be rewritten as:

$$v^{\prod_{i \in X_1 \setminus \{x_1\}} i} = g^{\pi_1 \times x_2} \tag{2}$$

With this new equation, the knowledge of the order of g is no longer necessary. We can now use the classical method like in the other Baby-Step Giant-Step algorithms. More precisely, in a first time we can compute the two following sets

$$\mathcal{S}_1 = \left\{ v^{\prod_{k \in X_1 \setminus \{i\}} k}; \forall i \in X_1 \right\} \text{ and } \mathcal{S}_2 = \{ g^{\pi_1 \times j}; \forall j \in X_2 \}.$$

In the two sets, two values meet for one value $v^{\prod_{k \in X_1 \setminus \{i_0\}} k}$ and one value $g^{\pi_1 j_0}$ such that x is equal to $i_0 \times j_0$.

However, we must be careful with the actual time complexity of this algorithm. More precisely, let us evaluate the time complexity of the construction of the set $\mathcal{S}_1$; in the different Baby-Step Giant-Step algorithms recalled in Section 2, the computation of each element of the relative sets requires (at most) a modular inversion and a group exponentiation. In our new algorithm, the value π_1 cannot be reduced modulo the unknown order n so that, computing each element in a classical way is equivalent to computing $(|X_1| - 1)$ group exponentiations with $\log_2 n$ bits exponents. Thus the naïve computation of the full set $\mathcal{S}_1$ has a time complexity $\mathcal{O}\big(|X_1|^2\big)$ group exponentiations. Taking into account the time complexity required for the computation of $\mathcal{S}_2$, we finally obtain a time complexity $\mathcal{O}\big(|X_1|^2 + |X_2|\big)$ group exponentiations, i.e $\mathcal{O}\big((|X_1|^2 + |X_2|) \log_2 n\big)$ group multiplications.

Since the complexity of the exhaustive search over all the possible $x_1 \in X_1$ and $x_2 \in X_2$ is obviously in $\mathcal{O}\big(|X_1| \times |X_2|\big)$ group exponentiations, the practical gain of our algorithm may not be significant (for instance if $|X_1| \cong |X_2|$), with a naïve computation of $\mathcal{S}_1$.

Note that there is no problem with the computation of the set $\mathcal{S}_2$ since after computing one time g^{π_1}, each element of $\mathcal{S}_2$ can be obtained using a single group exponentiation.

Let us now present how to construct efficiently the set $\mathcal{S}_1$.

3.3 Efficient Construction of $\mathcal{S}_1$

The Algorithm. For a better overview of the construction, we consider in the following a set X of 2^q elements denoted by x_i, $i \in [\![1, 2^q]\!]$ for which we want to obtain the set of values

$$\mathcal{S} = \left\{ v^{\prod_{x_i \in X \setminus \{x_j\}} x_i};\ \forall j \in [\![1, 2^q]\!] \right\}.$$

The method we present relies on an implicit binary tree structure. The algorithm starts from the root equal to v and it ends with 2^q leafs equal to the elements of $\mathcal{S}$. All the nodes can be computed using the following algorithm:

$node_{(0,0)} := v$
For $i \in [\![0, q-1]\!]$
  For $j \in [\![0, 2^i - 1]\!]$

(Left Son)

$$exp_L := \prod_{k=1+(2j+1)(2^{q-i-1})}^{(2j+2)(2^{q-i-1})} x_k$$

$$node_{(i+1,2j)} = (node_{(i,j)})^{exp_L}$$

(Right Son)

$$exp_R := \prod_{k=1+2j(2^{q-i-1})}^{(2j+1)(2^{q-i-1})} x_k$$

$$node_{(i+1,2j+1)} = (node_{(i,j)})^{exp_R}$$

This algorithm iteratively computes

$$node(i,j) = v^{\left(\prod_{k=1}^{j \times 2^{q-i}} x_k\right) \times \left(\prod_{\ell=1+(j+1)2^{q-i}}^{2^q} x_\ell\right)}$$

for $i \in [\![0, q]\!]$ and $j \in [\![0, 2^i - 1]\!]$, i.e v to the power the product of all the $x_k \in X$, but a "gap" of 2^{q-i} consecutive elements. This property is easily proved using a recursive argument. As a consequence, for all $i \in [\![0, 2^q - 1]\!]$ the leaf $node_{(q,i)}$, is equal to the value $v^{\prod_{x_j \in X \setminus \{x_{i+1}\}} x_j}$.

Example. Let $q = 2$. Using the algorithm for the first generation, the left son of the root, denoted by $node_{(1,0)}$ is equal to $v^{x_3 x_4}$ and the right son, denoted $node_{(1,1)}$, is equal $v^{x_1 x_2}$. The second generation is described in Fig. 1.

Complexity of the Construction. The advantage of our algorithm is twofold:

1. Once the two sons $node_{(i+1,2j)}$ and $node_{(i+1,2j+1)}$ of $node_{(i,j)}$ are computed, the $node_{(i,j)}$ is no longer required, so that it can be erased. Thus, as mentioned previously, the algorithm uses a binary tree structure but does not require the storage of the entire tree. As a consequence, the space requirement

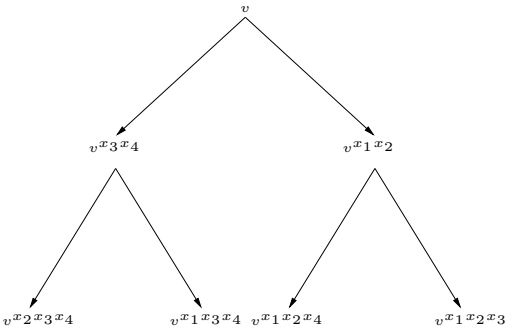


Fig. 1. Iteration of the algorithm for $q = 2$

during the algorithm execution is optimal, i.e equal to the space required for the storage of the set $\mathcal{S}$.

2. The different exponents x_k are used only once during each loop over i. Thus, the time complexity (in terms of group exponentiations) is equal to $\mathcal{O}\big(|X| \log_2 |X|\big)$ as there are exactly q loops involving $|X|$ group exponentiations. This complexity can also be obtained considering the overall number of group exponentiations given by:

$$\sum_{i=0}^{q-1} \sum_{j=0}^{2^i-1} (2 \times 2^{q-i-1}) = 2 \sum_{i=0}^{q-1} (2^{q-1}) = q \times 2^q$$

In terms of group multiplications complexity, without additional assumption on the structure of X, all the exponents we use have $\log_2 n$ bits so the complexity is $\mathcal{O}\big(|X| \log_2 |X| \log_2 n\big)$ group multiplications.

3.4 Complexity of the Full Algorithm

Our new variant of the Baby-Step Giant-Step algorithm described in Section 3.2, is based on the computation of two sets $\mathcal{S}_1$ and $\mathcal{S}_2$. Using the method of Section 3.3, the computation of the set

$$\mathcal{S}_1 = \left\{ v^{\prod_{k \in X_1 \setminus \{i\}} k}; \forall i \in X_1 \right\}$$

has time complexity $\mathcal{O}\Big(|X_1| \log_2 |X_1| \log_2 n\Big)$ group multiplications and the computation of

$$\mathcal{S}_2 = \{ g^{\pi_1 \times j}; \forall j \in X_2 \}$$

has time complexity $\mathcal{O}\Big(|X_2| \log_2 n\Big)$ group multiplications. Thus, the overall time complexity, for computing the two sets, is $\mathcal{O}\Big(\big(|X_2| + |X_1| \log_2 |X_1|\big) \log_2 n\Big)$.

Finally, finding two identical values in the two sets can be done efficiently (for example if one set is sorted), so that the time complexity of the algorithm mainly relies on the construction of the sets. Thus, the time complexity of the full algorithm is also $\mathcal{O}\Big(\big(|X_2| + |X_1| \log_2 |X_1|\big) \log_2 n\Big)$ group multiplications.

4 Attacks on GPS with Private Keys from CHES'04

In this section, we briefly recall the basic GPS scheme [4,13]. Next, we recall the private keys suggested by Girault and Lefranc at CHES 2004 conference and we finally present two attacks on such private keys. The first one relies on our new algorithm from Section 3 taking advantage of a first weakness of the private key and the second attack relies on a second weakness of the private keys.

4.1 The GPS Scheme

We denote by $\mathbb{Z}_n$ the residue class ring modulo n and $\mathbb{Z}_n^*$ the multiplicative group of invertible elements in $\mathbb{Z}_n$. The GPS identification scheme from [4,13], is an interactive protocol between a prover and a verifier which contains one or several rounds of three passes. The security is based on the intractability of *the short discrete logarithm problem* defined as follows.

Definition 2. *Let n be a composite integer the factorization of which is unknown. Let g be an element in $\mathbb{Z}_n^*$ of maximal order $\lambda(n)$. Let S be an integer such that $2^S < \lambda(n)$. The short discrete logarithm problem consists in computing the value $s \in [\![0, 2^S[\![$ given $v = g^s \bmod n$.*

Assumption. *The short discrete logarithm problem is polynomially intractable.*

During a round of identification, a prover uses his knowledge of a private value s related to the public value v by the equation $v = g^{-s} \bmod n$. More precisely, in typical applications, a prover holds a private key s and a public key (n,g,v) such that:

- $n = pq$ is the product of two prime integers such that factoring n is difficult,
- g is an element of $\mathbb{Z}_n^*$ of maximal order $\lambda(n)$,
- $v = g^{-s} \bmod n$.

There are four security parameters S, k, R and m defined as follows:

- S is the binary size of the private key s; $S = 160$ is a typical choice.
- k is the binary size of the challenges sent to the prover and determines the level of security of the scheme.
- R is the binary size of the exponents used in the commitment computation. It typically verifies $R = S + k + 80$.
- m is the number of rounds the scheme is iterated. Theoretically, m is polynomial in the size of the security parameter; but, in practice, m is often chosen equal to 1.

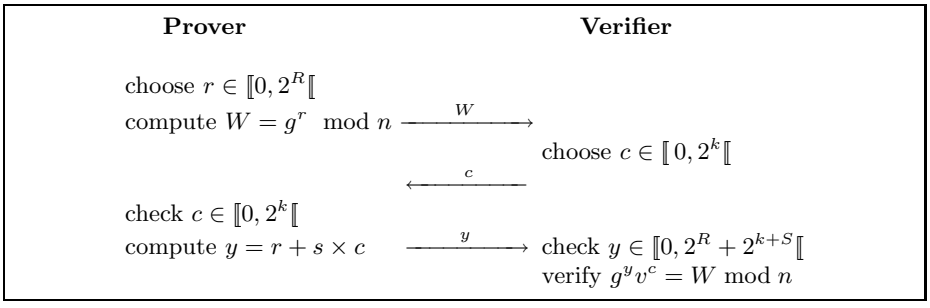


Fig. 2. The basic GPS identification scheme

Security of the Scheme. The security of the GPS scheme is recalled in the following Theorem (the proof is given in [13]).

Theorem 1. *The GPS identification scheme is a secure identification scheme under the intractability of the short discrete logarithm problem if* m *and* 2^k *are polynomial in* $|n|$, $m \times 2^{S+k-R}$ *is negligible in* $|n|$ *and* $\log |n| = o(m \times k)$.

4.2 The Girault-Lefranc Private Keys

As many other discrete-logarithm-based schemes, the GPS identification scheme, used with precomputations of the commitments $W = g^r \bmod n$, is a very efficient scheme for the prover. Thus, during the protocol, the computations of the prover can be reduced to the computation of the value $y = r + sc$ so that it is well designed for low cost chips. However, in new chips like RFID tags, even computing a multiplication may be too expensive. Thus, at the CHES 2004 conference, Girault and Lefranc [5] proposed three solutions to make easier the integration of the GPS identification scheme in such chips. Their goal was to remove (or at least reduce) the computation requirement for $y = r + sc$.

One of these solutions consists in using specific private keys equal to the product of low Hamming weight sub-private keys. More precisely, they consider a private key s equal to $s_1 \times s_2$ with s_1 in X_1 and s_2 in X_2 such that both s_1 and s_2 have a low Hamming weight.

With such private keys, the computation of $s \times c$ is then replaced successively by the computation of $s_2 \times c$ and $s_1 \times (s_2 c)$. Thus, since the Hamming weight of the sub-private keys s_1 and s_2 is low, the computation of y using the shift-and-add paradigm does not involve too many shifts.

Security of Such Private Keys. During this analysis, we require a security over the private key of 2^{80} group multiplications. However, the level of security is highly application dependant and some lower levels can be accepted.

The authors give some general security arguments on such private keys. Indeed, they suggest the use of sub-private keys s_1 and s_2 such that:

1. the private key $s = s_1 s_2$ is sufficiently large, i.e around 160-bit, such that the classical Baby-Step Giant-Step has a time complexity of around 2^{80} group multiplications;
2. the average Hamming weight of $s = s_1 s_2$ is around 64 such that Stinson's algorithm requires more than 2^{80} group multiplications.

Finally, since the group generator g is of unknown order $\lambda(n)$, the second variant of the Baby-Step Giant-Step algorithm recalled in Section 2.3 cannot be used. Thus, Girault and Lefranc consider that the best attack on such private keys is the exhaustive search over s_1 and s_2 which time complexity is obviously in $\mathcal{O}\big(|X_1| \times |X_2|\big)$ group exponentiations, where $|X_1|$ and $|X_2|$ denote the respective cardinalities of X_1 and X_2. Thus, they suggest to consider two adequate sets X_1 and X_2 such that $|X_1| \times |X_2| \approxeq 2^{80}$.

Numerical Application. In [5], Girault and Lefranc give the following example. To obtain a 160-bit private key s with an Hamming weight equal on average to 64, s_2 should be a 142-bit number with 16 random bits equal to 1 chosen among the 138 least significant ones and s_1 a 19-bit number with 5 random bits equal to 1 chosen among the 16 least significant ones.

With such sub-private keys, the cardinality of X_1 is equal to $\binom{16}{5} \approxeq 2^{12}$ and the cardinality of X_2 is equal to $\binom{138}{16} \approxeq 2^{68}$; so that the exhaustive search is at least as infeasible as the classical Baby-Step Giant-Step algorithm for a 160-bit key.

Then, if we assume that c is a 32-bit number, computing $r + s \times c$ in a classical way involves on average 16 additions of a 160-bit numbers, i.e 2560 bit additions. Using the structure of the private key equal to a product of low Hamming weight sub-private keys, then computing $s_2 \times c$ requires exactly 5 additions of a 32-bit number, then $s_1 \times (s_2 c)$ requires 16 additions of a 51-bit number and adding $s_1(s_2 c)$ to r requires a final addition of a 192-bit number. Finally, only 1168 bit additions are required.

4.3 Two Attacks

This first attack relies on our new Baby-Step Giant-Step algorithm described in Section 3. Thus, to prove the efficiency of our algorithm, we apply it to the numerical application given in Section 4.2.

In this example, we recall that the two sets X_1 and X_2 are of cardinalities respectively upper bounded by 2^{12} and 2^{68}. We also recall that the time complexity of our new algorithm is $\mathcal{O}\big(|X_2| + |X_1| \log_2 |X_1|\big)$ group exponentiations. Thus, applied to the given example, our algorithm recovers the private key s with about 2^{68} group exponentiations, which is significantly less than the complexity of the exhaustive search equal to 2^{80} group exponentiations.

A detailed enumeration of the number of group multiplication shows that the computation of the set S_2 in the algorithm of section 4.2 has the highest complexity; using precomputation in the usual square and multiply exponentiation algorithm leads to a complexity of 2^{73} group multiplications (the exhaustive search requires around 2^{84} group multiplications).

Whereas our first attack only takes advantage of the structure of product of the private keys, we now describe a second attack which takes advantage of both the product structure and the low Hamming weight of the sub-private keys.

Indeed, from the basic equation $v = g^s = g^{s_1 \times s_2}$, and denoting g^{s_1} by h, we then obtain the new equation

$$v = h^{s_2}.$$

With this change of base, the discrete logarithm of v in base h is a low Hamming weight number so that Stinson's algorithm can now be easily applied. The attack consists in using Stinson's algorithm (see section 2.2) for all possible bases h defined as g^i for any i in X_1. The complexity of this attack is then obviously $\mathcal{O}\big(|X_1| \times \ell\binom{\ell/2}{t/2}\big)$ group exponentiations, ℓ denoting the binary size of s_2 and t its Hamming weight. Note that the space requirement of this attack is the same as the one of Stinson's algorithm, i.e $\mathcal{O}(\binom{\ell/2}{t/2})$.

In the numerical application of section 4.2, we recalled that $|X_1| \approxeq 2^{12}$ and that s_2 is a 138-bit number with a Hamming weight equal to 16. As $\binom{138/2}{16/2} \approxeq 2^{33}$, we recover the private key with about 2^{52} group exponentiations.

A detailed analysis of the exact number of group multiplications required by this attack shows that the private key recovery requires 2^{54} group multiplications in the worst case.

To keep using Girault-Lefranc private keys, we suggest to consider new types of sub-private keys. Indeed, s_1 should be a 30 bit number with 12 non zero bits and s_2 should be a 130-bit number with 26 non zero bits.

With such sub-private keys, our first algorithm requires around 2^{94} group multiplications and the complexity of the second attack is then around 2^{80} group multiplications. However, the computation advantage of the method is obviously decreased. Using the same consideration as in the numerical application of section 4.2, the number of bit additions is then equal to 2100; the practical gain is less significant, specially with small challenges c.

5 Conclusion

We have proposed a new variant of the Baby-Step Giant-Step algorithm for discrete logarithms equal to products in groups of unknown order. More precisely, our algorithm recovers $x = x_1 \times x_2$ with $x_1 \in X_1$, $x_2 \in X_2$ from the given value g^x in time $\mathcal{O}\big((|X_2| + |X_1| \log_2 |X_1|) \log_2 n\big)$ group multiplications.

This new variant finds a direct application with the GPS scheme used with such private keys as described by Girault and Lefranc at CHES 2004 conference. Thus, whereas the time complexity of the best known attack (the exhaustive search) on such private key was 2^{84} group multiplications, using our new algorithm, this complexity falls downto 2^{73} group multiplications. Moreover, using the fact that such private keys require sub-private keys of low Hamming weight, we have constructed a second attack the time complexity of which is 2^{54} group multiplications.

Acknowledgements

The authors wish to thank Aline Gouget and Marc Girault for valuable and helpful discussions and comments.

References

1. H. Cohen. *A Course in Computational Algebraic Number Theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, 1993.
2. W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.
3. A. Fiat and A. Shamir. How to Prove Yourself : Practical Solutions to Identification and Signature Problems. In A. M. Odlyzko, editor, *Advances in Cryptology - Crypto '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer Verlag, 1986.
4. M. Girault. Self-Certified Public Keys. In D. W. Davies, editor, *Advances in Cryptology - Eurocrypt '91*, volume 547 of *Lecture Notes in Computer Science*, pages 490–497. Springer-Verlag, 1991.
5. M. Girault and D. Lefranc. Public Key Authentication with one Single (on-line) Addition. In M. Joye and J. J. Quisquater, editors, *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 413–427. Springer-Verlag, 2004.
6. L. C. Guillou and J. J. Quisquater. A Practical Zero-knowledge Protocol Fitted to Security Microprocessor Minimizing both Transmission and Memory. In C. G. Günther, editor, *Advances in Cryptology - Eurocrypt '88*, volume 330 of *Lecture Notes in Computer Science*, pages 123–128. Springer-Verlag, 1988.
7. J. Hoffstein and J.H. Silverman. Random Small Hamming Weight Products with Applications to Cryptography. Technical report, NTRU Cryptosystems.
8. National Institute of Standards and Technologies. Digital Signature Standard (DSS). Federal Information Processing Standards, Publication 186, november 1994.
9. A. M. Odlyzko. Discrete Logarithms: The Past and The Future. *Designs, Codes, and Cryptography*, 19(2/3):129–145, 2000.
10. T. Okamoto, H. Katsuno, and E. Okamoto. A Fast Signature Scheme based on new on-line Computation. In C. Boyd and W. Mao, editors, *Information Security Conference '02*, number 2851 in Lecture Notes in Computer Science, pages 111–121. Springer-Verlag, 2003.
11. T. Okamoto, M. Tada, and A. Miyaji. An Improved Fast Signature Scheme without on-line Multiplication. In M. Blaze, editor, *Financial Crypto*, volume 2357 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.
12. J. M. Pollard. Monte Carlo Methods for Index Computations (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.
13. G. Poupard and J. Stern. Security Analysis of a Practical "on the fly" Authentication and Signature Generation. In K. Nyberg, editor, *Advances in Cryptology - Eurocrypt '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 422–436. Springer-Verlag, 1998.
14. R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communication of the ACM*, 21(2):120–126, 1978.
15. C. P. Schnorr. Efficient Identification and Signatures for Smart Cards. In G. Brassard, editor, *Advances in Cryptology - Crypto '89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer-Verlag, 1990.

16. V. Shoup. Practical Threshold Signatures. In B. Preneel, editor, *Advances in Cryptology - Eurocrypt '00*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220. Springer-Verlag, 2000.
17. J. Stern and J. P. Stern. Cryptanalysis of the OTM Signature Scheme from FC'02. In R. N. Wright, editor, *Financial Cryptography '03*, volume 2742 of *Lecture Notes in Computer Science*, pages 138–148. Springer-Verlag, 2003.
18. D. R. Stinson. Some Baby-Step Giant-Step Algorithms for the Low Hamming Weight Discrete Logarithm Problem. *Mathematics of Computation*, 71(237):379–391, 2002.

Further Hidden Markov Model Cryptanalysis

P.J. Green[1], R. Noad[2], and N.P. Smart[2]

[1] Department of Mathematics, University of Bristol,
University Walk, Bristol, BS8 1TW, United Kingdom
P.J.Green@bristol.ac.uk
[2] Department of Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol, BS8 1UB, United Kingdom
{noad, nigel}@cs.bris.ac.uk

Abstract. We extend the model of Karlof and Wagner for modelling side channel attacks via Input Driven Hidden Markov Models (IDHMM) to the case where not every state corresponds to a single observable symbol. This allows us to examine algorithms where errors in measurements can occur between sub-operations, e.g. there may be an error probability of distinguishing an add (A) versus a double (D) for an elliptic curve system. The prior work of Karlof and Wagner would assume the error was between distinguishing an add-double (AD) versus a double (D). Our model also allows the modelling of unknown values, where one is unable to determine whether a given observable is add or double, and is the first model to allow one to analyse incomplete traces. Hence, our extension allows a more realistic modelling of real side channel attacks. In addition we look at additional heuristic approaches to combine multiple traces together so as to deduce further information.

1 Introduction

The randomization of algorithms as a technique to prevent side channel analysis has in recent years been a topic of intense research. However, many of the approaches using randomization have been made in an ad-hoc manner with little analysis as to whether the randomization introduced actually helps reduce the risk of side channel attacks.

As a trivial example of such a randomization consider the following two variants of the right-to-left binary exponentiation algorithm in an additive group. One is the standard, non-randomized, version whilst the second is a randomized version requiring a random coin per key symbol (usually a bit).

Non-Randomized Binary Method

$Q \leftarrow \mathcal{O}$
$T \leftarrow P$
For $i = 1$ to N
  If $(k_i = 1)$ $Q \leftarrow Q + T$
  $T \leftarrow 2T$
Return Q

Randomized Binary Method

$Q \leftarrow \mathcal{O}$
$T \leftarrow P$
For $i = 1$ to N
  If $(k_i = 1)$ $Q \leftarrow Q + T$
  Else if $(\text{coin}_i = 0)$ $R \leftarrow Q + T$
  $T \leftarrow 2T$
Return Q

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 61–74, 2005.

Various authors [4,8,6] have suggested the use of finite state machines or Hidden Markov Models (HMMs) as a mechanism to analyse the benefit of randomization techniques in side channel analysis. See [7] for an introduction and partial survey of the application of HMMs to side channel analysis. The idea is that the hidden states of the Markov Model represent the states of the algorithm implementing the countermeasure, whilst the observations represent the side channel itself.

In [4] Karlof and Wagner introduce a concept called an Input Driven Hidden Markov Model, this is a HMM which for each state of the algorithm associates an input state which drives the state transition. This input state is used to model the input of the fixed key to the algorithm. The internal state transitions not only depend on the current state and the random tape given to the algorithm, but also the key symbols. We note that this is only a notational simplification since the inputs in the Karlof/Wagner model can be modelled in a standard HMM by extending the state space of a standard HMM to consist of not only the state of the algorithm but also the corresponding key symbol.

The major innovation of the Karlof and Wagner approach was to allow the modelling of attacks involving multiple traces and the modelling of errors in the state measurements. However, a major drawback was that each internal state and observable had to correspond to a single key symbol. To see the advantages and the disadvantages of this approach we now give an overview of the approach taken by Karlof and Wagner, as applied to randomized group exponentiation algorithms, which are after all the main application area of side channel analysis on public key algorithms.

We shall adopt additive group notation, as is common in elliptic curve cryptography. Suppose we wish to compute $Q = kP$ for a fixed secret integer k and a (possibly fixed) public group element P. Almost all algorithms process the bits of k in chunks (e.g. a bit at a time, or in fixed/sliding window segments). In almost all algorithms the processing of each bit can be reduced to the computation of either a double D, an add-double AD or a double-add DA. Whether one has AD or DA depends on the precise group exponentiation algorithm used, for ease of explanation we shall suppose the algorithm either performs a D or an AD. In simple side channel analysis whether a D or an AD is performed can be deduced from the side channel, the observed sequence of D's and AD's we call a *trace*. Hence, the goal of the attacker is to deduce the value of k given the sequence of D's and AD's observed.

In practice, however, one may not be able to determine correctly the sequence of D's and AD's from the side channel, one may make errors in this observation, or in fact be unable for certain measurements to determine whether a given symbol is A or D. In addition since one is assuming a randomized exponentiation algorithm the attacker could repeat the side channel experiment, assuming the same k is used on each exponentiation (but not necessarily the same P). The attacker then needs to combine the information obtained from multiple traces in such a way as to obtain the secret k.

Karlof and Wagner propose a heuristic method, which they describe as a variant of the Viterbi algorithm although it is actually a variant of the Forward-

Backward (FB) algorithm, to solve the above problem. Using a prior probability distribution on the key symbols (say for example each bit is equally likely to be zero or one), they use their FB algorithm and a single trace so as to deduce an approximation to the posterior probability distribution on the key symbols. This posterior distribution is then used as the prior distribution for the next trace to be processed and so on. After the processing of all of the set of multiple traces one deduces the final estimated posterior distribution on the key symbols, which the attacker hopes reveals to him the actual key used. This form of belief propagation reduces an exponential increase in the number of states needed to process all traces in a parallel manner. However, it is clearly susceptible to the order in which the traces are fed into the algorithm and it does not work for some exponentiation algorithms.

A practical problem with the Karlof and Wagner approach is that each observable, i.e. D or AD, needs to correspond to a single key bit. This is fine when dealing with noiseless data; however, in measuring a power trace it is unlikely that we confuse a D with an AD since they take a significantly different period of time. It is far more likely that we confuse a single D with a single A, and vice versa, or be unable to distinguish a D from an A at all.

To see the problems that the model of Karlof and Wagner can produce, suppose we used a non-randomized right-to-left binary exponentiation algorithm, as above. Now if we saw the sequence DAD then the Karlof and Wagner algorithm would interpret this as a key with two bits. Since the output D corresponds to one bit, whilst the output AD corresponds to another. However, it could be that the measuring equipment mistook the A for a D and that the actual sequence executed was DDD, in other words a key with three bits. Hence, one can see that the error model of Karlof and Wagner does not fully model the errors that one can see in a real-life side channel measurement.

The main motivation for looking at these techniques is to handle situations where one is unable to accurately distinguish an A from a D. This happens when analysing exponentiation algorithms which have been implemented using arithmetic techniques which aim to make the distinguishing of an A from a D as difficult as possible. Such techniques have been proposed by various authors in particular Brier, Déchéne, Joye, Liardet, Quisquater and Smart [2,3,5].

In this paper we extend the Karlof and Wagner approach to cope with traces for which each key symbol potentially corresponds to multiple, or zero, observable symbols and where some of the observable symbols may be unknown or wrong. In particular we model the case where multiple runs of the exponentiation algorithm, with the same secret exponent, can lead to traces of different lengths. Hence, we need to model the case of variable length data. In Section 3 we present our FB algorithm for coping with a single trace. This approach extends the state space of the HMM to include a variable which counts how far one has processed along the output trace, each state transition within the HMM corresponds to the processing of a single key symbol; however, each transition may take up a varying number of output symbols. In Section 4 we describe some heuristic approaches to dealing with multiple traces, we examine the pros and

cons of each approach. In Section 5 we present some experimental results for our methods as applied to various exponentiation algorithms.

We end this introduction by thanking John Malone-Lee for various useful discussions and insight whilst the work in this paper was carried out.

2 Notation

In this section we introduce the notation we will use throughout the paper. In particular we highlight the difference between our approach and that of Karlof and Wagner.

If X is a discrete random variable then we let $p(X = x)$ denote the probability distribution function, which we shorten to $p(x)$ for compactness when the underlying random variable X is clear. We let $p(x|y)$ denote the probability that the random variable X is x, given that Y is y, a notation which is extended to $p(x_1, \ldots, x_s|y_1, \ldots, y_t)$ in the standard way.

We assume we are interested in analysing an exponentiation algorithm with respect to a given fixed, but hidden, exponent k of at most N symbols, which we shall call the key. The symbols (usually bits) of k we shall denote by the vector $\boldsymbol{k} = (k_1, \ldots, k_N)$. As the algorithm progresses the internal state of the algorithm passes through a sequence of states $\boldsymbol{q} = (q_0, \ldots, q_N)$. We assume there is one internal state transition for each key symbol. When running the exponentiation algorithm the attacker obtains a sequence of observable outputs $\boldsymbol{y} = (y_1, \ldots, y_L)$.

In the model of Karlof and Wagner the observable outputs are in one-to-one correspondence with the internal states, hence $L = N$. The values of the observables are taken from the set $\{D, AD\}$; each internal state corresponds to one of these symbols and errors in measurement are specified by given the error probability p_0 which is $p(y_n = AD|q_n = D) = p_0$ and $p(y_n = D|q_n = AD) = p_0$. The internal sequence of states $\boldsymbol{x} = (x_0, \ldots, x_N)$ of the HMM in Karlof and Wagner's approach is essentially given by $x_n = (k_n, q_n)$.

In our model the observable outputs are not in one-to-one correspondence with the internal states, hence $L \neq N$. The observables are taken from the language $\mathcal{O}$, which is generated from the alphabet $\{D, A, \emptyset, \perp\}$, where $\emptyset$ is the zero-length observable and $\perp$ denotes unknown; each internal transition corresponds to a number of these symbols. To keep track of the number of observable symbols consumed by the state transitions we introduce another variable $\boldsymbol{m} = (m_1, \ldots, m_N)$, which signals that at internal algorithm state q_n the algorithm has output a total of m_n observable symbols. In particular we have $L = m_N$. The internal state of the HMM in our approach is given by the triple $x_n = (k_n, q_n, m_n)$. The precise number of output symbols "consumed" on entering a given state depends on the previous internal state of the HMM and the new key bit. To simplify matters we assume that m_n depends only on m_{n-1} and q_n, which is the case in all algorithms under consideration.

Again we assume that internal state corresponds to one of the observables symbols in $\mathcal{O}$. Errors are then modelled by defining $p(\text{ observed} = o_j|\text{ expected} = o_i) = p_{i,j}$, for $o_i, o_j \in \mathcal{O}$,

3 HMMs with Variable Length Data

In this section we present how to use the FB algorithm to analyse our HMM for a single trace, where the length of the list of output symbols may not correspond to the length of the list of internal states. We first present the FB algorithm as a general tool, we then recap on its application to classic HMM, and finally we present the modifications needed to cope with our situation.

3.1 Forward-Backward Algorithm

The FB algorithm is an efficient method for computing all *marginal sums*

$$t_n(x_n) = \sum_{x_0} \sum_{x_1} \cdots \sum_{x_{n-1}} \sum_{x_{n+1}} \cdots \sum_{x_N} f(x_0, x_1, \ldots, x_N), \tag{1}$$

for $n = 0, 1, \ldots, N$, when the function being summed has a factorisation of the form

$$f(x_0, x_1, \ldots, x_N) = \prod_{n=1}^{N} g_n(x_{n-1}, x_n). \tag{2}$$

By substituting (2) into (1) and rearranging the factors and summation signs, it is easy to see that

$$t_n(x_n) = r_n(x_n) s_n(x_n)$$

for all n and x_n, where

$$r_n(x_n) = \sum_{x_{n-1}} g_n(x_{n-1}, x_n) \sum_{x_{n-2}} g_{n-1}(x_{n-2}, x_{n-1}) \cdots$$

and

$$s_n(x_n) = \sum_{x_{n+1}} g_{n+1}(x_n, x_{n+1}) \sum_{x_{n+2}} g_{n+2}(x_{n+1}, x_{n+2}) \cdots$$

These summations can be computed recursively via

$$r_n(x_n) = \sum_{x_{n-1}} g_n(x_{n-1}, x_n) r_{n-1}(x_{n-1}) \text{ for } n = 1, 2, \ldots, N \tag{3}$$

$$s_n(x_n) = \sum_{x_{n+1}} g_{n+1}(x_n, x_{n+1}) s_{n+1}(x_{n+1}) \text{ for } n = N-1, N-2, \ldots, 0 \tag{4}$$

starting from $r_0(x_0) \equiv 1$ and $s_N(x_N) \equiv 1$.

3.2 Classic HMM

In the standard application of the FB algorithm to HMMs, $f(x_0, x_1, \ldots, x_N)$ is the joint distribution of hidden variables $(x_0, x_1, \ldots, x_N)$ and corresponding observed data $\boldsymbol{y} = (y_1, y_2, \ldots, y_N)$. The FB algorithm can then be applied since the factors of f are

$$g_1(x_0, x_1) = p(x_0)p(x_1|x_0)p(y_1|x_1) \text{ and } g_n(x_{n-1}, x_n) = p(x_n|x_{n-1})p(y_n|x_n)$$

for $n = 2, 3, \ldots$. Hence, the FB algorithm allows us to compute *marginal posteriors* since

$$p(x_n|y_1, y_2, \ldots, y_N) = t_n(x_n) / \sum_{x_n} t_n(x_n).$$

The application of Karlof and Wagner can be interpreted as taking the internal states x_n to be (k_n, q_n), where k_n is the n-th symbol of the key and q_n is the internal state. The values y_n then correspond to the observations made during the side channel analysis and the probability $p(y_n|x_n)$ models the error probability of seeing certain outputs given the internal state of the exponentiation algorithm. The formulae given to evaluate $p(x_i|\boldsymbol{y})$ given in [4] is then simply an application of the standard FB algorithm to this situation.

3.3 HMM with Variable Length Data

We now turn to the situation where the indexing of the observable states y_i does not correspond in a one-to-one manner with the indexing of the internal states x_i. The FB method really pays no regard to the indexing of the data, so that, providing the joint distribution of hidden variables and data can be factorised in the form (2), we can apply the method.

We now use a HMM in which the state variable x_n is a triple (k_n, q_n, m_n), and we assume a Markov structure, with transition probabilities

$$p(x_n|x_{n-1}) = p(k_n)p(q_n|q_{n-1}, k_n)p(m_n|m_{n-1}, q_n).$$

We assume that the distribution of the data $\boldsymbol{y}$ given $x_0, x_1, \ldots, x_n$ factorises into a product

$$p(\boldsymbol{y}|x_0, x_1, \ldots, x_n) = \prod_{n=1}^{N} d_n(\boldsymbol{y}, x_{n-1}, x_n);$$

the interpretation is that d_n is the distribution of the nth 'chunk' of data, conditional on x_{n-1} and x_n, or in practice only on (m_{n-1}, m_n, q_n). The FB algorithm can then be used, with

$$g_1(x_0, x_1) = p(q_0)p(k_1)p(q_1|q_0, k_1)p(m_1|m_0, q_1)d_1(\boldsymbol{y}, x_0, x_n)$$

(with m_0 fixed at 0) and, for $n = 2, 3, \ldots$,

$$g_n(x_{n-1}, x_n) = p(k_n)p(q_n|q_{n-1}, k_n)p(m_n|m_{n-1}, q_n)d_n(\boldsymbol{y}, m_{n-1}, m_n, q_n).$$

Note that this allows (but does not require) that the data chunk pointers m_n are random and not determined by q_n and m_{n-1}. This is a slight extra generalisation on the situation we are in when analysing exponentiation algorithms.

Given that the g_n factors, the forwards and backwards recursions (3) and (4) are performed and the marginal posteriors can be computed as above:

$$p(k_n, q_n, m_n|\boldsymbol{y}) = t_n(x_n) / \sum_{x_n} t_n(x_n)$$

from which the marginal distribution for k_n alone can be found by summing out m_n and q_n.

In the above we set $m_0 = 0$, the corresponding condition at the other end of the sequence, that restores the forwards/backwards symmetry is given as follows: If we assume that all of the observed data sequence is generated by the N steps of the HMM, then m_N is also known, and is equal to the length of $\boldsymbol{y}$. The known values of m_0 and m_N can then be regarded as part of the observed data sequence, and their values fixed by specification of the end conditions, so that $r_0(x_0) = 1$ if and only if $x_0 = (q_0, m_0)$ has $m_0 = 0$, and $s_N(x_N) = 1$ if and only if $x_N = (k_N, q_N, m_N)$ has m_N equal to the length of $\boldsymbol{y}$.

As well as specifying the initial state, q_0, as part of the HMM, we also include a set of permissible terminating states. This allows us to more accurately model algorithms with specific termination points and gives a corresponding increase in the accuracy of the calculated belief values.

3.4 Useful Properties

In addition to accommodating errors at the level of individual symbols with a given probability, our model allows the specification of an error map specifying different probabilities for each symbol transformation, i.e. the probability of reading an "A" as a "D" could be 0.5 but the probability of reading a "D" as an "A" could be only 0.1. The model also allows for symbols to be marked as unknown - so that in the case of a highly ambiguous reading it is possible to enter nothing rather than input a potentially misleading value into the HMM. Both of these types of error are handled in the d_n function during the processing of the FB algorithm.

Some algorithms may terminate without generating an output symbol for all input symbols - for example, when the remaining symbols in an exponent are all zero. We can avoid artificial pre-processing of data to fix the length of traces for such algorithms by using the zero-length observable symbol $\emptyset$, and having a corresponding terminating state in the algorithms HMM with this observable that links only to itself. This technique is used when modelling the Liardet–Smart exponentiation algorithm [5] and the Oswald–Aigner exponentiation algorithm [8].

3.5 Implementation Notes

At each n we need to store and manipulate the tables r_n and s_n indexed by $x_n = (k_n, q_n, m_n)$. While k_n and q_n have small sets of possible values, the range of values of m_n for which both r_n and s_n are non-zero varies with n. However, this does not cause a problem in practice; since each internal state change will output between $l = \min_{y \in \mathcal{Y}} |y|$ and $h = \max_{y \in \mathcal{Y}} |y|$ observable symbols, we have $ln \leq m_n \leq hn$, and also $l(N - n) \leq (m_N - m_n) \leq h(N - n)$ by symmetry. This is exploited to save time in the FB algorithm by providing an initial reduction in the portion of the state space which has the potential to generate non-zero belief values.

During the processing of the forward values, we further reduce the possible range of values for m_{n+1} at step n based of the range of values for which m_n

generated non-zero beliefs and the possible transitions from q_n. This state space reduction is also performed for m_{n-1} when calculating the backward values.

4 Modelling with Multiple Traces

Given a single trace we can use belief propagation as in Section 3 to calculate exact beliefs for each key symbol. However, trying to deal with multiple traces in this model increases the state space exponentially in the number of traces. We need a heuristic to combine the results from analysing the traces separately. This heuristic should make at most a polynomial (in number of traces and input size) number of calls to the single trace algorithm.

In the single trace version, we calculate the belief for a symbol n from a trace $\boldsymbol{y}$ and prior key symbol distribution D:

$$b_n(\boldsymbol{y}, D) = p(k_n = 1|y, D) = \sum_{q \in S} \sum_{m=0}^{|\boldsymbol{y}|} \frac{t_n^{\boldsymbol{y},D}(\{q, m, 1\})}{\sum_{x_n} t_n^{\boldsymbol{y},D}(x_n)},$$

where $t_n^{\boldsymbol{y},D}$ is the same as the t_n function in Section 3 but we now make the dependence on $\boldsymbol{y}$ and D explicit. We now present two heuristic approaches to dealing with multiple traces, one a natural analogue of that of Karlof and Wagner to the case of variable length observable data, the second one is based on *loopy belief propagation* (see, for example, [11]).

We let $\mathcal{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{|\mathcal{Y}|}\}$ denote the set of traces obtained from the side channel, we let N denote the number of key symbols in k and let D denote the prior probability distribution on these symbols. The goal of our heuristics is to output a heuristic posterior distribution, which we also denote by D, which takes into account the information contained within the set of traces $\mathcal{Y}$.

4.1 Karlof-Wagner

The *belief propagation* method used by Karlof and Wagner as applied to our situation is described below.

For all $\boldsymbol{y} \in \mathcal{Y}$
  For $n = 1 \ldots N$
    $D'(k_n = 1) \leftarrow b_n(\boldsymbol{y}, D)$
  $D \leftarrow D'$
Return D

In other words we compute

$$p(k_n = 1|\mathcal{Y}, D) = b_n\left(\boldsymbol{y}_1, b\left(\boldsymbol{y}_2, \ldots b\left(\boldsymbol{y}_{|Y|}, D\right)\ldots\right)\right)$$

where $b(\boldsymbol{y}, D)$ is shorthand for $\{b_n(\boldsymbol{y}, D)\}_{n=0}^{N}$.

This method has the advantage of having a complexity which is linear in the number of traces, however the final heuristic posterior distribution on the

key symbols depends heavily on the order in which the traces are processed. As an example of this problem consider the (non-randomized) right-to-left binary algorithm on a two bit exponent. Processing traces $\boldsymbol{y}_1 = ADD$ and $\boldsymbol{y}_2 = DAD$ should give $P(k_n = 1) = \frac{1}{2}$; However, this method gives the distribution $\{0.9, 0.1\}$ when processing ADD first and $\{0.1, 0.9\}$ when processing DAD first.

4.2 Bitwise Average

We investigated a number of heuristic methods for combining the data from multiple traces, although some produced results better than the following method, their complexity was too high for practical use in large examples. The following method was the one which produced the best results with a reasonable performance.

Our new heuristic combining method is as follows: We calculate the beliefs for each trace, perform a bitwise average to combine them into a new key symbol distribution and repeat until the distribution converges.

Repeat
  $D' \leftarrow D$
  For all $\boldsymbol{y} \in \mathcal{Y}$
    For $n = 1 \ldots N$
      $D_{\boldsymbol{y}}(k_n = 1) \leftarrow b_n(\boldsymbol{y}, D)$
    $D \leftarrow \mathtt{Avg}_{\boldsymbol{y} \in \mathcal{Y}}(D_{\boldsymbol{y}})$
Until $D \approx D'$
Return D

In other words we set

$$D = \left\{ \mathtt{Avg}_{\boldsymbol{y} \in \mathcal{Y}} \left(b_n \left(\boldsymbol{y}, D \right) \right) \right\}_{n=0}^{N}$$

and repeat until the values in the distribution D have appeared to converge.

Intuitively, this method is inspired by the following technique, one could think of a factor graph (see, for example, [11]) which connects the corresponding hidden states in each trace with an averaging function. As this graph contains loops, we must perform *loopy belief propagation* - that is, we start with an initial set of messages, in our case the initial key symbol distribution and the traces, and iterate until we (hopefully) get convergence. Clearly the output of this heuristic is independent of the input order of the traces, however it is unclear how many iterations are necessary and whether the method converges for a given input sequence.

The method also does not take into account information which can be obtained by considering two or more traces at once, so called *cross trace analysis*. As an example of this consider the randomized binary algorithm on a 3 bit exponent, where no errors occur when interpreting the power trace. A trace of DDD indicates that there are exactly three '0'-bits and zero '1'-bits. Given a set of traces $\{DDD, ADADAD\}$ the output should be 000 with probability one, as the

first trace shows that all of the As in the second trace are spurious. However, as there is no interaction between traces the final result is not definite.

We now turn to the discussion of the averaging function $\texttt{Avg}(\cdot)$ in the above heuristic method. If one uses the arithmetic mean then problems can arise, as the following example demonstrates: If $p(k_n = 1|\boldsymbol{y}_1, D) = 1$ and $p(k_n = 1|\boldsymbol{y}_2, D) = 0.5$ averaging these values gives us $p(k_n = 1|\boldsymbol{y}_1, \boldsymbol{y}_2, D) = 0.75$. However, $p(k_n = 1|\boldsymbol{y}_1, D) = 1$ is saying that the key symbol is *definitely* 1 whereas $p(k_n = 1|\boldsymbol{y}_2, D) = 0.5$ says that $\boldsymbol{y}_2$ gives no information about the key symbol. Clearly then we should combine the traces such that $p(k_n = 1|\boldsymbol{y}_1, \boldsymbol{y}_2, D) = 1$.

Replacing the arithmetic mean with a weighted mean where the weight function is 0 at $x = 0.5$ and increases with $|x - 0.5|$ solves this problem by allowing us to calculate the combined belief as

$$\texttt{Avg}(b_1, b_2, \ldots, b_n) = \begin{cases} 0.5 & \sum_{i=0..n} w(b_i) = 0, \\ \frac{\sum_{i=0..n} w(b_i) b_i}{\sum_{i=0..n} w(b_i)} & \text{Otherwise.} \end{cases}$$

The most effective function we have found for producing the weight is $w(x) = 4x^2 - 4x + 1$. In the above example the above weighting would give us a new combined belief of $p(k_n = 1|\boldsymbol{y}_1, \boldsymbol{y}_2, D) = 1$.

5 Performance of Heuristics

Our main interest was in analysing exponentiation algorithms in the situation where defences already exist to make it hard to distinguish doubles from additions. In such a situation one is interested in how much defence one obtains against simple power analysis by using the naive (non-randomized) binary algorithm. In addition, there are certain exponentiation algorithms which have been proposed for precisely the situation where, hopefully, indistinguishable operations have been implemented. For example in [5] Liardet and Smart propose an algorithm to be used in conjunction with their indistinguishable addition formulae so as to help mitigate against differential power analysis. They do this by introducing a small amount of randomization into the exponentiation algorithm without increasing the run time considerably.

In [10] Walter analyses the Liardet–Smart exponentiation algorithm in the situation where there are no indistinguishable operations, and from the power trace one can work out exactly the sequence of additions and doublings which are carried out. Walter shows that in such a situation, for a 160-bit exponent, one can break the Liardet–Smart algorithm with ten traces and work effort around $O(2^{64})$, using $R = 5$ in the Liardet–Smart algorithm. If one increases the number of traces to twenty then the work effort goes down to $O(2^{40})$. This result is achieved by an exact analysis of the Liardet–Smart algorithm.

With our method we were able to investigate various exponentiation algorithms with various error models. To illustrate typical results we used 160-bit exponent values and two errors models, which we now describe. Clearly other more complicated error models are allowed in our analysis but for ease of presenting our results we focus on the following two:

5.1 Error Model A:

Here we used an error model which swapped an A for a D, and vice versa, with a given fixed probability p_0,

$$p(\text{ observed } = A|\text{ expected } = D) = p(\text{ observed } = D|\text{ expected } = A) = p_0,$$
$$p(\text{ observed } = A|\text{ expected } = A) = p(\text{ observed } = D|\text{ expected } = D) = 1 - p_0.$$

5.2 Error Model B:

This model assumes that a certain fixed proportion p_0 of the symbols are unable to be read.

$$p(\text{ observed } = \perp \mid \text{expected } = D) = p(\text{ observed } = \perp \mid \text{expected } = A) = p_0.$$
$$p(\text{ observed } = A|\text{ expected } = A) = p(\text{ observed } = D|\text{ expected } = D) = 1 - p_0.$$

For each algorithm we performed a number of experiments, and produced the results in Tables 1, 2 and 3, for the standard binary algorithm, the Liardet–Smart algorithm and the Oswald–Aigner algorithm (OA2). In these tables the column $\mathfrak{p}$ represents the average proportion of key bits correctly recovered. Given this we can compute the amount of additional work needed to recover the key, given that the HMM algorithm recovers the stated proportion of the key symbols. This workfactor is derived using the low-Hamming weight variant of the Baby-Step/Giant-Step algorithm for the discrete logarithm problem (DLP) [9]. Namely, if we can derive an approximation to a discrete logarithm such that proportion $\mathfrak{p}$ of the N bits are correct, then one can solve for the exact discrete logarithm in expected time

$$O\left(\sqrt{N \cdot \mathfrak{p}} \cdot \binom{N/2}{N \cdot \mathfrak{p}/2}\right),$$

which may be more efficient than the $O(2^{N/2})$ technique of using the standard Baby-Step/Giant-Step algorithm. For 160-bit exponents we obtain an improvement as soon as $\mathfrak{p} > 0.8$.

Table 1. Results for the Binary Exponentiation Algorithm

Error Model	p_0	Number of Traces 1 $\mathfrak{p}$	5 $\mathfrak{p}$	10 $\mathfrak{p}$	20 $\mathfrak{p}$	100 $\mathfrak{p}$
–	0.0	1.00	1.00	1.00	1.00	1.00
A	0.1	0.66	0.75	0.77	0.79	0.82
	0.2	0.59	0.67	0.68	0.69	0.70
B	0.1	0.80	0.87	0.89	0.89	0.90
	0.2	0.72	0.77	0.77	0.77	0.78

Table 2. Results for the Liardet–Smart Exponentiation Algorithm ($R = 5$)

Error Model	p_0	Number of Traces 1 $\mathfrak{p}$	5 $\mathfrak{p}$	10 $\mathfrak{p}$	20 $\mathfrak{p}$	100 $\mathfrak{p}$
–	0.0	0.40	0.62	0.79	0.89	0.97
A	0.1	0.43	0.50	0.55	0.59	0.62
	0.2	0.39	0.44	0.47	0.49	0.52
B	0.1	0.43	0.51	0.57	0.63	0.71
	0.2	0.39	0.45	0.48	0.51	0.57

Table 3. Results for the OA2 Exponentiation Algorithm

Error Model	p_0	Number of Traces 1 $\mathfrak{p}$	5 $\mathfrak{p}$	10 $\mathfrak{p}$	20 $\mathfrak{p}$	100 $\mathfrak{p}$
–	0.0	0.82	0.89	0.95	0.97	0.98
A	0.1	0.59	0.69	0.72	0.77	0.79
	0.2	0.54	0.63	0.67	0.70	0.73
B	0.1	0.66	0.79	0.82	0.83	0.84
	0.2	0.60	0.71	0.72	0.75	0.76

To compare our heuristic trace combining method to that used by Karlof and Wagner we present in Table 4 the average proportion of key bits recovered correctly using our error models, but using the heuristic belief propagation method of Karlof and Wagner.

From the tables we see that our trace combining method recovers more of the key than the trace combining method of Karlof–Wagner. Furthermore, the results in Table 4 for both Liardet–Smart and Oswald–Aigner without errors show a decrease in accuracy as the number of traces increases. This is due to the way their heuristic overemphasises the belief values generated by early traces and causes the belief values to increase very rapidly when given consistent trace data, but not decrease as rapidly when given evidence to the contrary. With the Liardet–Smart algorithm an A in the trace indicates that either the current or previous key bit is 1 whereas a D indicates that the current key bit is marginally more likely to be 0 than 1. When processing multiple traces, the Karlof-Wagner heuristic combines these slight biases until rounding errors in the floating point representation cause a belief of 1 in the key bit being a 0. If a future trace then indicates that the key bit is actually 1 a contradiction occurs in the forward–backward algorithm causing it to fail. A non-zero probability of error prevents the belief values from getting close enough to 1 for such a rounding error to occur, which explains why the heuristic does not fail in those case. The decrease in accuracy when processing the Oswald-Aigner algorithm is due to the same reason; a slight bias for a particular key bit value is compounded until it outweighs strong evidence to the contrary.

Table 4. Results for the Karlof–Wagner Heuristic

Error Model	p_0	Algorithm: Binary 1	Binary 20	Binary 100	L–S 1	L–S 20	L–S 100	OA2 1	OA2 20	OA2 100
–	0.0	1.0	1.0	1.0	0.42	0.00	0.00	0.82	0.39	0.39
A	0.1	0.66	0.72	0.72	0.29	0.48	0.53	0.64	0.76	0.77
	0.2	0.61	0.68	0.68	0.21	0.42	0.48	0.58	0.69	0.73
B	0.1	0.76	0.85	0.85	0.27	0.53	0.54	0.66	0.77	0.78
	0.2	0.67	0.78	0.78	0.17	0.47	0.50	0.62	0.74	0.75

We also see that Error Model B allows one to recover more of the key than Error Model A, this is because in Error Model A we feed the algorithm possibly incorrect information, whilst in Error Model B we only tell it when we are unsure of certain data values; a value of $\perp$ provides only correct information, whereas an observation error provides misinformation. The ability to mark that an observable action took place without having to commit to what the action was would be very useful in practice, where two sub-operations may have similar power-traces or timing characteristics.

We see that, in the case of the Liardet–Smart algorithm, our general HMM based method is almost as effective as Walter's method in the case of no errors and multiple traces. However, the results for a single trace with no errors appear to be worse than random guessing! This doesn't quite tell the whole story, as the 40% (on average) of correct bits have a high level of confidence in correctness whilst the remaining bits have very low level. Using this level of confidence to perform a weighted average when combining multiple traces is key to our trace combining heuristic.

This is the first work to look at the Liardet–Smart algorithm in the case of noisy trace data, as would happen when the algorithm is used in the context of indistinguishable addition/doubling formulae as proposed in [5]. In this situation the Liardet–Smart algorithm is relatively immune to simple power analysis via our HMM method. This contrasts with the case of the Oswald–Aigner method, which performs only marginally better than the standard binary method if there are multiple traces with errors.

References

1. I.F. Blake, G. Seroussi and N.P. Smart, editors. *Advances in Elliptic Curve Cryptography.* Cambridge University Press, 2005.
2. É. Brier, I. Déchène and M. Joye. Unified addition formulæ for elliptic curve cryptosystems. In *Embedded Cryptographic Hardware: Methodologies and Architectures.* Nova Science Publishers, 2004.
3. M. Joye and J.-J. Quisquarter. Hessian elliptic curves and side-channel analysis. In *Cryptographic Hardware and Embedded Systems – CHES 2001*, Springer-Verlag LNCS 2162, 402–410, 2001.

4. C. Karlof and D. Wagner. Hidden Markov model cryptanalysis. In *Cryptographic Hardware and Embedded Systems – CHES 2003*, Springer-Verlag LNCS 2779, 17–34, 2003.
5. P.-Y. Liardet and N.P. Smart. Preventing SPA/DPA in ECC systems using the Jacobi form. In *Cryptographic Hardware and Embedded Systems – CHES 2001*, Springer-Verlag LNCS 2162, 391–401, 2001.
6. E. Oswald. Enhancing simple power-analysis attacks on elliptic curve cryptosystems. In *Cryptographic Hardware and Embedded Systems – CHES 2002*, Springer-Verlag LNCS 2523, 82–97, 2002.
7. E. Oswald. Side-Channel Analysis. In [1], 69–86, 2005.
8. E. Oswald and M. Aigner. Randomized addition-subtraction chains as a countermeasure against power attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2001*, Springer-Verlag LNCS 2162, 39–50, 2001.
9. D. Stinson. Some baby-step giant-step algorithms for the low hamming weight discrete logarithm problem. *Math. Comp.*, **71**, 379–391, 2002.
10. C. Walter. Breaking the Liardet–Smart randomized exponentiation algorithm. In *Proceedings Cardis '02*, 59–68, USENIX Assoc., 2002.
11. J.S. Yididia, W.T. Freeman and Y. Weiss. Understanding Belief Propagation and its Generalizations. Mitsubishi Electric Research Laboratories Technical Report TR-2001-22, January 2002.

Energy-Efficient Software Implementation of Long Integer Modular Arithmetic*

Johann Großschädl[1], Roberto M. Avanzi[2], Erkay Savaş[3], and Stefan Tillich[1]

[1] Institute for Applied Information Processing and Communications, Graz University of Technology, Inffeldgasse 16a, A–8010 Graz, Austria
{Johann.Groszschaedl,Stefan.Tillich}@iaik.at
[2] Faculty of Mathematics and Horst Görtz Institute for IT-Security, Ruhr University Bochum, Universitätsstrasse 150, D–44780 Bochum, Germany
Roberto.Avanzi@ruhr-uni-bochum.de
[3] Faculty of Engineering and Natural Sciences, Sabanci University, Orhanli-Tuzla, TR–34956 Istanbul, Turkey
erkays@sabanciuniv.edu

Abstract. This paper investigates performance and energy characteristics of software algorithms for long integer arithmetic. We analyze and compare the number of RISC-like processor instructions (e.g. single-precision multiplication, addition, load, and store instructions) required for the execution of different algorithms such as Schoolbook multiplication, Karatsuba and Comba multiplication, as well as Montgomery reduction. Our analysis shows that a combination of Karatsuba-Comba multiplication and Montgomery reduction (the so-called KCM method) allows to achieve better performance than other algorithms for modular multiplication. Furthermore, we present a simple model to compare the energy-efficiency of arithmetic algorithms. This model considers the clock cycles and average current consumption of the base instructions to estimate the overall amount of energy consumed during the execution of an algorithm. Our experiments, conducted on a StrongARM SA-1100 processor, indicate that a 1024-bit KCM multiplication consumes about 22% less energy than other modular multiplication techniques.

1 Introduction

The reduction of energy consumption is a first-class design goal for embedded systems, driven mainly by the proliferation of mobile, battery-powered devices like cell phones, handheld computers, portable media players, and so on. The clock frequency of microprocessors exploded from 33 MHz in the early 1990s to more than 3 GHz in 2005. During the same period, the power consumption

* The work described in this paper has been supported by the Austrian Science Fund under grant number P16952-N04 ("Instruction Set Extensions for Public-Key Cryptography"), and in part by the European Commission through the IST Programme under contract IST-2002-507932 ECRYPT. Erkay Savaş is supported by the Scientific and Technical Research Council of Turkey under project number 104E007.

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 75–90, 2005.

of microprocessors increased by an order of magnitude, even though transistor sizes shrunk by roughly one half every 18-24 months and the supply voltages were scaled down from 5 V to less than 1.5 V. It is expected that the increase in transistor density of microchips will follow Moore's law for (at least) another ten years. Unfortunately, progress in battery technology has not kept up with Moore's law since the average annual growth in battery capacity is less than 12%. Dramatic improvements in battery technology are not foreseen in the next years, which means that the gap between power consumption of microprocessors and available battery capacity will widen in the future.

During the last 15 years, a significant effort has been spent on reducing the overall power and energy consumption of battery-operated devices. Low-power hardware design is a well-established area of research and numerous approaches for power and energy minimization have been proposed. Of fundamental importance in low-power VLSI design is the availability of supporting EDA tools and an appropriate design flow that considers power consumption in all phases of a design. However, since much of the activity of hardware is controlled by software, it is also necessary to analyze the software impact on the hardware energy consumption. A significant problem in this context is the lack of development tools which enable a software designer to systematically evaluate and reduce the energy consumption. While hardware designers have a number of different circuit and gate-level power analysis tools at their disposal, there exist no adequate tools for analyzing the power consumption at high levels of abstraction or to quantify the power cost of software. On the other hand, most software development tools allow functional verification and performance profiling, but provide no support for energy-related cost metrics [23].

Design methodologies for *energy-efficient software* are a relatively new field of research, whereby especially the trade-off of performance versus energy has received large attention. This is, to some extend, also a result of the exponential growth in processor performance, which allows to realize more and more computation-intensive applications in software instead of hardware. It was argued in [23] that *software offers a great potential for energy reduction, but software savings are more difficult to achieve than hardware savings.* A common finding of previous work [25,26,19,15] is that the energy consumption of software is closely tied to the execution time. However, reducing execution time is not the only way to extend battery lifetime. The software energy optimization techniques found in recent literature can be divided into three general categories [19]: reduction of the cost or frequency of memory accesses, selection of the least expensive instructions or instruction sequences, and processor-specific optimizations. References [26,19,4] demonstrate with a number of examples that smart software design can indeed lead to substantial energy savings.

As security and cryptography play an increasingly important role in battery-operated products, energy and power consumption are evolving to critical constraints for embedded cryptographic software. However, while there exists a rich literature dealing with the *performance* of cryptographic software [2,3,14,21], the aspect of *energy-efficiency* has not been widely researched so far, especially in

the context of software for public-key cryptography[1]. With the present paper we attempt to fill this gap. In the following sections, we analyze and compare the execution time and energy dissipation of different algorithms for long integer arithmetic. We show that different algorithms for one and the same arithmetic operation, e.g. multiple-precision multiplication, can require different amounts of energy and that these differences are not only given through unequal execution times, but also through the number of energy-intensive processor instructions executed by the algorithm. Typical examples of costly instructions (in terms of energy) in modern RISC processors are multiply instructions as well as load/store instructions [24]. The use of multiplication algorithms which require fewer load/store instructions (e.g. Comba's method [2]) or fewer multiply instructions (e.g. Karatsuba's method [11]) can reduce the total energy dissipation compared to the "conventional" schoolbook method [16], even when the execution times do not vary significantly.

2 Energy Characteristics of the StrongARM SA-1100

Intel's StrongARM SA-1100 is a high-performance, low-power RISC processor for portable wireless multimedia devices. The SA-1100 processor incorporates the efficiency of the ARMv4 instruction set architecture (ISA) [1] along with the quality of Intel design and process technology [9]. Because of its excellent performance and energy figures, the StrongARM SA-1100 has found widespread use in pocket computers and PDAs such as the HP Jornada 720, Sharp Zaurus SL-5500G, or Compaq iPAQ H3630.

2.1 SA-1100 Instruction Timing

The SA-1100 consists of a 32-bit RISC core with separate instruction and data caches (of size 16 kB and 8 kB, respectively), a memory management unit (MMU), and peripheral controllers (DRAM controller, serial ports, etc.) integrated onto a single chip. It can be run at a variety of clock frequencies, ranging from 39 MHz up to 220 MHz, with a nominal core supply voltage of between 1.5 and 2.0 V [10]. Key characteristics of the processor core are a classic five-stage pipeline (Fetch, Issue, Execute, Buffer, and Register Write) with static branch prediction, and a multiply/accumulate (MAC) unit featuring a (32×12)-bit Wallace tree multiplier. The instruction set of the StrongARM SA-1100 is specified in the ARM Architecture Reference Manual [1].

The SA-1100 employs an *early termination* mechanism for multiply and multiply/accumulate operations, which means that it detects "small" operands and completes a multiplication more quickly. For example, if bits 31-11 of the first operand are all 0, then the multiply operation completes in one cycle. When bits 31-23 are all 0, the multiply spends two cycles in the Execute stage of the pipeline. In all other cases, it spends three cycles in the Execute stage [8].

[1] Contrary to public-key cryptosystems, there exist papers about the energy-efficiency of block ciphers in software [6] and energy aspects of security protocols [7,18,12].

Table 1. Average current consumption of SA-1100 instructions (at 206 MHz) [24]

Instruction type	Avg. current	Avg. energy
Arithmetic/logical instructions	0.178 A	1.296 nJ
Multiply and MAC instructions	0.196 A	1.427–5.709 nJ
Load instructions (cache hit)	0.196 A	1.427 nJ
Store instructions (cache hit)	0.229 A	1.667 nJ
Other instructions	0.170 A	1.238 nJ

The SA-1100 executes "conventional" arithmetic/logical instructions at a rate of one instruction per clock cycle, i.e. every stage of the pipeline is occupied for a single cycle. Load instructions, such as `LWR`, also require one cycle in each pipeline stage, provided that they hit the data cache. However, the pipeline will stall for a cycle if the immediately following instruction uses the loaded value as operand. Store instructions (e.g. `STR`) normally require one clock cycle in each pipeline stage when they hit the data cache. Multiply instructions spend up to three clock cycles in the Execute stage of the pipeline, depending on the magnitude of the first operand. In addition, the "long" multiply instructions producing a 64-bit result (e.g. `UMULL`) require a second cycle in the Buffer stage of the pipeline [8].

2.2 SA-1100 Power Consumption

The energy consumed by a processor while running a certain program can be estimated through *instruction-level power analysis*, first proposed by Tiwari et al. [25,26]. This technique estimates the total amount of energy drawn during the execution of a program by summing up the energy consumed by each individual instruction. Therefore, an instruction-level energy model requires to determine the energy cost of the processor instructions. Tiwari et al. propose to measure the average current dissipation while the processor repeatedly executes a single instruction [25]. Advanced energy models also consider inter-instruction effects like switching activity of buses, pipeline stalls, or cache misses [26].

Sinha and Chandrakasan [24] developed an instruction-level energy profiling tool for the StrongARM SA-1100, called *JouleTrack*. Table 1 shows the average current consumption of SA-1100 instructions, measured at a clock frequency of 206 MHz and a supply voltage of 1.5 V. It is stated in [24] that, on average, arithmetic and logical instructions consume 0.178 A, multiplies 0.196 A, loads 0.196 A, stores 0.229 A, while the other instructions consume about 0.170 A. The StrongARM's total variation in current consumption is 0.072 A, which is 38% of the overall average current consumption [24]. Sinha and Chandrakasan also observed that the current consumptions are pretty uniform and depend only marginally on addressing modes or operand values. However, other processors can have a quite different current profile. For example, the current consumption of the multiply instruction in DSPs will typically be far greater than the current consumed by other instructions.

Load and store instructions are more expensive (in terms of current dissipation) than other instructions that involve just register accesses. Reading or writing a memory location causes switching on highly capacitive address and data buses, row and column decode logic, and data lines with a high fan-out [19]. Also multiply instructions generally have an above-average current consumption. The (32×12)-bit multiplier in the StrongARM SA-1100 is a fairly large circuit and hence a significant source of switching activities. Moreover, it must be considered that the *energy* depends not only on the current consumption, but also on the number of clock cycles that an instruction requires for its execution. The `UMULL` instruction, for example, requires three extra cycles until it leaves the pipeline (two extra cycles in the Execute stage and one extra cycle in the Buffer stage). Therefore, the energy consumption of `UMULL` is about 4.4 times higher than the energy of an "ordinary" arithmetic/logical instruction.

The product of average current consumption, supply voltage, and running time is exactly the energy that the processor dissipates during execution of a program. Although there is a strong relation between execution time and energy consumption, we stress the fact that optimizing for low energy is not the same as minimizing the execution time. Software energy savings can be achieved by reducing the running time or by reducing the average current dissipation of the instructions involved in the execution (or by a combination of both).

3 Multiple-Precision Multiplication

In this section we analyze three principal methods to perform a multiple-precision multiplication: the schoolbook method [16], Comba's method [2], and Karatsuba's method [11]. These three methods form the basis of the algorithms for Montgomery multiplication discussed in Section 4. The schoolbook method represents the most straightforward way to realize a multiple-precision multiplication and is covered in many textbooks. However, the two other methods may perform better in practice. Comba's method requires fewer memory accesses (in particular store operations), whereas Karatsuba's method reduces the number of multiply instructions.

Before describing the methods in detail, we introduce some notation. We represent long integers as arrays of w-bit digits. A typical choice for w is the word-size of the processor, which means $w = 32$ for the implementation that we describe in this paper. The bitlength of the integers is denoted by n, and s is the number of digits necessary to store them, whereby $s = \lceil n/w \rceil$. For example, a 256-bit integer requires $s = 8$ digits on a 32-bit architecture. We shall denote long integers by uppercase letters and use the corresponding lowercase letters for the individual w-bit digits, e.g. $A = (a_{s-1}, \ldots, a_1, a_0)$ with $0 \leq a_i < 2^w$.

3.1 Schoolbook Method

The schoolbook method, shown in Algorithm 1, consists of two nested loops, each looping through the digits of one operand. In each iteration of the outer

Algorithm 1. Multiple-precision multiplication (schoolbook method)

Input: Two s-digit operands $A = (a_{s-1}, \ldots, a_1, a_0)$, $B = (b_{s-1}, \ldots, b_1, b_0)$.
Output: The $2s$-digit product $P = A \cdot B = (p_{2s-1}, \ldots, p_1, p_0)$.

```
1: P ← 0
2: for i from 0 by 1 to s − 1 do
3:    u ← 0
4:    for j from 0 by 1 to s − 1 do
5:       (u, v) ← a_j × b_i + p_{i+j} + u
6:       p_{i+j} ← v
7:    end for
8:    p_{s+i} ← u
9: end for
```

loop, a digit b_i of the operand B is multiplied by all digits of the operand A, and the $(n + w)$-bit results are accumulated according to their weight. The schoolbook method is also called *operand scanning method* since the outer loop moves through the digits of an operand.

An ordered pair of the form (u, v) represents the $2w$-bit (i.e. double-precision) integer $u \cdot 2^w + v$. The schoolbook method performs an operation of the form $a \times b + p + u$ in its inner loop, whereby a, b, p, and u are all w-bit quantities. Therefore, the result of this inner-loop operation is at most $2w$ bits long. This makes the schoolbook method easy to implement in high-level programming languages which provide a double-precision integer datatype. For instance, common extensions of the C and C++ programming language support the datatype `unsigned long long` for 64-bit integers. The Java language provides the `long` type, which has a precision of 64 bits on all platforms.

The square of a long integer can be computed almost twice as fast as the product of two distinct integers, which can be observed from Equation (1).

$$A^2 = \sum_{i=0}^{s-1} \sum_{j=0}^{s-1} a_j \cdot a_i \cdot 2^{(i+j) \cdot w} = \sum_{i=0}^{s-1} a_i^2 \cdot 2^{2 \cdot i \cdot w} + 2 \cdot \sum_{i=0}^{s-2} \sum_{j=i+1}^{s-1} a_j \cdot a_i \cdot 2^{(i+j) \cdot w} \quad (1)$$

Long integer squaring is typically performed in two steps. In the first step, all inner-product terms $a_j \cdot a_i$ with $j \neq i$ are calculated and summed up as shown in Equation (1). The second step doubles the result obtained in the first step and adds the inner products from the "main diagonal", i.e. the terms a_i^2.

3.2 Comba's Method

Algorithm 2 illustrates an alternative method to accomplish a long integer multiplication. This method, first described by Comba [2], also consists of a nested loop structure with a relatively simple inner loop. The two outer loops of Algorithm 2 move through the digits p_i of the product P, and therefore Comba's method is also referred to as *product scanning method*. To obtain the i-th digit p_i of $P = A \cdot B$, all inner-product terms $a_j \times b_{i-j}$ with $0 \leq j \leq i$ are accumulated

Algorithm 2. Multiple-precision multiplication (Comba's method)

Input: Two s-digit operands $A = (a_{s-1}, \ldots, a_1, a_0)$, $B = (b_{s-1}, \ldots, b_1, b_0)$.
Output: The $2s$-digit product $P = A \cdot B = (p_{2s-1}, \ldots, p_1, p_0)$.

```
 1: (t, u, v) ← 0
 2: for i from 0 by 1 to s − 1 do
 3:    for j from 0 by 1 to i do
 4:       (t, u, v) ← (t, u, v) + a_j × b_{i−j}
 5:    end for
 6:    p_i ← v
 7:    v ← u, u ← t, t ← 0
 8: end for
 9: for i from s by 1 to 2s − 2 do
10:    for j from i − s + 1 by 1 to s − 1 do
11:       (t, u, v) ← (t, u, v) + a_j × b_{i−j}
12:    end for
13:    p_i ← v
14:    v ← u, u ← t, t ← 0
15: end for
16: p_{2s−1} ← v
```

and eventually added to carries from the computation of previous digits. The store operation corresponding to each digit of the result only takes place in the outer loop, when the digit is completely evaluated.

Comba's method performs multiply/accumulate (MAC) operations in its inner loop, which means that two w-bit digits are multiplied and the $2w$-bit product is added to a cumulative sum. This sum can easily get longer than $2w$ bits and hence we need three w-bit registers for its storage. Algorithm 2 represents these three registers by the triple (t, u, v). The operation carried out at line 7 and 14 is just a w-bit right-shift of (t, u, v). However, the extended precision of the cumulative sum makes an implementation of Comba's method rather difficult when using high-level programming languages like C/C++ or Java, since they have neither triple-precision data types, nor built-in support for handling carries in an efficient way. On the other hand, Comba's method is typically faster than the schoolbook multiplication when implemented in assembly language.

3.3 Karatsuba's Method

Karatsuba's method reduces a multiplication of two s-digit operands to three multiplications of size $s/2$, but at the cost of an increased number of additions [11]. The three half-size multiplications can either be performed with the schoolbook method, Comba's method, or again Karatsuba's method, provided that the operands are large enough. A product of two s-digit operands with methods such as the schoolbook method or Comba's requires calculating s^2 single-precision multiplications. Karatsuba's method performs only $3s^2/4$ single-precision multiplications. However, when applied recursively, Karatsuba's method results in an algorithm with complexity $O(s^{\log_2 3})$ where $\log_2 3 \approx 1.584$.

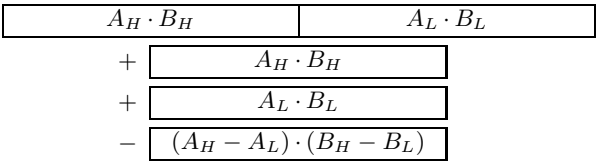


Fig. 1. Graphical representation of Karatuba's method

In order to explain Karatsuba's method, let us assume, for simplicity, that the bitlength n and the number of digits s are both even. The operands A and B are split into two parts of equal length, whereby A_L, B_L consist of the $s/2$ least significant digits, and A_H, B_H of the $s/2$ most significant digits of A and B, respectively. Since $A = A_H \cdot 2^{n/2} + A_L$ and $B = B_H \cdot 2^{n/2} + B_L$, the product $P = A \cdot B$ can be computed as according to the following equation.

$$P = A_H \cdot B_H \cdot 2^n + [A_H \cdot B_H + A_L \cdot B_L - (A_H - A_L) \cdot (B_H - B_L)] \cdot 2^{n/2} + A_L \cdot B_L \quad (2)$$

A graphical representation of Karatuba's method is given in Figure 1. It is also possible to do the calculation with the absolute value for $(A_H - A_L) \cdot (B_H - B_L)$ and to use the sign to decide whether this value is added to or subtracted from $A_H \cdot B_H + A_L \cdot B_L$ [13]. Note that carries may propagate from the most significant digits of $A_H \cdot B_H$, $A_L \cdot B_L$, and $(A_H - A_L) \cdot (B_H - B_L)$ when they are added. Karatsuba squaring is similar to multiplication, but with $A = B$ the equation reduces to three $(s/2)$-digit squares that have to be added according to Figure 1. The middle term $(A_H - A_L)^2$ is always positive, which simplifies the implementation of Karatsuba squaring [5].

3.4 Analysis of the Algorithms

Both the execution time and the energy consumption of the algorithms described in this section depend heavily on the concrete implementation. An implementer could, for instance, fully unroll the inner and outer loops of the algorithms. In this case, only the base instructions like multiplies, adds, loads and stores have to be performed. However, while loop unrolling allows to achieve the best possible performance, it can significantly increase the code size, especially when the number of digits is large. On the other hand, an implementation with "rolled" loops represents the other end of the spectrum. Rolled-loop implementations do not only execute the base instructions mentioned above, but also instructions which do not directly contribute to the calculation of the result. We may think about operations such as incrementing loop counters, branch instructions, register moves, or pointer arithmetic. While an implementation with rolled loops has the benefit of small code-size, it can be significantly slower than an optimized variant with unrolled loops. This makes it necessary to find a trade-off between performance (i.e. unrolled loops) and code-size (i.e. rolled loops). One possible solution is to *partially unroll* the loops. For instance, the body of the loop can be replicated multiple times (e.g. 8 or 16 times), which replaces a number of loop

Table 2. Comparison of base instructions for long integer multiplication algorithms

Algorithm	# MUL	# ADD	# LOAD	# STORE
Schoolbook Mul.	s^2	$4s^2$	$2s^2+s$	s^2+s
Schoolbook Sqr.	$\frac{1}{2}s^2+\frac{1}{2}s$	$2s^2+10s$	s^2+s	$\frac{1}{2}s^2+\frac{3}{2}s$
Comba Multiplication	s^2	$3s^2$	$2s^2$	$2s$
Comba Squaring	$\frac{1}{2}s^2+\frac{1}{2}s$	$\frac{3}{2}s^2+\frac{15}{2}s-3$	s^2+s	$2s$
Karatsuba-Schoolb. Mul.	$\frac{3}{4}s^2$	$3s^2+4s+2$	$\frac{3}{2}s^2+\frac{15}{2}s+1$	$\frac{3}{4}s^2+\frac{11}{2}s+1$
Karatsuba-Schoolb. Sqr.	$\frac{3}{8}s^2+\frac{3}{4}s$	$\frac{3}{2}s^2+19s+2$	$\frac{3}{4}s^2+\frac{15}{2}s+1$	$\frac{3}{8}s^2+\frac{25}{4}s+1$
Karatsuba-Comba Mul.	$\frac{3}{4}s^2$	$\frac{9}{4}s^2+4s+2$	$\frac{3}{2}s^2+6s+1$	$7s+1$
Karatsuba-Comba Sqr.	$\frac{3}{8}s^2+\frac{3}{4}s$	$\frac{9}{8}s^2+\frac{61}{4}s-7$	$\frac{3}{4}s^2+\frac{15}{2}s+1$	$7s+1$

iterations by non-iterated straight-line code. Partial loop unrolling eliminates, or substantially reduces, the effects of the loop overhead (i.e. incrementing the loop counter, branch instruction, etc.). In such case, the loop overhead is (almost) negligible, which means that the execution time and the energy consumption are primarily determined by the base instructions.

Table 2 summarizes the number of *base instructions* (i.e. multiplies, adds, loads, and stores) for the algorithms described before. The schoolbook method performs exactly s^2 iterations of the inner loop. In each iteration, an operation of the form $a \times b + p + u$ is executed, i.e. two w-bit digits are multiplied and another two w-bit digits are added to the product. Note that adding a single-precision digit to a double-precision digit actually involves two `ADD` instructions since a single-precision addition may produce a carry which has to be processed properly[2]. Furthermore, two load instructions (for a_j and p_{i+j}) and one store (for p_{i+j}) are executed in any iteration of the inner loop. The $2w$-bit quantity (u, v) is kept in registers and b_i is loaded once per iteration of the outer loop.

Comba's method also iterates the inner loop exactly s^2 times. Therefore, we have s^2 multiplications and $3s^2$ single-precision additions since the accumulation of a $2w$-bit product to the running sum in (t, u, v) requires one `ADD` and two `ADC` instructions. In any iteration of the inner loop, two operands are loaded from memory, but the stores only take place in the outer loops. Therefore, Comba's method requires only $2s$ `STORE` instructions. Both the Comba and the schoolbook squaring perform only $(s^2 + s)/2$ `MUL` instructions (see Equation 1), which reduces also the number of additions, loads and stores.

A Karatsuba multiplication of two s-digit operands basically consists of three $(s/2)$-digit multiplications and five $(s/2)$-digit additions or subtractions. Table 2 shows the number of base instructions when using the schoolbook method or Comba's method for the half-size multiplications (we do not apply Karatsuba's trick recursively). Note that the addition or subtraction of the s-digit products

[2] More precisely, an `ADD` and an `ADC` (add with carry) instruction are required. However, we ignore this distinction in our analysis and count only the number of single-precision additions, regardless of whether or not the carry flag is considered.

Table 3. Running time (in μs) and average current consumption I_{AVG} (in Ampere)

Algorithm	512 bit		1024 bit		1536 bit		2048 bit	
	Time	I_{AVG}	Time	I_{AVG}	Time	I_{AVG}	Time	I_{AVG}
Schoolbook Mul.	13.8μ	0.193	55.0μ	0.193	124μ	0.193	219μ	0.193
Comba Multiplication	11.3μ	0.191	45.1μ	0.190	101μ	0.190	180μ	0.190
Karatsuba-Schoolb. Mul.	11.6μ	0.194	43.7μ	0.193	96.3μ	0.193	169μ	0.193
Karatsuba-Comba Mul.	9.7μ	0.192	36.2μ	0.191	79.5μ	0.191	140μ	0.191

$A_H \cdot B_H$, $A_L \cdot B_L$, and $(A_H - A_L) \cdot (B_H - B_L)$ may produce a carry. For simplicity, we count one `ADD`, one `LOAD`, and one `STORE` for the processing of this carry.

Performance and Energy Evaluation. The product of average current consumption, supply voltage, and running time is exactly the energy that a processor consumes during the execution of a program. Consequently, we have to estimate the average current and running time in order to analyze the energy efficiency of the algorithms. However, as already mentioned, the running time depends heavily on implementation details like loop unrolling. Therefore, we only consider the base instructions (i.e. multiplications, adds, loads, and stores) for the analysis of the energy efficiency. This is clearly a coarse approach as it ignores pipeline stalls, cache misses, and, in the case of a rolled-loop implementation, the impact of "glue instructions" such as loop control, register moves, pointer management[3], and so on. Nonetheless, this approach is capable of making basic predictions about the execution time and energy consumption, especially for implementations with fully or partially unrolled loops. Note that our estimation can be easily refined to consider also other instructions, e.g. branches.

Table 3 shows the average current consumption and the running time of the multiplication algorithms on a StrongARM SA-1100 processor (at a frequency of 206 MHz and 1.5 V core supply voltage). The values stem from a theoretical evaluation with fully unrolled inner loops. We assumed that the average current of the base instructions is as specified in Section 2 (see Table 1) and that `ADD`, `LOAD`, and `STORE` execute in one clock cycle, while the `MUL` instruction requires four clock cycles, which is actually the case on the SA-1100 when the operands have a magnitude of 32 bits. The running times differ significantly, while the average power consumption shows only slight variations. However, it must be considered that the running time of the algorithms is quite long, and hence even a current saving of a few milli-Amperes can make a difference in the energy consumption. For instance, a 1024-bit schoolbook multiplication consumes an energy of about 15.9 μJ (at 1.5 V), while the Comba multiplication requires only 12.9 μJ. In other words, Comba's method requires 3.0 μJ (i.e. 18.9%) less energy than the schoolbook method. About 7.3% of this 3.0 μJ saving are due to the 3 mA lower current consumption, and the rest due to the shorter running

[3] The ARM architecture supports auto-increment/decrement addressing modes, and hence the pointer management does not fall into account on ARM processors.

Algorithm 3. Montgomery multiplication

Input: An s-digit modulus $M = (m_{s-1}, \ldots, m_1, m_0)$, operands $A = (a_{s-1}, \ldots, a_1, a_0)$ and $B = (b_{s-1}, \ldots, b_1, b_0)$ with $A, B < M$, and the constant $M' = -M^{-1} \bmod 2^n$.
Output: The Montgomery product $Z = A \cdot B \cdot 2^{-n} \bmod M$.
1: $P \leftarrow A \times B$
2: $Q \leftarrow P \times M' \bmod 2^n$
3: $Z \leftarrow (P + Q \times M)/2^n$
4: **if** $Z \geq M$ **then** $Z \leftarrow Z - M$ **end if**

time. The results for 1536 and 2048-bit operands are similar. Comba's method reduces the energy not only because it requires fewer `STORE` instructions, but also due to the fact that saved `STORE` instructions have an above-average current consumption (see Table 1).

Our theoretical evaluation shows that the Karatsuba-Comba multiplication is superior to all other methods with respect to running time and energy consumption, even for a short operand length like 512 bits. Besides the theoretical evaluation, we also implemented the algorithms and simulated them with *Joule-track* [24], an instruction-level energy profiler for the StrongARM SA-1100. The simulation results confirm that Karatsuba-Comba multiplication is faster and more energy-efficient than the other methods described in this section.

4 Montgomery Multiplication

The Montgomery multiplication algorithm [17] is an efficient method for performing modular multiplication with an odd modulus. Montgomery's algorithm replaces the trial division with simple shift operations, which are particularly suitable for implementations on general-purpose processors.

Given two integers A and B, and the modulus M, the Montgomery multiplication algorithm computes $Z = \text{MonMul}(A, B) = A \cdot B \cdot R^{-1} \bmod M$, whereby $A, B < M$ and R is a constant such that $\gcd(R, M) = 1$. Even though the algorithm works for any R which is relatively prime to M, it is more useful when the so-called Montgomery residual factor R is a power of two, e.g. $R = 2^n$ where $n = \lceil log_2(M) \rceil$. The Montgomery product $A \cdot B \cdot 2^{-n} \bmod M$ of the two integers A and B can be calculated as shown in Algorithm 3. First, the two operands are multiplied together to obtain the product P. The following two multiplications reduce the product modulo M, whereby only the lower part of the result of the first multiplication is needed, and from the second multiplication only the higher part. A final subtraction of M can be necessary to bring the result into the range of $[0, M-1]$. The constant M' depends only on the modulus M and hence it can be pre-computed. In summary, a Montgomery multiplication is only slightly more costly than two conventional multiplications of n-bit integers.

The Montgomery multiplication algorithm calculates the Montgomery product $A \cdot B \cdot 2^{-n} \bmod M$ instead of the actual residue $A \cdot B \bmod M$, i.e. the result carries the factor 2^{-n}. Therefore, Montgomery arithmetic requires a conversion

Algorithm 4. Montgomery multiplication (Coarsely Integrated Operand Scanning)

Input: An s-digit modulus $M = (m_{s-1}, \ldots, m_1, m_0)$, operands $A = (a_{s-1}, \ldots, a_1, a_0)$ and $B = (b_{s-1}, \ldots, b_1, b_0)$ with $A, B < M$, and the constant $m'_0 = -m_0^{-1} \bmod 2^w$.
Output: The Montgomery product $Z = A \cdot B \cdot 2^{-n} \bmod M$.

$Z \leftarrow 0$
for i from 0 by 1 to $s-1$ **do**
  $u \leftarrow 0$
  for j from 0 by 1 to $s-1$ **do**
    $(u, v) \leftarrow a_j \times b_i + z_j + u$
    $z_j \leftarrow v$
  end for
  $(u, v) \leftarrow z_s + u$
  $z_s \leftarrow v$
  $z_{s+1} \leftarrow u$
  $q \leftarrow z_0 \times m'_0 \bmod 2^w$
  $(u, v) \leftarrow z_0 + m_0 \times q$
  for j from 1 by 1 to $s-1$ **do**
    $(u, v) \leftarrow m_j \times q + z_j + u$
    $z_{j-1} \leftarrow v$
  end for
  $(u, v) \leftarrow z_s + u$
  $z_{s-1} \leftarrow v$
  $z_s \leftarrow z_{s+1} + u$
end for
if $Z \geq M$ **then** $Z \leftarrow Z - M$ **end if**

of operands and a re-conversion of the result in order to get rid of this factor [16]. We will not further discuss the basics of Montgomery multiplication since they are covered in a number of papers and textbooks, e.g. in [3,20,14,16].

4.1 Coarsely Integrated Operand Scanning (CIOS)

Koç et al. [14] describe a number of efficient software algorithms for calculating the Montgomery product on general-purpose processors. One of these methods is the so-called *Coarsely Integrated Operand Scanning* (CIOS) method, which can be phrased as shown in Algorithm 4. The CIOS method may be viewed as schoolbook multiplication with a "coarse" integration of the Montgomery reduction, i.e. multiplication and reduction steps are performed in the same outer loop, but different inner loops. Therefore, the CIOS method has the same inner-loop operation as the schoolbook method, which makes it simple to implement in both assembly and high-level programming languages. Koç et al. reported that the CIOS method achieves better performance than the other methods described in [14]. Therefore, we use the CIOS method as a "benchmark" for our energy evaluation. Further details about the CIOS method can be found in [14].

Algorithm 5. Montgomery reduction (product scanning form) [20]

Input: An s-digit modulus $M = (m_{s-1}, \ldots, m_1, m_0)$, operand $P = (p_{2s-1}, \ldots, p_1, p_0)$ with $P < 2M - 1$, and the constant $m'_0 = -m_0^{-1} \bmod 2^w$.
Output: The Montgomery residue $Z = P \cdot 2^{-n} \bmod M$.

1: $(t, u, v) \leftarrow 0$
2: **for** i from 0 by 1 to $s-1$ **do**
3: **for** j from 0 by 1 to $i-1$ **do**
4: $(t, u, v) \leftarrow (t, u, v) + z_j \times m_{i-j}$
5: **end for**
6: $(t, u, v) \leftarrow (t, u, v) + p_i$
7: $z_i \leftarrow v \times m'_0 \bmod 2^w$
8: $(t, u, v) \leftarrow (t, u, v) + z_i \times m_0$
9: $v \leftarrow u,\ u \leftarrow t,\ t \leftarrow 0$
10: **end for**
11: **for** i from s by 1 to $2s-2$ **do**
12: **for** j from $i-s+1$ by 1 to $s-1$ **do**
13: $(t, u, v) \leftarrow (t, u, v) + z_j \times m_{i-j}$
14: **end for**
15: $(t, u, v) \leftarrow (t, u, v) + p_i$
16: $z_{i-s} \leftarrow v$
17: $v \leftarrow u,\ u \leftarrow t,\ t \leftarrow 0$
18: **end for**
19: $(t, u, v) \leftarrow (t, u, v) + p_{2s-1}$
20: $z_{s-1} \leftarrow v,\ z_s \leftarrow u$
21: **if** $Z \geq M$ **then** $Z \leftarrow Z - M$ **end if**

4.2 Karatsuba-Comba-Montgomery (KCM) Multiplication

The Karatsuba-Comba-Montgomery (KCM) method combines Karatsuba and Comba-like multiplication techniques with Montgomery reduction [22]. Contrary to CIOS, the KCM method completely separates the multiplication of A by B and the reduction of the product modulo M. The KCM method employs Karatsuba-Comba multiplication for the former [21], while the latter is realized with a product scanning technique as shown in Algorithm 5 [20]. This algorithm accomplishes the Montgomery reduction in a similar way as Algorithm 3. The first outer loop (lines 2-10) of Algorithm 5 calculates the s digits of the product $Q = P \cdot M' \bmod 2^n$ and stores them in $(z_{s-1}, \ldots, z_1, z_0)$. Thereafter, the second loop (lines 11-20) produces the Montgomery residue $Z = (P + Q \cdot M)/2^n$. More details about Algorithm 5 and the KCM method can be found in [20].

4.3 Analysis of the Algorithms

Each of the two inner loops of the CIOS method is iterated s^2 times and hence s^2 `MUL` instructions are carried out. In addition, s single-precision multiplications are performed in the outer loop, which results in a total of $2s^2 + s$ `MUL` instructions. Only s^2 of these $2s^2 + s$ `MUL` instructions actually contribute to the multiplication of $A \cdot B$, while the remaining $s^2 + s$ `MUL` instructions contribute to the

Table 4. Comparison of base instructions for Montgomery multiplication algorithms

Algorithm	# MUL	# ADD	# LOAD	# STORE
CIOS Multiplication	$2s^2+s$	$4s^2+4s+2$	$4s^2+7s+2$	$2s^2+4s+1$
CIOS Squaring	$\frac{3}{2}s^2+\frac{5}{2}s$	$4s^2+7s+2$	$3s^2+6s+2$	$\frac{3}{2}s^2+\frac{11}{2}s+1$
KCM Multiplication	$\frac{7}{4}s^2+s$	$\frac{13}{4}s^2+8s+4$	$\frac{7}{2}s^2+11s+3$	$10s+1$
KCM Squaring	$\frac{11}{8}s^2+\frac{7}{4}s$	$\frac{17}{8}s^2+\frac{77}{44}s-5$	$\frac{11}{4}s^2+\frac{25}{2}s+3$	$10s+1$

Table 5. Running time (in μs) and average current consumption I_{AVG} (in Ampere)

Algorithm	512 bit		1024 bit		1536 bit		2048 bit	
	Time	I_{AVG}	Time	I_{AVG}	Time	I_{AVG}	Time	I_{AVG}
CIOS Multiplication	23.9μ	0.196	92.5μ	0.196	206μ	0.196	364μ	0.196
CIOS Squaring	20.3μ	0.195	76.5μ	0.195	169μ	0.195	297μ	0.195
KCM Multiplication	19.7μ	0.193	73.5μ	0.192	163μ	0.192	284μ	0.192
KCM Squaring	15.3μ	0.194	56.4μ	0.193	123μ	0.193	216μ	0.193

calculation of the Montgomery reduction. Also the reduction technique shown in Algorithm 5 performs $s^2 + s$ MUL instructions. However, the KCM method uses Algorithm 5 in combination with the Karatsuba-Comba method for the calculation of the product, and hence the overall number of MUL instructions is much smaller than in the CIOS method. Furthermore, the KCM method requires only a linear number of STORE instructions, since both Algorithm 5 and the Karatsuba-Comba method implement a product-scanning technique. The number of base instructions are summarized in Table 4.

Table 5 shows the running time and the average current consumption of the CIOS and the KCM method. These values have been obtained through a theoretical evaluation with the base instructions MUL, ADD, LOAD, and STORE as described in Section 3.4. The KCM method is faster and has a lower average current consumption than the CIOS method, mainly because it requires fewer MUL and STORE instructions. However, while the current values vary only by 4 mA, the running times differ significantly. For instance, a 1024-bit CIOS multiplication has a running time of 92.5 μs, but the KCM method requires only 73.5 μs, which means that the latter is 19.0 μs (20.5%) faster. The corresponding energy values differ by 6.0 μJ or 22.1% (27.2 μJ versus 21.2 μJ). About 8% of this saving of 6.0 μJ is due to the lower average current of the KCM base instructions. The same percentage holds for 1536 and 2048-bit operands.

In summary, more than 90% of the KCM method's energy advantage stems from the shorter execution time, while the remaining part is due to the lower power consumption. Consequently, there is a close relation between the performance and energy consumption of Montgomery multiplication algorithms. We have also simulated the algorithms with *JouleTrack*, and the simulation results confirm the superiority of the KCM method, even for 512-bit operands.

5 Conclusions

The contribution of this paper is twofold. We aimed at determining how to implement basic arithmetic algorithms for public-key cryptography with the goal to minimize the energy consumption. Several different algorithms have been considered. The higher goal, however, was to pave the way for a systematic approach to the evaluation of energy costs of arithmetic algorithms.

We performed a theoretical analysis with the help of base instructions (multiplication, addition, load, and store), and combined it with the actual energy consumption of these instructions on a specific architecture. Our results show that a combination of Karatsuba and Comba multiplication with Montgomery reduction (the KCM method) leads to the best energy efficiency. For example, a 1024-bit modular multiplication according to the CIOS method requires an energy of 27.2 μJ on the StrongARM SA-1100. The KCM method, on the other hand, needs only 21.2 μJ, which corresponds to an energy saving of more than 22%. This energy saving results from the fact that the KCM method requires fewer energy-intensive instructions like multiply and store instructions.

The power consumption of actual implementations of the considered algorithms was simulated using *JouleTrack*. We found the relative performance and energy figures from the simulation in perfect agreement with our theoretical model. Hence, by analyzing the energy cost of the base instructions, it is possible to obtain most of the information needed to evaluate the energy-efficiency of different algorithms for long integer arithmetic.

Acknowledgements. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

1. ARM Limited. ARM Architecture Reference Manual. ARM Doc No. DDI-0100, Issue H, Oct. 2003.
2. P. G. Comba. Exponentiation cryptosystems on the IBM PC. *IBM Systems Journal*, 29(4):526–538, Dec. 1990.
3. S. R. Dussé and B. S. Kaliski. A cryptographic library for the Motorola DSP56000. In *Advances in Cryptology — EUROCRYPT '90*, vol. 473 of *Lecture Notes in Computer Science*, pp. 230–244. Springer Verlag, 1991.
4. J. R. Goodman. *Energy Scalable Reconfigurable Cryptographic Hardware for Portable Applications*. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2000.
5. T. Granlund. GNU MP: The GNU Multiple Precision Arithmetic Library. Manual, available for download at `http://swox.com/gmp/gmp-man-4.1.4.pdf`, Sept. 2004.
6. C. T. Hager, S. F. Midkiff, J.-M. Park, and T. L. Martin. Performance and energy efficiency of block ciphers in personal digital assistants. In *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom 2005)*, pp. 127–136. IEEE Computer Society Press, 2005.

7. A. Hodjat and I. M. Verbauwhede. The energy cost of secrets in ad-hoc networks. In *Proceedings of the 5th IEEE CAS Workshop on Wireless Communications and Networking*. IEEE, 2002.
8. Intel Corporation. StrongARM SA-110 microprocessor instruction timing. Application note, order number 278194-001, Sept. 1998.
9. Intel Corporation. Intel® StrongARM® SA-1100 microprocessor for embedded applications. Brief datasheet, order number 278092-005, June 1999.
10. Intel Corporation. Intel® StrongARM® SA-1100 microprocessor. Specification update, order number 278105-025, Feb. 2000.
11. A. A. Karatsuba and Y. P. Ofman. Multiplication of multidigit numbers on automata. *Doklady Akademii Nauk SSSR*, 145(2):293–294, 1962.
12. R. Karri and P. Mishra. Optimizing the energy consumed by secure wireless sessions – Wireless Transport Layer Security case study. *Mobile Networks and Applications*, 8(2):177–185, Apr. 2003.
13. D. E. Knuth. *Seminumerical Algorithms*, vol. 2 of *The Art of Computer Programming*. Addison-Wesley, 3rd edition, 1998.
14. Ç. K. Koç, T. Acar, and B. S. Kaliski. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, June 1996.
15. H. Mehta, R. M. Owens, M. J. Irwin, R. Chen, and D. Ghosh. Techniques for low energy software. In *Proceedings of the 2nd International Symposium on Low Power Electronics and Design (ISLPED '97)*, pp. 72–75. ACM Press, 1997.
16. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
17. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, Apr. 1985.
18. N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha. Analyzing the energy consumption of security protocols. In *Proceedings of the 8th International Symposium on Low Power Electronics and Design (ISLPED 2003)*, pp. 30–35. ACM Press, 2003.
19. K. Roy and M. C. Johnson. Software design for low power. In *Low Power Design in Deep Submicron Electronics*, vol. 337 of *NATO Advanced Science Institutes Series*, chapter 6.3, pp. 433–460. Kluwer Academic Publishers, 1997.
20. M. P. Scott. Fast machine code for modular multiplication. Manuscript, available for download at `ftp://ftp.computing.dcu.ie/pub/crypto/fast_mod_mult2.ps`, Jan. 1995.
21. M. P. Scott. Comparison of methods for modular exponentiation on 32-bit Intel 80x86 processors. Informal draft, available for download at `ftp://ftp.computing.dcu.ie/pub/crypto/timings.ps`, June 1996.
22. Shamus Software Ltd. M.I.R.A.C.L. Users Manual. Available for download at `ftp://ftp.computing.dcu.ie/pub/crypto/manual.doc`, Nov. 2004.
23. T. Šimunić. *Energy Efficient System Design and Utilization*. Ph.D. Thesis, Stanford University, Stanford, CA, USA, Feb. 2001.
24. A. Sinha and A. P. Chandrakasan. JouleTrack - A web based tool for software energy profiling. In *Proceedings of the 38th Design Automation Conference (DAC 2001)*, pp. 220–225. ACM Press, 2001.
25. V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):437–445, Dec. 1994.
26. V. Tiwari, S. Malik, A. Wolfe, and T.-C. Lee. Instruction level power analysis and optimization of software. *Journal of VLSI Signal Processing*, 13(2–3):223–238, Aug. 1996.

Short Memory Scalar Multiplication on Koblitz Curves

Katsuyuki Okeya[1], Tsuyoshi Takagi[2], and Camille Vuillaume[1]

[1] Hitachi, Ltd., Systems Development Laboratory, Kawasaki, Japan
{ka-okeya, camille}@sdl.hitachi.co.jp
[2] Future University - Hakodate, Japan
takagi@fun.ac.jp

Abstract. We present a new method for computing the scalar multiplication on Koblitz curves. Our method is as fast as the fastest known technique but requires much less memory. We propose two settings for our method. In the first setting, well-suited for hardware implementations, memory requirements are reduced by 85%. In the second setting, well-suited for software implementations, our technique reduces the memory consumption by 70%. Thus, with much smaller memory usage, the proposed method yields the same efficiency as the fastest scalar multiplication schemes on Koblitz curves.

Keywords: *Elliptic curve cryptosystems, Koblitz curves, scalar multiplication, NAF, polynomial basis, normal basis, change-of-basis, smartcard.*

1 Introduction

Elliptic curves cryptosystems (ECC) offer an interesting alternative to standard prime-field based cryptosystem, because for the same security level, they are much faster and require less memory [11,13]. In particular, they are well-suited for implementations on low-end processors and memory-constrained devices such as smartcards. From the geometrical properties of elliptic curves, one can define two operations on the points of the curve: point addition and point doubling. Then, given a base point P and a scalar d, one can compute the scalar multiplication $Q = dP$. It is believed that the discrete logarithm problem on elliptic curves (EC-DLP), namely finding d from P and Q, is a hard problem, and many cryptosystems rely on the hardness of the EC-DLP. Since the scalar multiplication plays a crucial role in such cryptosystems, it is important to implement it efficiently. In practical cases, we distinguish two types of scalar multiplications. On the one hand, the scalar multiplication with known base point can be computed very efficiently. On the other hand, when the base point is random, much more computational effort is required.

A first approach to decrease the computational cost of the scalar multiplication with unknown base point is to deploy a new representation of the scalar in order to minimize the number of elliptic operations. The standard technique is to precompute some small multiples of the base point P and re-use these points in the scalar multiplication. One of the fastest recoding technique is the width

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 91–105, 2005.

w non-adjacent form (NAF_w), which reduces the number of point additions to $m/(w+1)$ on average for a m-bit scalar, provided that 2^{w-2} points are precomputed. Unfortunately, recoding techniques have no influence on the number of point doublings, and as a consequence, the speed-up which arises from the NAF_w is limited.

A second approach is to use special curves. In particular, on some special binary curves called Koblitz curves, all point doublings can be replaced by a much cheaper operation: the Frobenius automorphism [12]. Thus, NAF_w techniques and Koblitz curves nicely combine: the NAF_w reduces the number of point additions while point doublings are eliminated thanks to special algebraic properties of the curve [18]. Since the computational cost of precomputations grows with 2^{w-2} whereas that of the scalar multiplication itself decreases with $1/(w+1)$, obviously, there is an optimal value for w. In practical situations, $w = 5$ is optimal for Koblitz curves.

In [1], a method for computing the scalar multiplication on Koblitz curves without precomputations was proposed. The technique is faster than the NAF_2 on Koblitz curves, but slower than the NAF_3 which has one precomputed point. Additionally, it is well-known that on binary field, with special bases called normal bases, squares are virtually free. Techniques based on normal basis representations have been proposed to speed-up the scalar multiplication on Koblitz curves, with known point and large memory [4].

We propose a new method which can exploit the speed-up arisen from precomputations without actually storing all of the precomputed points: we keep the full power of precomputations without sacrifice on the side of memory. In fact, we embed the precomputation process in the scalar multiplication: the computations of the scalar multiplication are re-ordered and the precomputed points are generated sequentially. In our method, the order of computations does not play any role in the global efficiency, and operations can be freely re-ordered. Therefore, our scheme can be performed from right to left, or from left to right, or even following a random sequence, generalizing the notion of right-to-left or left-to-right computations. Our algorithm has two different settings. With the first setting, which is well-suited for hardware implementations, our method is as fast as the (optimal) NAF_5 but requires only one auxiliary point instead of seven, reducing the memory consumption by 85%. The second setting is well-suited for software implementations, that is, for general-purpose processors, but requires two auxiliary points instead of seven, reducing memory consumption by 70%.

2 Preliminaries

In this section, we discuss known facts: we describe the properties of polynomial and normal basis implementations of binary fields, and introduce Koblitz curves.

2.1 Polynomial Bases vs. Normal Bases

In the field $\mathbb{F}_{2^m}$, elements are represented with respect to a basis $(\epsilon_0, \ldots, \epsilon_{m-1})$, where $\epsilon_i \in \mathbb{F}_{2^m}$. Then, the element $b \in \mathbb{F}_{2^m}$ is represented with the vector with

binary entries $(b_0 \dots b_{m-1})_2$, corresponding to the polynomial $\sum_{i=0}^{m-1} b_i \epsilon_i$. For efficient implementations, two types of bases are usually considered: polynomial and normal bases.

In a polynomial basis, ϵ_i is the monomial X^i of degree i, and operations are computed modulo an irreducible polynomial $\Pi[X]$ of degree m, appropriately chosen in order to speed-up reduction. Also, there are efficient algorithms for multiplications and squarings in $\mathbb{F}_{2^m}$ using polynomial bases [6]. A normal basis consists of the m-tuple $(\beta_0, \dots, \beta_{m-1})$, where $\beta_i^2 = \beta_{i+1}$ for $i < m-1$ and $\beta_{m-1}^2 = \beta_0$. In other words, powers of two are simple cyclic shifts. Unfortunately, normal basis multiplications are slow. On the one hand, in hardware, with an appropriate circuitry and good parameter choices, the penalty arisen from slow multiplications can be minimized and normal bases yield an elegant and efficient solution to implement binary fields. On the other hand, in software, polynomial bases largely outperform normal bases [5].

Even though a full normal basis approach seems too slow to compete with polynomial basis implementations, with a mixed normal-polynomial approach, one can benefit from the advantages of both types of bases: fast multiplications with polynomial bases and fast computation of powers of two with normal bases. Indeed, there are techniques to convert elements between their normal and polynomial basis representations, using change-of-basis matrices [7]. The conversion time is roughly the same as one polynomial basis multiplication, and each matrix occupies m^2 bits in memory [4]. Note that change-of-basis matrices can be precomputed off-line and stored in ROM.

2.2 Koblitz Curves

Koblitz curves belong to a special class of elliptic curves defined over binary fields, and additionally offer a very efficient arithmetic, with no significant security flaw compared to general binary curves [12]. They are defined over a binary field $\mathbb{F}_{2^m}$ by the equation:

$$E_a : y^2 + xy = x^3 + ax^2 + 1, \tag{1}$$

where $a \in \{0, 1\}$. We denote by $E_a(\mathbb{F}_{2^m})$ the abelian group of the points of the Koblitz curve over $\mathbb{F}_{2^m}$, along with the point of infinity $\mathcal{O}$, neutral element of the addition law.

The main interest of Koblitz curves is that point doublings can be totally eliminated from the scalar multiplication, and replaced by the efficiently computable Frobenius automorphism $\Phi : (x, y) \mapsto (x^2, y^2)$. Since the quadratic equation $(x^4, y^4) + 2(x, y) = \mu(x^2, y^2)$ where $\mu = (-1)^{1-a}$ holds for every point of Koblitz curves, the Frobenius map can be seen as the complex $\tau = (\mu + \sqrt{-7})/2$, solution of the equation $\Phi^2 + 2 = \mu\Phi$. The approach for fast computations over Koblitz curves is to convert a scalar d to a radix τ expansion such as $d = \sum_{i=0}^{j} d_i \tau^i$, $d_i \in \{0, \pm 1\}$: in the scalar multiplication, point doublings are replaced by the efficiently computable Frobenius map. However, in order to fully take advantage of the Frobenius map, the τ expansion must be sparse and short.

In [18], Solinas proposed two efficient algorithms to satisfy these properties: partial reduction modulo $\delta = (\tau^m - 1)/(\tau - 1)$ and radix-τ NAF recoding.

One can see radix-τ integers as elements of the quadratic field $\mathbb{Z}[\tau]$, with norm $|\lambda| = l_0^2 + \mu l_0 l_1 + 2l_1^2$. Then the division with remainder $\gamma = \kappa \cdot \delta + \rho$ in $\mathbb{Z}[\tau]$ satisfies $|\rho| \leq 4/7 * |\delta|$. Here $\zeta P = (\zeta \bmod \delta)P$ holds for $\delta = (\tau^m - 1)/(\tau - 1)$ and $\zeta \in \mathbb{Z}[\tau]$, and thus the radix τ expansion of reduced scalars becomes shorter.

To generate a width w radix τ NAF expansion (TNAF_w), one can use the map $\Phi_w : u_0 + u_1 \cdot \tau \in \mathbb{Z}[\tau] \mapsto u_0 + u_1 \cdot t_w \bmod 2^w \in \mathbb{Z}/2^w\mathbb{Z}$, where $t_w = 2U_{w-1}U_w^{-1} \bmod 2^w$, U_w is the Lucas sequence U_w, defined by $U_0 = 0$, $U_1 = 1$ and $U_{w+1} = \mu U_w - 2U_{w-1}$ for $w \geq 1$, and "mods 2^w" means the signed residue modulo 2^w. Interestingly, the odd representatives modulo τ^w correspond exactly to the odd integers in $\mathbb{Z}/2^w\mathbb{Z}$ by Φ_w [18]. Therefore, the congruence classes modulo τ^w can be easily computed using Φ_w.

Algorithm 1. TNAF_w with partial modular reduction modulo δ [18]

INPUT: Scalar d ;
OUTPUT: TNAF_w(s);

1. $c_0 \leftarrow d$ partmod δ (partial modular reduction); $c_1 \leftarrow 0$; $i \leftarrow 0$;
2. **while** $c_0 \neq 0$ **or** $c_1 \neq 0$ **do**
   (a) **if** c_0 is odd **then** $u \leftarrow \Phi_w(c_0 + c_1\tau)$ **else** $u \leftarrow 0$;
   (b) $d_i^{(w)} \leftarrow u$; $c_0 \leftarrow c_0 - u$;
   (c) $(c_0, c_1) \leftarrow (c_1 + \mu c_0/2, -c_0/2)$; $i \leftarrow i + 1$;
3. **return** $(d_{i-1}^{(w)} \ldots d_1^{(w)} d_0^{(w)})$;

If the scalar is first reduced modulo δ, the length of the TNAF_w is at most $m+a+3$, and does not exceed $m+a$ with high probability [18]. Since the average nonzero density of the TNAF_w is $1/(w+1)$, the scalar multiplication requires $(m+a)/(w+1)$ point additions. Additionally, the points $P, 3P, \ldots, (2^{w-1} - 1)P$ must be precomputed. Therefore, the total cost of the scalar multiplication using TNAF_w is on average:

$$\mathcal{C}_{NAF} = (m + a) \cdot \mathrm{ECFRB} + \left(\frac{m + a}{w + 1} + 2^{w-2} - 1 \right) \cdot \mathrm{ECADD} + \mathrm{ECDBL} \quad (2)$$

where ECADD, ECFRB and ECDBL stand for the computational cost of point additions, τ multiplications and point doublings, respectively.

3 Short Memory Scalar Multiplication

We present the basic idea to compute the TNAF_5 with memory for only one auxiliary point instead of the seven usual precomputed points, assuming that a normal basis is used for representing elements of the underlying field.

3.1 Motivation

The general approach to decrease the cost of the scalar multiplication for an unknown base point is to precompute some multiples of the base point in order to minimize the number of point additions in the scalar multiplication. This method is particularly effective on Koblitz curves where point additions *only* are significant for the total computational cost. Especially, in the case of the TNAF_w method, for several binary fields of cryptographic interest ($m = 163, 233, 283, 409$), the width $w = 5$ is optimal[1]. Unfortunately, in practical situations, scarce memory resources might prevent us from choosing the optimal width. For example, for $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$, $\mathbb{F}_{2^{283}}$ and $\mathbb{F}_{2^{409}}$, the storage of 7 precomputed points occupies 294, 420, 504 and 728 bytes in RAM, respectively. Newer smartcards have larger RAM space, but the current trend is to develop multi-application cards with greedy memory requirements on the OS side. Generally, independently from how much RAM is available, only about 500 bytes are allocated for cryptography in total. Thus, having an important part, or even worse, the totality of the allocated memory occupied by precomputed values can be a serious drawback. Besides, in hardware and in software, it is advantageous to decrease memory requirements, and as a consequence, decrease the cost of the final device.

We propose a solution to drastically reduce the memory requirements of the optimal TNAF_5 scalar multiplication. Our approach is as follows: we re-group calculations involving the same precomputed point, which can be discarded immediately after such calculations are completed.

3.2 Sequential Precomputations

Since we aim at discarding precomputed points when they are not needed anymore, we need a procedure for computing them sequentially.

Definition 1. *We call sequential precomputations an ordered sequence of points where the kth point P_k can be computed from the base point P and the previous point P_{k-1}, with one point addition and several τ multiplications.*

In [18], the precomputed points $3P, 5P, \ldots, (2^{w-1} - 1)P$ are calculated by successively adding $2P$. Since the non-trivial point $2P$ must be readily available, the precomputations are not sequential in the sense of Definition 1. The computational cost of this technique is $(2^{w-2} - 1)\mathrm{ECADD} + \mathrm{ECDBL}$. Instead of $u = \pm 1, \ldots, \pm(2^{w-1} - 1)$, one can take $(u \bmod \tau^w)$ as coefficients in the TNAF_w. The precomputed points $(u \bmod \tau^w)P$ can be efficiently calculated thanks to relationships between the TNAF_2 representations of $(u \bmod \tau^w)$, for $u = 1, \ldots, (2^{w-1} - 1)$. Then, the computational cost of precomputations can be reduced to only $2^{w-2} - 1$ point additions [6]. With this technique, each point $(u \bmod \tau^w)P$ can be computed independently for $w < 5$: sequential precomputations are trivially possible for the TNAF_2, TNAF_3 and TNAF_4. However, for $w = 5$, it is necessary to store $(5 \bmod \tau^5)P$ during the whole precomputation

[1] On general curves, one can achieve a greater speed-up by using fractional width techniques. However, such a method has not been proposed yet for Koblitz curves.

process, which is unacceptable for our aims: we do not wish to store any non-trivial point. This is unfortunate because $w = 5$ is usually the optimal width. Thus, we have to look for new a technique which allows us to compute the points sequentially in the TNAF_5.

Unlike the standard method which utilizes the TNAF_2 representations of $(u \bmod \tau^5)$ for the precomputations, we use the binary expansion of $u \bmod \tau^5$. Then, we propose a precomputation technique which requires $2^{w-2}-1$ point additions and several τ multiplications, with memory for only one non-trivial point.

Table 1. Odd representatives modulo τ^5

	Case $a = 0$		**Case** $a = 1$	
u	$\alpha_u = u \bmod \tau^5$	binary exp. of α_u	$\alpha_u = u \bmod \tau^5$	binary exp. of α_u
1	1	1	1	1
3	$-\tau - 3$	$\tau^2 - 1$	$\tau - 3$	$\tau^2 - 1$
5	$-\tau - 1$	$-\tau - 1$	$\tau - 1$	$\tau - 1$
7	$-\tau + 1$	$-\tau + 1$	$\tau + 1$	$\tau + 1$
9	$-2\tau - 3$	$-\tau^4 + \tau - 1$	$2\tau - 3$	$-\tau^4 - \tau - 1$
11	$-2\tau - 1$	$\tau^3 + \tau^2 - 1$	$2\tau - 1$	$-\tau^3 + \tau^2 - 1$
13	$-2\tau + 1$	$\tau^3 + \tau^2 + 1$	$2\tau + 1$	$-\tau^3 + \tau^2 + 1$
15	$3\tau + 1$	$-\tau^3 - \tau^2 + \tau + 1$	$-3\tau + 1$	$\tau^3 - \tau^2 - \tau + 1$

Table 1 shows the representatives modulo τ^5 and their binary expansion, for the cases $a = 0$ and $a = 1$. Interestingly, one can find sequential relationships between the representatives: for instance, $\alpha_{11} = \tau^3 + \alpha_3$ and $\alpha_{15} = \tau - \alpha_{11}$. Indeed, thanks to these relationships, it is possible to compute the points $(u \bmod \tau^5)P$ sequentially: Table 2 presents a practical solution for such sequential precomputations, following the sequence $\{1, 3, 11, 15, 5, 13, 7, 9\}$. Then, the computational cost of our precomputation technique using a normal and a polynomial basis is respectively:

$$\mathcal{C}_{0,n}^{(5)} = 7 \cdot \mathrm{ECADD}_n + 7 \cdot \mathrm{ECFRB}_n \text{ and } \mathcal{C}_{0,p}^{(5)} = 7 \cdot \mathrm{ECADD}_p + 14 \cdot \mathrm{ECFRB}_p, \quad (3)$$

where the indexes n and p stand for the type of basis (normal or polynomial). In the following, the index 0 in $\mathcal{C}_0$ will refer to the cost of precomputations.

3.3 Short Memory Scalar Multiplication with a Normal Basis

On standard elliptic curves, the precomputation work must be done before the scalar multiplication itself; all precomputed points are stored in RAM. On Koblitz curves implemented with a normal basis, we will show that this is unnecessary. Indeed, when P is known, $\tau^i P$ can be computed with i-fold cyclic shifts of the coordinates of P. In other words, the order of the computations of the scalar multiplication $dP = \sum_{i=0}^{m+a} d_i \tau^i P$ does not matter. Especially, one can re-group calculations involving the same precomputed points, as shown in Algorithm 2.

Table 2. Sequential computation of $(u \bmod \tau^5)P$

u	$\alpha_u P = (u \bmod \tau^5)P, u = 0$	$\alpha_u P = (u \bmod \tau^5)P, u = 1$
1	$F(1, P, -^{\text{a}}) = \alpha_1 P = P$	$F(1, P, -) = \alpha_1 P = P$
3	$F(3, P, -) = \alpha_3 P = \tau^2 P - P$	$F(3, P, -) = \alpha_3 P = \tau^2 P - P$
11	$F(11, P, \alpha_3 P) = \alpha_{11} P = \tau^3 P + \alpha_3 P$	$F(11, P, \alpha_3 P) = \alpha_{11} P = -\tau^3 P + \alpha_3 P$
15	$F(15, P, \alpha_{11} P) = \alpha_{15} P = \tau P - \alpha_{11} P$	$F(15, P, \alpha_{11} P) = \alpha_{15} P = -\tau P - \alpha_{11} P$
5	$F(5, P, -) = \alpha_5 P = -\tau P - P$	$F(5, P, -) = \alpha_5 P = \tau P - P$
13	$F(13, P, \alpha_5 P) = \alpha_{13} P = P - \tau^2 \alpha_5 P$	$F(13, P, \alpha_5 P) = \alpha_{13} P = P - \tau^2 \alpha_5 P$
7	$F(7, P, -) = \alpha_7 P = -\tau P + P$	$F(7, P, -) = \alpha_7 P = \tau P + P$
9	$F(9, P, \alpha_7 P) = \alpha_9 P = -\tau^4 P - \alpha_7 P$	$F(9, P, \alpha_7 P) = \alpha_9 P = -\tau^4 P - \alpha_7 P$

[a] The symbol "−" refers to any point, as the third input parameter is not used in this case.

Algorithm 2. Short memory scalar multiplication on a normal basis

INPUT: Base point P, scalar d;
OUTPUT: $Q = dP$;

1. compute $d^{(5)} = \text{TNAF}_5(d)$; $Q \leftarrow \mathcal{O}$; $R \leftarrow \mathcal{O}$;
2. **for** u following the sequence $\{1, 3, 11, 15, 5, 13, 7, 9\}$ **do**
   (a) $R \leftarrow F(u, P, R)$ with Table 2; $k \leftarrow 0$;
   (b) **for** j **from** 0 **to** $m + a - 1$ **do**
      i. **if** $|d_j^{(5)}| = u$ **then**
         A. $R \leftarrow \tau^{j-k} R$; $k \leftarrow j$;
         B. $Q \leftarrow Q + \text{sign}(d_j^{(5)}) R$;
   (c) **if** $u \in \{3, 11, 5, 7\}$ **then** $R \leftarrow \tau^{m-k} R$;
3. **return** Q;

The average computational cost of Algorithm 2 is:

$$\mathcal{C}_{1,n}^{(5)} = \frac{m+a}{6} \cdot \text{ECADD}_n + \left(\frac{m+a}{6} + 4 \right) \cdot \text{ECFRB}_n + \mathcal{C}_{0,n}^{(5)}. \tag{4}$$

Where the index 1 in $\mathcal{C}_1$ stands for the total computational cost, whereas 0 in $\mathcal{C}_0$ stands for the cost of precomputations. It is exactly the same as that of the standard TNAF_5, with 4 additional τ-multiplications arising from step 2(c), which re-sets the current precomputed point to its original value $\alpha_u P$ in order to evaluate the next precomputed point. Interestingly, Algorithm 2 computes the scalar multiplication using the (optimal) TNAF_5 with only one auxiliary point, namely the current precomputed point R. In comparison, the standard TNAF_5 requires 7 non-trivial (that is, different from P) precomputed points. Thus, for a completely negligible overhead (4 ECFRB), we reduced the memory

consumption[2] by a factor 7. Or, putting it in a different way, for the same memory consumption as that of the TNAF_3, our method is as fast as the TNAF_5. Note that since sequential precomputations are trivially possible for $w = 2, 3, 4$, our technique is also applicable to the TNAF, TNAF_2, TNAF_3 and TNAF_4.

4 Short Memory with Change-of-Basis

In the following we show how a mixed normal-polynomial basis approach allows us to deploy the short memory method not only on hardware implementations but also on general-purpose processors.

4.1 General Idea

The interest of the latter idea is practically limited by the fact that a normal basis is necessary in order to efficiently compute multiplications by τ^i. In software, normal basis implementations are much slower than polynomial basis implementations. At this point, it would be interesting to take the best from the two approaches: fast computations of Frobenius map with a normal basis and fast field multiplications with a polynomial basis, and convert between the two representations when necessary. This mixed approach was already used for implementing a fast generator of pairs (k, kP) for signature schemes [4], and for defeating side channel attacks [16], but never to speed-up the scalar multiplication itself.

The general idea of our technique is to perform sequential precomputations with a polynomial basis, but then, convert the auxiliary precomputed point R_p to its normal basis representation R_n. After that, $\tau^i R_n$ can be computed with only two cyclic shifts, for any i. The point R_n with shifted coordinates can be converted back to its polynomial basis representation in order to perform point additions with the polynomial basis.

We remark an interesting property of this approach: to convert a point R_n represented with respect to the normal basis to its polynomial basis representation, one only needs the binary expansion of the coordinates of $R_n = (x_n, y_n)$. More precisely, the conversion to the polynomial basis is as follows: if the kth bit of x_n is 1, then add (i.e. xor) the kth line of the change-of-basis matrix to x_p. Of course, the same holds for y_n and y_p. As a consequence, it is not needed to explicitly compute $\tau^i R_n$: the knowledge original (unshifted) binary expansion of x_n and y_n is sufficient to convert $\tau^i R_n$ to its polynomial basis representation. Indeed, if the kth bit of x_n is 1, we simply add the $(k + i \bmod m)$ line of the change-of-basis matrix to x_p, and the same holds for y_n and y_p.

4.2 Short Memory TNAF_5 with Change-of-Basis

Unlike the normal basis short memory method, we use two auxiliary points instead of just one, in order to avoid additional conversions to the normal basis.

[2] By memory consumption, we refer to non-trivial precomputed points: we do not consider buffers, nor the base point or accumulators.

More precisely, the auxiliary precomputed point is stored in its polynomial and normal basis representations. Then, the average computational cost of the scalar multiplication using Algorithm 3 is:

$$\mathcal{C}_{1,cob}^{(5)} = \frac{m+a}{6} \cdot \mathrm{ECADD}_p + \left(\frac{m+a}{6} + 12\right) \cdot \mathrm{COB} + \mathcal{C}_{0,p}^{(5)}, \tag{5}$$

where COB stands for the computational cost for changing the basis. Note that the auxiliary precomputed point R_p is represented in affine coordinates, and thus, it is possible to use mixed projective-affine additions formulas [6].

Algorithm 3. Short memory scalar multiplication with change of basis, $w = 5$

INPUT: Base point P, scalar d;
OUTPUT: $Q = dP$;

1. compute $d^{(5)} = \mathrm{TNAF}_5(d)$; $Q \leftarrow \mathcal{O}$;
2. **for** u following the sequence $\{1, 3, 11, 15, 5, 13, 7, 9\}$ **do**
   (a) $R_p \leftarrow (u \bmod \tau^5)P$ with affine coordinates, polynomial basis;
   (b) $R_n \leftarrow$ convert R_p to normal basis;
   (c) **for** j **from** 0 **to** $m + a - 1$ **do**
      i. **if** $|d_j^{(5)}| = u \bmod \tau^5$ **then**
         A. $R_p \leftarrow$ convert $\tau^j R_n$ to polynomial basis;
         B. $Q \leftarrow Q + \mathrm{sign}(d_j^{(5)})R_p$ with polynomial basis, mixed coordinates;
   (d) **if** $u \in \{3, 11, 5, 7\}$ **then** $R_p \leftarrow$ convert R_n to polynomial basis;
3. **return** Q;

On the one hand, in the original TNAF_5 scalar multiplication computed with a polynomial basis, the Frobenius map must be explicitly computed, requiring $3 \cdot (m + a)$ squarings assuming projective coordinates. On the other hand, the short memory method does not compute the Frobenius map explicitly, but instead, several conversions to/from the polynomial basis and the normal basis are needed: one each time the auxiliary precomputed point is updated (i.e. 8), one before each point addition (i.e. $(m + a)/6$) and one when the new value of the auxiliary point requires the previous auxiliary point (i.e. 4). Since squarings and change-of-basis operations have a small computational cost compared to point additions, we can expect the original TNAF_5 and our method to have similar running times. In fact, depending on the relative speed of squarings and change-of-basis operations, the short memory method might be slightly slower or faster than the TNAF_5.

5 Comparisons and Properties

Finally, we discuss side channel attacks, generalize the computation strategy described in Algorithms 2 and 3, consider the two cases of hardware and software implementations, and show how the method compares with known techniques.

5.1 Discussion on Side Channel Attacks

On low-end processors running cryptography, side channel attacks are a serious threat. By re-grouping calculations involving the same precomputed point, the short memory method is naturally weaker than other methods, leaking even more information.

However, there are many situations where side channel analysis is not an issue. Consider EC-DSA, for instance. On the one hand, the signature generation should be protected against side channel attacks, as it involves secret parameters. Additionally, the signature generation is based on one scalar multiplication with known base point, which can be computed very efficiently thanks to comb methods [15]. On the other hand, the signature verification involves computations with random points, which are much less efficient, and only publicly available parameters. This is the ideal setting for the (unprotected) short memory method. Note that the short memory method does not combine well with interleave methods, but on Koblitz curves, the benefit obtained from interleave methods is small anyway, because point doublings are already replaced by the efficiently computable Frobenius automorphism.

In some situations, the base point is random and the scalar should be kept secret. Then, one can still benefit from the advantages of the short memory method and in the same, protect the scalar multiplication against side channel attacks. There are two types of side channel attacks based on power consumption analysis: simple power analysis (SPA) and differential power analysis (DPA). In the frame of SPA, the attack rely on one single power trace, whereas several power traces are analyzed with the help of a statistical tool in the case of DPA. To thwart SPA, one can use side channel atomicity, a technique virtually applicable to any algorithm [3]. The basic idea of side channel atomicity is to assemble atomic blocks which are indistinguishable by SPA, and implement every operation with atomic blocks. DPA can also be defeated with adequate countermeasures: the random exponent recoding technique applied to Koblitz curves [9] combines well with our method.

5.2 Recoding and Calculation Strategies

For the sake of simplicity, we introduced our technique with a repeated right-to-left scanning of the TNAF_w in Algorithms 2 and 3. However, since τ multiplications are computed with respect to a normal basis, the *order* of computations does not matter: one can compute $\tau^i P$ or $\tau^{-i} P$ with only two cyclic shifts, and the cost of the latter operation is independent from the cycle length i. In other words, instead of computing right-to-left, one can compute left-to-right, or even following a random re-ordering of the operations! Note that such alternative computation strategies have no influence on efficiency: the auxiliary point R in Algorithm 2, or R_p in Algorithm 3, is always represented with affine coordinates, whereas the accumulator Q can be represented with projective coordinates such as LD-coordinates. Therefore, one can always use mixed affine-projective formulas, independently from the computation strategy. In this sense, our technique

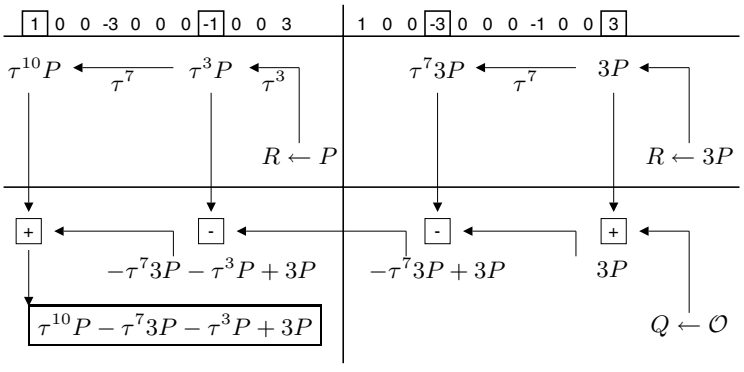


Fig. 1. Right-to-Left Computation Strategy

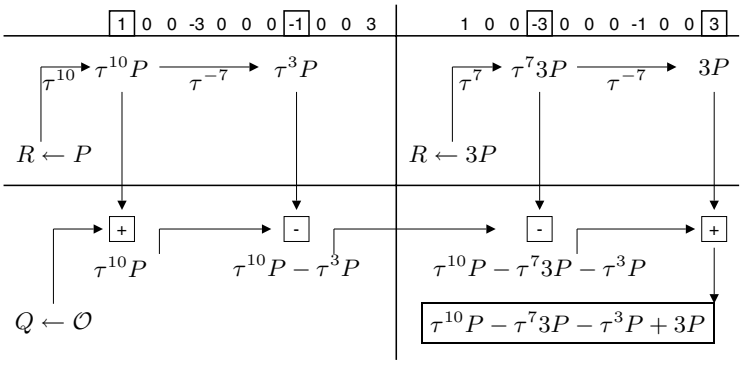


Fig. 2. Left-to-Right Computation Strategy

differs from standard methods were a left-to-right computation strategy is necessary when mixed addition formulas are used.

On-the-Fly Right-to-Left Strategy. In the standard approach, the scalar is converted to the TNAF_w using a right-to-left strategy whereas the scalar multiplication is computed left-to-right. Thus, the scalar must be stored in both of its original and TNAF_w representations, wasting $O(m)$ bits. With the short memory method, we are free from the left-to-right constraint for computations, and as a consequence, it becomes possible to recode the scalar on-the-fly, without effectively storing the coefficients of the TNAF_w. The idea is as follows: each time the auxiliary precomputed point is updated, the recoding is performed anew. For each individual recoding, additions (or subtractions) with the auxiliary point $(u \bmod \tau^w)P$ are performed only when the corresponding coefficient of the TNAF_w is $\pm(u \bmod \tau^w)$. Of course, one has to repeat the same recoding several times, but the computational cost of the recoding process is generally negligible compared to that of the scalar multiplication itself, and in the case of

the TNAF_5, the recoding must be calculated only 8 times, limiting the impact of the redundancy overhead.

Randomized Computation Sequence. For a given precomputed auxiliary point $(u \bmod \tau^w)P$, the order of computations can be freely chosen. In fact, it can even be randomized. For $w = 5$, there are some constraints on the precomputed auxiliary points themselves: $(3 \bmod \tau^5)P$ must be calculated before $(11 \bmod \tau^5)P$, for instance. But for $w < 5$, one can compute the points $(u \bmod \tau^w)P$ following any order, and then for fixed u, randomize the computation sequence again. Re-ordering operations can be used in order to prevent side-channel or fault attacks. However, this idea is based on temporal obfuscation: the order of the operations is randomized, but not the operations themselves. Therefore, alone, this technique might be insufficient to defeat side channel attacks, but combined with other countermeasures, it provides higher security for free.

5.3 Hardware Implementation

In hardware, the cylic shift is virtually free: for example, it can be emulated with a pointer on the least significant bit. Additionally, there are efficient normal basis multipliers for hardware implementations [14], and even when a polynomial basis representation is required for interoperability with other systems, the conversion from a normal basis to a polynomial basis representation can be carried out without storing the change-of-basis matrix [10]. Thus, the natural choice is a normal basis, where our method offers outstanding speed-ups with very small memory requirements. Alternatively, by combining Frobenius expansions and the point halving method, one can obtain a non-zero density of 2/7 with no precomputations [1]. Table 3 summarizes the computational costs and memory requirements for precomputations of our technique, which has one auxiliary precomputed point, the TNAF_2 and the τ/halve method, which have no precomputed tables, and with the TNAF_3 and the TNAF_5, which have one and seven precomputed points, respectively. Clearly, the short memory method outperforms the TNAF_2, TNAF_3 and the τ/halve techniques. The TNAF_5 is as fast as the short memory, but requires seven points whereas our method needs only one auxiliary point for precomputations, reducing the memory consumption for precomputations by 85%.

5.4 Software Implementation

A polynomial basis implementation is usually preferred in software, because field multiplications are much faster than when using a normal basis. However, the two approaches can be combined by using a normal basis for computing the Frobenius map and a polynomial basis for computing point additions.

In Table 4, we estimate the computational cost of the short memory method with change-of-basis and compare it with standard TNAF_w techniques, assuming mixed affine-LD projective coordinates for the scalar multiplication and affine coordinates for precomputations, and that the cost of field inversions, field squarings and change-of-basis operation is equivalent to that of 10, 1/7 and 1 field

Table 3. Speed and Memory Comparisons for Hardware Implementations

	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_{2^{233}}$	$\mathbb{F}_{2^{283}}$	$\mathbb{F}_{2^{409}}$	$\mathbb{F}_{2^{571}}$
TNAF_2	54 add.	78 add.	94 add.	136 add.	190 add.
τ/halve	47 add.	67 add.	81 add.	117 add.	163 add.
TNAF_3	42 add. 326 bits	59 add. 466 bits	72 add. 566 bits	103 add. 818 bits	144 add. 1142 bits
TNAF_5	34 add. 2282 bits	46 add. 3262 bits	54 add. 3962 bits	75 add. 5726 bits	102 add. 7994 bits
Short Memory (Algorithm 2)	34 add. 326 bits	46 add. 466 bits	54 add. 566 bits	75 add. 818 bits	102 add. 1142 bits

Table 4. Speed [a] and Memory Comparisons for Software Implementations

	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_{2^{233}}$	$\mathbb{F}_{2^{283}}$	$\mathbb{F}_{2^{409}}$	$\mathbb{F}_{2^{571}}$
TNAF_2	543 M	776 M	942 M	1362 M	1901 M
TNAF_3	437 M 42 bytes	620 M 60 bytes	750 M 72 bytes	1079 M 104 bytes	1502 M 144 bytes
TNAF_4	389 M 126 bytes	541 M 180 bytes	650 M 216 bytes	923 M 312 bytes	1274 M 432 bytes
TNAF_5	387 M 294 bytes	518 M 420 bytes	612 M 504 bytes	847 M 728 bytes	1150 M 1008 bytes
Short Memory (Algorithm 3)	395 M 84 bytes	520 M 120 bytes	609 M 144 bytes	834 M 208 bytes	1123 M 288 bytes

[a] Assumptions for relative costs with multiplication M as reference: squaring $S \approx M/7$, change-of-basis COB $\approx M$, inversion $I \approx 10M$.

multiplications, respectively. Our estimations show that under our assumptions, the short memory method is about as fast as the TNAF_5, with a small advantage for the TNAF_5 for small bitlengths. For example, for the field $\mathbb{F}_{2^{163}}$, the TNAF_5 is the fastest, but the difference with the short memory is only about 2%, and the short memory method requires about 70% less memory. However, for greater bitlengths, the short memory method is slightly faster than the TNAF_5: the overhead introduced by the change-of-basis operations is smaller than the speed-up obtained from saving τ multiplications. Note that the situation may be different in practical implementations, depending on the relative speed of change-of-basis operations and polynomial basis squarings. In particular, an efficient change-of-basis method would definitely give the advantage to the short memory method. The conventional change-of-basis techniques aim at achieving interoperability between full normal basis implementations and full polynomial

basis implementations [7]. In the case of the short memory method, we do not use normal bases for computing field multiplications. In other words, we do not need to use special normal bases such as optimal or Gaussian normal bases. Instead, one might look for normal bases with efficient conversion to polynomial bases.

6 Conclusion

We proposed a novel technique for computing the scalar multiplication on Koblitz curves. Our method keeps the full power of precomputations without effectively storing all of the precomputed values. In the case of hardware implementations, our method is as fast as the optimal TNAF_5, but reduces memory consumption for precomputations by 85%. In the case of software implementations, again, the running time of our method is roughly the same as the TNAF_5, but with 70% less memory for precomputations. Using our technique, one can deploy the optimal table size of the TNAF_w without sacrificing much memory to store it. Therefore, for environments with very scarce resources, in software or hardware, the short memory method offers significant improvements compared to known schemes, using the available memory where it is truly needed.

References

1. Avanzi, R. M., Ciet, M., Sica, F.: Faster scalar multiplication on Koblitz curves combining point halving with the Frobenius endomorphism. In Proc. of PKC'04, LNCS 2947, pp.28–40, 2004.
2. Brickell, E. F., Gordon, D. M., McCurley, K. S., Wilson, D. B.: Fast exponentiation with precomputation: algorithms and lower bounds. In Proc. of Eurocrypt'92, LNCS 658, pp.200–209, 1993.
3. Chevallier-Mames, B., Ciet, M., Joye, M.: Low-cost solutions for preventing simple side-channel analysis: side-channel atomicity. In IEEE Trans. Computers 53(6), pp. 760–768, 2004.
4. Coron, J.-S., M'Raïhi, D., Tymen, C.: Fast generation of pairs $(k, [k]P)$ for Koblitz elliptic curves. In Proc. of SAC'01, LNCS 2259 , pp. 151–164, 2001.
5. Dahab, R., Hankerson, D., Hu, F., Long, M., López, J., Menezes, A.: Software multiplication using normal bases. Technical report CACR 2004-12, University of Waterloo, 2004.
6. Hankerson, D., López, J., Menezes, A.: Software implementation of elliptic curve cryptography over binary fields. In Proc. of CHES'00, LNCS 1965, pp. 1–24, 2001.
7. IEEE P1363A. Standard specifications for public-key cryptography, annex A, number-theoretic background, 2000.
8. Joye, M., Tymen, C.: Compact encoding of non-adjacent forms with applications to elliptic curve cryptography. In Proc. of PKC'01, LNCS 1992, pp. 353–364, 2001.
9. Joye, M., Tymen, C.: Protections against differential analysis for elliptic curve cryptography: An algebraic approach. In Proc. of CHES'01, LNCS 2162, pp. 377–390, 2001.
10. Kaliski, B.S., Yin, Y.L.: Storage-efficient finite field basis conversion. In Proc. of SAC'98, LNCS 1556, pp. 81–93, 1999.

11. Koblitz, N.: Elliptic curve cryptosystems. In Mathematics of Computation, 48(177), pp. 203–209, 1987.
12. Koblitz, N.: CM-curves with good cryptographic properties. In Proc. of Crypto'91, LNCS 576, pp. 279–287, 1992.
13. Miller, V. S.: Use of elliptic curves in cryptography. In Proc. of Crypto'85, LNCS 218, pp. 417–426, 1986.
14. Massey, J., Omura, J. K.: Computational method and apparatus for finite field arithmetic. US Patent 4587627, 1986.
15. Möller, B.: Improved techniques for fast exponentiation. In Proc. of ICISC'02, LNCS 2587, pp 298-312, 2003.
16. Park, D. J., Sim, S. G., Lee, P. J.: Fast scalar multiplication method using change-of-basis matrix to prevent power analysis attacks on Koblitz curves. In Proc. of WISA 2003, LNCS 2908, pp. 474–488, 2003.
17. Solinas, J. A.: An improved algorithm for arithmetic on a family of elliptic curves. In Proc. of Crypto'97, LNCS 1294, pp. 357–371, 1997.
18. Solinas, J. A.: Efficient arithmetic on Koblitz curves. In Designs, Codes, and Cryptography, 19(2–3), pp. 195–249, 2000.

Hardware/Software Co-design for Hyperelliptic Curve Cryptography (HECC) on the 8051 μP

Lejla Batina[2], David Hwang[1], Alireza Hodjat[1],
Bart Preneel[2], and Ingrid Verbauwhede[1,2]

[1] University of California, El. Engineering Dept., Los Angeles, CA 90095
[2] Katholieke Universiteit Leuven, ESAT/COSIC, Kasteelpark Arenberg 10,
B-3001 Leuven-Heverlee, Belgium
{Lejla.Batina, Bart.Preneel, Ingrid.Verbauwhede}@esat.kuleuven.ac.be
{dhwang, ahodjat, ingrid}@ee.ucla.edu

Abstract. Implementing public-key cryptography on platforms with limited resources, such as microprocessors, is a challenging task. Hardware/software co-design is often the only answer to implement the computationally intensive operations with limited memory and power at an acceptable speed. This contribution describes such a solution for Hyperelliptic Curve Cryptography (HECC). The proposed hardware/software co-design of the HECC system was implemented and co-simulated using the GEZEL design environment [3]. As a low-cost platform, we chose an 8-bit 8051 microprocessor to which one small hardware co-processor was added for field multiplication. We show that the Jacobian scalar multiplication can be computed in 2.488 sec at 12 MHz on this platform if a minimal hardware module is added *i.e.* a hardware multiply-add unit. This optimal solution provides a factor of 26 speed-up over a software-only solution.

Keywords: HECC, GF(2^m), genus 2 curves, hardware/software co-design, embedded implementation.

1 Introduction

Public-key cryptosystems are present in almost all spheres of digital communication *e.g.* for financial, governmental and medical applications; they form an essential building block for network security protocols (e.g. SSL/TLS, IPsec, SSH). The best-known and most commonly used public-key cryptosystems are based on factoring (RSA) and on the discrete logarithm problem in GF(p) (Diffie-Hellman, ElGamal, Schnorr, DSA) [18]. They allow secure communications over insecure channels without prior exchange of a secret key and they also enable digital signatures. Elliptic Curve Cryptography (ECC), which was proposed in the mid 1980s by Miller [20] and Koblitz [14], is based on a different algebraic structure. ECC offers shorter certificates, lower power consumption and better performance on some platforms. Besides that, ECC offers more "security per bit" as no sub-exponential algorithm is known that solves the discrete logarithm

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 106–118, 2005.

problem in this group. However, only in the past few years has ECC started replacing some of the RSA applications.

In 1988 Koblitz suggested to use the generalization of Elliptic Curves (EC) for cryptography, the so-called Hyperelliptic Curves (HEC) [15]. While ECC applications are highly developed in practice, the use of HEC is still of pure academic interest. However, one advantage of HECC resides on the fact that the operand size for HECC is at least a factor of two smaller than the one of ECC. More precisely, while typical bit-lengths for ECC are at least 160 bits, for HECC this lower bound is around 80 bits (in the case of genus 2 curves). This fact makes HECC a very good choice for platforms with limited resources.

Almost all existing HECC implementations consider binary fields and curves of genus two or three; this choice is motivated by security reasons [9]. Software implementations were developed on general purpose processors and on embedded microprocessors e.g. on an ARM [21,26] and some research has been performed on a hardware implementation. However, this article describes the first HECC implementation using a hardware/software co-design. More precisely, we have implemented the HECC divisor multiplication on the 8051 microprocessor, which uses a small hardware co-processor to optimize the performance. This is the first step towards exploring all possibilities for hardware/software co-designed HECC implementations. Such an investigation is of special interest as embedded devices are believed to be of vital importance for a broad area of pervasive computing such as sensor networks and wireless applications.

First we examined the pure software *i.e.* C/assembly implementations. Next some small extra hardware was added, which facilitates the field operations, in particular the inversion and multiplication in the binary field. We conclude that even with very limited hardware resources one can obtain an attractive performance. We used formulae of Byramjee and Duquesne [8] to achieve optimized divisor doubling operation. For the optimal hardware/software co-design we used GEZEL as a design environment. GEZEL is especially suitable for the exploration of domain-specific coprocessor and multiprocessor micro architectures as it can provide cycle-true hardware/software co-simulation with various embedded core instruction set simulators.

The remainder of this paper is organized as follows. Section 2 lists some relevant previous work in HECC on embedded platforms. In Sect. 3 some background information on HECC is given. Details of our implementation are specified in Sect. 4 and results are listed in Sect. 5. Some directions for future work and conclusions are given in Sect. 6.

2 Previous Work

Algorithms for HECC and implementations have been studied intensively in the past years. A significant amount of work has been performed on investigating the formulae for the group operation [17,24,22,8]. Explicit formulae for genus 2 curves are given by Lange [17] for arbitrary fields and for various types of coordinates. There exist practical results for both software platforms (general purpose or

embedded processor) [26,21] and hardware devices, such as FPGAs [7,13]. The most detailed and complete reference dealing with software as well as hardware implementations is [24].

For embedded processors, a large amount of work has been performed for the ARM platform [26,23,4,21]. Pelzl *et al.* [21] have implemented the group operation of genus 2 and 3 for HECC on an ARM7 processor. They compared the results with ECC implementation (with corresponding security) and showed that HECC performance is comparable to the one of ECC. The performance for divisor scalar multiplication on the ARM microprocessor for genus 2 was further optimized in [23] and compared to genuses 3 and 4. They proved that genus 3 is the fastest, requiring less than 70 *ms* on an ARM7 running at 80 MHz. The work of Wollinger *et al.* [26] considered not just the ARM7TDMI but also the ColdFire and a PowerPC. In addition, they provided the first thorough comparison of ECC and HECC on those platforms.

The first complete hardware implementation of HECC was given by Boston *et al.* [7]. Wollinger *et al.* [25] investigated HECC implementation on a VLSI coprocessor. They used projective coordinates and completed their research on VLSI platforms started in [6,5]. They compared co-processors using affine and projective coordinates and concluded that the latter should be preferred for hardware implementations. They used a curve of a special form ($y^2 + xy = x^5 + f_1x + f_0$), which allowed for more optimized formulae. In [13] three different architectures on a FPGA have been examined for vast area of applications.

With respect to the platform, we mention here other relevant experiences with curve-based cryptography. Woodbury *et al.* [27] showed that EC point multiplication can be performed on an 8051 microcontroller in less than 2 sec as a pure software solution. However, they used a 134-bit OEF at lower security level. Gura *et al.* [10] compared ECC and RSA on 8-bit CPUs and proved that Public-key Cryptography is viable on small devices.

For hardware/software co-design the only relevant work that we are aware of is the one of Kumar and Paar [16]. They implemented ECC on an 8-bit AVR microcontroller with some extra hardware for field multiplications. They show that a 163-bit point multiplication can be calculated in 0.113 sec with a microcontroller running at 4 MHz. We can compare this to our solution as both implementations are for similar platforms and the fields offer the same level of security.

3 Hyperelliptic Curve Cryptography (HECC)

We now present the mathematical background for hyperelliptic curves including the algorithms for efficient arithmetic in the Jacobian group. More details on the theory of hyperelliptic curves can be found in [19].

3.1 Hyperelliptic Curves

Let $\overline{\mathrm{GF}}(2^m)$ be an algebraic closure of the field $\mathrm{GF}(2^m)$. Here we consider a hyperelliptic curve C of genus $g = 2$ over $\mathrm{GF}(2^m)$, which is given with an equation of the form:

$$C : y^2 + h(x)y = f(x) \quad in \quad \mathrm{GF}(2^m)[x, y], \tag{1}$$

where $h(x) \in \mathrm{GF}(2^m)[x]$ is polynomial of degree at most g ($deg(h) \leq g$) and $f(x)$ is a monic polynomial of degree $2g + 1$ ($deg(f) = 2g + 1$). Also, there are no solutions $(x, y) \in \overline{GF}(2^m) \times \overline{GF}(2^m)$ which simultaneously satisfy the equation (1) and the equations: $2v + h(u) = 0, h'(u)v - f'(u) = 0$. These points are called singular points. For the genus 2, in the general case the following equation is used $y^2 + (h_2x^2 + h_1x + h_0)y = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$. For our implementation we used the so-called type II curves [8], which are defined with $h_2 = 0, h_1 \neq 1$. In particular, the authors recommended to use curves of the form: $y^2 + xy = x^5 + f_3x^3 + x^2 + f_0$, since they combine a simpler arithmetic with a good security level.

A divisor D is a formal sum of points on the hyperelliptic curve C *i.e.* $D = \sum m_P P$ and its degree is $degD = \sum m_P$. Let Div denotes the group of all divisors on C and Div_0 the subgroup of Div of all divisors with degree zero. The Jacobian J of the curve C is defined as quotient group $J = Div_0/P$. Here P is the set of all principal divisors, where a divisor D is called principal if $D = div(f)$, for some element f of the function field of C ($div(f) = \sum_{P \in C} ord_P(f)P$). The discrete logarithm problem in the Jacobian is the basis of security for HECC. In practice, the Mumford representation according to which each divisor is represented as a pair of polynomials $[u, v]$ is usually used. Here, u is monic of degree 2, $degv < degu$ and $u|f - hv - v^2$ (so-called reduced divisors). For implementations of HECC, we need to implement the multiplication of elements of the Jacobian *i.e.* divisors with some scalar.

3.2 Algorithms for HECC

Divisor Multiplication. The divisor scalar multiplication is achieved by use of divisor addition and doubling. We used the NAF algorithm to reduce the number of additions.

Divisor Addition and Doubling. Let the quintuple $[U_1, U_0, V_1, V_0, Z]$ stand for $[x^2 + u_1x + u_0, v_1x + v_0] = [x^2 + \frac{U_1}{Z}x + \frac{U_0}{Z}, \frac{V_1}{Z}x + \frac{V_0}{Z}]$. This form allows us to complete both point operations without inversion. Only one inversion and four multiplication are required at the end to convert back from projective to affine coordinates. We used the formulae from [8] for doubling and we used the same approach to get formulae for addition in the case of mixed coordinates. The addition for type II curve has the same complexity as the one of Lange [17] i.e. it takes $44M$, but doubling has been further optimized to $31M$ (here M denotes number of multiplications/squaring). The formulae for the addition are given in Table 1. The numbers in parenthesis correspond to the case of mixed addition.

Finite Field Arithmetic. We used the polynomial basis representation with the irreducible polynomial being pentanomial in $\mathrm{GF}(2^{83})$. Each element of the field can be represented as an 11-byte word. The field addition of two vectors in hardware or software in $\mathrm{GF}(2^m)$ is simply the xoring of the two vectors. The field

Table 1. Formulae used for the divisor addition

Step	Calculations	# mult.
1	*Precomputation and resultant r:* $Z = Z_1 \cdot Z_2,\ \widetilde{U}_{21} = Z_1 \cdot U_{21}, \widetilde{U}_{20} = Z_1 \cdot U_{20},$ $\widetilde{V}_{21} = Z_1 \cdot V_{21}, \widetilde{V}_{20} = Z_1 \cdot V_{20},$ $t_1 = U_{11} \cdot Z_2 + \widetilde{U}_{21},\ t_2 = U_{10} \cdot Z_2 + \widetilde{U}_{20},$ $t_0 = U_{11} \cdot t_1 + t_2 \cdot Z_1,\ r = t_0 \cdot t_2 + t_1^2 \cdot U_{10}$	$12M(6M)$
2	*Compute almost inverse:* $t_1 = inv_1, t_3 = inv_0$	
3	*Compute almost s:* $t_4 = V_{10} \cdot Z_2 + \widetilde{V}_{20},\ t_5 = V_{11} \cdot Z_2 + \widetilde{V}_{21},$ $w_2 = t_0 \cdot t_4, w_3 = t_1 \cdot t_5;$ $s_1 = (t_0 + Z_1 \cdot t_1) \cdot (t_4 + t_5) + w_2 + w_3 \cdot (Z_1 + U_{11});$ $s_0 = w_2 + U_{10} \cdot w_3$	$8M(7M)$
4	*Precomputations:* $R = Z \cdot r, s_0 = s_0 \cdot Z, s_3 = s_1 \cdot Z,\ \widetilde{R} = R \cdot s_3;$ $S_3 = {s_3}^2, S = s_0 \cdot s_1,\ \widetilde{S} = s_3 \cdot s_1, \widetilde{\widetilde{S}} = s_0 \cdot s_3, \widetilde{\widetilde{R}} = \widetilde{R} \cdot \widetilde{S};$	$9M$
5	*Compute l:* $l_2 = \widetilde{S} \cdot \widetilde{U}_{21}, l_0 = S \cdot \widetilde{U}_{20}, l_1 = (\widetilde{S} + S) \cdot (\widetilde{U}_{21} + \widetilde{U}_{20})$ $+ l_2 + l_0,\ l_2 = l_2 + \widetilde{\widetilde{S}};$	$3M$
6	*Compute U':* $U_0{}' = {s_0}^2 + {s_1}^2 \cdot t_1 \cdot (t_1 + \widetilde{U}_{21}) + t_2 \cdot \widetilde{S} + R \cdot [t_1 \cdot r + s_1 \cdot Z];$ $U_1{}' = \widetilde{S} \cdot t_1 + R^2,\ l_2 = l_2 + U_1{}',$ $t_4 = U_0{}' \cdot l_2 + S_3 \cdot l_0,\ t_5 = U_1{}' \cdot l_2 + S_3 \cdot (U_0{}' + l_1);$ $Z' = \widetilde{R} \cdot S_3, U_1{}' = \widetilde{R} \cdot U_1{}',\ \ U_0{}' = \widetilde{R} \cdot U_0{}';$	$17M$
7	*Compute V':* $V_0{}' = t_4 + \widetilde{\widetilde{R}} \cdot \widetilde{V}_{20};$ $V_1{}' = t_5 + \widetilde{\widetilde{R}} \cdot (\widetilde{V}_{21} + Z);$	$2M$
total		$51M(44M)$

multiplication is the most costly operation in our system, since it is performed thousands of times during the course of a single divisor multiplication. While the inversion algorithm is actually more complex, it is only performed a single time (for the case of projective coordinates) and hence it is not the bottleneck in our initial implementation. We discuss our choices for field multiplication in more detail in Sect. 4.

4 Implementation

4.1 8051 Microprocessor

Here we give a brief overview of the 8051 microprocessor platform. An 8051 is an 8-bit microcontroller originally designed by Intel that consists of several components: a controller and instruction decoder, an ALU, 128 bytes of internal

memory (IRAM), up to 64K of external RAM (XRAM) addressed by a 16-bit DPTR register, and up to 64KB of external program memory or 4KB of internal program memory (ROM). The 8051 also has 128 bytes of special function registers (SFRs), which are used to store system values such as timers, serial port controls, input/output registers, etc. The architecture is shown in Figure 1, which is based on the Dalton 8051 core from UC Riverside [2].

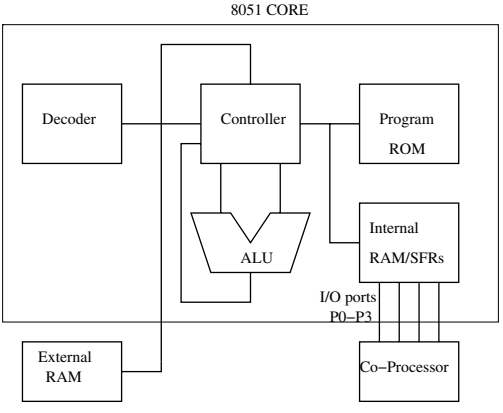


Fig. 1. The architecture of the 8051 microprocessor

An external RAM module (XRAM) can be attached to the 8051 core when the 128 bytes of internal RAM are insufficient, which is often the case in public-key cryptosystems. The 8051 interfaces to the outside world via a serial port as well as four input/output register ports, labeled P0 through P3.

It also should be noted that the 8051 in its original form relies on a clock division principle. That is, the external clock entering into the device is actually divided by 12 to produce the system clock. Thus, a 12-MHz external clock would produce an 8051 with a 1-MHz machine clock cycle, with most instructions requiring 1 or 2 machine cycles. Newer 8051 cores attempt to reduce the clock division [1]. The clock division principle can serve as an advantage to co-designed systems in that the coprocessor circuitry can inherently operate at 12x the internal 8051 machine rate.

4.2 Various Implementation Options

The paper presents two types of HECC implementations on the 8051 processor. The first type is a pure software implementation - either a pure C model operating on the 8051 or a mixed C/assembly model in which most of the functions are performed in C while the GF(2^{83}) finite field multiplier is performed in assembly. The second type is a mixed hardware/software model in which some of the functions are performed in C while the GF(2^{83}) finite field operations (multiplication/addition/inversion) are performed in hardware. The hardware operators

and the 8051 are connected by a memory-mapped interface, over the 8051's P0, P1, and P2 I/O port interfaces.

Software C/ASM Implementation. The first implementation is a pure C implementation, compiled onto the 8051 processor using the Keil suite. This implementation uses a single function in C to combine the multiplication and reduction functions. As a first improvement the multiplication routine is replaced by an assembly code.

Multiplication: In the software implementation, we used a modified form of Algorithm 4 of [11] to implement fast software multiplication. The algorithm is a fast comb-based multiplication method with windows implemented for a 32-bit processor with window size of 4. Based upon initial simulation results, for an 8-bit processor, we found that a window size of 2 provides faster performance.

Reduction: To reduce the multiplication result by the irreducible polynomial, a fast reduction technique was used. This technique was based on Algorithm 6 of [11]. We have used a similar approach but modified the algorithm to implement reduction using our GF(2^{83}) pentanomial and a word-size of 8 bits.

Inversion: The inversion function for this case is implemented as the Extended Euclidean Algorithm.

Hardware/Software Implementation. The second type of HECC implementation is a hardware/software co-design i.e. software routines were enhanced with binary field operations in hardware. In the first attempt we implemented a data path which includes a hardware GF(2^{83}) multiplier. Figure 2 shows this data path. The data IO ports from the 8051 processor are 8-bits long and the multiplication is performed on the GF(2^{83}) operands. There is an instruction register that controls the HW data path from the 8051 processor. The supported instructions for the data path of Figure 2 are as shown in Table 2:

Table 2. Instructions for the data path

Instruction	Definition
`LOADA`	Load 8-bits of data from the 8051 to Register A of HW data path
`LOADB`	Load 8-bits of data from the 8051 to Register B of HW data path
`DOMULT`	Perform GF(2^{83}) mult. on A and B and put the results in C
`GETC`	Return 8-bits of data from Register C of HW data path to the 8051

Due to the fact that the data is transferred back and forth from the CPU to the HW multiplier there is a lot of I/O overhead. In order to optimize the total performance we tried to reduce the I/O transfers with minimum additional memory storage added to the data path. The key observation is that in the schedule of divisor's double and add operations (see Table 1) there are many expressions of the following form: $k_1 = f_3 \cdot t_0 + t_1$.

Initially for such expression, f_3 and t_0 were moved to the hardware multiplier, the multiplication was performed in the hardware, then the result was returned back to the CPU and the addition with t_1 was performed in the SW. In order to speed up this expression the hardware multiplier was replaced with a GF(2^{83}) "multiply-and-add" data path. For this purpose a hardware adder and a feedback line that can keep the result of the multiplication in hardware was added to the original data path and therefore, the number of I/O transfers decreased with not much of extra hardware. For the new datapath (Figure 3), the instructions shown in Table 3 were added.

Table 3. The new instructions for the data path

Instruction	Definition
`MOVE_CTOB`	Move the data in Register C to Register B
`DOADD`	Perform GF(2^{83}) addition on A and B and put the results in C

Moreover, in the software routines that implement the divisor's double and add operations, we moved the coprocessor's instructions up and down in the schedules of the divisor's operations, so that we do not have to repeatedly load the same values into the internal register A of the data path. The performance gain of these optimizations will be provided in the next section.

In addition, for the best performance in the final HW/SW implementation of HECC on the 8051 processor, the GF(2^{83}) inversion operation was performed in HW. The same HW datapath is used to implement the inversion algorithm which consists of repeated multiplications. The details of the hardware GF(2^{83}) multiplication and inversion are given after introducing our design environment.

Design Environment: At this stage we briefly introduce the design environment GEZEL [3] in which we model the co-designed system. In our application, we used the Dalton 8051 ISS to perform cycle-accurate simulations for our software only (C and C/ASM) implementation. For the hardware/software system, we designed our co-processor multiplier using GEZEL's hardware description language. The language syntax is primarily used to describe the FSMD (finite state machine plus datapath) system model. Thus, a datapath for the co-processor was designed and its corresponding control logic was also designed in the GEZEL language.

After the design of the hardware co-processor, we attached the co-processor to the input/output ports of the 8051 ISS (P0-P3) using the GEZEL design environment, and then performed timing and functional verification. GEZEL gave us the ability to co-simulate the 8051 with clock division circuitry as it interfaced with a 12 MHz hardware module in a cycle-exact manner. Upon verification of the functionality of the multiplier co-processor, the GEZEL code was automatically converted to RTL VHDL and input into Synplicity for FPGA synthesis.

Multiplier: In the first version of the multiplier, the multiplier implements a finite field multiplication and simultaneously a corresponding reduction in a

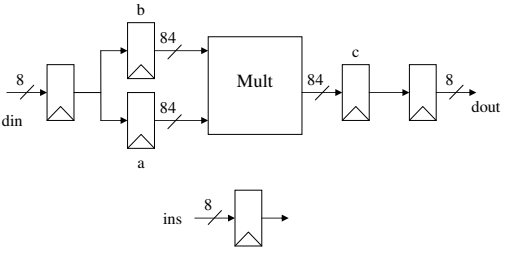


Fig. 2. Data path for the initial design

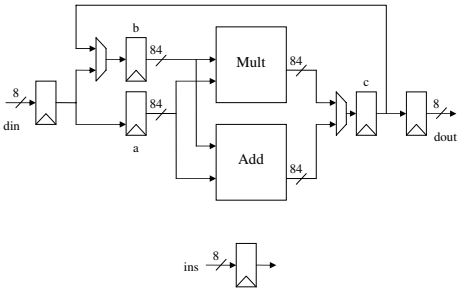


Fig. 3. Data path of the new co-processor

bit-serial implementation. A bit-serial implementation was chosen for area compactness as well as to take advantage of the 12x increase in effective clock rate of the co-processor over the 8051 core. In the second version, the multiplier was enhanced with the additional "multiply-and-add" instruction and datapath element, as described previously.

Inversion: Inversion in binary fields can be replaced by a chain of multiplications (and squarings). It is of interest if squarings are faster than multiplication such as for normal bases. First by means of Fermat's little theorem we have: $a^{-1} = a^{2^m-2} = (a^{2^{m-1}-1})^2$, for all $a \in \mathrm{GF}(2^m)$. The technique to compute this in optimal way is the basis for the idea of Itoh and Tsujii [12]. Their method is especially suited for normal basis but can be applied on polynomial basis as well.

Here we consider the case for m odd, so $m-1$ is even. Then we can write: $a^{2^{m-1}-1} = a^{(2^{\frac{m-1}{2}}-1)(2^{\frac{m-1}{2}}+1)} = (a^{2^{\frac{m-1}{2}}-1})^{2^{\frac{m-1}{2}}} a^{2^{\frac{m-1}{2}}-1}$. In our case for $\mathrm{GF}(2^{83})$ we get: $a^{-1} = a^{2^{83}-2} = (a^{2^{82}-1})^2 = ((a^{2^{41}-1})^{2^{41}} a^{2^{41}-1})^2$, which means that we need to use formula for $a^{2^{m-1}-1}$, but now $m-1$ is odd. In this case: $a^{2^{m-1}-1} = aa^{2^{m-1}-2} = a(a^{2^{m-2}-1})^2$.

By repeated use of these formulae we can compute the inverse by only $90M$. The total number of multiplications (or squarings) required to compute an inverse in $\mathrm{GF}(2^m)$ is given with: $\lfloor log_2(m-1) \rfloor + w(m-1) - 1$. Here $w(k)$ denotes the Hamming weight of some positive integer k.

5 Results

Here we give detailed results on all three platforms and we discuss them further. In Table 4 the timings for all finite field operations are given for hardware and software. Timings for all basic operations are shown and in the last row, the "multiply-and-add" operation is also added. One can notice that inversion in software takes a very long time because it is implemented using the Extended Euclidean Algorithm. The software implementation of inversion by means of Fermat would be already much faster, but we decided to move this operation in hardware anyway. Namely, we concluded that although our software implementations could possibly be further optimized, it would still be difficult to achieve an efficient HECC implementation.

Another observation is that the numbers for addition and multiplication in hardware are the same. The reason for that is because the majority of the time for multiplication and addition on hardware is spent on the IO transfers. Therefore, the time to perform single multiplication (83 cycles) or an addition (1 cycle) is not more than even one 8-bit IO transfer from 8051 to the accelerator. Moreover, this time (2.3 ms) is also very close to the time it takes to do $ab + c$ (2.5 ms), and this is for the same reason as well. However, the fact that this operation is used repeatedly allowed for a speed-up in the new datapath. Sizes of XRAM and ROM are given in bytes (B).

Table 4. Implementation results for operations in GF(2^{83}) for hardware and software routines

Operation	Perf. [♯ Cl. Cyc.]	Perf. [ms]@12MHz	XRAM [B]	ROM [B]
Addition (SW)	38 K	3.2	54	608
Multiplication (SW)	650 K	54.1	122	2065
Inversion (SW)	467.2 M	38.9 K	160	2383
Addition (HW)	28.2 K	2.3	53	934
Multiplication (HW)	28.2 K	2.3	53	934
Inversion (HW)	788.5 K	65.7	75	1835
$ab + c$ **(HW)**	30.5 K	2.5	44	942

The results for the scalar multiplication of divisor for various implementation options are given in Table 5. FPGA area is given in number of LUTs without XRAM and ROM which are specified separately. As can be seen in Table 5, a significant increase in performance is provided by moving the finite-field multiplication from C to assembly, as shown in the first two rows. An additional improvement is made when the multiplication is moved into hardware; however at this point the timing does not improve dramatically because at this point the inversion algorithm (rather than the finite-field multiplication) is the critical path element of the system. Moving the inversion into hardware rapidly reduces the timing (from 52 to 4.1518 seconds). An additional 40% timing reduction occurs after the point operation signal flow graphs are analyzed and manipulated,

Table 5. Implementation results for divisor multiplication in GF(2^{83}) for all three platforms

Implementation	FPGA [♯ LUTs]	XRAM [Bytes]	ROM [Bytes]	Perf.[s] @12MHz
C (Inversion in SW)	3300	820	11754	191.7
C+ASM(Inversion in SW)	3300	820	12284	64.9
C+HW multiplier (Fig. 2-Inversion in SW)	3600	820	11754	52
C+HW multiplier (Fig. 2-Inversion in HW)	3600	927	12789	4.1518
C+HW multiplier (Fig. 3-Inversion in HW)	3781	936	11524	2.4880

and the new "multiply-and-add" operation is created and used. From this table it can also be seen that the number of LUTs does not change whether inversion is performed in hardware or software. This is due to the fact that even if inversion is done in software, the same accelerator is used for field multiplication.

Now we compare our performance results with other work on embedded processors. Table 6 shows that our result features a practical HECC implementation in constrained environments. First, it should be mentioned that it is extremely difficult to compare the performance of cryptographic primitives on different embedded processors, since each processor presents a unique architecture and memory structure. The discussion below is primarily to reference prior art.

The first two references relate to the ARM7, which is a 32-bit platform and features completely different architecture than the 8051. Even so, the second reference is of the same order as ours using frequency scaling for rough normalization. The most suitable comparison to this work is [16] and [10]. Gura *et al.* achieve the shown performance using a "faster" 8051, i.e. an 8051 whose clock division was much less than 12x. They also demonstrate the well-known fact that the AVR is much faster than the 8051 (though exactly how much faster is subject to debate). This provides perspective when comparing to the ECC implementation of Kumar and Paar [16].

Table 6. Implementation results for divisor multiplication on various embedded platforms

Reference	PKC	Field	Platform	Frequency [MHz]	Performance [ms]
[23]	HECC	GF(2^{83})	ARM7	80	71.56
[4]	HECC	GF(2^{80})	ARM7	80	374
[16]	ECC	GF(2^{163})	AVR	4	113
[10]	ECC	GF(2^{160})	8051	12	4580
this work	HECC	GF(2^{83})	8051	12	2488

6 Conclusions and Future Work

This paper shows that even on a small 8-bit processor one can implement hyperelliptic curve cryptography efficiently. We have designed a small hardware module that results in a significant speed-up compared with a software-only solution. We believe that hardware/software co-design offers a new alternative for low-power and low-footprint devices. We plan to explore other trade-offs between hardware and software in order to find the best partition. Additional options can be made available by exploiting parallelism between HECC operations.

References

1. Dallas semiconductor ds89c420 ultra-high-speed microcontroller. `http://www.maxim-ic.com/quick_view2.cfm/qv_pk/2963`.
2. Dalton 8051 processor. `http://www.cs.ucr.edu/~dalton/8051/`.
3. GEZEL design environment. `http://www.ee.ucla.edu/~schaum/gezel`.
4. S. Baktır, J. Pelzl, T. Wollinger, B. Sunar, and C. Paar. Optimal tower fields for hyperelliptic curve cryptosystems. In *Proceedings of 38th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, USA, November 7-10 2004.
5. G. Bertoni, L. Breveglieri, T. Wollinger, and C. Paar. Finding optimum parallel coprocessor design for genus 2 hyperelliptic curve cryptosystems. In *Proceedings of ITCC, April 5-7, 2004*, Las Vegas, Nevada, USA, 2004.
6. G. Bertoni, L. Breveglieri, T. Wollinger, and C. Paar. *Hyperelliptic Curve Cryptosystem: What is the Best Parallel Hardware Architecture?*, chapter in Embedded Cryptographic Hardware: Design and Security. Nova Science, 2004.
7. N. Boston, T. Clancy, Y. Liow, and J. Webster. Genus two hyperelliptic curve coprocessor. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Proceedings of 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 2523 in Lecture Notes in Computer Science, pages 400–414. Springer-Verlag, 2002.
8. B. Byramjee and S. Duquesne. Classification of genus 2 curves over F_{2^n} and optimization of their arithmetic. Cryptology ePrint Archive: Report 2004/107.
9. P. Gaudry. An algorithm for solving the discrete log problem on hyperelliptic curves. In B. Preneel, editor, *Advances in Cryptology: Proceedings of EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 19–34, 2000.
10. N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In M. Joye and J. J. Quisquater, editors, *Proceedings of 6th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, Lecture Notes in Computer Science 3156, pages 119–132, 2004.
11. D. Hankerson, J. L. Hernandez, and A. Menezes. Software implementation of elliptic curve cryptography over binary fields. In Ç. K. Koç and C. Paar, editors, *Proceedings of 2nd International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 1965 in Lecture Notes in Computer Science, pages 1–24. Springer-Verlag, 2000.
12. T. Itoh and S. Tsujii. Effective recursive algorithm for computing multiplicative inverses in GF(2^m). *Electronics Letters*, 24(6):334–335, 1988.

13. H. Kim, T. Wollinger, Y. Choi, K. Chung, and C. Paar. Hyperelliptic curve coprocessors on a FPGA. In *Workshop on Information Security Applications - WISA*, Jeju Island, Korea, August 23-25 2004.
14. N. Koblitz. Elliptic curve cryptosystem. *Math. Comp.*, 48:203–209, 1987.
15. N. Koblitz. A family of Jacobians suitable for Discrete Log Cryptosystems. In S. Goldwasser, editor, *Advances in Cryptology: Proceedings of CRYPTO'88*, number 403 in Lecture Notes in Computer Science, pages 94–99. Springer-Verlag, 1988.
16. S. Kumar and C. Paar. Reconfigurable instruction set extension for enabling ECC on an 8-bit processor. In *Proceedings of International Conference on Field-Programmable Logic and Applications (FPL) 2004*, Antwerp, Belgium, August 30-September 1, 2004.
17. T. Lange. Formulae for arithmetic on genus 2 hyperelliptic curves. *Applicable Algebra in Engineering, Communication and Computing*, 15(5):295–328, February 2005.
18. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
19. A. Menezes, Y.-H. Wu, and R. Zuccherato. *An elementary introduction to hyperelliptic curves*, chapter Appendix, pages 155–178. Springer-Verlag, 1998. N. Koblitz: Algebraic Aspects of Cryptography.
20. V. Miller. Uses of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology: Proceedings of CRYPTO'85*, number 218 in Lecture Notes in Computer Science, pages 417–426. Springer-Verlag, 1985.
21. J. Pelzl, T. Wollinger, J. Guajardo, and C. Paar. Hyperelliptic curve cryptosystems: Closing the performance gap to elliptic curves. In C. Walter, Ç. K. Koç, and C. Paar, editors, *Proceedings of 5th International Workshop on Cryptograpic Hardware and Embedded Systems (CHES)*, number 2779 in Lecture Notes in Computer Science, pages 351–365. Springer-Verlag, 2003.
22. J. Pelzl, T. Wollinger, and C. Paar. High performance arithmetic for hyperelliptic curve cryptosystems of genus two. In *Proceedings of ITCC, April 5-7, 2004*, Las Vegas, Nevada, USA, 2004.
23. J. Pelzl, T. Wollinger, and C. Paar. *Special Hyperelliptic Curve Cryptosystems of Genus Two: Efficient Arithmetic and Fast Implementation*, chapter in Embedded Cryptographic Hardware: Design and Security. Nova Science Publishers, 2004.
24. T. Wollinger. *Software and Hardware Implementation of Hyperelliptic Curve Cryptosystems*. PhD thesis, Ruhr-University Bochum, Germany, 2004.
25. T. Wollinger, G. Bertoni, L. Breveglieri, and Christof Paar. Performance of HECC coprocessors using inversionfree formulae. International Workshop on Information Security & Hiding, Singapore (ISH '05).
26. T. Wollinger, J. Pelzl, V. Wittelsberger, C. Paar, G. Saldamli, and Ç. Koç. Elliptic and hyperelliptic curves on embedded μP. *ACM Transactions on Embedded Computing Systems*, 3(3):509–533, 2004.
27. A. D. Woodbury, D. V. Bailey, and C. Paar. Elliptic curve cryptography on smartcards without coprocessors. In *Proceedings of Fourth Smart Card Research and Advanced Applications (CARDIS 2000) Conference*, 2000.

SHARK: A Realizable Special Hardware Sieving Device for Factoring 1024-Bit Integers

Jens Franke[1], Thorsten Kleinjung[1], Christof Paar[2], Jan Pelzl[2], Christine Priplata[3], and Colin Stahlke[3]

[1] University of Bonn, Department of Mathematics, Beringstraße 1, D-53115 Bonn, Germany
{franke, thor}@math.uni-bonn.de
[2] Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
{cpaar, pelzl}@crypto.rub.de
[3] EDIZONE GmbH, Siegfried–Leopold–Straße 58, D-53225 Bonn, Germany
{priplata, stahlke}@edizone.de

Abstract. Since 1999 specialized hardware architectures for factoring numbers of 1024 bit size with the General Number Field Sieve (GNFS) have attracted a lot of attention ([Ber], [ST]). Concerns about the feasibility of giant monolytic ASIC architectures such as TWIRL have been raised. Therefore, we propose a parallelized lattice sieving device called SHARK, which completes the sieving step of the GNFS for a 1024-bit number in one year. Its architecture is modular and consists of small ASICs connected by a specialized butterfly transport system. We estimate the costs of such a device to be less than US$ 200 million. Because of the modular architecture based on small ASICs, we claim that this device can be built with today's technology.

Keywords: Integer factorization, GNFS, lattice sieving, RSA 1024 bit, special hardware.

1 Introduction

The General Number Field Sieve (GNFS) is asymptotically the best known algorithm to factor numbers with large factors. In practice it seems to be the best algorithm for both software and hardware for factoring 1024-bit numbers, such as they appear in RSA based cryptographic protocols. The GNFS has two expensive parts: the sieving part and the matrix step. This paper describes SHARK, a specialized hardware architecture which completes the sieving step of the GNFS for a 1024-bit number in one year. It is much cheaper than general purpose hardware that solves the same problem (e.g. personal computers). The architecture consists of 2300 identical isolated machines sieving in parallel. In the following we describe one of these machines.

We estimate the costs of one machine to be US$ 70 000. It uses lattice sieving. The actual sieving is done in very fast accessible memory ("cache"). If this

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 119–130, 2005.

memory would be extremely cheap, we could construct a machine that sieves in some extremely large memory chip. Since this kind of memory is expensive we only use 32 MB of sieving cache memory.

The sieving area is split into many small parts such that each part fits in the sieving cache. After the sieving of one small part is completed, the machine moves on to the next part until the whole sieving area has been scanned.

The tricky part is to sort the sieving contributions such that all sieving contributions for a certain part are loaded into the sieving cache just before the sieving of that part starts. To achieve this, the data produced by the lattices corresponding to the larger primes of the factor base are sent through a specialized transport system with butterfly topology.

The output of the sieve consists of potential sieving reports that still need to be checked for smoothness. This is done (after a quick compositeness test) by special hardware devices using the Elliptic Curve Method (ECM). The algorithm has been adapted for hardware implementations (see [FKPPPSS]). The use of ECM in special hardware is preferable for lowering the costs of the machine. However, in this paper we use a choice of parameters with a moderate ECM support in order to focus on the sieving part of the machine. There are better choices with much more ECM, as indicated at the end of Section 3. Notice that the importance of using special hardware for factoring the potential sieving reports grows with the bit length of the number to be factored.

The estimated costs of computing power for factoring 1024-bit numbers have been derived from software experiments. Together with the experience from recent factoring records in software (see [RSA576] and [RSA200]), this leads to a realistic choice of parameters and good estimates for the amount of computing power and storage needed by each part of the machine.

Section 2 summarizes the necessary background on the GNFS and, in particular, on lattice sieving. It also discusses parameter choices. The SHARK architecture is introduced in Section 3 and an overview of the whole machine is given. A detailed description of the hardware modules and a cost estimate is presented in Section 4. We finish with some conclusions and remarks in Section 5.

2 The General Number Field Sieve and Lattice Sieving

In GNFS we are given two homogeneous polynomials $F_i \in \mathbb{Z}[X, Y]$, $i = 1, 2$, satisfying certain conditions. The task of the sieving step is to collect sufficiently many coprime pairs of integers (a, b), $b > 0$, such that both integers $F_i(a, b)$ decompose into prime factors smaller than a given bound L. Such pairs (a, b) are also called relations. The number of relations needed depends on the bound L. Collecting $2\pi(L) \approx \frac{2L}{\log L}$ relations is usually far more than enough. For more details on GNFS see [LL].

The collection of relations is usually done by a combination of a sieving technique and a method for factoring smaller numbers, e.g. ECM or MPQS.

For this purpose we choose two factor bases $\mathcal{F}_i$ each consisting of pairs (p, r), where $p < B_i$ is a prime and r an integer such that p divides $F_i(a, b)$ whenever $p \mid a - br$. The sieving technique identifies pairs (a, b) such that both values $F_i(a, b)$ are divisible by many primes $< B_i$. The cofactors ($F_i(a, b)$ divided by all prime factors $< B_i$) are subsequently handled by a factoring method for small numbers. If both decompose into prime factors $< L$ a relation is found.

Our proposed sieving device will carry out the collection of relations by lattice sieving in the way described in [FK] (see also *The lattice sieve* by J.M. Pollard in [LL]). Let the dimensions of the sieving rectangle be $I \times J$ and let (q, s) be a special q, i.e. q is a prime and s an integer such that q divides $F_1(a, b)$ whenever $q \mid a - bs$. We consider the lattice $\Lambda_{(q,s)} := \{(\alpha, \beta) \in \mathbb{Z}^2 \mid \alpha \equiv \beta s \pmod{q}\}$ associated to (q, s), calculate a reduced basis (a_1, b_1), (a_2, b_2) of $\Lambda_{(q,s)}$ and define the sieving rectangle to consist of the points $i(a_1, b_1) + j(a_2, b_2)$ for $-\frac{I}{2} \leq i < \frac{I}{2}$ and $0 < j \leq J$.

The factor base elements (p, r) with $p \leq I$ have to be adapted to the lattice given by (q, s), yielding $(p, \tilde{r})$. Then we proceed with $(p, \tilde{r})$ the same way as in line sieving. The factor base elements (p, r) with $p > I$ are handled differently. First, we transform the elements to obtain vectors v and w which allow us to quickly identify the points of the intersection of the sieving rectangle and the lattice $\Lambda_{(p,r)}$ corresponding to (p, r). This is done by starting at the point $(0, 0)$ and continuing from there by a sequence of additions of either v or w or $v + w$ by a simple rule as described in [FK]. At each of these locations we have to add a contribution of $\log p$ to the sieving array. We are interested in those points of the sieving array where the sum of all contributions is bigger than some bound.

In GNFS we have to perform two sieves, an algebraic sieve and a rational sieve. Moreover, we perform a trial division sieve which is a modification of [GLM] described in [FK].

For estimating the costs of a factorization of a 1024-bit number we use the following parameters which are based on the polynomial pair of degree 5 and 1 of [ST]. The factor base bounds are $B_1 = 4 \cdot 10^{10}$ on the algebraic side ($1.7 \cdot 10^9$ prime ideals) and $B_2 = 2 \cdot 10^{10}$ on the rational side ($9 \cdot 10^8$ prime ideals). The size of the sieving rectangle is $2^{20} \times 2^{19}$. If a point of the sieving rectangle passes both sieves and both cofactors are at most 2^{125}, we check for smoothness (and aborting as soon as it fails) by quick compositeness tests and ECM. If this is successful and all factors are at most $L = 2^{42}$ we obtain a relation. We will do lattice sieving for all $3.7 \cdot 10^9$ special q in $[4 \cdot 10^{10}, 1.33 \cdot 10^{11}]$ which we estimate to yield $2.7 \cdot 10^{11}$ relations. The last number was obtained by integrating smoothness probabilities over sieving rectangles. In the whole process, about $1.7 \cdot 10^{14}$ numbers are processed by ECM.

If one desires a smaller matrix, more relations are needed. In this case, we propose to do lattice sieving for all $4.4 \cdot 10^9$ special q in $[4 \cdot 10^{10}, 1.5 \cdot 10^{11}]$ which we estimate to yield $3.1 \cdot 10^{11}$ relations. This increases the number of machines needed for the sieving from 2300 to 2800.

As for cost estimates for other sieving devices the costs will be reduced if one spends more effort in finding a good polynomial pair.

3 SHARK – Architectural Overview

The SHARK machine consists of parts I, II, III and a transport system (see Figure 1). The sieving area is split into small parts consisting of 2^{14} lattice points. For the sieving process one byte per point has a sufficient precision to sum up the logarithms of the primes. Therefore, sieving one part is done in 16 kB of fast accessible memory (comparable to the first level cache of a general purpose CPU).

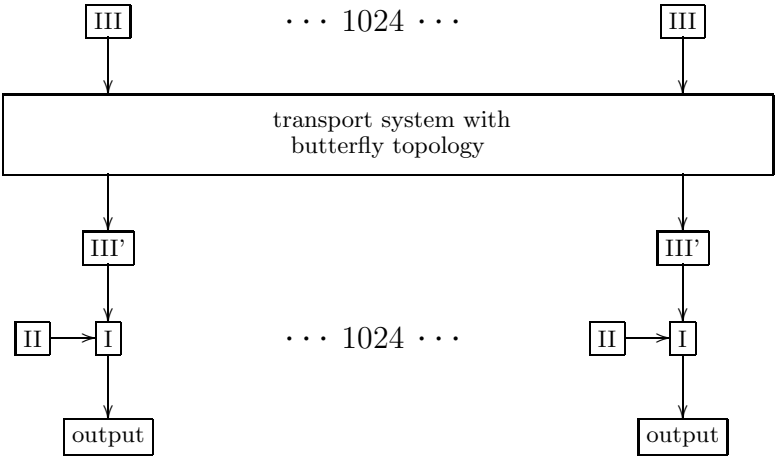


Fig. 1. High-Level Schema of the SHARK Sieving Machine

We split the factor base into small, medium and larger primes which will be dealt with in the three different parts of the machine. Part III of the machine takes care of the larger primes, extracts the necessary data for the sieving process and sends it through a specialized transport system with butterfly topology. The transport system sends the data only to that part of the machine where it is needed. Part III has 1024 small units working in parallel, each dealing with just 1/1024 of the sieving area. Therefore, the transport system has 1024 inputs.

Part II of the machine processes the medium primes. Since the lattices corresponding to these primes are much denser, their data do not need to be sent to all parts of the machine, but can be sorted locally. As visible in Figure 1, part II consists of 1024 small parts, each dealing locally with a small part of the sieving area. These 1024 parts do not communicate among each other.

Part I of the machine consists of 1024 small local units that do not communicate among each other. It generates the very dense lattices for the small primes of the factor base and sieves with these data on 2^{14} lattice points. Additionally, part I collects the sieving data from part II and part III that are necessary for the sieving on the 2^{14} lattice points and sieves with these data. The survivors of this small part of the sieving area are potential sieving reports, and they are

sent as output to an ECM unit to be checked for smoothness. Then, part I turns to the next 2^{14} lattice points.

Within one year, 2300 such machines will output about $1.7 \cdot 10^{14}$ potential sieving reports that need to be tested for smoothness, e.g. with the Elliptic Curve Method (ECM). This could be done by conventional PCs within the required time.

As soon as special hardware for ECM becomes available (see [FKPPPSS]), adapting the parameters of SHARK can save up to 50% of the overall costs, depending on the efficiency of the ECM implementation. E.g. increasing the bound for cofactors from 2^{125} to 2^{163} we only need 1300 machines producing $1.3 \cdot 10^{16}$ potential sieving reports to be processed by ECM.

4 Description of the SHARK Modules

The key to the modular architecture is the partitioning of the sieving area and of the factor base. This algorithmic aspect of the sieving is explained in the first subsection, whereas the three parts of the machine (I, II and III), reflecting the partitioning of the factor base, are described subsequently.

4.1 Sieving

In GNFS we have to perform two sieves: an algebraic sieve and a rational sieve. Notice that we do not need to choose a linear polynomial, the following will also work with two polynomials of degree > 1. These two sieving tasks are almost identical except that for the second sieve we only consider the surviving points of the first sieve. Since we want to know the factorizations of the polynomial values for the surviving points, we also perform a trial division sieve to recover the factors found by the sieves.

We divide a sieving task into three phases: the generation of sieving contributions, the actual sieving, and the evaluation of the sieving area.

The first phase is the generation of triples $(p, \log p, e)$, where p is a prime, $\log p$ the (scaled) logarithm of p, and e a position in the sieving area. If a prime ideal has a contribution to a sieve location, a corresponding triple is produced. In the second phase the contributions are summed up. A sieving array is initialized by zero and for each triple, $\log p$ is added at position e, i.e., for each e the sum

$$\sum_{(p_i, \log p_i, e_i) \text{ with } e_i = e} \log p_i$$

is calculated.

The evaluation phase isolates those sieving locations where the contribution exceeds a certain bound (also depending on the location). For these survivors we can perform a trial division sieve, creating for each survivor a list of its prime divisors, in the following way. We clear the sieving array and fill the positions of the survivors with different identifiers $(1, 2, 3, \ldots)$. Afterwards, for each triple the prime p is stored in the list given by the identifier at position e (if the identifier

is not zero). Note that the generation of triples is only done once while they are used twice: $\log p$ and e for the actual sieving and p and e for the trial division sieve.

We will use lattice sieving which means that we often change the lattice corresponding to a special q. At every change we have to carry out initializations for all elements of the factor base (see [FK]). These initializations amount to roughly one inversion and one half of an extended gcd per factor base element. They are done locally at the places of the machine where the factor base elements are stored. The machine is divided into (roughly) three parts: Part I deals with the small elements (p, r) of the factor base $(1 < p < 2^{14})$, part II with the medium elements and part III processes the large elements $(2^{22} < p)$.

We now describe the general structure of the components of the machine and their interaction over time. Our sieving area has size $2^{20} \times 2^{19}$. Since we omit those pairs for which both coordinates are even we will sieve over three subareas of size $2^{19} \times 2^{18}$. We divide these subareas into 32 parts, each of size $2^{19} \times 2^{13}$. These are called *ranges* and have the following meaning: During a certain period of time all parts of the machine with the exception of part I will prepare data for the algebraic sieve for the n-th range. In the next period of time these parts will do the same for the rational sieve for the n-th range while part I will complete the algebraic sieve for range n using the data prepared in the previous period of time. The rational sieve for range n will be completed in the next period of time by part I while the other parts prepare data for the algebraic sieve for range $n + 1$ etc. Hence there is a need to buffer the prepared data for two sieves over a range.

Each range is divided into 1024 parts of size $2^{19} \times 2^3$ which we will call *domains*. There are also 1024 identical parts of the machine (one for each domain), which will handle sieving contributions of prime ideals of type I and II. The contributions of prime ideals of type III are processed in a different way. These prime ideals are split into 1024 parts and for each part all contributions for a range are prepared and sent to the correct part of the machine. This sorting will be done by a transport system with butterfly topology.

Sieving for a domain is done in 256 steps handling 2^{14} points each. For this purpose, data for prime ideals of type II and III (which have to be stored anyway) are written to the correct array out of 256 arrays. Data for prime ideals of type I are generated on the fly and combined with the data from the corresponding array.

We now describe the individual parts in more detail.

4.2 Part III

This part generates triples for prime ideals of type III. It consists of 1024 identical units each containing 64 MB DRAM and a generation unit. The DRAM is used to hold the factor bases and related information. For each element (p, r) of the factor bases we store an 8-tuple $(p, r, \log p, v_x, v_y, w_x, w_y, e)$ where $\binom{v_x}{v_y} = v$ and $\binom{w_x}{w_y} = w$ are vectors used to update the contribution location and e is the next

contribution location for this prime ideal. For our choice of parameters we can store such an 8-tuple in 25 byte using 36 bit for p and r, 8 bit for $\log p$, 20 bit for v_x, v_y, w_x, w_y and 40 bit for e.

The generation unit has two tasks. After changing a special q it calculates for each prime ideal the values v_x, v_y, w_x and w_y for this lattice and sets e to the first location where the prime ideal contributes. During the sieving phase it reads all 8-tuples one by one, generates the triples for all locations in the sieved domain where this prime ideal has a contribution, and writes the 8-tuple back to memory (actually only e will change). The generated triples are sent to the transport system.

For the initialization task the generation unit has to perform calculations of the complexity of an extended gcd. The actual generation of triples requires only simple instructions such as conditional additions or load/store operations. Accessing the DRAM does not need to be faster than in a conventional PC. The same is true for parts I and II as well.

4.3 Transport System

The transport system has 1024 input channels and 1024 output channels. The purpose of the transport system is to deliver each triple $(p, \log p, e)$ from an input channel to a certain output channel determined by 10 bits of e. Triples have a size of at most 80 bit and may arrive simultaneously at different input channels. We will tolerate a small loss of triples arising from data collisions. For instance, the loss of one triple out of 2^{40} will at most affect one potential sieving report per special q.

We now describe a structure which will comply to the requirements above (see Figure 2). It consists of $11 \cdot 1024$ simple nodes connected in a butterfly topology, i.e., nodes $N_{i,j}$ and $N_{i',j'}$ $(0 \le i, i' < 11,\ 0 \le j, j' < 1024)$ are connected if $i' = i - 1$ and either $j' = j$ or $j' = j$ xor $2^{i'}$. Data always flow from nodes $N_{i,j}$

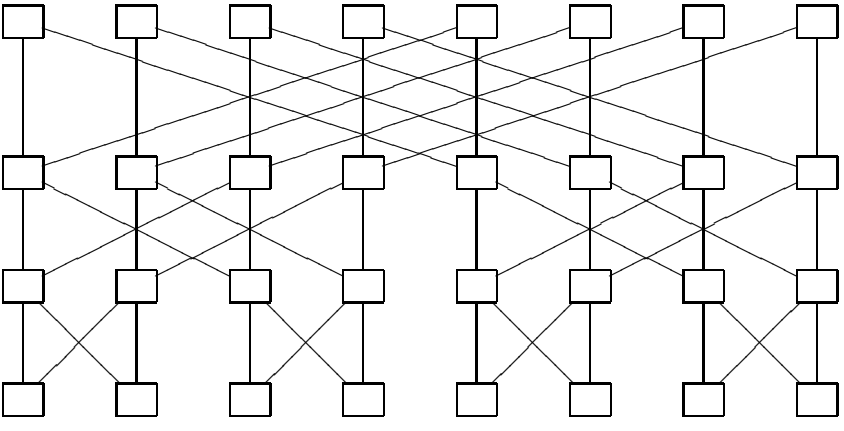


Fig. 2. Butterfly Topology of Width 8

with a higher i to those with a lower i. A typical node consists of two input lines where each is connected to a small buffer such that they can simultaneously receive triples, two output lines and a logic which reads a triple from an input buffer, examines a certain bit of e and delivers this triple to the corresponding output line. For the nodes $N_{i,j}$ in the top layer (i.e. $i = 10$) one of the input lines is an input channel of the transport system and the other input line is not connected. The nodes in the bottom layer (i.e. $i = 0$) send all output to one output line which is the output channel of the transport system.

In order to avoid long cable lengths, the transport system of width 1024 should not be realized in a plane but in a cube (of side length around 1 meter). Each of the 1024 inputs receives 80 bit every 50 ns on average, at peak times every 10 ns. The output rate is more balanced. Using 8 bit wide connections, the clock rate needs to be 1 GHz, or 100 MHz for 80 bit buses. It is also possible to balance the inputs using buffers in part III, such that the needed clock rate can be reduced.

This still might be technically difficult. But for a physical realization it is not necessary to manufacture a separate chip for each node or to strictly adhere to the topology. We might also group several nodes on a chip or implement a different sorting structure as long as the performance is not worse than that of the butterfly topology. Grouping several nodes will also reduce the costs for connecting them. Perhaps the whole transport system could be realized as a mesh which sorts the data.

4.4 Part III'

This part also consists of 1024 identical units each of which handles triples for one domain of the processed range. Each unit connects directly to an output channel of the transport system and receives triples which are to be sorted and stored in a double buffer via a 64 kB cache. The double buffer has a size of $2 \cdot 16$ MB DRAM and each half is used to store triples from prime ideals of type III generated for one domain. They are stored in one of 256 arrays according to 8 bit of e. Since at this stage 18 bit of e are fixed we can omit them and store a triple in 7 byte. Therefore it is possible to store 9300 triples per array which is far more than the expected 7700 triples per array on average. The two halves of the double buffer are written alternately by this part. While one half is written, the other half is read by part I (see below) at a rate of not more than 4 GBit per second.

4.5 Part II

Part II again consists of 1024 identical units. Each unit is responsible for the generation of triples for prime ideals of type II for one domain of the processed range. Since it will generate triples in a line sieve like fashion, it is essentially a simplified version of part III and part III'. The output rate is not more than 4 GBit per second.

This unit consists of 8 MB DRAM, a generation unit, a sorter and a 64 kB-cached double buffer of size $2 \cdot 12$ MB DRAM. The 0.6 million factor base elements

of type II (size between 2^{14} and 2^{22}) can be stored in 8 MB, each using 14 byte. These 14 byte take into account some auxiliary data needed for line sieving and the change from one domain of a range to the corresponding domain in the next range. The generation unit has a slightly easier initialization task than that of part III but the actual generation tasks are comparable. It sends the generated triples to a sorting unit which stores them via a 64 kB cache in one half of the double buffer. For prime ideals of type II 5 byte per triple are sufficient such that each array can hold 9800 triples which is more than the 7400 needed on average.

4.6 Part I

This part again appears in 1024 identical units. Each unit has more complex tasks than the units in the other parts of the machine. It generates triples for prime ideals of type I, adds up these contributions, adds up the contributions from prime ideals of type II and III generated by the other parts of the machine, combines these sums and evaluates them. This process is now described in more detail.

In this part the sieving for a domain will be done in $2 \cdot 256$ steps: first, 256 algebraic sieves, each over an area of 2^{14}, and then 256 rational sieves over the same areas. Each of these sieving steps consists of several phases: first, an initialization of the sieving caches, then the actual summation of the contributions, then an evaluation, and finally the trial divison sieve. The speed needed to access the sieving caches is the same as for conventional processors accessing their first level cache.

The prime ideals of type I together with auxiliary data are stored in less than 50 kB DRAM. A generation unit comparable to that of part II accesses this memory and generates triples for a sieving area of size 2^{14}. These triples are directly sent to a sieving unit which performs the actual sieving in a cache of 2^{14} byte. Since there is no buffering of the triples they have to be generated a second time during the trial division sieve. The initialization of the cache with zeros is also done by the sieving unit.

At the same time another sieving unit which also controls a cache of 2^{14} byte reads the triples generated by parts II and III of the machine and does the actual sieving in this cache. Since parts II/III and part I are processing on different sieving sides (i.e., algebraic/rational) there will be no conflict in accessing the double buffers. Reading the triples will also be fast since triples for an area of size 2^{14} are stored in one array.

Apart from those units described so far there is a more complex evaluation processor which has 8 MB DRAM. It is also connected to the two sieving units and to their caches (see Figure 3). During the actual sieving phase it computes thresholds for the evaluation phase. After all triples have been processed by the sieving units, the processor evaluates the sieving area by adding up corresponding bytes of the two sieving caches and comparing the result to a previously computed threshold. Whenever the sum surpasses the threshold the position is marked in both sieving caches, otherwise it is set to zero (in a sieve on the ratio-

nal side we also set to zero a position which has not survived the corresponding algebraic sieve). When this has been done for the whole area of size 2^{14}, the trial division phase begins. The sieving units read (resp. receive) again triples and send those triples which correspond to a marked position in the sieving area to the evaluation processor which stores them in its DRAM. After a trial division sieve on the rational side has been finished, the evaluation processor outputs the survivors and all data obtained from the trial division sieves for this sieving area.

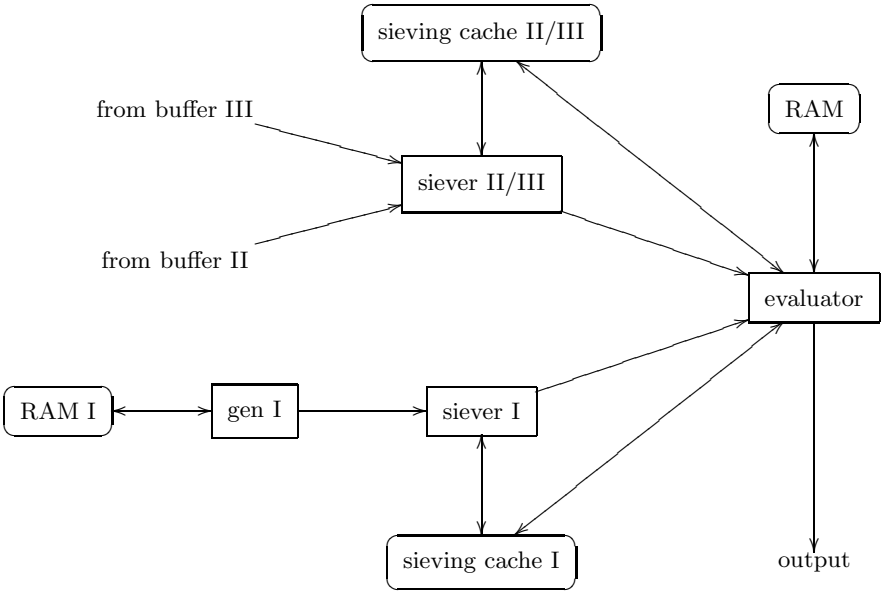


Fig. 3. Block Diagram of Part I

4.7 Cost Estimates

The width of the transport system is crucial for the costs of the whole machine. We first give a simple analysis of the behaviour of costs (money $\times$ time) for varying widths. The total costs consist of the costs for the transport system, the costs for memory and the costs for the ASICs outside the transport system. The third summand remains constant since doubling the width of the transport system will double the number of these ASICs but also halve the time spent for one special q. Furthermore the total memory of the machine remains constant. This has the consequence that doubling the width of the transport system will decrease the costs as long as standard memory chips of smaller size get cheaper. Notice that we want to use standard memory chips, because we assume these to be cheaper than customized memory ASICs. The first summand always grows when doubling the width, since a transport system of width 2^{n+1} consists of two systems of width 2^n and its top layer together with all connections of the top

layer. There will be a certain width for which minimal costs will be attained. In our design this will probably be bigger than 1024 but this is technically more demanding. Therefore we chose a width of 1024.

Apart from a few PCs for controlling the sieving process and collecting the output, one machine consists of 136 GB DRAM, 160 MB cache and various ASICs. Most of the ASICs only perform quite simple tasks. Only the evaluator needs a considerable area (around 20 mm^2). We estimate that all ASICs of one of the 2300 machines occupy a third of the area of a 300 mm wafer. Even taking into account a whole wafer, the silicon costs including memory are less than US$ 30 000. Doubling this number for overhead (packaging, cooling, ...) and adding US$ 10 000 for the PCs and special ECM hardware we obtain US$ 70 000 per machine. Notice that the costs for the ECM hardware for this choice of parameters are just a few dollars and thus negligible.

At a clock frequency of 1 GHz one machine takes around 20 s per special q such that 2300 machines are needed for $3.7 \cdot 10^9$ special q. This amounts to production costs of US$ 160 million (without development). Considering the ASIC area, we estimate that each machine has a power consumption of at most 30 kW which induces a power bill of US$ 60 million per factorization.

5 Conclusions and Remarks

Conclusions. SHARK appears to be the first proposal for an architecture for sieving a 1024-bit number within a year which is realizable with conventional technology and costs less than a thousand million US$. The main difference to other proposed architectures is (in contrast to a giant monolytic ASIC) its modular design composed of small ASICs connected by conventional data buses. The modularity is achieved by dividing the factor base into several parts and sorting the sieving data with a butterfly transport system. All choices of parameters are a result of intense software experiments with a complete implementation of the GNFS for factoring large numbers.

Remarks. Our architecture permits many reasonable modifications: the size of the transport system could be smaller or larger, the partition of the factor base in three parts could vary, ECM could be used more intensely to permit less sieving, many other parameters could be changed. This permits using the architecture also for other bit lengths. 768-bit numbers can be sieved by a similar architecture. While scaling the system for larger numbers, the role of an efficient hardware (like ECM in ASICs, see [FKPPPSS]) to factorize the cofactors becomes more and more important. The transport system has to become very large and at some point the complexity of the connections between the layers will be practically impossible.

Future Work. The efficiency of the machine heavily depends on the different processing of factor base elements of different size. We will analyse different methods for processing very large elements, small prime powers and different classi-

fications of sizes in more than three categories. Some initializations and choices of parameters can still be optimized. A crucial point for the scalability to larger numbers than 1024 bit will be the size of the butterfly transport system. We will investigate different realizations and try to make it larger than 1024 channels. A large butterfly transport system can also be used for solving the matrix in GNFS. We will analyse how to optimize the matrix step in this way and how to lower the size of the butterfly transport system needed for solving the matrix.

References

[Ber] D. J. Bernstein, *Circuits for Integer Factorization: A Proposal*, Manuscript, November 2001. http://cr.yp.to/papers.html#nfscircuit

[FK] J. Franke and T. Kleinjung, *Continued Fractions and Lattice Sieving*, in: Special-Purpose Hardware for Attacking Cryptographic Systems – SHARCS 2005, Paris, 2005. http://www.ruhr-uni-bochum.de/itsc/tanja/SHARCS/talks/FrankeKleinjung.pdf

[FKPPPSS] J. Franke, T. Kleinjung, C. Paar, J. Pelzl, C. Priplata, M. Šimka and C. Stahlke, *An Efficient Hardware Architecture for Factoring Integers with the Elliptic Curve Method*, in: Special-Purpose Hardware for Attacking Cryptographic Systems – SHARCS 2005, Paris, 2005. http://www.ruhr-uni-bochum.de/itsc/tanja/SHARCS/talks/ecm_paper.pdf

[GS] W. Geiselmann and R. Steinwandt, *Yet another sieving device*, CT-RSA 2004, LNCS **2964**, Springer, 2004, 278–291.

[GLM] R. A. Golliver, A. K. Lenstra and K. S. McCurley, *Lattice sieving and trial division*, in: Algorithmic Number Theory (ed. by L. M. Adleman, M.-D. Huang), LNCS **877**, Springer, 1994, 18–27.

[LL] A.K. Lenstra and H.W. Lenstra, Jr. (eds.), *The Development of the Number Field Sieve*, Lecture Notes in Math. **1554**, Springer, 1993.

[LTSKDHL] A. K. Lenstra, E. Tromer, A. Shamir, W. Kortsmit, B. Dodson, J. Hughes and P. Leyland, *Factoring Estimates for a* 1024*-bit RSA Modulus*, in: Proc. ASIACRYPT 2003, LNCS **2894**, Springer, 2003, 55–74.

[RSA576] J. Franke, T. Kleinjung et al., *RSA*-576, Email announcement, 2003. http://www.crypto-world.com/announcements/rsa576.txt

[RSA200] J. Franke, T. Kleinjung et al., *RSA*-200, Email announcement, May 2005. http://www.crypto-world.com/announcements/rsa200.txt

[ST] A. Shamir and E. Tromer, *Factoring Large Numbers with the TWIRL Device*, in: Proc. Crypto 2003, LNCS **2729**, Springer, 2003, 1–26. http://www.wisdom.weizmann.ac.il/~tromer/papers/twirl.ps.gz

Scalable Hardware for Sparse Systems of Linear Equations, with Applications to Integer Factorization

Willi Geiselmann[1], Adi Shamir[2], Rainer Steinwandt[1,3], and Eran Tromer[2]

[1] IAKS, Arbeitsgruppe Systemsicherheit, Prof. Dr. Th. Beth,
Fakultät für Informatik, Universität Karlsruhe, 76131 Karlsruhe, Germany
{geiselma, steinwan}@ira.uka.de
[2] Department of Computer Science and Applied Mathematics,
Weizmann Institute of Science, Rehovot 76100, Israel
{adi.shamir, eran.tromer}@weizmann.ac.il
[3] On leave to Department of Mathematical Sciences,
Florida Atlantic University, Boca Raton, FL 33431-0991, USA

Abstract. Motivated by the goal of factoring large integers using the Number Field Sieve, several special-purpose hardware designs have been recently proposed for solving large sparse systems of linear equations over finite fields using Wiedemann's algorithm. However, in the context of factoring large (1024-bit) integers, these proposals were marginally practical due to the complexity of a wafer-scale design, or alternatively the difficulty of connecting smaller chips by a huge number of extremely fast interconnects.

In this paper we suggest a new special-purpose hardware device for the (block) Wiedemann algorithm, based on a pipelined systolic architecture reminiscent of the TWIRL device. The new architecture offers simpler chip layout and interconnections, improved efficiency, reduced cost, easy testability and greater flexibility in using the same hardware to solve sparse problems of widely varying sizes and densities. Our analysis indicates that standard fab technologies can be used in practice to carry out the linear algebra step of factoring 1024-bit RSA keys.

As part of our design but also of independent interest, we describe a new error-detection scheme adaptable to any implementation of Wiedemann's algorithm. The new scheme can be used to detect computational errors with probability arbitrarily close to 1 and at negligible cost.

Keywords: Factorization, number field sieve, sparse systems of linear equations.

1 Introduction

In recent years, various special-purpose hardware implementations of the Number Field Sieve (NFS) algorithm have been proposed for factoring large (e.g., 1024-bit) integers. These devices address two two critical steps of the NFS: the sieving step [1,2,3,4,5,6,7] and the linear algebra step [8,9,10,11].

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 131–146, 2005.

This work focuses on the linear-algebra step of the NFS. While the cost of this step seems to have been reduced to below that of the sieving step (for 1024-bit composites) by the most recent proposals [10,11], practically these designs are not fully satisfactory: they require (various combinations of) extremely large chips, non-local wiring and high-bandwidth chip interconnects, and thus pose significant technological hurdles.

Below we describe a new systolic design for the NFS linear algebra step, and specifically for the matrix-by-vector multiplications which dominate the cost of the Wiedemann algorithm. This design is both more efficient and more realistic than previous ones. In its simplest form, it consists of a one dimensional chain of identical chips with purely local interconnects, which from a practical standpoint makes it an attractive alternative to previous wafer-scale mesh proposals. For higher efficiency it can be generalized to a two-dimensional array of chips, but unlike previous proposals, this device has standard chip sizes, purely local interconnects, and can use standard DRAM chips for some of its components. In addition, the new design is highly scalable: there is no need to commit to particular problem sizes and densities during the chip design phase, and there is no need to limit the problem size to what can be handled by a single wafer. Since a single chip design of small fixed size can handle a wide range of sparse matrix problems (some of which may be related to partial differential equations rather than cryptography), the new architecture can have additional applications, greatly reduced technological uncertainties, and lower initial NRE cost.

Unlike previous routing based proposals, whose complex data flows required simulation of the whole device and were not provably correct, the present device has a simple and deterministic data flow, so that each unit can be simulated independently. This facilitates the simulation and actual construction of meaningful proof-of-concept sub-devices.

We have evaluated the cost of this device for a specific choice of matrix parameters, which is considered a conservative estimate for the matrix size in factoring 1024-bit integers using NFS. The estimated area×time cost is 6.5 lower than the best previous proposal; the concrete cost estimate is 0.4M US$×year (i.e., excluding non-recurring R&D costs, US$ 0.4M buys enough hardware to obtain a throughput of one solved linear algebra instance per year).

The present design adapts efficiently and naturally to operations over any finite field GF(q), since it does not depend on the in-transit pairwise cancellation of values in GF(2). In particular, it can support the new algorithm of Frey [12,13]. In fact, it can be used with minor modifications over any ground field, such as the rationals or complex numbers.

Section 2 recalls basic facts about Wiedemann's algorithm and its context in the NFS. Section 3 describes the new hardware architecture. In any large-scale computation the handling of faults is crucial; Section 4 presents a particularly efficient error detection scheme, which can also be adapted to other implementations of block Wiedemann. Section 5 gives a preliminary cost analysis for parameters currently considered as plausible for 1024-bit numbers, and compares it to previous proposals.

2 Preliminaries

For an introduction to the NFS algorithm we refer to [14], and for a detailed account to [15]. Here it is sufficient to keep in mind that the overall running time of the NFS algorithm is dominated by the *sieving step* and the *linear algebra step*. In this paper we exclusively consider the linear algebra step, defined as follows. We are given a $D \times D$ matrix A over GF(2), whose columns correspond to *relations* found in the preceding sieving step (after some pre-processing). Our goal is to find a few vectors in the kernel of A, i.e., several sets of relations that sum to the zero vector. This matrix is large but sparse, with a highly non-uniform distribution of row densities. As in previously proposed devices [8,9,10,11], we employ the block Wiedemann algorithm [16,17] for solving sparse systems of linear equations. Basically, the block Wiedemann algorithm reduces the above to the problem of computing sequences of the form

$$Av, A^2v, \ldots, A^tv \tag{1}$$

for some $v \in \mathrm{GF}(2)^D$. Such a sequence can be computed by means of t matrix-by-vector multiplications, where the matrix A remains fixed and the vector varies. Overall, roughly $2D$ such multiplications are needed, divided into $2K$ chains, where $K > 32$ is the *blocking factor*. The resulting products are not explicitly output after each multiplication; depending on the phase of Wiedemann's algorithm, only their inner product with some fixed vectors or their (partial) sums are needed.

Parameters for 1024-bit Composites. At present there is considerable uncertainty about the size and density of the matrix one would encounter in the factorization of a 1024-bit composite, for several reasons: freedom in the choice of the NFS parameters, freedom in the application of pre-processing to the matrix (e.g., to cancel out "large primes"), and lack of complete analysis of this aspect of the NFS algorithm. For concreteness and ease of comparison, in the following we shall assume the "large matrix" parameters from [9], namely a size of $D \times D$ for $D \approx 10^{10}$ and density of 100 entries per column. This leaves a generous conservative margin compared to the smaller matrix expected to be produced by TWIRL [4].

For the sake of concreteness, we propose a concrete instance of our architecture where various design parameters are chosen suitable for the above NFS parameters. In the following, these concrete parameters are designated by angular brackets (e.g., $D \langle\!\langle = 10^{10} \rangle\!\rangle$). Section 5 provides additional details and discusses the cost of the device for these parameters.

3 The New Architecture

We shall unravel the architecture in several stages, where each stage generalizes the former and (when appropriately parameterized) improves its efficiency.

3.1 Basic Scheme

The proposed hardware device is preloaded with a compressed representation of the sparse matrix $A \in \mathrm{GF}(2)^{D \times D}$, as will be detailed below. For each multiplication chain, we load the input vector v and iteratively operate the device to compute the vectors $Av, A^2v, \ldots, A^tv$ and output the appropriate sums or inner products.

We begin by describing an inefficient and highly simplified version of the device, to illustrate its high-level data flow.[1] This simplified device consists of $D \langle\!\langle = 10^{10} \rangle\!\rangle$ *stations* connected in a pipeline. The i-th station is in charge of the i-th matrix row, and contains a compressed representation of the $\langle\!\langle \approx 100 \rangle\!\rangle$ non-zero entries in that row. It is also in charge of the i-th entry of the output vector, and contains a corresponding accumulator $W'[i]$.

In each multiplication, the input vector $v \in \mathrm{GF}(2)^D$ is fed into the top of the pipeline, and moves down as in a shift register. As the entries of v pass by, the i-th station looks at all vector entries v_j passing through it, identifies the ones corresponding to the non-zero matrix entries $A_{i,j}$ in row i, and for those entries adds $A_{i,j} \cdot v_j$ to its accumulator $W'[i]$. Once the input vector has passed all stations in the pipeline, the accumulators $W'[\cdot]$ contain the entries of the product vector Av. These can now be off-loaded and fed back to the top of the pipeline in order to compute the next multiplication.

The one-dimensional chain of stations can be split across several chips: each chip contains one or more complete stations, and the connections between stations may span chip boundary. Note that since communication is unidirectional, inter-chip I/O latency is not a concern (though we do need sufficient bandwidth; the amount of bandwidth needed will increase in the variants given below, and is taken into account in the cost analysis of Section 5).

3.2 Compressed Row Handling

Since the matrix A is extremely sparse, it is wasteful to dedicate a complete station for handling each row of A, as it will be idle most of the time. Thus, we partition A into $u \langle\!\langle = 9600 \rangle\!\rangle$ horizontal stripes and assign each such stripe to a single station (see Figure 1). The number of rows per station is $\mu \approx D/u \langle\!\langle = 2^{20} \rangle\!\rangle$, and each station contains μ accumulators $W'[i]$ with i ranging over the set of row indices handled by the station.

Each station stores all the non-zero matrix entries in its stripe, and contains an accumulator for each row in the stripe. As before, the input vector v passes through all stations, but now there are just u of these (rather than D). Since the entries of v arrive one by one, each station implicitly handles a $\mu \times D$ submatrix of A at each clock cycle.

[1] This basic version is analogous to the electronic pipeline-of-adders version of TWINKLE [2], and many of the improvements described in the following have corresponding analogues in the TWIRL architecture [4].

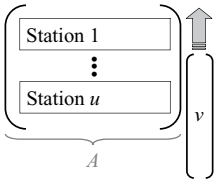


Fig. 1. Distributing the entries of A onto stations

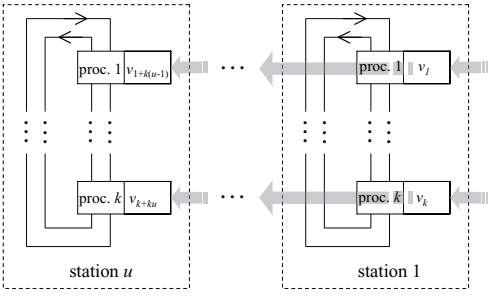


Fig. 2. Subdivision of a chip into stations and processors

3.3 Compressed Vector Transmission

For additional efficiency, we add parallelism to the vector transmission. Instead of each station processing a single entry of v in each clock-cycle, we process v in chunks of $k \langle\!\langle = 32 \rangle\!\rangle$ consecutive entries.[2] The inter-station pipeline is thickened by a factor of k. The vector v now passes in chunks of k entries over an inter-station pipeline (in Figure 2 from right to left); in each clock cycle, each station obtains such a chunk from the previous station (to its right), processes it and passes it to the next station (to its left). The first (rightmost) station gets a new part of the vector received from the outside. At each clock cycle, each station now implicitly handles a $\mu \times k$ submatrix of A.

Each station is comprised of k processors, each connected to a separate pipeline line (see Figure 2), and these k processors inside each station are connected via $\gamma \langle\!\langle = 2 \rangle\!\rangle$ *intra-station channels*, which are circular shift registers spanning the station. The μ accumulators $W'[i]$ contained in this station are partitioned equally between the k processors.

For processing a k-element chunk of the vector, each of the k processors has to decide whether the vector element v_i it currently holds is relevant for the station it belongs to, i.e., whether any of the μ matrix rows handled by this station contains a non-zero entry in column i. If so, then v_i should be communicated to the processor handling the corresponding accumulator(s) and handled there. This is discussed in the following subsection.

3.4 Processing Vector Elements

Fetching Vector Elements. The relevance of a vector entry v_i to a given station depends only on i, which is uniquely determined by the clock cycle and the processor (out of the k) it reached. Consequently, each processor needs to read the content of one pipeline line (to which it is attached) at a predetermined set of

[2] The choice of k depends mainly on the number of available I/O pins for inter-chip communication.

clock cycles, specific to that processor, which is constant across multiplications and easily precomputed. This set of cycles is encoded in the processor as follows.

Each processor contains a *fetches table* which instructs it when to read the next vector element from the pipeline. It contains *fetch events*, represented as triplets (τ, f, ℓ) where τ is an $\delta_{\mathrm{u}} \langle\!\langle = 7 \rangle\!\rangle$-bit integer, f is a one-bit flag and ℓ is a $\lceil \log_2(\gamma) \rceil$-bit integer. Such a triplet means: "ignore the incoming vector entries for τ clock cycles; then, if $f = 1$, read the input vector element and transmit it on the ℓ-th intra-station channel".[3] The table is read sequentially, and is stored in compact DRAM-type memory.

Updating the Accumulators. Once a relevant vector element v_i has been fetched by some processor and copied to an intra-station channel, we still need to handle it by adding $A_{j,i} \cdot v_i$ to the accumulator $W'[j]$, for every row j handled by this station for which $A_{j,i} \neq 0$. These accumulators (usually just one) may reside in any processor in this station. Thus, each processor also needs to occasionally fetch values from the intra-station channels and process it. Similarly to above, the timing of this operation is predetermined, identical across multiplications and easily precomputed.

To this end, each processor also holds an *updates table* containing *update events* represented as a 5-tuple (τ, f, ℓ, j', x) where τ is an $\delta_{\mathrm{f}} \langle\!\langle = 7 \rangle\!\rangle$-bit integer, f is a one-bit flag, ℓ is a $\lceil \log_2(i) \rceil$-bit integer, j' is a $\lceil \log_2(\mu/k) \rceil$-bit integer and x is a field element.[4] Such a 5-tuple means: "ignore the intra-station channels for τ clock cycles; then, if $f = 1$, read the element $y \in \mathrm{GF}(q)$ currently on channel ℓ, multiply it by x, and add the product to the j'-th accumulator in this processor." This table is also read sequentially and stored in compact DRAM-type memory.

During a multiplication, each processor essentially just keeps pointers into those two tables (which can actually be interleaved in a single DRAM bank), and sequentially executes the events described therein.

An update operation requires a multiplication over $\mathrm{GF}(q)$ and addition of the product to an accumulator stored in DRAM (which is very compact but has high latency). These operations occur at non-regular intervals, as prescribed by the updates table; the processors use small queues to handle congestion, where a processor gets several update events within a short interval. Crucially, the load on these queues is known in advance as a side effect of computing the tables. If some processor is over-utilized or under-utilized, we can change the assignments of rows to stations, or permute the matrix columns, to even the load.

Handling Dense Rows. All the entries arriving from the intra-station channels while the updated vector is stored into the DRAM have to be held in the processor's queues. As the random-access latency of DRAM is quite large ($\approx$ 70ns), the entries must not arrive too fast. Some of the rows of A are too dense, and could cause congestions of the queues and intra-station channels. To overcome this problem we split such dense rows into several sparser rows, whose sum equals

[3] The flag f is used to handle the cases where the interval between subsequent fetches is more than $2^{\delta_{\mathrm{u}}} - 1$.

[4] Over GF(2), $x = 1$ always and can thus be omitted.

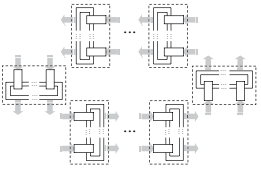


Fig. 3. Arranging the stations into a circle

the original. In this way we also ensure that all stations have a similar load and handle the same number of rows. This increases the matrix size by an insignificant amount ($\langle\!\langle \approx 10^6 \rangle\!\rangle$ additional rows[5]), and the post-processing required to re-combine the split rows is trivial.

Precomputation and Simulation. The content of the two tables used by each processor fully encodes the matrix entries. These tables are precomputed once for each matrix A, e.g., using ordinary PCs. Once computed, they allow us to easily simulate the operation of any processor at any clock cycle, as it is completely independent of the rest of the device and of the values of the input vectors. We can also accurately (though inefficiently) simulate the whole device. Unlike the mesh-based approaches in [9,10,11], we do not have to rely on heuristic run time assumptions for the time needed to complete a single matrix-vector multiplication.

3.5 Skewed Assignment for Iterated Multiplication

In the above scheme, once we have started feeding the initial vector v into the pipeline, after $(D/k) + u$ clock cycles[6] the vector v has passed through the complete pipeline and the vector $A \cdot v$ is stored in the stations. More precisely, each of the u stations contains $\mu = D/u$ consecutive components of v, and we next want to compute the matrix-by-vector product $A \cdot Av$. Thus, we need to somehow feed the computed result Av back into the inter-station pipeline.

To feed the vector Av back into the inter-station pipeline, first we physically close the station interconnects into a circle as depicted in Figure 3; this can be done by appropriate wiring of the chips on the PCB. We also place a memory bank of D/u GF(q) elements at each of the u stations. Collectively, denote these banks by W. At the beginning of each multiplication chain, the initial vector v is loaded into W sequentially, station by station.

During a multiplication, the content of W is rotated, by having each station treat its portion of W as a FIFO of k-tuples: in each clock cycle it sends the last k-tuple of its portion of W to the next station, and accepts a new k-tuple from the previous station. Meanwhile, the processors inside each station function exactly

[5] Extrapolated from a pre-processed RSA-155 NFS matrix from [18], provided to us by Herman te Riele.

[6] Actually slightly more, due to the need to empty the station channels and processor queues.

as before, by tapping the flow of k-tuples of vector elements in W at some fixed point (e.g., the head of the FIFO in that station). Thus, after D/k clock cycles, we have completed a full rotation of the content of W and the multiplication result is ready in the accumulators W. A key point here is that each station sees the contents of W in cyclic order starting at a different offset, but owing to the commutativity of addition in GF(q) this does not affect the final result.

Having obtained the matrix-by-vector, we can now continue to the next multiplication simply by switching the roles (or equivalently, the contents) of the memory banks W and accumulators W': this amounts to a simple local operation in each processor (note that size and distribution among processors of the cells $W[\cdot]$ and the cells $W'[\cdot]$ is indeed identical). Thus, the matrix-by-vector multiplications can be completed at a rate of one per D/k cycles.

3.6 Amortizing Matrix Storage Cost

Recall that in the block Wiedemann algorithm, we actually execute $2K$ multiplication chains with different initial vectors but identical matrix A. These are separated into two phases, and in each phase we can handle these K chains in parallel. An important observation is that we can handle these K chains using a single copy of the matrix (whose representation, in the form of the two event tables, has so far dominated the cost). This greatly reduces the amortized circuit cost per multiplication chain, and thus the overall cost per unit of throughput.

The above is achieved simply by replacing every field element in W and W' by a K-tuple of field elements, and replacing all field additions and multiplications with element-wise operations on the corresponding K-tuples. The event tables and the logic remain the same. Note that the input and output of each station (i.e., the pipeline width) is now $k \cdot K$ field elements.

3.7 Two-Dimensional Chip Array

As described above, each of the processors inside the station incorporates two types of memory storage: a fixed storage for the representation of the matrix elements (i.e., the event tables), and vector-specific storage (W and W') which increases with the parallelization factor K. Ideally, we would like to use a large K in order to reduce the amortized cost of matrix storage. However, this is constrained by the chip area available for W and W'.

To obtain further parallelization without increasing the chip sizes, we could simply run several copies of the device in parallel. By itself, this does not improve the cost per unit of throughput. But now all of these devices use identical storage for the matrix representation, and access it sequentially at the same rate, so in fact we can "feed" all of them from a single matrix representation. In this variant, the event tables are stored in an external DRAM bank, and are connected to the chips hosting the processors and chain-specific storage through a unidirectional pipeline, as illustrated in Figure 4. Note that communication remains purely local—there are no long broadcast wires.

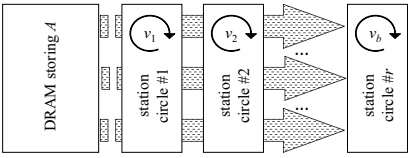


Fig. 4. Using external memory to store the matrix A and b parallel devices, each hosting a circle of stations

This variant splits each of the monolithic chips used by the previous variants into a standard DRAM memory chip for matrix storage, plus a chain of small ASIC chips for the processors and the storage of the vectors. By connecting $b \langle\!\langle = 90 \rangle\!\rangle$ such ASIC chips to each DRAM chip, we can increase the blocking factor K by a factor of b without incurring the cost of duplicate matrix storage.

4 Fault Detection and Correction

4.1 A Generic Scheme

To successfully complete the Wiedemann algorithm, the device must compute all the matrix-by-vector multiplications without a single error.[7] For the problem parameters of interest the multiplications will be realized by tens of thousands of chips operating over several months, and it would be unrealistic to hope (or alternatively, expensive to ensure) that all the computations will be faultless. The same concern arises for other special-purpose hardware designs, and also for software implementations on commodity hardware. It is thus crucial to devise algorithmic means for detecting and correcting faults.

A simple real time error-detection scheme would be to apply a linear test: during a preprocessing stage, choose a random $d \times D$ matrix B for an appropriate d, precompute on a reliable host computer and store in the hardware the $d \times D$ matrix $C = BA$, and verify that $Bw' = Cw$ whenever the hardware computes a new product $w' = Aw$. Over GF(q) each row of the matrix B reduces the probability of an undetected error by a factor of q, and thus for $q = 2$ we need at least a hundred rows in B to make this probability negligible. Since each one of the dense $100 \times D$ matrices B and C contains about the same number of 1's as the sparse $D \times D$ matrix A (with one hundred 1's per row), this linear test can triple the storage and processing requirements of the hardware, and meshes poorly with the overall design whose efficiency relies heavily on the sparseness of the matrix rows. Note that we cannot solve this problem by making the $100 \times D$ matrix B sparse, since this would greatly reduce the probability of detecting single bit errors.

[7] Note the contrast with the NFS sieving step, which can tolerate both false positive and false negative errors in its smoothness tests.

In the following we describe an alternative error-detection scheme, which provides (effectively) an arbitrarily small error probability at a negligible cost, under reasonable assumptions. It inspects only the computed (possibly erroneous) matrix-by-vector products, and can thus be applied to any implementation of Wiedemann's algorithm. We will consider its operation over any finite field GF(q), though for integer factoring via NFS only $q = 2$ is of interest.

Detection. Let $w_0, w_1, w_2, \ldots \in \mathrm{GF}(q)^D$ denote the sequence of vectors computed by the device, where $w_0 = v$. To verify that indeed $w_i = A^i v$ for all $i > 0$, we employ the following randomized linear test. For a small integer $d \langle\!\langle = 200 \rangle\!\rangle$, choose a single vector $b \in \mathrm{GF}(q)^D$ uniformly at random, and precompute on a reliable computer the single vector $c^{\mathrm{T}} = b^{\mathrm{T}} A^d$ (here $^{\mathrm{T}}$ denotes transpose). After each w_i is computed, compute also the inner products $b^{\mathrm{T}} w_i$ and $c^{\mathrm{T}} w_i$ (which are just field elements). Save the last d results of the latter in a small shift register, and after each multiplication test the following condition:

$$b^{\mathrm{T}} w_i = c^{\mathrm{T}} w_{i-d} \quad . \tag{2}$$

If equality does not hold, declare that at least one of the last d multiplications was faulty.

Correctness. If no faults have occurred then (2) holds since both sides equal $b^{\mathrm{T}} A^i v$. Conversely, we will argue that the first faulty multiplication $w_j \neq A w_{j-1}$ will be detected within d steps with overwhelming probability, under reasonable assumptions.

Let us first demonstrate this claim in the simplest case of some transient error ε which occurs in step j. This changes the correct vector $A^j v$ into the incorrect vector $w_j = A^j v + \varepsilon$. All the previous w_i for $i < j$ are assumed to be correct, and all the later w_i for $i > j$ are assumed to be computed correctly, but starting with the incorrect w_j in step j. It is easy to verify that the difference between the correct and incorrect values of the computed vectors w_i for $i > j$ evolves as $A^{i-j}\varepsilon$, and due to the randomness of the matrix A generated by the sieving step these error vectors are likely to point in random directions in the D-dimensional space $\mathrm{GF}(q)^D$. The device has d chances to catch the error by considering pairs of computed vectors which are d apart, with the first vector being correct and the second vector being incorrect. The probability that all these d random error vectors will be orthogonal to the single random test vector b is expected to be about q^{-d}, which is negligible; the computational cost was just two vector inner products per matrix-by-vector multiplication.

The analysis becomes a bit more involved when we assume that the hardware starts to malfunction at step j, and adds (related or independent) fault patterns to the computed result after the computation of each matrix-vector product from step j onwards. Let the result of the i-th multiplication be $w_i = A w_{i-1} + \varepsilon_i$, where the vector ε_i is the error in the output of this multiplication. We consider the first fault, so $\varepsilon_i = 0$ for all $i < j$. Assume that $j \geq d$ ($j < d$ will be addressed below). By the linearity of the multiplication and the minimality of j, we can expand the above recurrence to obtain $w_i = A^i v + \sum_{i'=j}^{i} (A^{i-i'} \varepsilon_{i'})$ $(i \geq j)$.

Plugging this into (2) and canceling out the common term $b^{\mathrm{T}}A^i v$, we get that for $j \le i < j + d$, (2) is equivalent to:

$$b^{\mathrm{T}} r_i = 0 \qquad \text{where} \qquad r_i = \sum\nolimits_{i'=j}^{i} (A^{i-i'} \varepsilon_{i'}) \ . \tag{3}$$

We assume that each error ε_i is one of at most $(qD)^\alpha$ possibilities for some $\alpha \ll D/d$ (e.g., $\langle\!\langle \alpha = 10^5 \rangle\!\rangle$), regardless of A and b. This suffices to enumerate all reasonably likely combinations of local faults (corrupted matrix entries, faulty pipeline connections, errors in $\mathrm{GF}(q)$ multipliers, memory bit flips, etc.). We also make the simplifying (though not formally correct) assumption that $A^{10}, ..., A^{d-1}$ are random matrices drawn uniformly and independently.[8] Then for any fixed values of ε_i, the vectors in the set $R = \{r_i\}_{i=j+10}^{j+d-1}$ are drawn uniformly and independently from $\mathrm{GF}(q)^D$ (recall that $\varepsilon_j \neq 0$), and thus the probability that the span of R has dimension less than $|R| = d - 10$ is smaller than $dq^{-(D-d)}$ (which is a trivial upper bound on the probability that one of the $d-10$ vectors falls into the span of the others). By assumption, there are at most $(qD)^{\alpha d}$ possible choices of $(\varepsilon_i)_{i=j+1}^{j+d}$. Hence, by the union bound, the probability that the span of R has dimension less than $d - 10$ is at most $(qD)^{\alpha d} \cdot dq^{-(D-d)} = d \cdot q^{\alpha d \log_q D + d - D}$, which is negligible. Conditioned on the span of R having full rank $d-10$, the probability of the random vector b being orthogonal to the span of R is $q^{-(d-10)}$, which is also negligible. Hence, with overwhelming probability, at least one of the tests (3) for $j + 10 < i < j + d$ will catch the fault in w_j.

Startup and Finalization. Note that the test (2) applies only to $i > d$, and moreover that our analysis assumes that the first d multiplications are correct. Thus, for each of the $2K$ multiplication chains of block Wiedemann, we start the computation by computing the first d multiplications on a reliable general-purpose computer, and then load the state (including the queue of $c^{\mathrm{T}}w_i$ values for $i = 0, \ldots, d$) into the device for further multiplications.

Also note that in the analysis, the results of the j-th multiplications are implicitly checked by (2) for $i = j, \ldots, j + d - 1$. Thus, in order to properly check the last d multiplications in each chain, we run the device for d extra steps and discard the resulting vectors but still test (2).

Recovery. The above method will detect a fault within d clock cycles (with overwhelming probability), but will not correct it. Once the fault is detected, we must backtrack to a known-good state without undoing too much work.

[8] The sieving and preprocessing steps of NFS yield a matrix A that has nearly full rank and is "random-looking" except for some biases in the distribution of its values: A is sparse (with density $\langle\!\langle \approx 100/10^{10} \rangle\!\rangle$) and its density is decreasing with the row number. The first few self-multiplications increase the density exponentially and smoothen the distribution of values, so that A^{10} has full and uniform density. The independence approximation is applicable since we are looking at simple local properties (corresponding to sparse error vectors), which are "mixed" well by the matrix multiplication. While the resulting matrices do have some hidden structure, realistic fault patterns are oblivious to that structure.

Assuming a sufficiently low probability of error, it is simplest to dump a full copy of the current vector w_i from the device into a general-purpose computer, at regular but generously-spaced intervals; this can be done by another special station tapping the pipeline. The backup vectors may be stored on magnetic media, and thus their storage has negligible cost. When a fault is detected, the faulty component can be replaced (or a spare device substituted) and the computation restarted from the last known-good backup.

4.2 Device-Specific Considerations

Implementation. The above scheme requires only the computation of two inner products ($b^{\mathrm{T}}w_i$ and $c^{\mathrm{T}}w_i$) for each multiplication. In the proposed hardware device, this is achieved by one additional station along the pipeline, which taps the vector entries flowing along the pipeline and verifies their correctness by the above scheme. This station contains the entries of b and c in sequential-access DRAM. For each of the K vectors being handled, it processes a k-tuple of vector entries at every clock cycle, keeps the d most recent values of $c^{\mathrm{T}}w_i$ in a local FIFO queue at this station, and performs the test according to (2).

Halving the Cost. The storage cost can be halved by choosing b pseudorandomly instead of purely randomly; the number of multipliers can also be nearly halved by choosing b to be very sparse.

Using Faulty Chips. In addition to the above high-level error-recovery scheme, it is also useful to work around local faults in the component chips: this increases chip yield and prevents the need to disassemble multi-chip devices if a fault was discovered after assembly. To this end, the proposed device offers a significant level of fault tolerance due to its uniform pipelined design: we can add a "bypass" switch to each station, which effectively removes it from the pipeline (apart for some latency). Once we have mapped the faults, we can work around any fault in the internals of some station (this includes the majority circuit area) by activating the bypass for that station and assigning its role to one of a few spare stations added in advance. The chip containing the fault then remains usable, and only slightly less efficient.

5 Cost and Performance

5.1 Cost for 1024-Bit NFS Matrix Step

As explained in Section 2, there is considerable uncertainty about the size and density of the matrices that would appear in the factorization of 1024-bit composites using the Number Field Sieve. For concreteness and ease of comparison, throughout Section 3 and in the following we assume the rather conservative "large matrix" parameters (see Section 2).

Clearly there are many possibilities for fixing the different parameters of our device, depending on such parameters as desired chip size and number of

chips. One may even consider combining the above design with the splitting of the processed matrix into submatrices as put forward in [10], thereby giving up the homogeneity and purely local communication but decreasing the dimension of the vectors that have to be handled. In the following we consider a specific parameter set, which focuses on practicality with today's technology.

We assume 90nm chip manufacturing technology with DRAM-type process[9], a net chip area of 1 cm^2, a per-chip I/O bandwidth of 1024 Gbit/s, and a clock rate of 1GHz. A DRAM access is assumed to take 70 clock cycles. These parameters are quite realistic with current technology.

We employ a 300×90 array of ASIC chips. Each column of 300 chips contains $u = 9600$ stations (32 per chip). Each station consists of $k = 32$ processors, communicating over $\gamma = 2$ intra-station channels, with a parallelization factor of 10. Each of the 300 rows, of 90 chips each, is fed by a 108Gbit DRAM module. Overall, the blocking factor is $K = 10 \cdot 90 = 900$. This array can complete all multiplication chains in ≈ 2.4 months.

The total chip area, including the matrix storage, is less than 90 full 30cm wafers. Assuming a "silicon cost" of US$ 5000 per wafer, and a factor 4 increase for overheads such as faulty chips, packaging, testing and assembly, the total cost is under US$ 2M.

Comparison to Previous Designs. A mesh-based design as considered in [11], adapted to 90nm technology and using 85×85 chips of size 12.25 cm^2 each, will require about 11.7 months to process the above matrix. The higher complexity of this design limits the clocking rate to 200 MHz only. Comparing throughput per silicon area, the new device is 6.5 more efficient; it also has much smaller individual chips and no need for non-local wiring.

Implications for 1024-Bit Factorization. With the above device and matrix size, the cost of the NFS linear algebra step is 0.4M US$×year, which is significantly lower than that of the NFS sieving step using the TWIRL device [4]. Moreover, TWIRL is expected to produce a matrix significantly smaller than the conservative estimate used above, so the cost of the linear algebra step would be lower than the above estimate. Since TWIRL, being a wafer-scale design, is also more technologically challenging, this reaffirms the conclusion that at present the bottleneck of factoring large integers is the NFS sieving step [9].

5.2 Further Details

To derive concrete cost and performance estimates for the 1024-bit case, several implementation choices for parameters, such as δ_u, δ_f, γ, τ, have been determined experimentally as follows. For the above problem and technology parameters, and a large randomly drawn matrix, we used a software simulation of a station to check for congestions in bus and memory accesses, and chose design parameters for which such congestions never occur experimentally. Recall that the device's

[9] Amortized DRAM density is assumed to be $0.1\mu m^2$ per bit, and the logic is assumed to have an average density of $1.4\mu m^2$ per transistor.

operation is deterministic and repetitive (see Section 3.4), so the simulation accurately reflects the device's operation with the given parameters.

In the following we briefly mention some aspects of the circuit area and its analysis, as used to derive the above estimate. Note that we employ the split design of Section 3.7, which puts the matrix storage in plain DRAM chips and the logic and vector storage in ASIC chips. For these parameters, memory storage dominates area: approximately 97% of the ASIC chip area is occupied by the DRAM which stores the intermediate vectors (i.e., W and W'). Thus, the suitable chip production process is a DRAM process optimized for maximum memory density (at the expense of slightly larger logic circuits); similar cases arose in previous proposals [9,10,11]. Each of the $k \langle\!\langle = 32 \rangle\!\rangle$ processors in each of the 32 stations in each of the 300×90 ASIC chips contains the following logic:

- A K/b-bit register for storing the K/b-tuples of GF(2) elements flowing along from the inter-station pipeline ($\approx 8 \cdot K/b$ transistors).
- A K/b-bit register for each of the $\gamma \langle\!\langle = 2 \rangle\!\rangle$ intra-station channels ($\approx 8 \cdot \gamma \cdot K/b$ transistors).
- A FIFO queue of depth $\langle\!\langle 2 \rangle\!\rangle$ for storing elements arriving on the inter-station pipeline along with the number of the internal bus onto which the respective element is to be written. For this $\approx 2 \cdot 8 \cdot (K + \lceil \log_2(\gamma) \rceil)$ transistors per queue entry are sufficient.
- A FIFO queue of depth $\langle\!\langle 4 \rangle\!\rangle$ for storing elements arriving on the intra-station channels that have to be XORed to the vector. Each entry consists of a K/b-tuple of bits for the vector and a row number in the submatrix handled by the station has to be stored. This occupies $\approx 4 \cdot 8 \cdot (K/b + \lceil \log_2 \lceil D/(ku) \rceil \rceil)$ $\langle\!\langle = 4 \cdot 8 \cdot (10 + 15) \rangle\!\rangle$ transistors per queue entry.

In addition to the registers and queues, we need some logic for counters (to identify the end of a vector and to decide when to read another element from a bus), multiplexers, etc. For the parameters of interest, < 1500 transistors are sufficient for this. Overall, the 32×32 processors on each chip occupy $\langle\!\langle \approx 3.2\text{mm}^2 \rangle\!\rangle$.

The DRAM needed splits into three parts.

- For storing $2 \cdot K/b$ vectors in $\mathrm{GF}(2)^{\lceil D/(uk) \rceil}$,: $2 \cdot K/b \cdot D/(uk)$ bit $\langle\!\langle \approx 650\ \text{Kbit} \rangle\!\rangle$.
- For the fetches table: $\delta_\text{u} + 1 + \lceil \log_2(\gamma) \rceil$ bits per entry.
- For the updates table: $\delta_\text{f} + 1 + \lceil \log_2(\gamma) \rceil + \lceil \log_2 \lceil D/(uk) \rceil \rceil$ bits per entry.

Overall, the DRAM on each chip occupies $\langle\!\langle \approx 67\text{mm}^2 \rangle\!\rangle$. The time for each of the $\approx 2D/K$ matrix-by-vector multiplications is $\approx e + D/k$ clock cycles, where e gives some leeway for emptying queues and internal buses (for the parameters we are interested in $e \ll 1000$ is realistic).

6 Conclusion

We have described a pipelined systolic design for the matrix-by-vector multiplications of the block Wiedemann algorithm, which exhibits several advantages over the prior (mesh-based) approach. It has lower cost and modest technological

requirements; specifically, unlike previous proposals it uses standard chip sizes and purely local communication. The architecture is scalable, and offers the flexibility to handle problems of varying sizes. The operation is deterministic and allows local simulation and verification of components. We have also described an efficient error detection and recovery mechanism, which can also be adapted to other software or hardware implementations of Wiedemann's algorithm.

For 1024-bit RSA keys, executing the linear algebra step of the NFS using this device appears quite realistic with present technology, at a cost lower than that of the NFS sieving step.

References

1. Shamir, A.: Factoring Large Numbers with the TWINKLE Device. In: CHES 1999. Volume 1717 of LNCS, Springer (1999) 2–12
2. Lenstra, A.K., Shamir, A.: Analysis and Optimization of the TWINKLE Factoring Device. In: EUROCRYPT 2000. Volume 1807 of LNCS, Springer (2000) 35–52
3. Geiselmann, W., Steinwandt, R.: A Dedicated Sieving Hardware. In: PKC 2003. Volume 2567 of LNCS, Springer (2003) 254–266
4. Shamir, A., Tromer, E.: Factoring Large Numbers with the TWIRL Device. In: CRYPTO 2003. Volume 2729 of LNCS, Springer (2003) 1–26
5. Geiselmann, W., Steinwandt, R.: Yet Another Sieving Device. In: CT-RSA 2004. Volume 2964 of LNCS, Springer (2004) 278–291
6. Franke, J., Kleinjung, T., Paar, C., Pelzl, J., Priplata, C., Stahlke, C.: SHARK - A Realizable Special Hardware Sieving Device for Factoring 1024-bit Integers. In: SHARCS 2005. (2005)
7. Franke, J., Kleinjung, T., Paar, C., Pelzl, J., Priplata, C., Simka, M., Stahlke, C.: An Efficient Hardware Architecture for Factoring Integers with the Elliptic Curve Method. In: SHARCS 2005. (2005)
8. Bernstein, D.J.: Circuits for Integer Factorization: a Proposal. At the time of writing available electronically at `http://cr.yp.to/papers/nfscircuit.pdf` (2001)
9. Lenstra, A.K., Shamir, A., Tomlinson, J., Tromer, E.: Analysis of Bernstein's Factorization Circuit. In: ASIACRYPT 2002. Volume 2501 of LNCS, Springer (2002) 1–26
10. Geiselmann, W., Steinwandt, R.: Hardware for Solving Sparse Systems of Linear Equations over GF(2). In: CHES 2003. Volume 2779 of LNCS, Springer (2003) 51–61
11. Geiselmann, W., Köpfer, H., Steinwandt, R., Tromer, E.: Improved Routing-Based Linear Algebra for the Number Field Sieve. In: Proceedings of ITCC '05 – Track on Embedded Cryptographic Systems, IEEE Computer Society (2005) 636–641
12. Frey, G.: A First Step Towards Computations in Brauer Groups and Applications to data Security. Invited talk at WARTACRYPT '04 (2004)
13. Frey, G.: On the Relation between Brauer Groups and Discrete Logarithms. Unpublished manuscript (2004)
14. Pomerance, C.: A Tale of Two Sieves. Notices of the ACM (1996) 1473–1485
15. Lenstra, A.K., Hendrik W. Lenstra, J., eds.: The development of the number field sieve. Volume 1554 of Lecture Notes in Mathematics. Springer (1993)
16. Coppersmith, D.: Solving Homogeneous Linear Equations over GF(2) via Block Wiedemann Algorithm. Mathematics of Computation **62** (1994) 333–350

17. Villard, G.: Further analysis of Coppersmith's block Wiedemann algorithm for the solution of sparse linear systems. In: International Symposium on Symbolic and Algebraic Computation — ISAAC '97, ACM (1997) 32–39
18. Cavallar, S., Dodson, B., Lenstra, A., Lioen, W., Montgomery, P., Murphy, B., te Riele et al., H.: Factorization of a 512-bit RSA modulus. In: EUROCRYPT 2000. Volume 1807 of LNCS, Springer (2000) 1–17

Design of Testable Random Bit Generators

Marco Bucci and Raimondo Luzzi

Infineon Technologies Austria AG,
Babenbergerstrasse 10, A-8020 Graz (AUSTRIA)
{marco.bucci, raimondo.luzzi}@infineon.com

Abstract. In this paper, the evaluation of random bit generators for security applications is discussed and the concept of stateless generator is introduced. It is shown how, for the proposed class of generators, the verification of a minimum entropy limit can be performed directly on the post-processed random numbers thus not requiring a good statistic quality for the noise source itself, provided that a sufficient compression is adopted in the post-processing unit. Assuming that the noise source is stateless, a straightforward entropy estimator to drive an adaptive compression algorithm is proposed. Examples of stateless sources are also discussed.

Finally, an attack scenario against a noise source is defined and an effective approach to the attack detection is presented. The entropy estimator and the attack detection together guarantee the unpredictability of the generated random numbers.

Keywords: Random bit source, random numbers, entropy, ring oscillators, jitter.

1 Introduction

Random numbers are extensively used in many cryptographic operations. Public/private key pairs for asymmetric algorithms are generated from a random bit stream; a random bit generator (RBG) is also needed for key generation in symmetric algorithms, for generating challenges in authentication protocols, and for creating padding bytes and blinding values [1].

Even if, historically, the only requirement for an RBG was to fulfill bunches of statistical tests aimed to reveal defects in the generated data, nowadays in the technical community is well accepted that, for random numbers used in cryptography, a flat statistic is not sufficient and their unpredictability is the main requirement: a potential attacker must not be able to carry out any useful prediction about the generator's output even if its design is known. As a consequence, the focus is on the verification of a minimum entropy requirement and statistical tests are significant only if the statistical model of the random source under evaluation is known [2].

The German IT security certification authority (BSI) has adopted this approach in its AIS 31 publication [3] where the physical noise source is separated from the digital post-processing and, for devices belonging to the functionality class P2[1], criteria

[1] Devices intended for generation of signature key pairs, generation of DSS signatures, generation of session keys for symmetric encryption mechanisms must belong to the functionality class P2.

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 147–156, 2005.

and statistical tests are defined for the noise source output (*digitized noise signal*) in order to verify a minimum entropy limit for the post-processor output (*internal random numbers*). Namely, the entropy requirement on the final data is guaranteed by defining a minimum entropy limit for the raw data from the source and, at the same time, requesting that the adopted post-processing algorithm does not reduce its input entropy.

In this paper we propose to go further ahead with this approach defining an RBG based on a stateless (memoryless) noise source and a stateless post-processing algorithm. Since the noise source is assumed memoryless, the generated symbols are independent and, since the post-processor is also memoryless, the internal random numbers are independent too. Therefore, the entropy limit can be verified directly after the post-processor, controlling that the assumed compression ratio in the post-processor is well chosen with respect to the available entropy per bit from the source. In this scheme, very fast noise sources, but with a low entropy per bit ("spread" entropy sources), can be adopted, provided that a sufficient compression is applied. In other terms, the relevant figure of merit becomes the *entropy throughput* (entropy/second) instead of the entropy per bit, and the design of the noise source is not anymore constrained by the statistical quality, but efficiency and robustness can be taken as the main goals.

Moreover, under the hypothesis of independent symbols, a straightforward entropy estimator can be used to evaluate the amount of entropy produced by the source thus allowing an adaptive post-processing (compression) and an on-line test of the source. In fact, even if the source entropy throughput can be evaluated during the characterization of selected prototypes, tolerances of components, temperature drifts and ageing affect the statistical properties of the noise source thus requiring an on-line test during the RBG operation [2].

In addition to technological and environmental variances, attack scenarios when the generator is operated in a real application should be also considered. In this case, the effective entropy is basically the attacker's error rate when observing or forcing the noise source output. Since the estimation of the attacker's error rate should be very conservative, a strong post-processing is required anyway (minimum compression ratio), and requesting a high statistical quality for the noise source operated in a controlled environment loses significance.

In Section 2, the proposed RBG is described focusing on how the hypothesis of statistical independence for both the raw data and the internal random numbers is fulfilled. Examples of stateless noise sources are reported in Section 3 while, in Section 4, the important topic of robustness against attacks is discussed proposing an effective approach to their detection.

2 Stateless Random Bit Generators

Random bit generators used in applications where the unpredictability is a key requirement are based on non-deterministic phenomena that act as the source of randomness. In integrated circuit implementation, electronic noises (thermal and shot) and time jitter are usually the only available randomness sources.

Every noise source, even if well-designed, produces a bit stream that usually shows statistical defects due to bandwidth limitation, fabrication tolerances, ageing and tem-

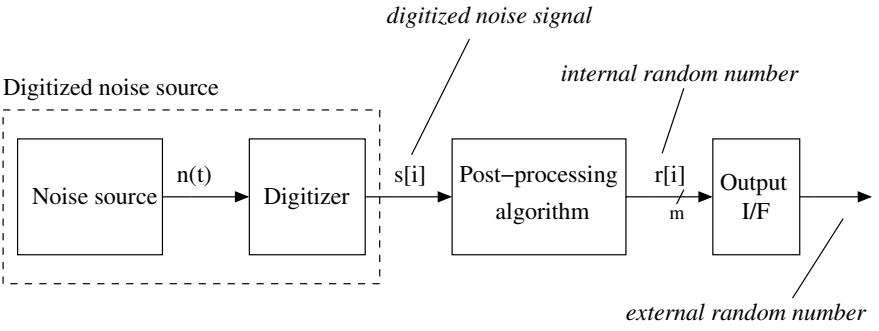


Fig. 1. Generic architecture for an RBG

perature drifts, deterministic disturbances and, in case, signals forced by an attacker. As a consequence, the noise source should be always followed by a strong digital post-processing as shown in Figure 1, where the definitions reported in [3] have been adopted.

A noise source generates an analog signal $n(t)$ which is input into a digitizer. The digitizer samples the analog signal and converts the sampled values into a stream of random bits $s[i]$ (*digitized noise signal*). The non-deterministic source and the digitizer together form the digitized noise source. The random bits $s[i]$ are then fed into the post-processing unit which then produces a stream of m-bit random words $r[i]$ (*internal random numbers*).

The post-processing unit fulfills two purposes: Firstly, it is necessary to adjust the probability distribution of the raw random bits $s[i]$ thus overcoming statistical defects present in the non-deterministic noise source or in the digitizer (e.g. input offset of a voltage comparator). The probability distribution of the resultant random words $r[i]$ is much closer to a uniform distribution than that of the input stream $s[i]$.

Secondly, the post-processor is used to increase the entropy per bit of the output stream. The entropy per bit is increased adopting a compressing function that collects ("distills") the entropy in the input stream $s[i]$ to produce a lower speed output stream with increased randomness.

Due to the presence of mathematical post-processing in an RBG, it is not straightforward for a certification authority to discriminate between a truly RBG and a pseudo random generator which is able to deceive statistical tests producing a uniform probability distribution without, however, producing any entropy.

In order to guarantee that the generated random numbers are indeed unpredictable and not just uniformly distributed, the AIS 31 guidelines [3] require, in P2.c), - in addition to the requirement that the statistical behavior of the internal random numbers should be inconspicuous - that "*the prospects of success for systematic guessing of the external random numbers*[2] *(realised through systematic exhaustion attacks) - even if external random number sub-sequences are known - should at best be negligibly higher*

[2] As shown in Figure 1, random numbers that the RBG external interface delivers to the application are named external random numbers.

than would be the case if the external random numbers had been generated by an ideal random number generator".

To certify that this is the case, in P2.d)(vii), [3] states that the sequence of raw random bits $s[i]$ has "*to meet particular criteria or pass statistical tests intended to rule out features such as multi-step dependencies*" and a suite of statistical tests to be fulfilled is provided. Basically, by testing the random bits before the post-processing it can be guaranteed that the random words $r[i]$ do indeed have entropy as a physical non-deterministic noise source is involved in their generation.

Although the AIS 31 has finally focused the attention on the entropy, the described approach has the disadvantage to consider as the relevant figure of merit the entropy per bit produced by the source and not its entropy throughput (entropy/second). Therefore, fast sources but with low entropy per bit, that could produce good quality random numbers if a sufficient compression is applied and, in general, have a more robust implementation than sources with a better probability distribution, are ruled out.

Moreover, it is not always possible to separate the noise source from the post-processing. For example, RBG's based on discrete chaotic systems [4,5] present an intrinsic pseudo-random behavior superimposed to a random evolution. Therefore, even if statistical tests applied on the source output $s[i]$ pass, that is not sufficient to state that a chaotic source produces enough entropy per bit.

The approach proposed in this paper is to consider the noise source only as a source of entropy, without requiring good statistical properties for it, and to transfer to the post-processor the task of adjusting the statistic quality and increasing the entropy per bit by means of a sufficiently high compression. Of course that requires a different procedure to certify the generator since, in general, the noise source will not pass the AIS 31 suite of statistical tests as requested in P2.d)(vii).

Actually, this different approach is still compatible with [3], where *alternative criteria* to statistical tests on the source are also provided. In particular, according to the alternative criteria - type 1, "*the applicant may alternatively submit the following proof:*

- *Internal random number sequences pass statistical tests specified in P2.i)(vii).*
- *Clear proof that the internal random numbers achieve the goal set with criterion P2.d)(vii).*
  The proof must the provided taking into account the mathematical post-processing and on the basis of the empirical properties of the digitized noise signal sequence."

Criterion P2.d)(vii) and the statistical tests specified in P2.i(vii) are aimed "*to guarantee P2.c) for selected prototypes by verifying a minimum entropy limit for each internal random bit with a negligibly small error probability*". In other words, criterion P2.d)(vii) requires that the random stream after the post-processor meets a minimum entropy limit in order to guarantee the unpredictability of the generated numbers.

Now, if the adopted noise source is assumed to be stateless (memoryless), then the generated symbols $s[i]$ are independent. As a consequence, if the post-processing unit is also stateless, the internal random numbers $r[i]$ are independent too and the obtained entropy per bit can be evaluated from the uniformity of their estimated (empirical) probability distribution. In particular, a χ^2 test with $k-1$ degrees of freedom can be applied, where $k = 2^m$ is the cardinality of the output alphabet [1]. Obviously, to keep the test

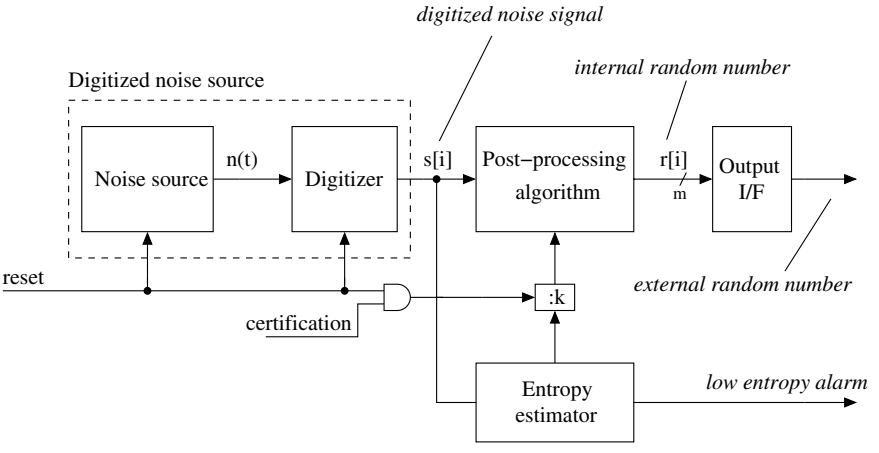


Fig. 2. A stateless random bit generator

feasible, the output parallelism m must be limited; a typical output interface for a RBG is 8-bit wide.

The stateless hypothesis can be fulfilled by resetting to a constant value every state variable both in the source[3] and the post-processor[4] before the generation of a new bit $s[i]$ and a new word $r[i]$ respectively, as depicted in Figure 2.

While it is perfectly clear what resetting the digital post-processor means, the implementation of the noise source reset depends on the particular source we adopt and two examples of stateless sources will be discussed in Section 3.

It is interesting to observe that the source reset before each bit generation is not strictly necessary. In fact, if a compression factor k is used in the post-processor, the independence of the input $(k \cdot m)$-bit sub-sequences $\{s[(k \cdot m)i + j]\}$, $j = 0, \ldots, (k \cdot m) - 1$, is sufficient to prove the independence of the m-bit output symbols $r[i]$. Therefore, resetting the noise source every time a new output word $r[i]$ is generated would be sufficient.

Nevertheless, the additional hypothesis of independence of each raw bit allows to adopt the counting of the transitions in the sequence $s[i]$ as a straightforward *entropy estimator* (Figure 2) for the source [6]:

$$N_{trans} = \sum_i \left(s[i] \oplus s[i-1] \right). \tag{1}$$

A lack of transitions with respect to what is expected from an ideal random source signals a lack of entropy. Using (1), a post-processor with adaptive compressing can be defined [7], processing raw bits from the source until a minimum number of transitions has been counted. For example, if 16 transitions must be counted before an 8-bit output word $r[i]$ is released, that is equivalent to have a mean dynamic compression equal to

[3] Resetting the noise source implies resetting the underlying noise process too. Alternatively, a bit generation frequency sufficiently slower than the noise process bandwidth must be adopted.

[4] Patent pending.

4 for an ideal input sequence, which increases automatically as the entropy from the source decreases.

The mean compression is chosen according to the entropy throughput expected from the source. In addition, an upper limit for the compression ratio can be fixed (e.g. $k = 16$), after that an alarm is asserted thus detecting faults in the source.

Even if the post-processing reset is necessary during the entropy evaluation (certification), it can be disabled during the normal operation (Figure 2). In fact, if a linear post-processing algorithm is employed (e.g. a linear-feedback shift register), it can be easily proved that the output is equivalent to the sum of the reset post-processing output and the output when the state is maintained and there is no input string from the source (free evolution). Therefore, the resulting entropy per bit is not lower than that evaluated during the RBG certification.

3 Examples of Stateless Sources

Even if every noise source can be turned into a stateless source simply switching off and on again its power supply before the generation of each new bit[5], in practice, an ad hoc designed source is necessary. In particular, a *reset state* must be implemented where every memory of the previous generated bit is canceled. A recovery time from the reset negligible with respect to the bit generation time is the main condition to fulfill.

A first example of stateless noise source can be obtained from the well-know oscillator-based random bit generator [8,9,10,11] if both oscillators are stopped after each bit generation (Figure 3), thus avoiding the phase shifting between f_{fast} and f_{slow}. If digital ring oscillators are employed to implement f_{fast} and f_{slow}, the start-up time is usually a small fraction of the their oscillating period.

On the trailing edge of a start pulse, both oscillators are enabled and, after $T_{slow}/2$, the f_{slow} raising edge samples f_{fast} and disables both oscillators. A new start pulse is necessary to trigger the generation of a new bit $s[i]$.

The phase shift between the oscillators is not the only state variable which must be canceled in this system. In fact, the digitized (a flip-flop in Figure 3) has a memory and its behavior in commutation depends somehow on the current state (e.g. different switching timings). As depicted in Figure 3, this further state variable can be easily canceled resetting the flip-flop too.

An RBG based on this noise source is reported in [12] where an additional feedback loop is employed to control a starting delay on f_{fast} in order to force the slow oscillator to sample the fast one close to one of its edges. Such compensation delay is controlled according to the mean value of the generated bit stream $\{s[i]\}$ and, as a consequence, the source is not stateless any more. Anyway, if the feedback loop is stopped once the steady-state has been reached, thus alternating compensation and normal operation phases, the stateless hypothesis still holds.

As a second example, a noise source based on a well known chaotic circuit [13] is depicted in Figure 4. The circuit state is the voltage across the capacitor C and a reset

[5] External asynchronous disturbances and signals forced by an attacker are neglected in this context. An attack scenario and how to detect it is discussed in Section 4.

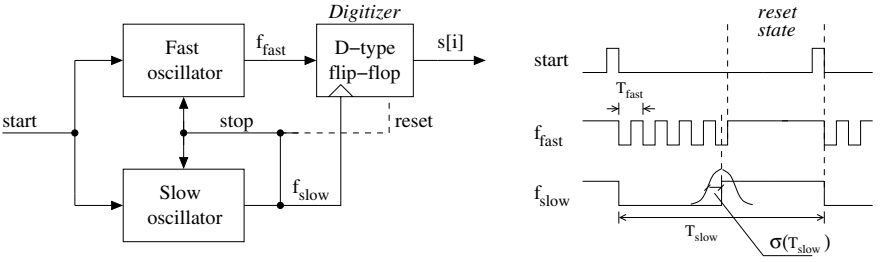


Fig. 3. A stateless oscillator-based noise source

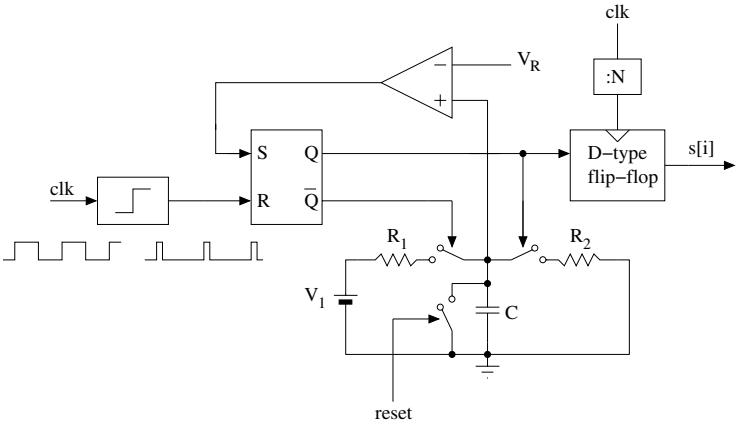


Fig. 4. A stateless noise source based on a chaotic circuit

state can be easily implemented with an additional switch. In general, that holds for every chaotic source based on as a switched capacitor circuit.

On the rising edges of the clock clk, the capacitor C is charged with a time constant $\tau_1 = R_1 C$. When the voltage across C reaches a reference level V_R, C is discharged with a time constant $\tau_2 = R_2 C$. If τ_1, τ_2 and the clock period are properly chosen, the circuit shows a chaotic behavior. Every N clock periods an output bit $s[i]$ is generated and C is reset.

The down-sampling parameter N must be properly chosen in order to obtain a sufficient number of transitions (1). Its value is strictly dependent on the actual divergence speed of the chaotic circuit (Lyapunov exponent). As a consequence, the transition test (1) detects whether or not the circuit is actually having a chaotic behavior, i.e. if all the critical parameters of the circuit are currently in range.

4 Attack Detection in Stateless Noise Sources

An entropy estimator based on the transition counter (1) has been introduced in Section 3, stating that a lack of transitions signals a lack of entropy. Unfortunately, if attack sce-

narios are taken into account, the inverse implication does not hold. In fact, an attacker will force the noise source with a pseudo-random disturbance thus deceiving the transition counter. Therefore, an additional attack detection mechanism is necessary, while the transition counter is solely intended to detect faults in the noise source.

From the attacker point of view, the actual entropy of the noise source is his error probability when observing or forcing the generator. An RBG can be observed with the same techniques employed against cryptographic processors, in particular side-channel attacks. However, in this case, averaging techniques are not helpful thus limiting the effectiveness of the attack. On the other hand, since noise sources are based on very weak random signals, forcing attacks by means of strong disturbances are a threat for RBG's and must be taken into account.

The attack model depicted in Figure 5 is assumed in the following: the attacker can superimpose an own random signal $d(t)$ to the noise output $n(t)$ thus obtaining a probability distribution that, after the digitizer, is indistinguishable from what the generator produces in normal conditions. It is then clear that an attack detection is possible only before the digitization.

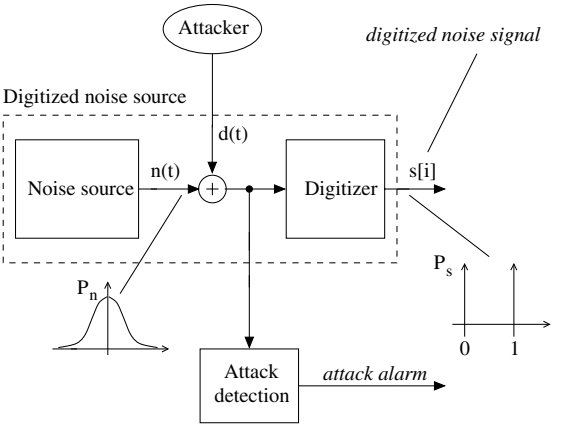
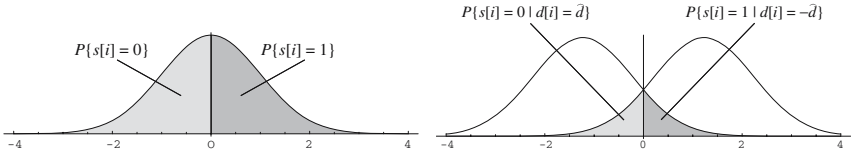


Fig. 5. Forcing attack on a noise source

In order to force the source, the attacker shifts the source probability distribution P_n adding a disturbance $\pm\widehat{d}$ and, to obtain an effective entropy reduction (taking into account the compression in the post-processor), his error probability $Pr\{s[i] = 1 \mid d[i] = -\widehat{d}\}$ must be sufficiently small (Figure 6). In other terms, a disturbance amplitude $\widehat{d}$ larger than the P_n standard deviation is required.

Therefore, an attacker cannot force a noise source without increasing its intensity and the attack detection is based on a source intensity test, assuming that its intensity in normal conditions is known.

A simple statistical test to detect the attack is counting the number of samples falling beyond two fixed thresholds as shown in Figure 7, triggering an alarm if the counted value is too high with respect to the expected value.



(a) normal conditions.

(b) conditional distributions and residual error under attack.

Fig. 6. Source probability distribution

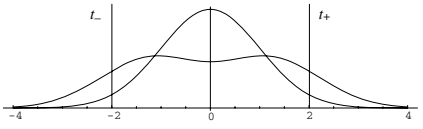


Fig. 7. Noise source intensity test

Such approach is general and can be applied to the noise signal $n(t)$ produced by every kind of stateless noise sources. As an example, for the oscillator-based RBG discussed in Section 3, the attack detection is implemented adding a second flip-flop which samples f_{fast}/n, where the down-scaling factor n is chosen according to the ratio $\sigma(T_{slow})/T_{fast}$ measured in a controlled environment. If the number of transitions in this second stream is too high an alarm is generated or, in a borderline condition, the compression rate can be increased to counterbalance the entropy loss.

5 Conclusions

The concept of stateless random bit generator has been introduced, defining a class of generators where both the noise source and the post-processing unit are assumed stateless. The assumption is satisfied introducing a reset state to cancel the memory of the previous generated bits. The noise source is reset after each bit generation while, for the post-processor, a reset after a word generation is required. The stateless property implies the independence of the external random numbers thus allowing to shift the verification of a minimum entropy limit after the post-processing.

In this approach, the noise source plays the role of entropy source and the relevant figure of merit is its entropy throughput. In other words, the statistic quality of the source itself is not a requirement, provided that a sufficient compression is implemented in the post-processing algorithm. The independence of the raw bits from the source allows to define a straightforward on-line entropy estimator to drive an adaptive compressing thus adapting the compression ratio to the actual entropy available from the source.

Implementation aspects have been also discussed pointing out that, while the post-processing reset is not an issue, the implementation of the source reset depends on the source architecture. Two examples have been provided to clarify the general principle.

The stateless hypothesis is sufficient to define an entropy estimator only if disturbances from an attacker aimed to force the source are not taken into account. As a consequence, in a real application, a further attack detection mechanism is required and an effective statistical test on the source before the digitizer has been discussed.

References

1. A. J. Menezes, P. C. Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, FL, 2001.
2. W. Schindler, *Efficient Online Tests for True Random Number Generators*, Proc. 3rd Workshop Cryptographic Hardware and Embedded Systems (CHES '01), LNCS (Springer-Verlag), vol. 2162, pp. 103-117, 2001.
3. W. Killmann and W. Schindler, *AIS 31: Functionality classes and evaluation methodology for true (physical) random number generators*, version 3.1, Bundesamt fur Sicherheit in der Informationstechnik (BSI), Bonn, 2001.
4. T. Stojanovski and L. Kocarev, *Chaos-Based Random Number Generators - Part I: Analysis*, IEEE Trans. Circuits and Systems I, vol. 48, no. 3, pp. 281-288, Mar. 2001.
5. T. Stojanovski, J. Pihl, and L. Kocarev, *Chaos-Based Random Number Generators - Part II: Practical Realization*, IEEE Trans. Circuits and Systems I, vol. 48, no. 3, pp. 382-385, Mar. 2001.
6. V. Bagini and M. Bucci, *A Design of Reliable True Random Number Generator for Cryptographic Applications*, Proc. 1st Workshop Cryptographic Hardware and Embedded Systems (CHES '99), LNCS (Springer-Verlag), vol. 1717, pp. 204-218, 1999.
7. E. Trichina, M. Bucci, D. De Seta, and R. Luzzi, *Supplementary Cryptographic Hardware for Smart Cards*, IEEE Micro, vol. 21, no. 6, pp. 26-35, Nov./Dec. 2001.
8. M. Dichtl and N. Janssen, *A High Quality Physical Random Number Generator*, Proc. Sophia Antipolis Forum Microelectronics (SAME 2000), pp. 48-53, 2000.
9. B. Jun and P. Kocher, *The Intel Random Number Generator*, Cryptographic Research Inc., white paper prepared for Intel Corp., Apr. 1999, http://www.cryptography.com/resources/white papers/IntelRNG.pdf.
10. C. S. Petrie and J. A. Connelly, *Modeling and Simulation of Oscillator-Based Random Number Generators*, Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS '96), vol. 4, pp. 324-327, 1996.
11. M. Bucci, L. Germani, R. Luzzi, A. Trifiletti, and M. Varanonuovo, *A High-Speed Oscillator-Based Truly Random Number Source for Cryptographic Applications*, IEEE Trans. Computers, vol. 52, no. 4, pp. 403-409, April 2003.
12. H. Bock, M. Bucci, and R. Luzzi, *An Offset-Compensated Oscillator-based Random Bit Source for Security Applications*, Proc. 6th Workshop Cryptographic Hardware and Embedded Systems (CHES '04), LNCS (Springer-Verlag), vol. 3156, pp. 268-281, 2004.
13. S. Mandal and S. Banerjee, *An Integrated CMOS Chaos Generator*, Proc. 1st Indian National Conf. Nonlinear Systems & Dynamics (NCNSD '03), pp. 313-316, Dec. 2003.

Successfully Attacking Masked AES Hardware Implementations*

Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology,
Inffeldgasse 16a, 8010 Graz, Austria
{Stefan.Mangard, Norbert.Pramstaller, Elisabeth.Oswald}@iaik.TUGraz.at
http://www.iaik.TUGraz.at/research/sca-lab

Abstract. During the last years, several masking schemes for AES have been proposed to secure hardware implementations against DPA attacks. In order to investigate the effectiveness of these countermeasures in practice, we have designed and manufactured an ASIC. The chip features an unmasked and two masked AES-128 encryption engines that can be attacked independently.

In addition to conventional DPA attacks on the output of registers, we have also mounted attacks on the output of logic gates. Based on simulations and physical measurements we show that the unmasked and masked implementations leak side-channel information due to glitches at the output of logic gates. It turns out that masking the AES S-Boxes does not prevent DPA attacks, if glitches occur in the circuit.

Keywords: AES, ASIC, DPA, Masking, Power Analysis.

1 Introduction

Power analysis attacks pose a serious threat to implementations of cryptographic algorithms. This is why there has been a lot of research during the last years to develop countermeasures. In particular, there have been quite some efforts to find methods to protect implementations of the Advanced Encryption Standard (AES) [10] against differential power analysis (DPA) attacks [7].

A commonly used approach to protect implementations of AES against DPA attacks is to randomize all intermediate results that occur during the computation of the algorithm. Usually, this is done by adding a random value to the intermediate results. This approach is called masking. The first article describing a masking scheme for AES was published by Akkar *et al.* in 2001 [2].

During the last years, several alternative masking schemes have been proposed (see [3], [6], [12], [16], and [17]). These publications focus on alternative methods to mask the AES S-Box. All other operations of AES are linear and

* This work has been supported by the Austrian Science Fund (FWF Project Number P16110) and by the European Commission under the Sixth Framework Programme (Project SCARD, Contract Number IST-2002-507270).

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 157–171, 2005.

hence they are easy to mask. Most of the articles that have been published on masking so far, are mainly theoretical. The security or insecurity of the different masking schemes has primarily been analyzed by assuming certain power consumption characteristics of the hardware that is used to implement the schemes.

The current article is different. We have designed and manufactured a chip that features an unmasked version and two masked versions of an AES-128 encryption engine. For the masked versions we have used the approach presented by Oswald *et al.* [12] and the original approach of Akkar *et al.* [2]. We have restricted our implementation to these two masking schemes for the following reasons.

The approach presented in [3] is very similar to the one of Oswald *et al.* Both schemes are provably secure in theory and therefore we have implemented only one of them. The masking scheme proposed in [17] has not been considered because it has been shown in [1] that this scheme does not prevent standard first-order DPA attacks. The approach of Golić and Tymen [6] seems not to be suitable for hardware implementations due to its big area requirements. The approach of Trichina and Korkishko [16] is based on masking at the gate level. However, we only wanted to compare implementations of masking schemes that are applied at the algorithm level.

Our chip implementing the unmasked AES processor and the two masked versions has been designed using a 0.25 μm CMOS technology. In order to perform DPA attacks on this chip, we have built a dedicated printed circuit board (PCB) that provides easy access to the power supply of the chip.

In this article, we present and compare the results of two types of DPA attacks. The first type of DPA attacks was targeted at intermediate results of AES that are stored in registers in our AES implementations. Attacks on registers of an unmasked AES implementation have also been analyzed by Örs *et al.* in [11]. Like the attacks of Örs *et al.*, also our attacks on the unmasked implementation have been successful. As expected, the cipher key of the masked versions could not be revealed by this type of attack. The second type of DPA attacks we have performed, was targeted at intermediate results that occur only at the output of logic gates. This is an important type of attack because in typical AES hardware implementations not all intermediate results that are suitable for a DPA attack are stored in registers.

In this article, we present successful DPA attacks of this type on the unmasked as well as on the masked AES implementations of our chip. Masking does not prevent this kind of DPA attacks because of glitches that occur in the masked S-Boxes of our chip. Glitches are switching operations of logic gates that are caused by timing properties of gates and by interconnection delays.

The fact that glitches lead to a side-channel leakage of masked gates has already been shown in [8] based on SPICE simulations. Also Suzuki *et al.* [15] have recently discussed the effect of glitches on the DPA-resistance of masked circuits. The current article shows that it is actually possible to exploit the side-channel leakage of masked AES implementations in practice.

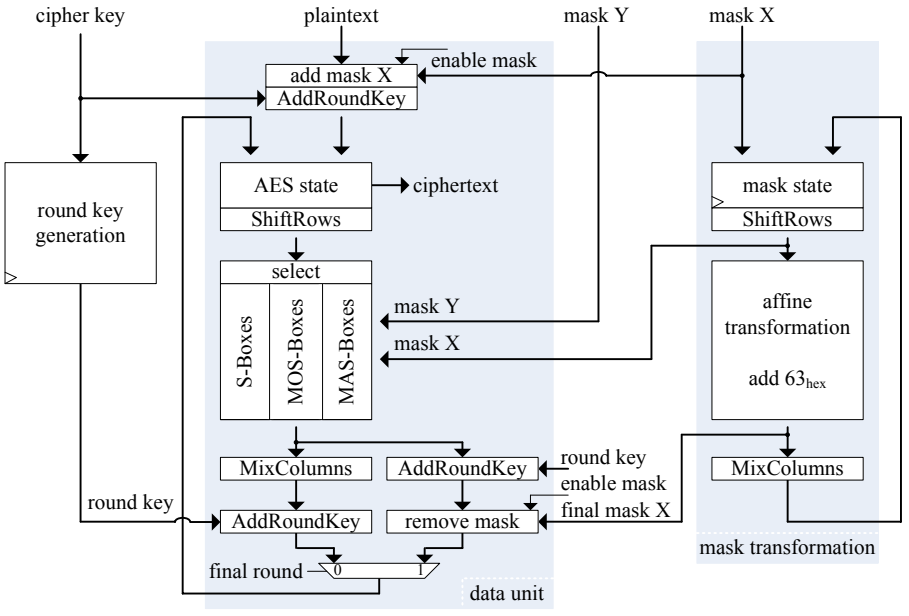


Fig. 1. Architecture of the AES chip

It is important to point out that it is not necessary to perform higher-order DPA attacks (see [9] and [18]) in order to exploit this side-channel leakage. All attacks presented in this article are first-order DPA attacks.

The remainder of this article is organized as follows: Section 2 describes the architecture of our AES chip. Results of DPA attacks that have been targeted at the output of registers are discussed in Section 3. Section 4 presents the results of the attacks on logic gates. This section provides an extensive discussion of the measurement results and it also analyzes glitches based on simulations. A summary of the results of the attacks on logic gates is presented in Section 5. Future research topics and a conclusion are stated in Section 6.

2 Architecture of the AES Chip

The architecture of our AES chip is schematically depicted in Figure 1. Based on this architecture, AES-128 encryptions can be performed in three different modes. In the first mode, an unmasked encryption is computed. The second mode performs a masked encryption based on the masking scheme proposed by Oswald *et al.*, and the third mode encrypts plaintexts based on the masking scheme proposed by Akkar *et al.*

During the design of the chip, special attention has been paid to ensure that only those parts of the chip are active that are actually needed for the selected mode—all other parts are completely disabled.

The main components of the architecture of our chip are the round-key generation, the data unit, and the mask transformation (see Figure 1). The round-key generation calculates the round keys as specified in [10]. The data unit implements all round transformations: AddRoundKey, ShiftRows, SubBytes, and MixColumns. The S-Boxes that are needed for the SubBytes transformation have been implemented separately for the unmasked mode and for the two masked modes. We refer to the masked S-Boxes as MOS-Boxes (masked as proposed in [12]) and MAS-Boxes (masked as proposed in [2]). Our architecture is based on a 32-bit datapath and therefore, four S-Boxes, four MOS-Boxes, and four MAS-Boxes are present in the design.

The mask transformation is the third main component of the architecture. It computes how the input mask (mask X) is altered by the linear transformations of the AES algorithm. Transformed mask values are required as input for the MAS-Boxes and the MOS-Boxes, as well as for the mask removal in the final round of an encryption. The multiplicative mask Y is only required for the MAS-Boxes.

3 DPA Attacks on Registers

In our architecture, the register labelled "AES state" in Figure 1 stores the AES state [10] after each round of an AES-128 encryption. If masking is enabled, this register stores the corresponding masked AES state, *i.e.* the sum of the unmasked AES state and the mask stored in the register labelled "mask state".

For the DPA attacks on this register, we assume that the attacker knows the ciphertext, *i.e.* she/he knows the content of the register after the final round of AES has been computed. In the final round, no MixColumns transformation is performed. Hence, it is possible to calculate one byte of the AES state of round nine based on one byte of the ciphertext and one byte of round key ten. We have exploited this property to successfully mount a DPA attack on the unmasked implementation of AES.

We have revealed round key number ten by attacking one byte of this key after the other. The DPA attack was done by formulating hypotheses about the number of transitions that occur at the output of the register "AES state" at the moment of time when the ciphertext is stored. The correlation between the hypotheses and the power consumption of the chip was measured using Pearson's correlation coefficient. The cipher key of the unmasked implementation was found based on $120,000$ measurements.

After the successful DPA attack on the unmasked implementation, we have performed the same attack on the masked ones. However, using the same hypotheses as in the unmasked case, it was not possible to reveal the cipher key of the masked implementations. Not even an attack based on one millon measurements was successful. This is actually an expected result. The hypotheses of the attacker do not correlate with the power consumption because the content of "AES state" register is masked. Table 1 shows a summary of our attacks on the "AES state" register.

Table 1. Summary of the attacks on the register storing the (masked) AES state

Attacked AES implementation	Number of needed measurements
S-Box	120,000
MOS-Box	not possible with 1,000,000
MAS-Box	not possible with 1,000,000

4 DPA Attacks on Logic Gates

In hardware implementations of cryptographic algorithms, many intermediate results that can be used for DPA attacks are usually not stored in registers. Our implementations for example do not store the output of the S-Box operations. We only store the AES state after each round.

In this section, we discuss DPA attacks on intermediate results that occur at the output of logic gates. Attacks of this kind cannot be conducted as easily as attacks on registers. The reason for this is that the transitions occurring at the output of logic gates are very hard to predict for an attacker. Registers switch their output only once per clock cycle. This transition of the output value leads to the power consumption that is attacked. Logic gates in CMOS circuits however, switch their output potentially several times per clock cycle—there occur glitches. This is a consequence of the timing properties of logic gates and the interconnection delays. Information about glitches in CMOS circuits can for example be found in [14].

We discuss the challenges of performing DPA attacks on logic gates based on our unmasked AES implementation in Section 4.1. DPA attacks on the masked implementations of AES are subsequently presented in Sections 4.2 and 4.3.

4.1 Attacks on the Unmasked Implementation

The output of the S-Box operation in the first round is an ideally suited target for a DPA attack on logic gates. This intermediate result is not directly stored in a register and it can be calculated based on one byte of plaintext and one byte of the cipher key. All attacks we discuss in this section are targeted at the logic gates computing this intermediate result. However, before discussing the attacks on the actual chip, we analyze and attack the power consumption characteristics of an unmasked S-Box based on simulations.

Attacking an Unmasked S-Box Based on Simulations. The S-Boxes used in our architecture have been implemented as proposed by Wolkerstorfer *et al.* in [19]. A block diagram of this implementation is shown in Figure 2. In order to analyze the power consumption of S-Box 1 of our unmasked implementation, we have performed simulations based on a back-annotated netlist of this S-Box.

Simulations of this kind can be used to determine the number of transitions that occur at the nodes of the S-Box circuit upon a change of the S-Box input. Figure 3 for example shows how the output bits of the S-Box change, if the input switches from 10_{hex} to FF_{hex}. This transition at the input leads to many

transitions at the output during a time frame of more than 2 ns. As it can be seen in Figure 3, many glitches occur in our S-Box implementation.

In addition to the transitions at the output of the S-Box, there also occur many transitions at the internal nodes. In order to assess the overall power consumption of the combinational circuit implementing the S-Box, we have performed simulations for all possible input transitions ($2^8 * 2^8 = 2^{16}$ simulations). During each simulation, we have counted the number of transitions that occur at the nodes of the S-Box circuit. This counting was done based on an in-house tool that has been developed to analyze the switching activity of nodes in combinational circuits. Using the output of this tool, we have calculated the average number of transitions that occur for each of the 256 possible S-Box outputs.

Figure 4 shows the result of these simulations. The upper plot shows the average number of transitions occurring in the S-Box for each output value. The capacitive load of the wires in the S-Box do not differ significantly and hence, this transition count can be used as estimation for the actual power consumption of the S-Box. The lower plot in Figure 4 shows the Hamming weight of the output values. In many DPA attacks that have been published, hypotheses about the Hamming weight of an intermediate result have been used to perform an attack. Figure 4 however, indicates that the power consumption of our S-Box implementation is unrelated to the Hamming weight of the output value of the S-Box. The correlation between the two curves shown in Figure 4 is 0.035. Therefore, DPA attacks that are based on the Hamming weight model may not be successful. In order to verify this statement, we have performed DPA attacks based on our simulations.

For these attacks, we have first generated 100,000 random plaintexts and we have randomly chosen a cipher key. Subsequently, we have determined the number of transitions that occur in the attacked S-Box during the encryption of the plaintexts. This number of transitions was used as estimation for the power consumption of the S-Box.

This estimated power consumption was attacked by predicting the Hamming weights of the intermediate results $i_1 \dots i_{12}$ (see Figure 2) and the Hamming weight of the S-Box output. However, none of these 13 DPA attacks was successful, *i.e.* the correct key did not lead to the highest correlation.

Subsequently, we have also performed attacks based on predicting each individual bit of the 4-bit intermediate results $i_1 \dots i_{12}$ and of the 8-bit S-Box output. Most of these 56 DPA attacks also failed. However, there were some intermediate results that lead to successful DPA attacks. For example, a DPA attack based on bit 2 of i_8 revealed the correct key based on 250,000 measurements.

The value of this bit seems to match the power consumption of the S-Box quite well. However, we consider this property to be highly specific to our implementation and therefore we do not provide a detailed discussion about which bits lead to a successful attack and which did not. The transitions that occur in the circuit implementing the S-Box depend on many factors. The main factor is the HDL description of the S-Box. However, the transitions occurring in the S-Box also depend on the used cell library, the placement, the routing and of

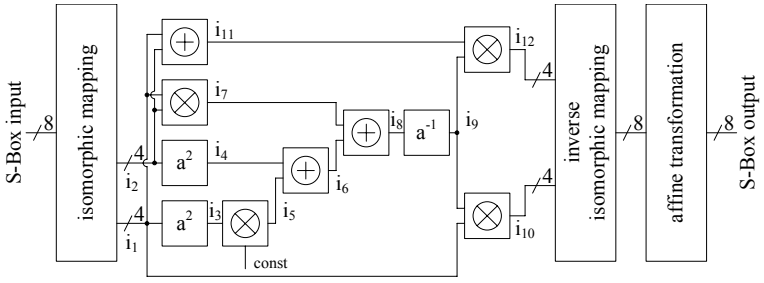


Fig. 2. Architecture of the unmasked S-Box implementation

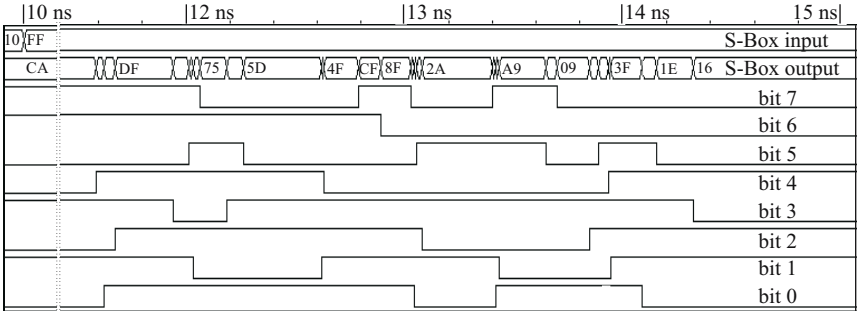


Fig. 3. The transitions that occur at the output of the unmasked S-Box, if the input changes from 10_{hex} to FF_{hex}

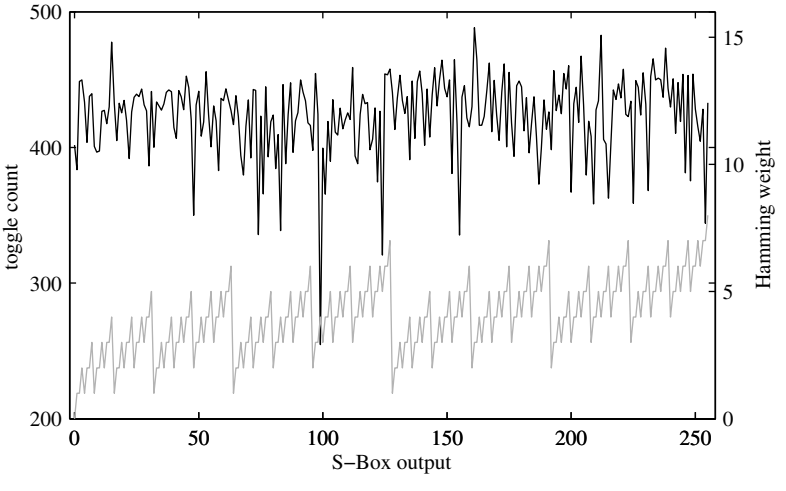


Fig. 4. The average number of transitions that occur in the unmasked S-Box for the 256 possible output values (upper plot); the lower plot shows the Hamming weight of the output values

course on the way the synthesizer maps the HDL description of the S-Box to the cell library.

In our case, the power consumption characteristic of the S-Box shown in Figure 4 is correlated to bit 2 of i_8. Yet, there is no guarantee that this property is maintained if a different HDL description, synthesizer, placement tool, or cell library is used. In fact, it may turn out that in a different implementation, another bit of the intermediate results or even of the S-Box output are correlated to the power consumption of the S-Box.

The overall conclusion of our simulations is that DPA attacks based on simple power models, like the Hamming weight, work only for very few intermediate bits of our S-Box implementation. DPA attacks are only possible, if the power consumption values that are predicted by the attacker match the actual power consumption of the S-Box at least to some degree. In the following paragraphs, we empirically verify these results by performing the same attacks on the actual chip.

Attacking an Unmasked S-Box on the Actual Chip. Using the setup described in Appendix A, we have encrypted one million random plaintexts with the unmasked AES implementation on our chip. During each encryption, the power consumption was recorded with a digital oscilloscope.

Based on these one million power traces, we have performed the same attacks as in the simulation. This means that we have mounted attacks based on the bits and the Hamming weights of the intermediate results $i_1 \dots i_{12}$ and of the S-Box output. The target of all our attacks was S-Box 1 in the first AES round.

We have measured a high correlation between at least one key hypothesis and the power consumption of the chip in all attacks. However, in almost all attacks it was not the correct key hypothesis that lead to the peak in the correlation trace.

Figure 5, for example, shows the result of a DPA attack based on one million measurements that was mounted on the least significant bit of the output of S-Box 1. The black trace shows the correlation we have measured for the correct key hypothesis, while the gray traces show the correlation for all other hypotheses. Some of the wrong key hypotheses lead to significant peaks. These peaks occur in the correct clock cycle, *i.e.* they occur in the clock cycle when the attacked S-Box operation is performed.

An attacker observing this result, can learn the moment of time when the S-Box operation is performed. However, the attacked byte of the cipher key is not revealed directly. This holds true for almost all attacks we have performed. In these attacks, it usually took not more than $250,000$ power measurements to determine in which clock cycle the S-Box operation is performed. However, in general the correct key could not be revealed—not even based on one million measurements.

The correct key could only be revealed by very few attacks. Like in the simulation, the attack on bit 2 of i_8 lead to the best results. $140,000$ measurements were needed in order to successfully perform a DPA attack based on bit 2 of i_8.

In addition to the attacks on bits and Hamming weights, we have also performed a DPA attack using a more advanced power model of the S-Box. In fact,

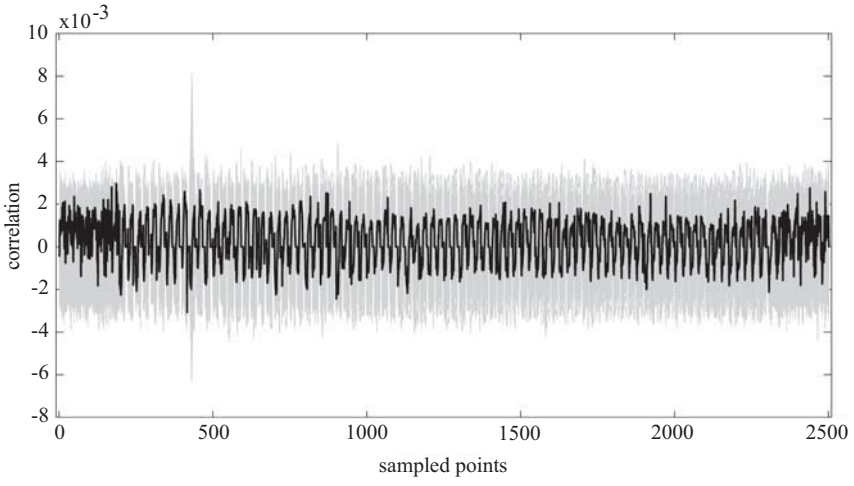


Fig. 5. The correlation for the correct key hypothesis (black) and the correlation for the wrong key hypotheses (gray) calculated based on one million measurements

we have used the average transition count that is shown in Figure 4 as power model for our attack. This means that instead of predicting a bit of the S-Box output, we were predicting the number of transitions that occur in the S-Box. This approach is to some degree comparable to the template attacks described in [4].

The DPA attack we have performed based on predicting the number of transitions turned out to be very powerful. Only 25,000 measurements were needed in order to determine the attacked byte of the cipher key. This result confirms that it is legitimate to use the transition count as a model for the power consumption in the context of DPA attacks.

All in all, the results of the DPA attacks on the unmasked implementation have confirmed the results of our simulations. The big majority of DPA attacks using simple power models were not successful. Only those attacks using a power model that at least to a certain degree matches the actual power consumption of the S-Box, have been successful. The better the used power model was, the less measurements were needed for the attack. Our results have been confirmed by DPA attacks on all four unmasked S-Boxes of our chip.

4.2 Attacks on the Implementation of the Scheme by Oswald et al.

After the attacks on the unmasked S-Boxes of the chip, we have performed attacks on the implementation of the masked S-Boxes that are based on the approach of Oswald *et al.* (the MOS-Boxes).

Like in the unmasked case, we have first performed some simulations based on the back-annotated netlist of a MOS-Box. However, we have not performed simulated DPA attacks for the MOS-Box. Essentially, we have only derived a

power model of the MOS-Box based on our simulations. This power model was created by counting the number of transitions occurring in the MOS-Box during the encryption of 100,000 random plaintexts. For each of these encryptions, a randomly generated mask was used.

Nevertheless, it turned out that the power consumption of the MOS-Box depends on the data input of the MOS-Box. The 256 possible data inputs lead to different numbers of transitions in the MOS-Box. In fact, there were significant differences and hence, our MOS-Box implementation is leaking side-channel information.

This can be explained by glitches that occur in the MOS-Box. In [12], Oswald *et al.* prove that all intermediate results that occur in their masking scheme are independent of the plaintexts. However, this proof is done at the algorithm level. At this level, all the additional intermediate results that occur in an actual CMOS implementation due to glitches are not considered.

In order to verify that the side-channel leakage is indeed caused by glitches in our MOS-Box implementation, we have additionally performed a functional simulation of the MOS-Box circuit. For this simulation, we have used the same back-annotated netlist as in the previous simulations. However, we have ignored all the timing information and hence, no glitches occurred in the MOS-Box during the functional simulation. As expected, the number of transitions that occurred in the MOS-Box during the functional simulation did not depend on the input of the MOS-Box. During this simulation, only intermediate results occurred that have been proven to be independent of the data input of the MOS-Box.

However, unfortunately the timing characteristics of a circuit cannot be ignored in practice. The DPA attacks we have performed on the MOS-Box implementations on our chip confirm that the power consumption of the implementation with glitches actually leaks side-channel information. Like in the unmasked case, we have used one million measurements to perform the DPA attacks on our chip. The attacks we have performed first, were based on predicting individual bits and the Hamming weight of the output of MOS-Box 1 during round one. The predictions were only based on the plaintexts—the masks are unknown to the attacker.

Like in the attacks on the unmasked S-Box, it was not possible to determine byte one of the cipher key based on the attacks on the S-Box output—not even with one million measurements. However, in all attacks it was again possible to determine in which clock cycle the attacked MOS-Box operation is performed. Roughly 250,000 measurements were needed to obtain this information.

In order to prove that it is possible to successfully attack the MOS-Box implementation with glitches, we have performed a DPA attack using the power model we had previously derived from the simulation with glitches based on the back-annotated netlist. Using this power model, it was possible to successfully attack the MOS-Box implementation based on 30,000 measurements. Comparable results were also achieved when we targeted the remaining three MOS-Box implementations on our chip. Hence, DPA attacks on the MOS-Boxes of our chip can be performed successfully, if a reasonable power model is used.

4.3 Attacks on the Implementation of the Scheme by Akkar et al.

The implementation of the masked S-Boxes that are based on the approach of Akkar *et al.* (the MAS-Boxes) have been attacked in the same way as the MOS-Boxes in the previous subsection. This means that we have first derived a power model of a MAS-Box based on simulating its back-annotated netlist. Like before, this was done by counting the number of transitions occurring in the MAS-Box during 100,000 masked encryptions of random plaintexts.

The number of transitions in the MAS-Box depends on the data input—just like in the case of the unmasked S-Box and the MOS-Box. It is important to point out that the observed dependency was not only caused by the zero-value problem [6] of the scheme of Akkar *et al.* Also non-zero data inputs lead to significantly different transition counts. These differences can again be explained by the glitches that occur in the circuit.

In order to verify that also our MAS-Box implementation can be attacked successfully in practice, we have measured the power consumption of our chip during one million masked encryptions. Using these power measurements, we have first performed attacks based on predicting the individual bits and the Hamming weight of the output of MAS-Box 1 in round one. The masks were again considered to be unknown to the attacker.

The attacks on the output of MAS-Box 1 have not been successful based on one million measurements. Yet, it was again possible to determine in which clock cycle the attacked MAS-Box operation was performed. Compared to the attacks on the other S-Box implementations, however, significantly more measurements were needed to obtain this information. It took 900,000 measurements.

An intuitive argument for this big difference is that our MAS-Box implementation is significantly bigger than the S-Box or the MOS-Box implementation. Furthermore, roughly half of the operations in the MAS-Box operate on masks only. For an attacker, this part of the MAS-Box acts like a big noise engine. No glitches leading to a data-dependent power consumption can occur in this part of the MAS-Box. A data-dependent power consumption can only be caused by glitches in operations that involve the masked data.

Nevertheless, we have been able to successfully perform DPA attacks on the MAS-Boxes of our chip. Using the power model derived from the simulation of the MAS-Box, we have successfully attacked all four MAS-Boxes on our chip. For these attacks, 130,000 measurements were needed.

5 Summary of the DPA Attacks on Logic Gates

In the previous section, we have discussed different DPA attacks on the unmasked and on the two masked AES implementations on our chip. The targets of these attacks were the S-Box operations in the first round of AES.

The main result of the attacks is that all three AES implementations leak side-channel information. CMOS implementations of the masking schemes proposed in [12] and [2] leak side-channel information due to glitches. We have analyzed this fact based on simulations of back-annotated netlists of all S-Box

Table 2. Summary of the DPA attacks on the different S-Box implementations

S-Box Implementation	Number of measurements needed to determine clock cycle	Number of measurements needed to determine key (using power model)
Unmasked S-Box	220,000	25,000
MOS-Box	250,000	30,000
MAS-Box	900,000	130,000

implementations. These simulations have shown that the number of transitions that occur in the S-Boxes depends on the S-Box input. Even in the masked cases, this dependency has been observed.

In addition to the simulations, we have performed DPA attacks on an actual chip. It has turned out that the attacks on the unmasked and the masked implementations lead to similar results. DPA attacks using simple power models, such as the Hamming weight or the value of a bit, were in general not successful. However, these attacks revealed in which clock cycle the attacked S-Box operation is performed. The number of measurements that were needed to obtain this information from the different AES implementations is shown in column two of Table 2.

All AES implementations have been successfully attacked using power models that have been derived based on simulations. The number of measurements that were needed to perform these attacks are shown in column three of Table 2. The attacks obviously pose a serious threat to unmasked as well as masked CMOS implementations of AES S-Boxes.

Designers of AES hardware implementations also need to be aware of the fact that their design might be susceptible to DPA attacks using simple power models. In our experiments, an attack on bit 2 of i_8 of the unmasked S-Box was successful. Actually, there is no guarantee that the power consumption of a masked AES hardware implementation is in general uncorrelated to similar hypotheses of an attacker. Depending on the implementation, it might also happen that the power consumption of a masked AES hardware implementation is correlated to the Hamming weight of the S-Box output.

6 Future Work and Conclusion

In this article, we have shown that it is possible to mount successful first-order DPA attacks on masked ASIC implementations of AES. The attacks we have presented are based on power models that have been derived from simulations of back-annotated netlists.

However, an attacker usually does not have easy access to the back-annotated netlist of a product. This is why we are currently closely analyzing the characteristics of the side-channel leakage that is caused by glitches. Our goal is to determine whether or not there exists a general power model that can be used to attack masked AES S-Boxes. In this context, we also plan to analyze in de-

tail why our implementation of the MOS-Boxes is not more secure than our implementation of the MAS-Boxes.

Although, these questions remain unanswered at this time, our experiments clearly show that masked hardware implementations are not as secure in practice as one might have expected. We have shown that there is a side-channel leakage of masked CMOS implementations due to glitches. We have observed this side-channel leakage in simulations based on back-annotated netlists as well as in power measurements of an ASIC implementation.

The conclusion of this article is that it is crucial for the DPA resistance of a design to think about glitches when masking schemes are implemented. Glitches should either be completely avoided [13] or the used masking scheme needs to be adapted in a way that it also works in the presence of glitches [5].

Acknowledgements

The analyzed chip with the unmasked and the two masked AES-128 encryption engines has been designed and implemented in cooperation with the Integrated Systems Laboratory at the Swiss Federal Institute of Technology Zurich. We would like to thank Frank K. Gürkaynak and Simon Häne for their generous support.

References

1. Mehdi-Laurent Akkar, Régis Bevan, and Louis Goubin. Two Power Analysis Attacks against One-Mask Methods. In *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 332–347. Springer, 2004.
2. Mehdi-Laurent Akkar and Christophe Giraud. An Implementation of DES and AES, Secure against Some Attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001.
3. Johannes Blömer, Jorge Guajardo, and Volker Krummel. Provably Secure Masking of AES. In *Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers*, volume 3357 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2005.
4. Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2535 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2003.
5. Wieland Fischer and Berndt M. Gammel. Secure Masking in the Presence of Glitches. In *Cryptographic Hardware and Embedded Systems – CHES 2005, 7th International Workshop, Edinburgh, Scotland, August 29 - September 1, 2005, Proceedings*, Lecture Notes in Computer Science. Springer, 2005.
6. Jovan D. Golić and Christophe Tymen. Multiplicative Masking and Power Analysis of AES. In *Cryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2535 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2003.

7. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
8. Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.
9. Thomas S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In *Cryptographic Hardware and Embedded Systems – CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000.
10. National Institute of Standards and Technology (NIST). FIPS-197: Advanced Encryption Standard, November 2001. Available online at `http://www.itl.nist.gov/fipspubs/`.
11. Siddika Berna Örs, Frank K. Gürkaynak, Elisabeth Oswald, and Bart Preneel. Power-Analysis Attack on an ASIC AES Implementation. In *Proccedings International Conference on Information Technology - ITCC 2004, Las Vegas, USA, Proceedings*, 2004.
12. Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A Side-Channel Analysis Resistant Description of the AES S-box. In *Fast Software Encryption, 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Proceedings*, volume 3557 of *Lecture Notes in Computer Science*, Springer, 2005.
13. Thomas Popp and Stefan Mangard. Masked Dual-Rail Pre-Charge Logic: DPA-Resistance without Routing Constraints. In *Cryptographic Hardware and Embedded Systems – CHES 2005, 7th International Workshop, Edinburgh, Scotland, August 29 - September 1, 2005, Proceedings*, Lecture Notes in Computer Science. Springer, 2005.
14. Jan M. Rabaey. *Digital Integrated Circuits*. Prentice Hall, 1996. ISBN 0-13-178609-1.
15. Daisuke Suzuki, Minoru Saeki, and Tetsuya Ichikawa. Random Switching Logic: A Countermeasure against DPA based on Transition Probability. Cryptology ePrint Archive (`http://eprint.iacr.org/`), Report 2004/346, 2004.
16. Elena Trichina and Tymur Korkishko. Small Size, Low Power, Side Channel-Immune AES Coprocessor: Design and Synthesis Results. In *Proceedings of the Fourth Conference on the Advanced Encryption Standard (AES)*, 2004.
17. Elena Trichina, Domenico De Seta, and Lucia Germani. Simplified Adaptive Multiplicative Masking for AES. In *Cryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2535 of *Lecture Notes in Computer Science*, pages 187–197. Springer, 2003.
18. Jason Waddle and David Wagner. Towards Efficient Second-Order Power Analysis. In *Cryptographic Hardware and Embedded Systems – CHES 2004, 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004, Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2004.
19. Johannes Wolkerstorfer, Elisabeth Oswald, and Mario Lamberger. An ASIC implementation of the AES SBoxes. In *Topics in Cryptology - CT-RSA 2002, The Cryptographer's Track at the RSA Conference 2002, San Jose, CA, USA, February 18-22, 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 67–78. Springer, 2002.

A Measurement Setup

A dedicated printed circuit board has been developed for mounting the DPA attacks on our chip (see Figure 6). We use an FPGA as interface between a standard PC and the chip. The communication between the PC and the FPGA is performed via an optically decoupled parallel interface.

Measurements are performed as follows. First, the input data of the chip is loaded into the FPGA via the parallel port. Then, the FPGA loads the data into the chip and starts an unmasked or masked AES encryption. The chip triggers a digital oscilloscope, which records the power consumption of the chip during the encryption.

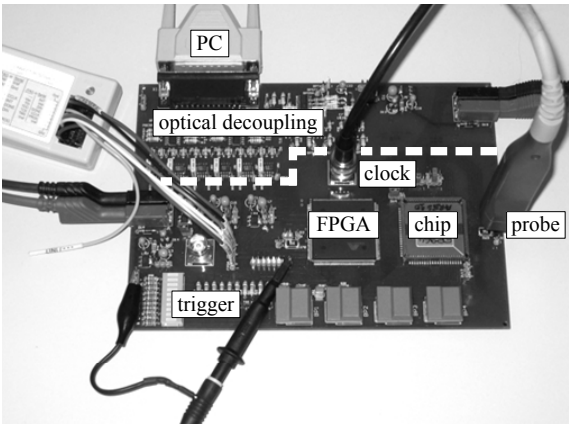


Fig. 6. Measurement setup for performing DPA attacks on the AES chip

Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints*

Thomas Popp and Stefan Mangard

Institute for Applied Information Processing and Communications (IAIK),
TU Graz, Inffeldgasse 16a, A-8010 Graz, Austria
{Thomas.Popp, Stefan.Mangard}@iaik.at

Abstract. During the last years, several logic styles that counteract side-channel attacks have been proposed. They all have in common that their level of resistance heavily depends on implementation constraints that are costly to satisfy. For example, the capacitive load of complementary wires in an integrated circuit may need to be balanced. This article describes a novel side-channel analysis resistant logic style called MDPL that completely avoids such constraints. It is a masked and dual-rail pre-charge logic style and can be implemented using common CMOS standard cell libraries. This makes MDPL perfectly suitable for semi-custom designs.

Keywords: Side-Channel Analysis, DPA, Hardware Countermeasures, MDPL, Masking Logic, Dual-Rail Pre-Charge Logic.

1 Introduction

During the last years, many logic styles that counteract side-channel analysis (SCA) attacks have been proposed. The big advantage of counteracting SCA attacks at the logic level is that this approach treats the problem right where it arises. If the basic building blocks, *i.e.* the logic cells, are resistant against SCA attacks, a designer can build a digital circuit with an arbitrary functionality and it will also be resistant against SCA attacks. Having SCA-resistant cells means that hardware as well as software designers do not need to care about SCA attacks any more. This greatly simplifies the design flow of a cryptographic device. Only the designers of the logic cells themselves need to be aware of SCA attacks.

An asynchronous logic style that makes devices more resistant against SCA attacks has for example been presented in [13]. However, it has been shown in [5] that this logic style has some weaknesses.

So far, the most promising logic styles to make devices resistant against SCA attacks are dual-rail pre-charge (DRP) logic styles that consume an equal amount of power for every transition of a node in a circuit. The most relevant logic styles

* This work has been supported by the European Commission under the Sixth Framework Programme (Project SCARD, Contract Number IST-2002-507270).

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 172–186, 2005.

of this kind are SABL [18,19], WDDL [20], and Dual-Spacer DRP [16]. In these DRP logic styles, the signals are represented by two complementary wires. The constant power consumption is achieved by guaranteeing that in every clock cycle one of these two wires is charged and discharged again. Which one of the two wires performs this charge and discharge operation depends on the logical value that the wires represent.

Obviously, a constant power consumption can only be achieved, if the complementary wires have the same capacitive load. Otherwise, the amount of energy needed per clock cycle would depend on which of the two nodes is switched and therefore would be correlated to the logical value. Unfortunately, the requirement to balance the capacitive load of two wires is hard to fulfill in a semi-custom design flow.

In a semi-custom design flow, so-called EDA tools place and route a digital circuit automatically. There exist only sub-optimal mechanisms to tailor the place and route operation such that the capacitive load of two wires is equal. Such a partial solution is for example parallel routing as introduced by Tiri and Verbauwhede [21]. However, integrating such mechanisms becomes more and more difficult for deep submicron process technologies where the transistor sizes and wiring widths continuously shrink. Hence, the capacitance of a wire more and more depends on the state of adjacent wires rather than on the length of the wire and its capacitance to VDD or GND. Therefore, it is very hard to guarantee a certain resistance against SCA attacks, if a DRP circuit is placed and routed automatically. Placing and routing a circuit manually, *i.e.* doing a full-custom design, significantly increases the design costs.

In this article, we present an approach that can easily be integrated in existing semi-custom design flows, *i.e.* there are no constraints for the place and route operation. The basic idea of our approach is to use masked cells to randomize the power consumption of cryptographic devices.

So far, masking has mainly been used at the software level. In [3] for example, Chari et al. analyze a secret sharing scheme where each intermediate bit of the original calculation is probabilistically split into k shares and every subset of $k-1$ shares is statistically independent of the original bit. A similar approach is described by Goubin and Patarin in [6] to mask the S-boxes of DES.

Masking on the gate level was considered for the first time in a patent [11] of Messerges et al. in 2001. However, the masked gates described in [11] are extremely big because they are built based on multiplexors. A different approach has been pursued later on by Gammel et al. in [7]. This patent shows how to mask complex circuits such as crypto cores, arithmetic-logic units, and complete micro controller systems. Other masked logic styles have been presented in [10] and in [22] where masked cells are built from standard CMOS cells.

The problem with all the mentioned masked logic styles is that glitches occur in these circuits. As shown in [9] and in [17], glitches in masked CMOS circuits reduce the SCA resistance significantly. Therefore, glitches must be considered when introducing an SCA countermeasure based on masking. In [17], a masked logic style called random switching logic (RSL) is presented. It avoids glitches

in the circuit. Yet, RSL needs a careful timing of enable signals. Furthermore, a new standard cell library must be compiled where all combinational gates have enable inputs.

In the current article, we present the so-called masked dual-rail pre-charge logic (MDPL) that uses masking at the gate level and that avoids glitches in the circuit by using a dual-rail pre-charge approach. There are no constraints for the place and route process. All MDPL cells can be built from standard CMOS cells that are commonly available in standard cell libraries.

Section 2 of this article explains the functionality and implementation details of MDPL cells. Furthermore, the overall architecture of MDPL circuits is introduced. In Section 3, experimental results of MDPL cells and circuits are presented. Conclusions are formulated in Section 4.

2 Masked Dual-Rail Pre-charge Logic

Currently the most promising DPA-resistant logic styles require the balancing of complementary wires. In the following sections, the masked dual-rail pre-charge logic (MDPL) is introduced, which completely avoids this constraint. Furthermore, MDPL can be implemented using a commonly available standard CMOS cell library. Hence, MDPL can be used easily in semi-custom design flows.

2.1 Masking CMOS Logic

Currently, the most widely used logic style to implement digital integrated circuits is CMOS [23]. A main characteristic of CMOS logic is that it requires primarily dynamic power while its static power consumption is almost zero. The dynamic power consumption is caused by transitions of logic signals that occur in the CMOS circuit. The type and the probability of signal transitions depend on the logical function of a circuit and on the processed data. As a result, the power consumption of a CMOS circuit depends on the data that is being processed and hence, DPA attacks as described in [8] are possible.

In a digital CMOS circuit, there are essentially four transitions that can occur at a node of the circuit at a given moment of time. These include the two degenerated events where the node's value stays the same. Table 1 lists all transitions that can occur at a node n storing the data value d_{t-1} at the time $t-1$ and the value d_t at the time t. The energy that is dissipated in order to perform the respective transition is denoted by $E_{00} \dots E_{11}$. Each of these transitions occurs with a certain probability, denoted by $p_{00} \dots p_{11}$.

In a DPA attack on a cryptographic device, several power traces or EM traces of the device are recorded while it operates on different input data. The traces are then split into two sets according to the value d_t at a certain time t. d_t is calculated based on the input data and a key hypothesis. Subsequently, the attacker determines the difference of the means (DM) of the two sets of traces. We refer to these means as $M_{d_t=0}$ and $M_{d_t=1}$.

Of course, at the time t not only the value of node n performs a transition. Several other nodes also switch their value at this moment of time. However, the

Table 1. Transitions of the value d of a node in a CMOS circuit

d_{t-1}	d_t	Energy	Probability
0	0	E_{00}	p_{00}
0	1	E_{01}	p_{01}
1	0	E_{10}	p_{10}
1	1	E_{11}	p_{11}

energy dissipation that is caused by these other nodes can be modeled as gaussian noise (see for example [12]). Therefore, the expected value of the difference of the means, $\mathcal{E}(DM_{d_t})$, can be calculated as shown in Equation 1.

$$\mathcal{E}(DM_{d_t}) = \mathcal{E}(M_{d_t=1}) - \mathcal{E}(M_{d_t=0}) = \frac{p_{11}E_{11} + p_{01}E_{01}}{p_{11} + p_{01}} - \frac{p_{00}E_{00} + p_{10}E_{10}}{p_{00} + p_{10}} \quad (1)$$

In case of standard CMOS logic ($E_{00} \approx E_{11} \ll E_{10} \neq E_{01}$), this difference is different from zero and can therefore be detected by an attacker. At all other moments of time, except for t, the partitioning of the traces according to d_t is meaningless. Consequently, the expected value for the difference of the means at these moments of time is zero. Furthermore, if a wrong key value is used to calculate d_t, the partitioning of the traces is again meaningless and also leads to an expected value of zero. As a result, the attacker in general gets a significant peak for a single key hypothesis, which is then the correct key.

The straightforward method to prevent an attacker from seeing such a peak is to use cells with the property that $E_{00} = E_{01} = E_{10} = E_{11}$. This is in fact the motivation for using dual-rail pre-charge logic styles such as SABL [18] or WDDL [20]. DRP logic styles have the property that transitions need the same amount of energy, if all pairs of complementary wires are perfectly balanced, *i.e.* have the same capacitive load. However, as already discussed, this requirement is very hard or even impossible to guarantee. This is the motivation for MDPL. MDPL is based on a completely different approach to prevent DPA attacks.

Resistance against DPA attacks at the gate level cannot only be achieved by consuming the same amount of energy for all transitions, but also by randomizing the logic signals in the circuit. This is the basic idea of MDPL. Instead of representing a logical value d by two complementary signals $\mathbf{d}$ and $\overline{\mathbf{d}}$, we represent d by $d_m = d \oplus m$, where $\oplus$ denotes the addition modulo 2 and m is a mask value. The mask is randomly generated and updated in every clock cycle.

Table 2 shows all transitions that can occur at a node n storing the masked value d_m when the time moves from $t-1$ to t. It also shows the required energy and the probability of each transition. The probability in line 1 is for example calculated as follows: $\frac{1}{4}p_{00} = p(m_{t-1} = 0) \cdot p(m_t = 0) \cdot p(d_{t-1} = 0) \cdot p(d_t = 0)$.

When a DPA attack is performed on the masked circuit, the power traces are split according to the value of d_t at time t. However, in the circuit now the masked value d_{m_t} is actually processed. In order to calculate $\mathcal{E}(M_{d_t=0})$, the odd lines are used while $\mathcal{E}(M_{d_t=1})$ is calculated based on the even lines. As shown in Equation 2, the two expected values are equal. Therefore, the expected value of

Table 2. Transitions of the value d_m of a masked node

Line no.	d_{t-1}	m_{t-1}	$d_{m_{t-1}}$	d_t	m_t	d_{m_t}	Energy	Probability
1	0	0	0	0	0	0	E_{00}	$\frac{1}{4}p_{00}$
2	0	0	0	1	1	0	E_{00}	$\frac{1}{4}p_{01}$
3	1	1	0	0	0	0	E_{00}	$\frac{1}{4}p_{10}$
4	1	1	0	1	1	0	E_{00}	$\frac{1}{4}p_{11}$
5	0	0	0	0	1	1	E_{01}	$\frac{1}{4}p_{00}$
6	0	0	0	1	0	1	E_{01}	$\frac{1}{4}p_{01}$
7	1	1	0	0	1	1	E_{01}	$\frac{1}{4}p_{10}$
8	1	1	0	1	0	1	E_{01}	$\frac{1}{4}p_{11}$
9	0	1	1	0	0	0	E_{10}	$\frac{1}{4}p_{00}$
10	0	1	1	1	1	0	E_{10}	$\frac{1}{4}p_{01}$
11	1	0	1	0	0	0	E_{10}	$\frac{1}{4}p_{10}$
12	1	0	1	1	1	0	E_{10}	$\frac{1}{4}p_{11}$
13	0	1	1	0	1	1	E_{11}	$\frac{1}{4}p_{00}$
14	0	1	1	1	0	1	E_{11}	$\frac{1}{4}p_{01}$
15	1	0	1	0	1	1	E_{11}	$\frac{1}{4}p_{10}$
16	1	0	1	1	0	1	E_{11}	$\frac{1}{4}p_{11}$

the difference of the means $\mathcal{E}(DM_{d_t})$ is zero and DPA attacks are not possible any more.

$$\mathcal{E}(M_{d_t=0}) = \mathcal{E}(M_{d_t=1}) = \frac{1}{4}\left(E_{00} + E_{01} + E_{10} + E_{11}\right) \tag{2}$$

Note that it is necessary to prevent glitches in a masked CMOS circuit in order to be resistant against DPA attacks. The fact that glitches lead to a leakage of side-channel information in masked CMOS circuits has been shown in [9] and in [17]. In CMOS circuits, the value of a node may switch several times before it reaches the correct value. The reason is that the input signals of the cell driving a node in general arrive at different moments of time. Glitches typically account for a significant amount of the dynamic power consumption of a CMOS circuit [15] and depend on the data that is processed. MDPL completely avoids glitches in the masked circuit as it is explained in the next section.

2.2 MDPL Cells

MDPL is a masked logic style that prevents glitches by using the DRP principle. Hence, for each signal d_m also the complementary signal $\overline{d_m}$ is present in the circuit. Every signal in an MDPL circuit is masked with the same mask m. The actual data value d of a node n in the circuit results from the signal value d_m that is physically present at the node and the mask m: $d = d_m \oplus m$. In the following, we show the implementation of an MDPL AND gate. All other combinational MDPL gates are based on this gate.

An MDPL AND gate takes six dual-rail inputs (a_m, $\overline{a_m}$, b_m, $\overline{b_m}$, m, $\overline{m}$) and produces two output values (q_m, $\overline{q_m}$). The truth table of an MDPL AND is

Table 3. Truth table of an MDPL AND gate

Line no.	a_m	b_m	m	q_m	$\overline{a_m}$	$\overline{b_m}$	$\overline{m}$	$\overline{q_m}$
1	0	0	0	0	1	1	1	1
2	0	0	1	0	1	1	0	1
3	0	1	0	0	1	0	1	1
4	0	1	1	1	1	0	0	0
5	1	0	0	0	0	1	1	1
6	1	0	1	1	0	1	0	0
7	1	1	0	1	0	0	1	0
8	1	1	1	1	0	0	0	0

Table 4. Truth table of an MDPL OR gate

Line no.	a_m	b_m	m	q_m	$\overline{a_m}$	$\overline{b_m}$	$\overline{m}$	$\overline{q_m}$
1	0	0	0	0	1	1	1	1
2	0	0	1	0	1	1	0	1
3	0	1	0	1	1	0	1	0
4	0	1	1	0	1	0	0	1
5	1	0	0	1	0	1	1	0
6	1	0	1	0	0	1	0	1
7	1	1	0	1	0	0	1	0
8	1	1	1	1	0	0	0	0

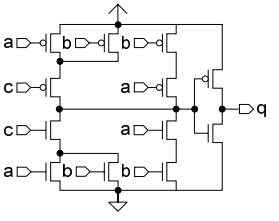


Fig. 1. Schematic of a CMOS majority gate

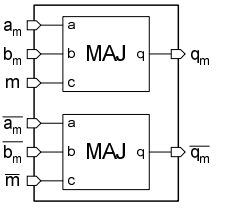


Fig. 2. Schematic of an MDPL AND gate

shown in Table 3. The outputs of the MDPL AND gate are calculated according to the following equations: $q_m = ((a_m \oplus m) \wedge (b_m \oplus m)) \oplus m$ and $\overline{q_m} = ((\overline{a_m} \oplus \overline{m}) \wedge (\overline{b_m} \oplus \overline{m})) \oplus \overline{m}$

In Table 3, it can be seen that q_m and $\overline{q_m}$ can be calculated by the so-called majority (MAJ) function. The output of this function is 1, if more inputs are 1 than 0. Otherwise, the output is 0: $q_m = MAJ(a_m, b_m, m)$ and $\overline{q_m} = MAJ(\overline{a_m}, \overline{b_m}, \overline{m})$. A majority gate is a commonly used gate and it is available in a typical CMOS standard cell library. The schematic of a CMOS majority gate is shown in Figure 1.

In an MDPL circuit, all signals are pre-charged to 0 before the next evaluation phase occurs. A so-called pre-charge wave is started from the MDPL D-flip-flops, similar to WDDL [20]. First, the outputs of the MDPL D-flip-flops are switched to 0. This causes the combinational MDPL cells directly connected to the outputs of the D-flip-flops to pre-charge. Then, the combinational gates in the next logic level are switched into the pre-charge phase and so on. Note that also the mask signals are pre-charged. In Table 3, it can be seen that the pre-charge wave propagates correctly through the MDPL AND gate (see line 1 and line 8, respectively). The output signals of the MDPL AND gate are pre-charged if all inputs are pre-charged. All combinational MDPL gates are implemented in that way. Therefore, in the pre-charge phase, the pre-charge wave can propagate through the whole combinational MDPL circuitry and all signals are pre-charged correctly.

A majority gate in a pre-charge circuit switches its output at most once per pre-charge phase and at most once per evaluation phase, *i.e.* there occur no glitches. In a pre-charge circuit, all signals perform monotonic transitions in the evaluation phase ($0 \rightarrow 1$ only) and in the pre-charge phase ($1 \rightarrow 0$ only), respectively. Furthermore, the majority function is a so-called monotonic increasing (positive) function. Monotonic transitions at the inputs of such a gate lead to an identically oriented transition at its output. Hence, a majority gate performs at most one ($0 \rightarrow 1$) during the evaluation phase and at most one ($1 \rightarrow 0$) during the pre-charge phase. Since an MDPL AND gate is built from majority gates, an MDPL AND gate will produce no glitches.

Figure 2 shows the schematic of an MDPL AND gate. As already discussed, it is constructed using two CMOS majority gates. An MDPL NAND can be built by swapping (=inverting) the complimentary wires of the output signal. An MDPL OR can be generated by swapping the complementary masking wires (see Table 4) and an MDPL NOR is built by swapping the wires of the mask and that of the output signal. Note that the swapping of complementary wires does not invalidate the considerations made for the MDPL AND gate concerning propagation of the pre-charge wave and glitches. Therefore, also the MDPL NAND, OR and NOR gates propagate the pre-charge wave correctly and produce no glitches.

Figure 3 shows the schematic of an MDPL XOR gate that is built using three MDPL NAND gates. Note that the connections for the mask signals have been omitted for the sake of clarity. Using two 3-input XOR's as an MDPL XOR (like the MAJ gate is used in the MDPL AND) would also lead to a functionally correct gate. However, it would not be free of glitches since an XOR is not a monotonic function. The use of the MDPL NAND gates in order to implement the MDPL XOR gate prevents glitches. Furthermore, also the pre-charge wave is propagated correctly. An MDPL XNOR is realized by swapping the complementary wires of the output signal.

The implementation of an MDPL D-flip-flop is shown in Figure 4. The MDPL XOR gate at the input is used to switch the mask m of the current clock cycle to the new mask m_n of the next clock cycle. Hence, the CMOS D-flip-flop stores a value at the positive clock edge that is already masked with the mask of the next

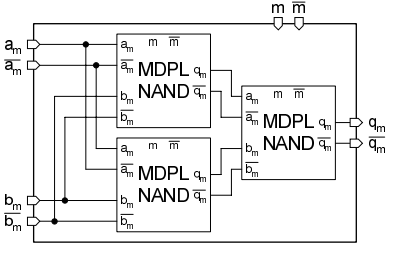


Fig. 3. Schematic of an MDPL XOR gate

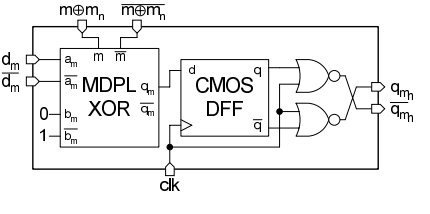


Fig. 4. Schematic of an MDPL D-flip-flop

clock cycle. Note that the fixed input signals of the MDPL XOR gate still allow the gate to pre-charge correctly if all other inputs are 0. The MDPL D-flip-flop must be supplied with the special signals $m \oplus m_n$ and $\overline{m \oplus m_n}$ in order to switch masks.

The two CMOS NOR gates at the output of the MDPL D-flip-flop are required to start the pre-charge wave when the clock signal clk is 1. During $clk = 0$, the circuit is in the evaluation phase and the MDPL D-flip-flop provides the differentially encoded data value at its output. In the MDPL D-flip-flop, there is a timing constraint that must be satisfied: When the positive edge of the clock signal clk arrives at the cell, the CMOS NOR gates must switch their outputs to the pre-charge level 0 before the CMOS D-flip-flop stores the new input value at its output. Otherwise, there may be glitches introduced in the circuit. However, this timing constraint is satisfied because NOR gates are faster than D-flip-flops. Additionally, the CMOS D-flip-flop and the CMOS NOR gates are leaf cells of the clock tree that is build during the design process. Therefore, the skew between the clock signals connected to the CMOS D-flip-flop and the NOR gates is minimal.

There is also an implementation of the MDPL D-flip-flop possible that does not have such a timing constraint. Yet, it is much bigger and requires a doubling of the clock frequency in order to keep the same data rate. Details can be found in Appendix A. The fixed inputs of the MDPL XOR gate that is used in the MDPL D-flip-flop allows an optimization of the flip-flop. This optimization is presented in Appendix B.

Table 5 summarizes the basic MDPL cells and their respective implementation with standard CMOS cells. The table also shows the area complexity of the MDPL cells when the $0.35\mu m$ standard cell library C35B3 of austriamicrosystems [1] is used for the implementation. The area requirements of the MDPL cells are furthermore compared to the area requirements of their standard CMOS counterparts. Note that for the MDPL D-flip-flop, the optimized implementation as introduced in Appendix B is considered. The size of an MDPL circuit compared to the size of a standard CMOS implementation depends on the used cell types.

Table 5. MDPL cells and their CMOS implementations

MDPL cell	CMOS implementation of MDPL cell	Area (gate equivalents) of MDPL cell	Area (gate equivalents) of std. CMOS cell	Ratio $\frac{MDPL}{CMOS}$
Inverter	Wire swapping	0	0.67	0
Buffer	2×Buffer	2	1	2
AND, OR (2-in)	2×MAJ (3-in)	4	1.67	2.4
NAND, NOR (2-in)	2×MAJ (3-in)	4	1	4
XOR (2-in)	6×MAJ (3-in)	12	2.33	5.1
XNOR (2-in)	6×MAJ (3-in)	12	2	6
D-Flip-Flop	2×AND, 2×OR (both 2-in) 2×MAJ (3-in), 1×D-FF	17.67	5	3.5

2.3 MDPL Circuits

The main features of MDPL circuits are that they can be based on a typical standard cell library and that there are no routing constraints concerning the balancing of complementary wires. Note that the dual-rail pre-charge property of MDPL is exclusively used to prevent glitches in the circuit. Hence, complementary wires are present, but they do not need to be balanced.

The design flow that is used to implement an MDPL circuit is almost the same as that one used when implementing a standard CMOS circuit. The only additional tool that is required is a converter that translates the synthesized CMOS netlist into an MDPL netlist. The MDPL cells in general cannot be used for synthesis since its masked and dual-rail attributes cannot be handled by typical state-of-the-art synthesizers. Furthermore, this would require the compilation of an MDPL synthesis cell library, which causes additional effort.

The converter replaces all CMOS cells by their MDPL versions as indicated in Table 5. It also adds the complementary wires, swaps a complementary wire pair if an inverter is present in the CMOS netlist and adds the mask nets. Dedicated single-rail nets like the clock net must be kept single-rail. We have built such a converter and have successfully translated and implemented an AES module. Details of this implementation can be found in Section 3.

The basic architecture of an MDPL circuit is shown in Figure 5. The combinational MDPL gates are supplied with the mask signals m and $\overline{m}$, the MDPL D-flip-flops must be supplied with the mask signals $m \oplus m_n$ and $\overline{m \oplus m_n}$. Note that also the mask signals are pre-charged in an MDPL circuit.

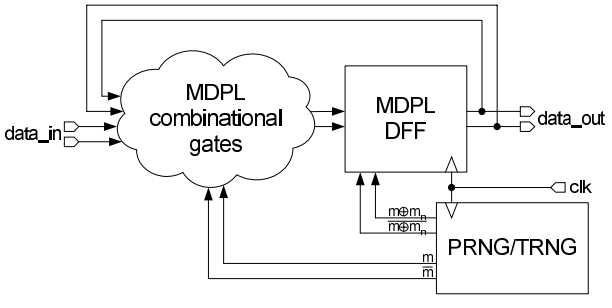


Fig. 5. Architecture of an MDPL circuit

The masks of the MDPL circuit are generated by a pseudo random-number generator (PRNG) that is seeded by a true random-number generator (TRNG) [2]. An MDPL circuit requires only one new masking bit per clock cycle.

In an MDPL circuit, all masks are dual-rail pre-charge signals. Therefore, it is not possible to perform SPA attacks an the masks. The complementary mask networks have approximately the same load. Hence, it is not possible to determine the masks based on one measurement. If the masks were not encoded

in this way, SPA attacks would be possible. The mask networks are very big and hence, it would be possible for an attacker to determine whether a mask is switched or not.

3 Experimental Results

In this section, we present the results of some practical investigations of MDPL circuits. We compare how the DPA resistance of the dual-rail pre-charge logic style WDDL [20] and of MDPL depend on the balancing of complementary wires. Furthermore, we show the results of an AES module implemented in MDPL. For our circuit implementations we have used the $0.35\mu m$ standard cell library C35B3 of austriamicrosystems [1].

First, we have studied the effects of unbalanced routing on the dual-rail pre-charge logic style WDDL and compared them to the behavior of MDPL under the same condition. As expected, the results show that MDPL is completely independent to unbalanced routing while it has a significant effect on WDDL.

Figure 6 shows a comparison of the difference-of-mean-energies (DME) of CMOS, WDDL and MDPL implementations of a NAND gate. The energy consumption of the NAND gates were simulated by Spice simulations (*spectre* from Cadence) for all possible input values and input transitions. These energies were then split into two groups according to the respective output value of the gate. In a last step, the difference of the means of the energies in both groups were calculated. The DME value represents the height of the DPA peak if the output signal of such a NAND gate is attacked.

In case of the CMOS NAND gate, the DME value and thus the hight of the DPA peak raises linearly with the load at the output of the gate. For the differential NAND gates implemented in WDDL and MDPL, the effect of an unbalanced differential output load is also shown in Figure 6. An unbalance of 5% means that the output q has a 5% lower capacitance than the output $\overline{q}$ of the cell. The figure clearly shows that the DPA resistance of WDDL depends significantly on the degree of balancing. MDPL is completely immune to an unbalanced differential output load – the three lines lie upon each other. The difference between the MDPL NAND 0% and the WDDL NAND 0% is caused by charge sharing effects and small imbalances in the MDPL NAND gate which is more complex than the WDDL NAND gate (more internal nodes).

In a second experiment, we have simulated the power traces of an AES module [4] while it encrypts different data blocks. The power traces were then used in a DPA attack that targeted the intermediate result of the first SubBytes operation after the initial AddRoundKey operation [14]. More precisely, one output bit of an 8-bit S-box was used as the selection bit in a standard DPA attack [8]. For the power simulation, the gate-level netlist of the AES module without parasitics was used. The power simulation was done for the 256 different input values of the data byte that corresponds to the key byte under attack.

In Figure 7, the result of the DPA attack on the AES module implemented in CMOS is shown. The correct key (43d) is clearly identifiable. Figure 8 shows

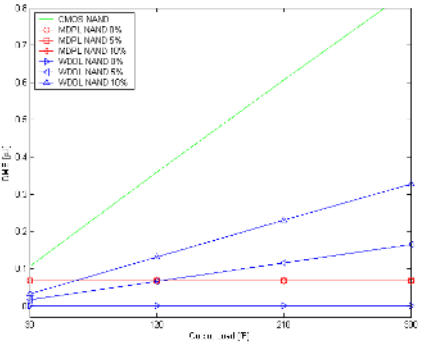


Fig. 6. Comparison of the DME of CMOS, WDDL and MDPL implementations of a NAND gate

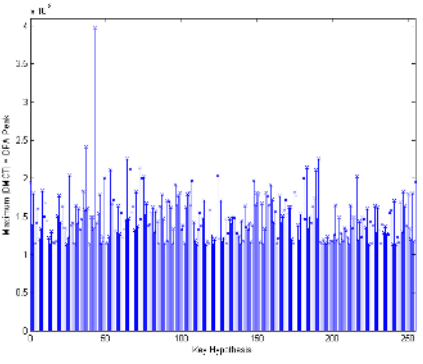


Fig. 7. DPA peaks for all key hypothesis for a DPA attack on an AES implemented in CMOS

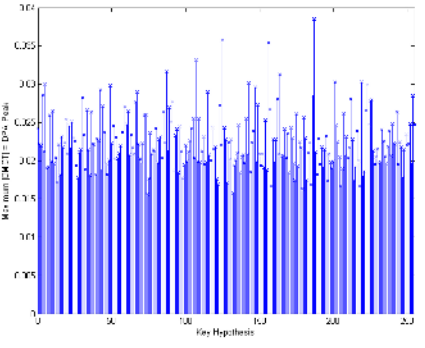


Fig. 8. DPA peaks for all key hypothesis for a DPA attack on an AES implemented in MDPL

Table 6. Comparison of AES implementations in CMOS and MDPL

	CMOS	MDPL	Ratio $\frac{MDPL}{CMOS}$
Area (gate equivalents)	3628	16465	4.54
Speed (MHz; worst-case speed corner)	16.91	9.82	0.58

the result of the DPA attack on the same AES implemented using MDPL. In this case, the correct key cannot be disclosed by the attack.

In Table 6, a comparison of the main properties of the CMOS and the MDPL implementation of the AES module is shown. The increase in the area is significant, yet the DPA resistance of the MDPL circuit does not depend on the balancing of complementary wires. Speed is halved because not only the logic signals but also the pre-charge wave needs to propagate through the MDPL circuit.

4 Conclusions

We presented the DPA-resistant logic style MDPL which has two major advantages: it can be implemented using commonly available standard cells and, most importantly, its security does not rely on balanced complementary wires. Experimental results show that MDPL is effective against DPA attacks and that an unbalanced load of complementary wires does not affect the DPA resistance of the MDPL cells. The trade-off is in increased area requirements and power consumption and in a reduced circuit speed.

References

1. austriamicrosystems. `http://www.austriamicrosystems.com`.
2. Holger Bock, Marco Bucci, and Raimondo Luzzi. An Offset-Compensated Oscillator-Based Random Bit Source for Security Applications. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004, Sixth International Workshop, Boston, USA, August 11-13, 2004, Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 268–281. Springer, 2004.
3. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
4. Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong Authentication for RFID Systems using the AES Algorithm. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004, Sixth International Workshop, Boston, USA, August 11-13, 2004, Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 357–370. Springer, 2004.
5. Jacques J. A. Fournier, Simon Moore, Huiyun Li, Robert D. Mullins, and George S. Taylor. Security Evaluation of Asynchronous Circuits. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 137–151. Springer, 2003.
6. Louis Goubin and Jacques Patarin. DES and Differential Power Analysis – The Duplication Method. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
7. Franz Klug, Oliver Kniffler, and Berndt Gammel. Rechenwerk, Verfahren zum Ausführen einer Operation mit einem verschlüsselten Operanden, Carry-Select-Addierer und Kryptographieprozessor. German Patent DE 10201449 C1, January 2002.
8. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

9. Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.
10. Renato Menicocci and Johan Pascal. Elaborazione Crittografica di Dati Digitali Mascherati. Italian Patent IT MI0020031375A, July 2003.
11. Thomas S. Messerges, Ezzy A. Dabbish, and Larry Puhl. Method and Apparatus for Preventing Information Leakage Attacks on a Microelectronic Assembly. US Patent 6,295,606, September 2001. Available online at `http://www.uspto.gov/`.
12. Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Examining Smart-Card Security under the Threat of Power Analysis Attacks. *IEEE Transactions on Computers*, 51(5):541–552, January 2002.
13. Simon Moore, Ross J. Anderson, Paul Cunningham, Robert D. Mullins, and George S. Taylor. Improving smart card security using self-timed circuits. In *Proceedings of the Eighth International Symposium on Asynchronous Circuits and Systems (ASYNC 02)*, pages 211–218. IEEE Computer Society, 2002.
14. National Institute of Standards and Technology (NIST). FIPS-197: Advanced Encryption Standard, November 2001. Available online at `http://www.itl.nist.gov/fipspubs/`.
15. Jan M. Rabaey. *Digital Integrated Circuits*. Prentice Hall, 1996. ISBN 013-1786091.
16. Danil Sokolov, Julian Murphy, Alex Bystrov, and Alex Yakovlev. Improving the Security of Dual-Rail Circuits. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 282–297. Springer, 2004.
17. Daisuke Suzuki, Minoru Saeki, and Tetsuya Ichikawa. Random Switching Logic: A Countermeasure against DPA based on Transition Probability. Cryptology ePrint Archive (`http://eprint.iacr.org/`), Report 2004/346, 2004.
18. Kris Tiri, Moonmoon Akmal, and Ingrid Verbauwhede. A Dynamic and Differential CMOS Logic with Signal Independent Power Consumption to Withstand Differential Power Analysis on Smart Cards. In *28th European Solid-State Circuits Conference (ESSCIRC 2002)*, 2002.
19. Kris Tiri and Ingrid Verbauwhede. Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 137–151. Springer, 2003.
20. Kris Tiri and Ingrid Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004), 16-20 February 2004, Paris, France*, pages 246–251. IEEE Computer Society, 2004.
21. Kris Tiri and Ingrid Verbauwhede. Place and Route for Secure Standard Cell Design. In *CARDIS 2004 - Sixth Smart Card Research and Advanced Application IFIP Conference, 23-26 August 2004, Toulouse, France*, 2004.
22. Elena Trichina. Combinational Logic Design for AES SubByte Transformation on Masked Data. Cryptology ePrint Archive (`http://eprint.iacr.org/`), Report 2003/236, 2003.
23. Neil H. E. Weste and Kamran Eshraghian. *Principles of CMOS VLSI Design - A Systems Perspective*. Addison-Wesley, 2nd edition, 1993. ISBN 0-201-53376-6.

A An MDPL D-Flip-Flop Without Timing Constraints

As mentioned in Section 2.2, the MDPL D-flip-flop can be designed in a way that it has no timing constraints that must be satisfied. In the original MDPL D-flip-flop implementation shown in Figure 4, the CMOS NOR gates must switch the outputs to pre-charge level before the CMOS D-flip-flop stores a new value at the positive clock edge. Otherwise, the MDPL D-flip-flop may emit glitches which potentially cause side-channel leakage.

In general, CMOS NOR gates switch significantly faster than CMOS D-flip-flops. The clock signal connected to the CMOS NOR gates and the CMOS D-flip-flop arrive nearly at the same time since these cells are leaf nodes of the clock tree. Therefore, the skew should be minimal. Consequently, the timing constraint should be satisfied if the clock tree is created correctly.

If it is necessary to avoid such a timing constraint at all, it is possible to use an MDPL D-flip-flop implementation as shown in Figure 9. This version uses four CMOS D-flip-flops. One column of the flip-flops stores the pre-charge signal 0, hence these two flip-flops need to be reset in the beginning. The other column stores the differential encoded data value. Note that for a correct power-on reset of these two flip-flops, the above one needs to be reset to 0 while the below one needs to be preset to 1.

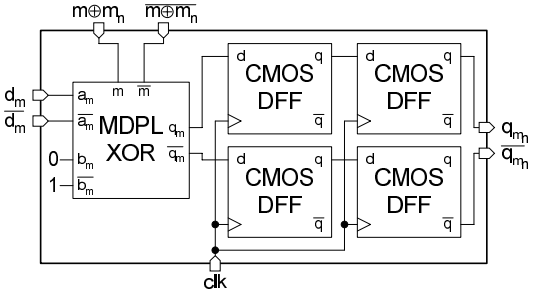


Fig. 9. Schematic of an MDPL D-flip-flop with no timing constraints

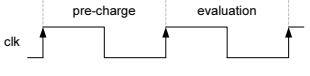


Fig. 10. Sequence of pre-charge and evaluation phases in an MDPL circuit using MDPL D-flip-flops without timing constraints

The sequence of the pre-charge and evaluation phases with respect to the clock signal when using the MDPL D-flip-flop without timing constraints is shown in Figure 10. A disadvantage is that the clock rate must be doubled in order to keep the data rate of the circuit constant. This increases the power consumption significantly.

B The MDPL D-Flip-Flop with an Optimized MDPL XOR

The schematic of the basic MDPL D-flip-flop introduced in Figure 4 can be optimized. The reason is that two inputs of the used MDPL XOR are fixed to 0 and 1, respectively. Those inputs are connected to two MDPL NAND gates inside the MDPL XOR. Each of these two MDPL NAND gates can be reduced to a CMOS AND and a CMOS OR gate as shown in Figure 11. This reduces the area requirements of the MDPL D-flip-flop (implemented using the $0.35\mu m$ standard cell library C35B3 of austriamicrosystems [1]) from 19 gate equivalents to 17.67 gate equivalents. AND and OR functions are both monotonic increasing functions, and so glitches cannot occur in the cell.

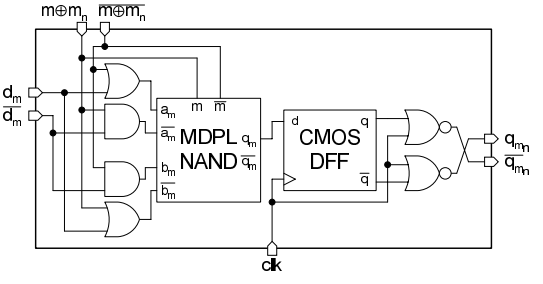


Fig. 11. Schematic of an MDPL D-flip-flop with optimized MDPL XOR

Masking at Gate Level in the Presence of Glitches

Wieland Fischer and Berndt M. Gammel

Infineon Technologies AG, St.-Martin-Straße 76,
D-81541 Munich, Germany
{Wieland.Fischer, Berndt.Gammel}@infineon.com

Abstract. It has recently been shown that logic circuits in the implementation of cryptographic algorithms, although protected by "secure" random masking schemes, leak side-channel information, which can be exploited in differential power attacks [14]. The leak is due to the fact that the mathematical models describing the gates neglected multiple switching of the outputs of the gates in a single clock cycle. This effect, however, is typical for CMOS circuits and known as *glitching*. Hence several currently known masking schemes are not secure in theory or practice. Solutions for DPA secure circuits based on logic styles which do not show glitches have several disadvantages in practice. In this paper, we refine the model for the power consumption of CMOS gates taking into account the side-channel of glitches. It is shown that for a general class of gate-level masking schemes a universal set of masked gates does not exist. However, there is a family of masked gates which is theoretically secure in the presence of glitches if certain practically controllable implementation constraints are imposed. This set of gates should be suitable for automated CMOS circuit synthesis.

Keywords: Cryptanalysis, side-channel attacks, power analysis, DPA, digital circuits, logic circuits, masking, random masking, masked logic circuits, glitches.

1 Introduction

Cryptanalysis based on side-channel information exploits the information leaked during the computation of an algorithm. Side-channel information can be contained in the characteristic power consumption, the timing, or the electromagnetic emanation of the device during the processing of secret information. Power analysis attacks exploit the fact that, in general, the instantaneous power consumption of a circuit depends on the data being processed by the circuit. The effect is prominent especially in the widely used CMOS design style. *Differential power analysis* (DPA), first introduced in [12], allows the attacker to exploit correlations between the observable instantaneous power consumption and intermediate results involving the secret. During the last years it has become more and more obvious that it is extremely difficult to protect a security device against

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 187–200, 2005.

DPA [1,2,3,4,5,6,8,9,13,14,15,16,18,21,22,23]. In the spirit of power analysis attacks Electromagnetic Emanation Analysis (EMA) extracts secret information from the electromagnetic radiation emited during the operation of the device [7].

The first class of *ad-hoc approaches* against power analysis attacks tries to reduce the signal-to-noise ratio of the side-channel leakage and finally to hide the usable information in the noise. Suggested methods are detached power supplies [20], the addition of power noise generators, or the application of a probabilistic disarrangement of the times at which the attacked intermediate results are processed. The latter can be achieved by inserting random delays or applying randomizations to the execution path. While such measures certainly increase the experimental and computational working load of the attacker they do not render the attack infeasible. In practice, typically several countermeasures are combined [5,13]. This can reduce the correlation down to a level that makes a DPA practically impossible. However, higher order differential attacks or the possibility of obtaining a spatial resolution of the power consumption and an increased signal-to-noise ratio by observing local electromagnetic emanations may again open a backdoor for professional attackers.

Circuit design approaches, the second class of countermeasures, aim at removing the root cause for side-channel leakage information. In standard CMOS style circuits the power consumption depends strongly on the the processed data. In some dynamic and differential logic styles, like Sense Amplifier Based Logic (SABL) [21], which is based on Differential Cascode Voltage Switching Logic (DCVLS), the power consumption can be made almost independent of the processed data. However, data independent power consumption requires a maximum activity factor and hence maximum power consumption. It is also essential that the load capacitances of the differential outputs are matched. Remaining asymmetries (parasitics, cross-coupling) make a DPA still possible. Disadvantages of this circuit style are the lack of standard cell libraries and tools, which leads to a full-custom design style. Area (power consumption) of a SABL circuit design are approximately 3.5 times (4.5 times) larger than for a corresponding CMOS design. As for all two-cycle schemes the performance is reduced by a factor of two. The Wave Dynamic Differential Logic style (WDDL) adopts the ideas of SABL. It implements the behavior of a dynamic and differential logic, but is based on standard CMOS cells [22]. Area and power consumption are approximately 3.5 times larger than for a CMOS design. The performance is two times smaller.

Masking approaches, the third class of measures, counteract DPA by randomizing intermediate results occurring during the execution of the cryptographic algorithm. The idea behind this approach is that the power consumption of operations on randomized data should not be correlated with the actual plain intermediate data [15]. Algorithmic countermeasures in the context of symmetric ciphers based on secret sharing schemes have been independently proposed by Goubin and Paterin [9] and Chari et al. [4]. A theory of securing a circuit at the gate level against side-channel attacks (focused on probing) was developed in [10].

Masking at algorithm level for asymmetric algorithms [6,17], as well as for symmetric algorithms, e.g., DES and AES [2,3], has been developed. Cryptographic algorithms often combine Boolean functions (like logical XOR or AND operations) and arithmetic functions (operations in fields with characteristic bigger than two). Masking operations for these two types of functions are referred to as *Boolean and arithmetic masking*, respectively. This poses the problem of a secure conversion between the two types of maskings in both directions [2].

It is appealing to apply the idea of randomizing intermediate results already on the level of logic gates. *Masking at gate level* leads to circuits where no wire carries a value which is correlated to an intermediate result of the algorithm. Clearly this approach is more generic than the algorithmic approach. Masking at gate level is independent of the specifically implemented algorithm. Once a secure masking scheme has been developed the generation of the masked circuit from the algorithm can be automated, and a computer program can convert the digital circuit of any cryptographic algorithm to a circuit of masked gates. This would also relieve the designers or implementers of cryptographic algorithms from the complex task of elaborating a specific solution against side-channel leakage for each new implementation variant or algorithm. Various generic masking schemes have been proposed. In [16] the multiplexor gate (MUX) used in the implementation of nonlinear operations, like S-boxes, is replaced by a masked MUX gate. In [11] the basic operations of an arithmetic-logic unit (ALU) are protected with one or more random masks at each masked gate. In [23] correction terms for the AND gate in the nonlinear components of the S-box of the AES are introduced. It has been shown that it is possible to break masking schemes that rely on one mask using advanced DPA methods [1].

The security analyses of masking schemes, conducted so far, were based on the implicit assumption that the input signals of any (masked) gate in a combinational CMOS circuit arrive at the same time. Recently it has been shown [14], that this assumption is not sufficient: the output of the gate possibly switches several times during one clock cycle. The transitions at the output of a gate, previous to the stable state right before the next clock edge is attained, are known as *glitches*. Glitches are a typical phenomenon in CMOS circuits and extensively discussed in the literature on VLSI design [19]. Because a glitch can cause a full swing transition at the output of the gate, just like the 'proper' transition to the final value, a glitch is *not* a negligible higher order effect. As made evident in [14] glitches do not just add a background noise due to uncorrelated switching activity – the dissipated energy of nonlinear masked gates is correlated to the processed values whenever the input values do not arrive simultaneously (forcing the output of the gate to toggle several times). Hence glitches can carry side-channel information and their effect must be included in the analysis of any secure masking scheme.

In the next section a model for the power consumption of CMOS gates is developed which takes into account the side-channel of glitches. Based on this model the notion of G-equivariance is introduced and it is shown that in the stated gate and energy models G-equivariance is a necessary condition for ran-

domized gates to prevent a differential power attack. We will show that in a class of gates which is preferred for implementation reasons, there exists no G-equivariant gate that can be used to realize a nonlinear logical function. However, for a model with weakened conditions an explicit construction of a universal set of semi-G-equivariant gates is given. The necessary constraints on gate design and signal routing should be realizable in practice using available design tools.

2 The Glitch Problem

In this section the glitch problem described in [14] is reformulated in a more theoretical and abstract way. First, the abstraction of the energy consumption of a single gate, which is the target of a DPA attack, is recapitulated. The most simple energy model which is commonly used is mentioned and a more general definition is given. Then the basic attack on such a gate with statistical means is described. The definitions of randomized gates (in the classical meaning) is given and it is shown how a DPA may still be successful if the more general energy model is applicable.

2.1 The General Power Consumption Model of a Gate

In a DPA the attacker tries to find a correlation between externally known (and guessed) data and internally processed signals. Since he will not be able to gain these internal signal data directly, he is obliged to use physical effects (the side-channels) which are again somehow correlated to the internal signals/data. One of these side-channels is the current consumption. Since we are interested in a protection of this side channel on gate level, out first step has to be the definition of a power consumption model of a gate: A gate g with n inputs and one output will be interpreted as a function $g\colon \mathbb{F}_2^n \to \mathbb{F}_2$. Our premise is that the power consumption during one clock cycle (in a synchronous design) only depends on the input at the time t_0 shortly before the clock edge (the old input) and after the clock edge (the new input), e.g., at t_1 shortly before the next clock edge. We do not consider dependencies on other signals in the surrounding circuit, like cross-coupling phenomena. The following definition of an energy function of a gate suggests itself:

Definition 1. *Let $g\colon \mathbb{F}_2^n \to \mathbb{F}_2$ be a gate. Denote the input at time t_0, at or shortly before the rising edge of a clock cycle, as $a = (a_1, \ldots, a_n) \in \mathbb{F}_2^n$ and the input at time t_1, at or shortly before the next rising edge, as $x = (x_1, \ldots, x_n) \in \mathbb{F}_2^n$. Then the energy consumption of the gate during this transition is given by the real number $E_g(a, x) \in \mathbb{R}$. Hence the* ***Energy function of the gate*** *g is defined to be the map*

$$\begin{aligned} E_g\colon \mathbb{F}_2^n \times \mathbb{F}_2^n &\longrightarrow \mathbb{R} \\ (a, x) &\longmapsto E_g(a, x). \end{aligned}$$

The energy function of a gate may be different for individual gates in a circuit, even if they are functionally equal. The reason is that the energy depends mainly on the individual capacitive load the gate has to drive.

The Simplistic Model: In a simplistic energy consumption model (coined on, e.g., CMOS logic style) one mainly identifies the power consumption of a gate with the energy needed to drive the output capacitance if the output toggles. The energy consumption of a gate is described only by its digital output behavior. Hence it is determined by the output values of g at times t_0 and t_1 and a fixed tuple $(E_{g,0\to0}, E_{g,0\to1}, E_{g,1\to0}, E_{g,1\to1}) \in \mathbb{R}^4$. If for example at time t_0 the output value of g is 1 and at time t_1 it is 0 then the energy for this clock cycle is $E_{g,1\to0}$. Hence, in this model the energy function of the gate g is given by: $E_g(a,x) := E_{g,g(a)\to g(x)}$.

Differential Power Analysis of This Model: Assume we have a cryptographic algorithm with some secret (key) implemented as a CMOS circuit. Further assume that there is a gate $g\colon \mathbb{F}_2^2 \to \mathbb{F}_2$ within this circuit. The input values of g at time t_0 are $(a,b) \in \mathbb{F}_2^2$ and later at t_1 are $(x,y) \in \mathbb{F}_2^2$. Since an attacker will survey the energy consumption of this gate during several runs of the algorithm with different messages, these values may be seen as random variables $a,b,x,y\colon \Omega \longrightarrow \mathbb{F}_2$ on some probability space (Ω, Σ, P). This gives rise to the following concatenation

$$\mathcal{E}_g := E_g \circ (a,b,x,y)\colon \Omega \longrightarrow \mathbb{F}_2^2 \times \mathbb{F}_2^2 \longrightarrow \mathbb{R}$$

With the knowledge of the secret key (or parts of it), which is called the *hypothesis*, one may construct a partition of Ω into two disjoint measurable[1] subsets A and B such that $\Omega = A \cup B$, with the property:

$$\mathbb{E}(\mathcal{E}_g|A) \neq \mathbb{E}(\mathcal{E}_g|B),$$

while this construction done with a wrong hypothesis yields: $\mathbb{E}(\mathcal{E}_g|A) = \mathbb{E}(\mathcal{E}_g|B)$. One classical example, cf. [12], is the partition of Ω into

$$A = \{\omega \in \Omega : g(x(\omega), y(\omega)) = 1\} \quad \text{and} \quad B = \{\omega \in \Omega : g(x(\omega), y(\omega)) = 0\}$$

With the simplistic energy model we obtain

$$\mathbb{E}(\mathcal{E}_g|A) = \alpha E_{g,0\to1} + \bar{\alpha} E_{g,1\to1} \quad \text{and} \quad \mathbb{E}(\mathcal{E}_g|B) = \beta E_{g,0\to0} + \bar{\beta} E_{g,1\to0}$$

for $\alpha := P(\{\omega \in \Omega : g(a(\omega), b(\omega)) = 0\}|A)$, $\bar{\alpha} := 1 - \alpha$ as well as $\beta := P(\{\omega \in \Omega : g(a(\omega), b(\omega)) = 0\}|B)$, $\bar{\beta} := 1 - \beta$. In general these two expectation values are not equal (if the hypothesis was correct). This gives rise to the classical DPA.

Remark 1. It is clear that, if $E_{g,0\to0} = E_{g,0\to1} = E_{g,1\to0} = E_{g,1\to1}$, then indeed the two expectation values are always equal, independent of whether the hypothesis was right or wrong. Hence no DPA is possible. In general terms, if the energy function

$$E_g\colon \mathbb{F}_2^2 \times \mathbb{F}_2^2 \longrightarrow \mathbb{F}_2 \text{ is constant,} \qquad (*)$$

[1] Although very important, we are not going to specify the measurability any further.

then the gate does not leak information and a DPA on the gate is not possible. In practice these conditions are only met if a logic style is chosen for the implementation which guarantees the constancy of the energy function itself. This corresponds to the second class of countermeasures (see discussion in the introduction). If this is not desirable, one still may be able to use the additional conditions given by a, b, x, y: We only have to fulfill the condition

$$\mathcal{E}_g \colon \Omega \longrightarrow \mathbb{R} \text{ is constant,} \qquad (**)$$

which is weaker than the former one. However, if we want to find gates for general purposes we have to fulfill this condition (**) for any a, b, x, y. Unfortunately, this is equivalent to condition (*).

Randomized Logic as Countermeasure: In fact, there still may be the possibility that $\mathcal{E}_g$ is constant in some conditional sense, even if E_g is not. This can be in the class of randomized (masked) gates: Randomizing a signal (in our context) means substituting one digital signal $a \in \mathbb{F}_2$ by a number of signals $a_1, \ldots, a_n \in \mathbb{F}_2$ with $a = a_1 + \cdots + a_n$ so that there exists no correlation between a and each summand a_i. For practical reasons, we will be restricted ourselves to the case $n = 2$.

One philosophy is to interpret the randomized signal (a_1, a_2) as the pair of the masked signal $a_m = a_1$ and its mask $m_a = a_2$ (cf. notation in e.g. [14]). But this is just terminology and we will only follow it in our discussion for presenting the randomized gates as in [14]. However this point of view has an impact on the philosophy of randomized (or masked) gates: Since the signals a, b are now split up in two portions, one has to substitute the old gate $g \colon \mathbb{F}_2^2 \to \mathbb{F}_2$ by a new gate.

The first choice would be $g' \colon \mathbb{F}_2^2 \times \mathbb{F}_2^2 \to \mathbb{F}_2$, such that $g(a, b) = g'(a_1, a_2, b_1, b_2)$, with $a = a_1 + a_2$ and $b = b_1 + b_2$. But since the output should also be randomized, one possibility would be $g' \colon \mathbb{F}_2^2 \times \mathbb{F}_2^2 \times \mathbb{F}_2 \to \mathbb{F}_2$, with the property $g(a, b) = g'(a_m, m_a, b_m, m_b, m_c) + m_c$ and $a = a_m + m_a$, $b = b_m + m_b$. This property defines g' uniquely. In the following g' is called **the masked lifting** of g, since the output of g' is the output of $c := g(a, b)$ masked with m_c. Fig. 1 shows an example for a circuit using masked liftings of gates (left hand sketch) and a realization of a lifting of an AND gate [8,23] (right hand sketch).

Another choice is using two gates $(g_1, g_2) \colon \mathbb{F}_2^2 \times \mathbb{F}_2^2 \to \mathbb{F}_2^2$ with the property $g(a, b) = g_1(a_1, a_2, b_1, b_2) + g_2(a_1, a_2, b_1, b_2)$. Here g_1 and g_2 are not uniquely defined by this equation. But, of course, if g_1 is given then g_2 will be fixed. The pair (g_1, g_2) is called **a randomized lifting** of g.

Using the simplistic energy consumption model from above

$$E_{g'}((\tilde{a}, \tilde{b}, m_c), (\tilde{x}, \tilde{y}, m_z)) = E_{g', g'(\tilde{a}, \tilde{b}, m_c) \to g'(\tilde{x}, \tilde{y}, m_z)},$$

where $(\tilde{a}, \tilde{b}, m_c) \in \mathbb{F}_2^2 \times \mathbb{F}_2^2 \times \mathbb{F}_2$ is the input at time t_0, $(\tilde{x}, \tilde{y}, m_z) \in \mathbb{F}_2^2 \times \mathbb{F}_2^2 \times \mathbb{F}_2$ the input at time t_1 with the abbreviations $\tilde{a} = (a_m, m_a)$, etc., the energy consumption $E_g((a, b), (x, y))$ has to be substituted by the (conditional resp. a, b, x, y) expectation value

$$\mathbb{E}(E_{g'}((\tilde{a}, \tilde{b}, m_c), (\tilde{x}, \tilde{y}, m_z))),$$

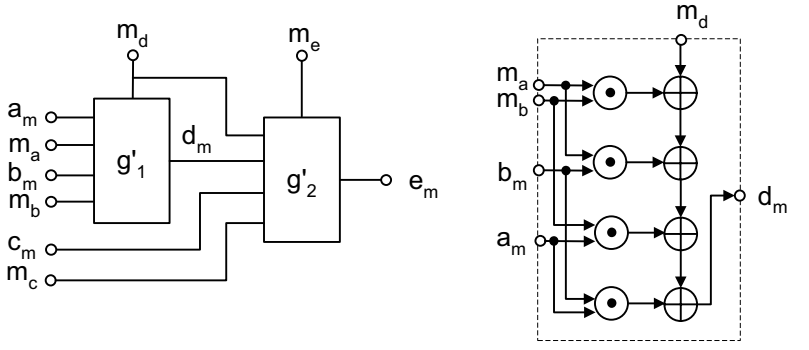


Fig. 1. Example for a combinational circuit consisting of two masked liftings of gates. The figure to the right shows a masked AND gate as given in [23,8]. The $\odot$ and $\oplus$ symbols denote a logical AND and XOR gate, respectively.

where $\tilde{a} = (a_m, m_a)$, $\tilde{b} = (b_m, m_b)$, m_c, $\tilde{x} = (x_m, m_x)$, $\tilde{y} = (y_m, m_y)$, m_z are interpreted as random variables with $a = a_m + m_a$, etc. An attacker will not be able to know the exact (microscopic) signals $(\tilde{a}, \tilde{b}, m_c), (\tilde{x}, \tilde{y}, m_z)$, but rather only the (macroscopic) signals a, b, x, y.

Indeed, if $m_c, m_z \colon \Omega \to \mathbb{F}_2$ are uniformly distributed random variables, independent to the random variables $g(a, b), g(x, y)$ then the masked lifting g' of a gate g does not leak information: $\mathbb{E}(E_{g'}((\tilde{a}, \tilde{b}, m_c), (\tilde{x}, \tilde{y}, m_z)))$ is independent of a, b, x, y. This was stated in [23,8].

2.2 Power Consumption of a Gate in the Presence of Glitches

As realized in [14], in realistic CMOS implementations the different signals x_m, m_x, y_m, m_y, m_z may not arrive at the gate g' the same time. In the example circuit of Fig. 1 signal d_m may arrive with a delay at the input of gate g'_2 compared to signals m_d, c_m, m_c due to the gate delay imposed by g'_1. Furthermore, all input signals of gate g'_2 have in general different additional delay contributions due to the propagation delay caused by wire capacitances. These delays depend on the route of the signal and are fixed when the circuit is laid out.

Consider the example that the signals arrive in the distinct order $y_m \longrightarrow m_y \longrightarrow m_z \longrightarrow x_m \longrightarrow m_x$. In this case the output value of the gate changes not only once during the clock cycle but five times leading to the consecutive output transitions $c_1 := g(a_m, m_a, b_m, m_b, m_c) \longrightarrow c_2 := g(a_m, m_a, y_m, m_b, m_c) \longrightarrow c_3 := g(a_m, m_a, y_m, m_y, m_c) \longrightarrow c_4 := g(a_m, m_a, y_m, m_y, m_z) \longrightarrow c_5 := g(x_m, m_a, y_m, m_y, m_z) \longrightarrow c_6 := g(x_m, m_x, y_m, m_y, m_z)$. Therefore the energy consumption will be given by the sum $E_{g',c_1 \to c_2} + E_{g',c_2 \to c_3} + E_{g',c_3 \to c_4} + E_{g',c_4 \to c_5} + E_{g',c_5 \to c_6}$.

Hence a new power model is required such that $E_{g'}((\tilde{a}, \tilde{b}, m_c), (\tilde{x}, \tilde{y}, m_z))$ is given by the sum from above. Unfortunately, with this model, it was shown in [14] that $\mathbb{E}(E_{g'}((\tilde{a}, \tilde{b}, m_c), (\tilde{x}, \tilde{y}, m_z)))$ is not independent of a, b, x, y any more, opening a door for DPA.

One can conceive an even worse situation: if a well-equipped attacker is able to measure the different partial energies of the five transitions the constraints for a gate to be resistant against DPA are even more difficult to fulfill.

3 Abstraction and Analysis of the Glitch Problem

The last section has motivated the following strategy and definitions. First the abstract model of the gates together with their energy model will be defined. Then conditions imposed on the gates will be formulated, which ensure that a differential power attack cannot be mounted.

3.1 The Power Consumption Model in the Presence of Glitches

Because of glitches the gate $g\colon \mathbb{F}_2^n \to \mathbb{F}_2$ can switch up to n times within one clock period and because every transition of the output consumes an amount of power $E_{g,0\to0}, E_{g,0\to1}, E_{g,1\to0}$, or $E_{g,1\to1}$, the notion of the energy function has to be generalized. Also, since the four values from above may strongly depend on the individual gate and its position in a circuit, it makes sense to treat these values as indeterminates. Therefore, it is natural to value the energy function not in $\mathbb{R}$ but rather in the 4-dimensional vector space $V := \mathbb{R}\cdot e_{00} \oplus \mathbb{R}\cdot e_{01} \oplus \mathbb{R}\cdot e_{10} \oplus \mathbb{R}\cdot e_{11}$. For a certain implementation one may concatenate the energy function with the evaluation function $ev\colon V \to \mathbb{R}, (x_{00}, x_{01}, x_{10}, x_{11}) \mapsto \sum_{i,j} x_{ij} E_{g,i\to j}$. We first give the formal definition of our power consumption model:

Definition 2. *Let $g\colon \mathbb{F}_2^n \to \mathbb{F}_2$ be a gate with n inputs and one output. The **(partial) energy functions** of the gate g are given by:*

$$\begin{aligned} E_{g,i}\colon \mathbb{F}_2^n \times \mathbb{F}_2^n \times \mathrm{Map}(\{1,\dots,n\},\{1,\dots,n,\}) &\longrightarrow V \\ (\tilde{a},\tilde{x},\varphi) &\longmapsto e_{g(\tilde{b}_i),g(\tilde{b}_{i+1})}, \end{aligned}$$

for $i = 1,\dots,n$. Here $\tilde{b}_i = (b_{i1},\dots,b_{in}) \in \mathbb{F}_2^n$ is defined by $\tilde{b}_1 := \tilde{a}$ and

$$b_{(i+1)j} := \begin{cases} x_j, & \text{if} \quad \varphi(j) = i, \\ b_{ij}, & \text{else}, \end{cases}$$

in particular we have $\tilde{x} = \tilde{b}_{n+1}$.

The map φ describes the order of the incoming (changing) input signals. If $\varphi(j) = 1$ then signal j changes first and the signal to change next is the one with $\varphi(j) = 2$, and so on. Since two or more signals may arrive at the same time the map φ does not need to be a permutation. The old energy description of a gate can be obtained by fixing $\varphi \equiv 1$ (or any constant between 1 and n).

The n+1 tuples $\tilde{b}_i$ are the different input value during the clock cycle at $n+1$ possible different moments in time: $\tilde{b}_0 = \tilde{a}$ is the input value at t_0 and $\tilde{b}_{n+1} = \tilde{x}$ is the final input value at t_1. $\tilde{b}_1, \dots, \tilde{b}_n$ are the consecutive input signals in between.

We see the order of the signals φ as a constant associated for each single gate within a circuit. This order is fixed at the design time of the circuit and is given by parameters such as the depth of logic tree at each input of the gate and the precise route of the signals.

Remark 2. This definition of the energy consumption of a gate reflects the assumption (idealization) that the implementation of a gate does not have any usable internal side-channels. This means, for instance, that the gate itself is inherently glitch free and there is only one signal change at the output if one input signal changes. Also the output delay must not depend on the input value. It can safely be assumed that these prerequisites can be realized in practice with relatively high accuracy if a masked logic cell is crafted for use in a library.

3.2 Randomized Signal Pairs

Randomization in our context means splitting up a signal a into a pair (a_1, a_2) of signals such that $a = a_1 + a_2$ and the individual bits a_1 and a_2 are unknown, i.e., random and uniformly distributed. Since we are, first of all, interested in the randomized realization of (macroscopic) 2-1 gates like AND, OR, etc. we will restrict ourselves to gates with two (macroscopic) inputs, a, b, which means four actual inputs a_1, a_2, b_1, b_2 for a randomized lifting of the gate. Fig. 2 depicts a combinational circuit where two normal gates $g_1(a, b)$ and $g_2(d, c)$ have been replaced by two randomized liftings of gates $(g_{11}(a_1, a_2, b_1, b_2), g_{12}(a_1, a_2, b_1, b_2))$ and $(g_{21}(d_1, d_2, c_1, c_2),\ g_{22}(d_1, d_2, c_1, c_2))$, which have been selected to sustain the old functionality of the circuit. The following two definitions describe this situation.

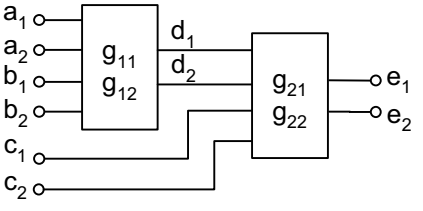


Fig. 2. Example of a combinational circuit using randomized liftings of gates

Definition 3. *A **randomized signal pair** is a 4-tuple (a_1, a_2, b_1, b_2) of random variables $a_1, a_2, b_1, b_2 \colon \Omega \longrightarrow \mathbb{F}_2$ such that the following properties are fulfilled:*

1. *a_1, a_2, b_1, b_2 are uniformly distributed, i.e., $P(a_1 = 0) = P(a_2 = 0) = P(b_1 = 0) = P(b_2 = 0) = 1/2$.*
2. *The random variables a_i and b_j are independent for $1 \leq i, j \leq 2$.*

Remark 3. The pairs a_1, a_2 and b_1, b_2 are in general not independent!

Definition 4. *If we define* $a := a_1 + a_2 \colon \Omega \to \mathbb{F}_2$ *and* $b := b_1 + b_2 \colon \Omega \to \mathbb{F}_2$ *then* $(a, b) \colon \Omega \to \mathbb{F}_2^2$ *is a pair of random variables and we say* $(\tilde{a}, \tilde{b})$ ***is a lifting of the pair*** (a, b)*, where* $\tilde{a} := (a_1, a_2)$ *and* $\tilde{b} := (b_1, b_2)$*.*

In the following we do *not* try to find a single gate g' which exactly lifts the functionality of a specific gate g. Instead we follow the general strategy to search for a *universal set* of lifted gates. That is a family of gates, which have the property that the energy of the macroscopic transition $(a, b) \to (x, y)$ does not leak information and which can be combined to realize any logical function.

In the next section we give a precise formulation of the necessary conditions for lifted gates which do not leak information also in the presence of glitches.

3.3 The Criterion of Glitch-Equivariance of Gates

The notion of glitch-equivariant gates will be introduced. Gates satisfying this criterion do not leak information about the macroscopic transition $(a, b) \to (x, y)$, because they have no flaw in the side-channel of glitches.

Based on the model for the energy function of a masked CMOS gate, Definition 2, and the notion of a randomized signal pair, Definition 3, the following definition describes necessary conditions for the resistance of masked gates in a DPA attack in the presence of gitches.

Definition 5. *A gate* $g \colon \mathbb{F}_2^2 \times \mathbb{F}_2^2 \to \mathbb{F}_2$ *is called* ***G-equivariant*** *if for any* $\varphi \in \mathrm{Map}(\{1, \ldots, 4\}, \{1, \ldots, 4\})$ *and* $i = 1, 2, 3, 4$*, the expectation values of the partial energies* $\mathbb{E}(E_{g,i}((\tilde{a}, \tilde{b}), (\tilde{x}, \tilde{y}), \varphi)) \in V$ *are independent of any choice of randomized signal pairs* $(\tilde{a}, \tilde{b}), (\tilde{x}, \tilde{y})$*.*

Since the family of the randomized signal pairs can be very large we need a simpler criterion in order to decide if a gate is G-equivariant.

Lemma 1. *1) A gate* $g \colon \mathbb{F}_2^2 \times \mathbb{F}_2^2 \to \mathbb{F}_2$ *is G-equivariant if and only if for any* φ *and* i *the expectation value* $\mathbb{E}(E_{g,i}((\tilde{a}, \tilde{b}), (\tilde{x}, \tilde{y}), \varphi)) \in V$ *is independent of any choice of randomized signal pairs* $(\tilde{a}, \tilde{b}), (\tilde{x}, \tilde{y})$ *which are liftings of any* constant *pairs* $(a, b), (x, y)$*.*
2) A gate $g \colon \mathbb{F}_2^2 \times \mathbb{F}_2^2 \to \mathbb{F}_2$ *is G-equivariant if and only if for any* φ *and* $i = 1, 2, 3, 4$*, the* 2^4 *values are equal:*

$$\sum_{\substack{a_1+a_2=a \\ b_1+b_2=b \\ x_1+x_2=x \\ y_1+y_2=y}} E_{g,i}((a_1, a_2, b_1, b_2), (x_1, x_2, y_1, y_2), \varphi), \quad \text{with } a, b, x, y \in \mathbb{F}_2,$$

From the definition of G-equivariance it is immediately obvious that gates satisfying this criterion overcome the problem of side-channel leakage in the presence of glitches (the dominant effect captured by the stated model). It is a simple task to perform an exhaustive search on all 2^{16} possible gates $g \colon \mathbb{F}_2^2 \times \mathbb{F}_2^2 \to \mathbb{F}_2$ using Lemma 1 to obtain a complete list of all G-equivariant gates.

Table 1. The Boolean functions of the 50 G-equivariant gates

$c, c+a_i, c+b_i, c+a_i+b_j, c+a_ib_j, c+a_i+a_ib_j, c+b_i+a_jb_i, c+a_i+b_j+a_ib_j$

There are 50 G-equivariant gates. In the algebraic normal forms, given in Tab. 1, the indices i, j can take on the values 1 or 2 and the constant c is either 0 or 1.

Unfortunately, in the set of G-equivariant gates there are no two gates which can be paired to a lifting of any nonlinear gate (like AND or OR). Thus we have shown that:

Theorem 1. *There is no universal set of masked gates of the form* (g_1, g_2) *with* $g_1, g_2 \colon \mathbb{F}_2^2 \times \mathbb{F}_2^2 \to \mathbb{F}_2$ *satisfying the G-equivariance criterion.*

4 The Logic Family of Semi-G-equivariant Gates

The negative result from the last section leads to the question, whether the strong condition of G-equivariance can be mediated for the realization of a masked CMOS circuit in practice.

Consider the replacement of all simple gates g_i with input a_i, b_i and output c_i by gates $\tilde{g}_i$ with input $\tilde{a}_i = (a_{i1}, a_{i2}), \tilde{b}_i = (b_{i1}, b_{i2})$ and output $\tilde{c}_i = (c_{i1}, c_{i2})$. It is obvious that the pair of correlated signals (say a_{i1}, a_{i2}) of a macroscopic signal (a_i) have always the same gate depth, since they always pass through the same gates. The requirement for the implementation of a masked gate g_i, that the gate delay for both outputs, (c_{i1}, c_{i2}), should be identical, can be fulfilled in practice. Under this condition the cumulative gate delay for each signal of a pair of correlated signal would be equal. The remaining source for different propagation times of the two correlated signals are different routes leading to different capacitances at the outputs of the gate. With contemporary routing technology, however, it is possible to control routing in a way that both signals paths have the same capacitances (with high accuracy). If these design and routing constraints are met the signals of each pair of correlated signals arrive simultaneously at the inputs gate of the next gates. This practically realizable setup for a CMOS circuit implementation rules out certain combinations of the arrival times of signals. Specifically, the conditions in Definition 2 can be reduced to all maps φ with $\varphi(1) = \varphi(2)$ (for a_1, a_2) and $\varphi(3) = \varphi(4)$ (for b_1, b_2).

Definition 6. *A gate* $g \colon \mathbb{F}_2^2 \times \mathbb{F}_2^2 \to \mathbb{F}_2$ *is called* ***semi-G-equivariant*** *if for any* $\varphi \in \mathrm{Map}(\{1, \ldots, 4\}, \{1, \ldots, 4\})$ *with* $\varphi(1) = \varphi(2)$ *and* $\varphi(3) = \varphi(4)$ *the expectation value of the partial energies* $\mathbb{E}(E_{g,i}((\tilde{a}, \tilde{b}), (\tilde{x}, \tilde{y}), \varphi)) \in V$ *is independent of any choice of randomized signal pairs* $(\tilde{a}, \tilde{b}), (\tilde{x}, \tilde{y})$.

An exhaustive search on all 2^{16} gates yields 58 semi-G-equivariant gates. The list of 58 semi-G-equivariant gates comprises the 50 gates from Tab. 1 and additionally the 8 gates given in Tab. 2 below.

Table 2. Boolean function of the additional 8 semi-G-equivariant gates

$$\boxed{c + a_i + b_j + a_1b_1 + a_1b_2 + a_2b_1 + a_2b_2}$$

The 8 additional semi-G-equivariant gates now allow pairings to liftings of non-linear gates. A semi-G-equivariant AND gate can be realized, for instance, by the lifting

$$AND'(a_1, a_2, b_1, b_2) = (a_1 + b_1, a_1 + b_1 + a_1b_1 + a_1b_2 + a_2b_1 + a_2b_2)$$

using entries of Tab. 1 and Tab. 2. One immediately finds 8 realizations for an AND gate. Correspondingly, one possible realization of an OR gate is given by

$$OR'(a_1, a_2, b_1, b_2) = (a_2 + b_2, a_1 + b_1 + a_1b_1 + a_1b_2 + a_2b_1 + a_2b_2).$$

Thus there is a universal set of masked semi-G-equivariant gates. The described gates could be used to craft a set of library cells suited for an automated masked CMOS design. Any implementation of a masked semi-G-equivariant gate of course must avoid unmasked intermediate values on internal cell structures.

5 Conclusions

It has been shown that within the defined gate and energy models G-equivariance is a necessary condition on randomized gates to withstand a differential power attack (in an otherwise unconstrained CMOS circuit). However, there exists no universal set of G-equivariant gates in the considered general class of randomized gates. If practically controllable implementation constraints are imposed a set of masked gates, which are theoretically secure in the presence of glitches, can be constructed. The power model developed in this paper is inevitably a coarse abstraction of the complicated physical processes of the energy dissipation in an active CMOS circuit. Next-higher order effects may be related to the transient behavior of a switching event of a CMOS gate. Such effects may include partial swings of the outputs of gates (overlapping glitches) or cross-couplings between neighboring wires which lead to mutual information leakage. Such higher-order effects, however, are not specific to CMOS circuits, but affect also other circuit styles, such as dynamic and differential logic styles. Further experimental investigations will be necessary to quantify the side-channel leakage signal-to-noise ratio of circuits built with semi-G-equivariant gates, as well as to determine the factor for the design size increase.

References

1. M.-L. Akkar, R. Bevan, and L. Goubin: Two Power Analysis Attacks against One-Mask Methods, *11th International Workshop on Fast Software Encryption – FSE 2004,* (B. K. Roy and W. Meier, eds.), Lecture Notes in Computer Science, vol. 3017, pp. 332–347, Springer-Verlag, 2004.

2. M.-L. Akkar and C. Giraud: An Implementation of DES and AES, Secure against Some Attacks, *Cryptographic Hardware and Embedded Systems – CHES 2001,* (Ç. K. Koç, D. Naccache, and C. Paar, eds.), Lecture Notes in Computer Science, vol. 2162, pp. 309–318, Springer-Verlag, 2001.
3. J. Blömer, J. G. Merchan, and V. Krummel: Provably Secure Masking of AES, *Selected Areas in Cryptography – SAC 2004,* Lecture Notes in Computer Science, vol. 3357, pp. 69–83, Springer-Verlag, 2004.
4. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi: Towards Sound Approaches to Counteract Power-Analysis Attacks, *Advances in Cryptology – CRYPTO'99,* (M. J. Wiener, ed.), Lecture Notes in Computer Science, vol. 1666, pp. 398–412, Springer-Verlag, 1999.
5. C. Clavier, J.-S. Coron, and N. Dabbous: Differential Power Analysis in the Presence of Hardware Countermeasures, *Cryptographic Hardware and Embedded Systems – CHES 2000,* (Ç. K. Koç and C. Paar, eds.), Lecture Notes in Computer Science, vol. 1965, pp. 252–263, Springer-Verlag, 2000.
6. J.-S. Coron: Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems, *Cryptographic Hardware and Embedded Systems – CHES 1999,* (Ç. K. Koç and C. Paar, eds.), Lecture Notes in Computer Science, vol. 1717, pp. 292–302, Springer-Verlag, 1999.
7. K. Gandolfi, C. Mourtel, and F. Olivier: Electromagnetic Analysis: Concrete Results, *Cryptographic Hardware and Embedded Systems – CHES 2001,* (Ç. K. Koç, D. Naccache, and C. Paar, eds.), Lecture Notes in Computer Science, vol. 2162, pp. 251–261, Springer-Verlag, 2001.
8. J. D. Golić and R. Menicocci: Universal Masking on Logic Gate Level, *Electronics Letters* **40(9)**, pp. 526–527 (2004).
9. L. Goubin and J. Patarin: DES and Differential Power Analysis – The Duplication Method, *Cryptographic Hardware and Embedded Systems – CHES 1999,* (Ç. K. Koç and C. Paar, eds.), Lecture Notes in Computer Science, vol. 1717, pp. 158–172, Springer-Verlag, 1999.
10. Y. Ishai, A. Sahai, and D. Wagner: Private Circuits: Securing Hardware against Probing Attacks, *Advances in Cryptology – CRYPTO 2003,* (D. Boneh, ed.), Lecture Notes in Computer Science, vol. 2729, pp. 463–481, Springer-Verlag, 2003.
11. F. Klug, O. Kniffler, B. M. Gammel: Rechenwerk und Verfahren zum Ausführen einer arithmetischen Operation mit verschlüsselten Operanden, *German Patent DE 10201449 C1*, Jan. 16, 2002.
12. P. C. Kocher, J. Jaffe, and B. Jun: Differential Power Analysis, *Advances in Cryptology – CRYPTO'99,* (M. J. Wiener, ed.), Lecture Notes in Computer Science, vol. 1666, pp. 388–397, Springer-Verlag, 1999.
13. S. Mangard: Hardware Countermeasures against DPA – A Statistical Analysis of Their Effectiveness, *Topics in Cryptology – CT-RSA 2004,* (T. Okamoto, ed.), Lecture Notes in Computer Science, vol. 2964, pp. 222–235, Springer-Verlag, 2004.
14. S. Mangard, T. Popp, and B. M. Gammel: Side-Channel Leakage of Masked CMOS Gates, *Topics in Cryptology – CT-RSA 2005,* (A. Menezes, ed.), Lecture Notes in Computer Science, vol. 3376, pp. 351–365, Springer-Verlag, 2005.
15. T. S. Messerges: Securing the AES Finalists Against Power Analysis Attacks, *7th International Workshop on Fast Software Encryption – FSE 2000,* (B. Schneier, ed.), Lecture Notes in Computer Science, vol. 1978, pp. 150–164, Springer-Verlag, 2001.
16. T. S. Messerges, E. A. Dabbish, and L. Puhl: Method and Apparatus for Preventing Information Leakage Attacks on a Microelectronic Assembly, *US Patent 6,295,606*, Sept. 25, 2001, (available at `http://www.uspto.gov/`).

17. T. S. Messerges, E. A. Dabbish, and R. H. Sloan: Power Analysis Attacks of Modular Exponentiation in Smartcards, *Cryptographic Hardware and Embedded Systems – CHES 1999,* (Ç. K. Koç and C. Paar, eds.), Lecture Notes in Computer Science, vol. 1717, pp. 144–157, Springer-Verlag, 1999.
18. T. S. Messerges, E. A. Dabbish, and R. H. Sloan: Examining Smart-Card Security under the Threat of Power Analysis Attacks, *IEEE Transactions on Computers*, **51(5)**, pp. 541–552, 2002.
19. J. M. Rabaey: *Digital Integrated Circuits,* Prentice Hall, 1996, ISBN 0-13-178609-1.
20. A. Shamir: Protecting Smart Cards from Passive Power Analysis with Detached Power Supplies, *Cryptographic Hardware and Embedded Systems – CHES 2000,* (Ç. K. Koç and C. Paar, eds.), Lecture Notes in Computer Science, vol. 1965, pp. 71–77, Springer-Verlag, 2000.
21. K. Tiri and I. Verbauwhede: Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology, *Cryptographic Hardware and Embedded Systems – CHES 2003,* (C. D. Walter, Ç. K. Koç, and C. Paar, eds.), Lecture Notes in Computer Science, vol. 2779, pp. 137–151, Springer-Verlag, 2003.
22. K. Tiri and I. Verbauwhede: A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation, *Proc. of Design, Automation and Test in Europe Conference – DATE 2004,* IEEE Computer Society, pp. 246–251, 2004.
23. E. Trichina: Combinational Logic Design for AES SubByte Transformation on Masked Data, Cryptology ePrint Archive, Report 2003/236 (available at `http://eprint.iacr.org/`).

Bipartite Modular Multiplication

Marcelo E. Kaihara and Naofumi Takagi

Department of Information Engineering, Nagoya University,
Nagoya, 464-8603, Japan
{mkaihara, ntakagi}@takagi.nuie.nagoya-u.ac.jp

Abstract. This paper proposes a new fast method for calculating modular multiplication. The calculation is performed using a new representation of residue classes modulo M that enables the splitting of the multiplier into two parts. These two parts are then processed separately, in parallel, potentially doubling the calculation speed. The upper part and the lower part of the multiplier are processed using the interleaved modular multiplication algorithm and the Montgomery algorithm respectively. Conversions back and forth between the original integer set and the new residue system can be performed at speeds up to twice that of the Montgomery method without the need for precomputed constants. This new method is suitable for both hardware implementation; and software implementation in a multiprocessor environment. Although this paper is focusing on the application of the new method in the integer field, the technique used to speed up the calculation can also easily be adapted for operation in the binary extended field $GF(2^m)$.

1 Introduction

Modular multiplication is one of the basic arithmetic operations that are extensively used in many public-key cryptographic applications, such as RSA [10], ElGamal [5], Diffie-Hellman key exchange [4], DSA [1], and others. Because of its computational intensity, implementation in dedicated hardware is required for high-performance systems. Various techniques for speeding up modular multiplication have been reported in literature. Among them, two major approaches stand out: One is based on the interleaved modular multiplication algorithm where the multiplier is processed from the most significant position [2,3,6,8,11,12]. The other one is based on the Montgomery algorithm where the multiplier is processed from the least significant position [7,9,13,15]. The techniques for speeding up these two approaches have been developed separately.

This paper proposes a method that takes advantage of these techniques and the ones that may eventually be devised in the future, to further boost speed. The key that enables the linking of these two approaches is a new representation of residue classes modulo M. Assuming M is an n-word odd integer, where the radix of each word is $r = 2^k$, this new representation maps an integer U in the range $[0, M-1]$ to the number $U \cdot R \mod M$ in the same range. R is a constant of value $r^{\alpha n}$, coprime to M, where α is a rational number such that $0 < \alpha < 1$,

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 201–210, 2005.

and, αn is an integer. The novelty in this representation is that the transformation constant R has a value less than the modulus M, a condition not allowed by the Montgomery representation. Modular multiplication is then performed in this new residue system. The new values for the transformation constant enable the splitting of the multiplier into two parts which can then be processed separately, in parallel. The upper part and the lower part of the multiplier can be processed using the interleaved modular multiplication algorithm and the Montgomery algorithm, respectively. The possibility of selecting the parameter α between the values 0 and 1, encompasses the application of this method to all combinations of algorithms of different performance derived from the interleaved modular multiplication algorithm and the Montgomery algorithm. If applied to algorithms with similar performance and the multiplier is split into two equal parts, it is theoretically possible to achieve the maximum speed of twice that of these two algorithms when performed individually. The latter condition is represented with the value of the parameter α so that $\alpha n = \lceil \frac{n}{2} \rceil$.

Two other advantages of this new method are: Firstly, compared to the Montgomery method, conversion speed between the original integer set and the new residue system is potentially doubled; and secondly, precomputation of constants is no longer necessary.

Due to the parallel processing, the proposed method is suitable for hardware implementation and also for software implementation in a multiprocessor environment.

The remainder of this paper is organized as follows: Section 2 reviews the interleaved modular multiplication algorithm and the Montgomery algorithm. The new computation method is introduced in Section 3. Section 4 explains hardware implementation of the method. Section 5 contains our concluding remarks.

2 Preliminaries

2.1 Interleaved Modular Multiplication Algorithm

Given a modulus M, and two elements of the residue class ring of integers modulo M, X and Y, we define the ordinary modular multiplication as:

$$X \times Y \triangleq X \cdot Y \mod M \tag{1}$$

Let the modulus M be an n-word number, where the radix of each word is $r = 2^k$. The i-th word ($i = 0, 1, \cdots, n-1$) of Y is denoted by y_i. Namely, $Y = \sum_{i=0}^{n-1} y_i \cdot r^i$. The interleaved modular multiplication algorithm for calculating the ordinary modular multiplication is shown below [2,3,11].

[Interleaved Modular Multiplication Algorithm]
Input: $M : r^{n-1} < M < r^n$
$X, Y : 0 \leq X, Y < M$
Output: $Z = X \cdot Y \bmod M$

Algorithm:
$Z := 0;$
for $i := n-1$ **downto** 0 **do**
$Z := r \cdot Z + y_i \cdot X;$
$q_C := \lfloor \frac{Z}{M} \rfloor;$
$Z := Z - q_C \cdot M;$
endfor

In this algorithm, the words of the multiplier are processed from the most significant position first.

2.2 Montgomery Multiplication Algorithm

Montgomery introduced a powerful algorithm for calculating modular multiplication where the multiplier is processed from the least significant position first [7]. Given an n-word odd modulus M and an integer U in the range $[0, M-1]$, the image, or the M-residue of U is defined as $X = U \cdot R_M \mod M$ where R_M is a constant coprime to M and $R_M > M$. In order to reduce computation effort, this constant is usually set to the value of r^n. If X and Y are the images of U and V respectively, the Montgomery multiplication of these two images, $X * Y$, is defined as:

$$X * Y \triangleq X \cdot Y \cdot R_M^{-1} \mod M \tag{2}$$

The result is the image of $U \cdot V \bmod M$. If the i-th word of M is denoted as m_i, then $M = \sum_{i=0}^{n-1} m_i \cdot r^i$. In a similar way, if the number that represents the partial products is denoted as $Z = \sum_{i=0}^{n-1} z_i \cdot r^i$, the resulting Montgomery algorithm is described below.

[Montgomery Multiplication Algorithm]
Input: $M : r^{n-1} < M < r^n$ and $\gcd(M, 2) = 1$
$X, Y : 0 \leq X, Y < M$
Output: $Z = X \cdot Y \cdot r^{-n} \bmod M$
Algorithm:
$Z := 0;$
for $i := 0$ **to** $n-1$ **do**
$Z := Z + y_i \cdot X;$
$q_M := (-z_0 \cdot m_0^{-1}) \bmod r;$
$Z := (Z + q_M \cdot M)/r;$
endfor
if $Z \geq M$ **then** $Z := Z - M$

The transformations back and forth between the ordinary representation and the M-residue representation can be performed using the same algorithm provided that the constant $R_M^2 \mod M$ is precomputed. An integer U can be transformed to the M-residue representation by applying the Montgomery algorithm

to this integer and the constant $R_M^2 \mod M$. Transformation of an image X back into the original integer set can be done by applying the Montgomery multiplication algorithm to this image and the number 1.

3 A New Modular Multiplication Method

In this section, a new fast method for calculating modular multiplication is presented. The calculation is performed using a new representation of residue classes modulo M. In contrast to the M-residue representation introduced by Montgomery which requires the constant R_M to be coprime to M and greater in value than M, we have changed the condition of $R_M > M$ and defined a new residue class representation using a new constant $R = r^{\alpha n}$, where R is coprime to M, and, α is a rational number so that $0 < \alpha < 1$ and αn is an integer. The resulting image of an integer U is $X = U \cdot r^{\alpha n} \mod M$. Given X and Y, two images of integers U and V respectively, multiplication modulo M in the new residue system is defined as:

$$X \circledast Y \triangleq X \cdot Y \cdot r^{-\alpha n} \mod M \tag{3}$$

The existence of $r^{-\alpha n} \mod M$ is assured by the relative primalty condition between $r^{\alpha n}$ and M. Since M is odd for cryptographic applications, by utilising $r = 2^k$, the primalty condition is satisfied.

Transformation from the original representation to the new residue system is accomplished by performing conventional modular multiplication between the integer value and the constant $r^{\alpha n}$. The inverse transformation from the new residue system back to the original representation can be performed by multiplying either of the images with the constant $r^{-\alpha n}$ in modulo M, which can be done using the Montgomery algorithm as explained at the end of this section. That the new multiplication modulo M over the images of U and V results in a image of $U \cdot V \mod M$ can easily be demonstrated as follows.

$$\begin{aligned} & X \cdot Y \cdot r^{-\alpha n} \mod M \\ = & (U \cdot r^{\alpha n}) \cdot (V \cdot r^{\alpha n}) \cdot r^{-\alpha n} \mod M \\ = & (U \cdot V) \cdot r^{\alpha n} \mod M \end{aligned} \tag{4}$$

Isomorphism between the original integer set $\mathcal{Z}_{\mathcal{M}}$ with the operation $\times$, and the new residue system $\mathcal{Z}'_{\mathcal{M}}$ with the operation $\circledast$, holds as illustrated in Fig. 1.

As we will now show, modular multiplication can be efficiently computed using this new representation of residue classes. Let us consider the multiplier Y split into two parts Y_H and Y_L, i.e. $Y = Y_H \cdot r^{\alpha n} + Y_L$. Then, the multiplication modulo M of the images X and Y can be computed as follows:

$$X \circledast Y = (X \times Y_H + X \circledast Y_L) \mod M \tag{5}$$

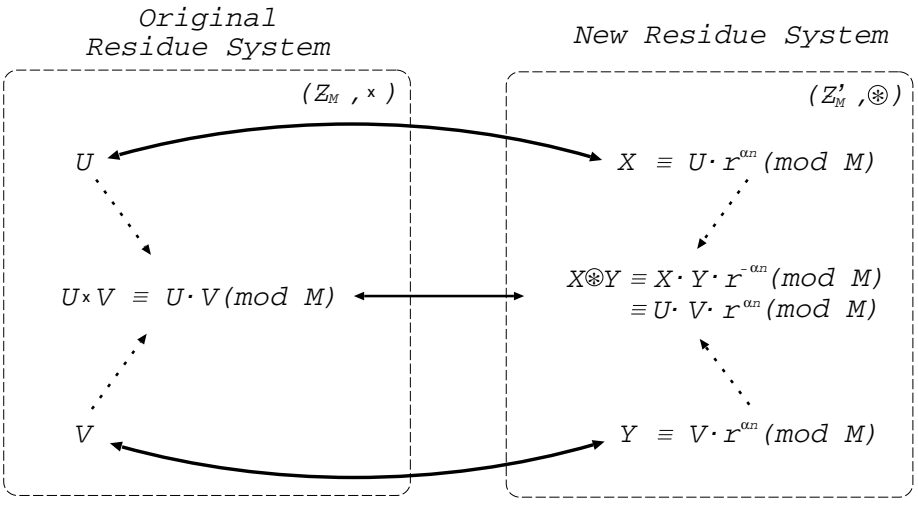


Fig. 1. Mapping between the original residue system and the new residue system

The left term, $X \times Y_H$, is calculated using the interleaved modular multiplication algorithm that processes Y_H from the most significant position first, while the second term, $X \circledast Y_L$, is calculated using the Montgomery algorithm which processes Y_L from the least significant position first. These two calculations are performed in parallel. Since the split operands Y_H and Y_L are shorter in length than Y, the calculations $X \times Y_H$ and $X \circledast Y_L$ are performed faster than the individual execution of the interleaved modular multiplication algorithm and the Montgomery algorithm with the unsplit operands.

The correctness of the above formula can be seen using the following equality:

$$\begin{aligned}
&(X \times Y_H + X \circledast Y_L) \mod M \\
&= (X \cdot Y_H + X \cdot Y_L \cdot r^{-\alpha n}) \mod M \\
&= X \cdot (Y_H \cdot r^{\alpha n} \cdot r^{-\alpha n} + Y_L \cdot r^{-\alpha n}) \mod M \\
&= X \cdot (Y_H \cdot r^{\alpha n} + Y_L) \cdot r^{-\alpha n} \mod M \\
&= X \cdot Y \cdot r^{-\alpha n} \mod M = X \circledast Y
\end{aligned} \tag{6}$$

Below is the algorithm that computes modular multiplication using the new representation. In this algorithm, A is a variable that stores the multiplicand; B_H and B_L are variables that store the upper and the lower parts of the multiplier respectively. $Interleaved\_modmul\ (A, B_H)$ is a function that calculates $A \times B_H$ by using the interleaved modular multiplication algorithm. $Montgomery\_modmul\ (A, B_L)$ is a function that calculates $A \circledast B_L$ by using the Montgomery algorithm. $\{C1; C2; \}$ means that two calculations, $C1$ and $C2$, are performed in parallel.

[Algorithm KT] (New Modular Multiplication)
Input: $M : r^{n-1} < M < r^n$, $\gcd(M, 2) = 1$
$X, Y \in \mathcal{Z}'_{\mathcal{M}}$
Output: $Z = X \cdot Y \cdot r^{-\alpha n} \bmod M \quad (Z \in \mathcal{Z}'_{\mathcal{M}})$
Algorithm:

Step 1: $A := X$; $M := M$; $S := 0$; $T := 0$;
$B_H := Y_H$; $B_L := Y_L \quad /* \, Y = Y_H \cdot r^{\alpha n} + Y_L \, */$
Step 2: $\{S := Interleaved\_modmul(A, B_H)$;
$T := Montgomery\_modmul(A, B_L); \}$
Step 3: $Z := (S + T) \bmod M$;

When using the interleaved modular multiplication algorithm and the Montgomery algorithm of similar performance, α can be set to the value so that $\alpha n = \lceil \frac{n}{2} \rceil$; the split two parts of the multiplier, Y_H and Y_L, are at most $\lceil \frac{n}{2} \rceil$-words wide. This means that, it is theoretically possible to obtain a maximum acceleration of twice the speed of the original algorithms performed individually, when these conditions are met. Fig. 2 shows the multiplication procedure with the parameter $\alpha = 1/2$.

Transformation of an integer U from the original integer set to the new residue system can be performed by executing $X = Interleaved\_modmul(U, r^{\alpha n})$. The inverse transformation of an image X from the new residue class representation back to the original integer set is accomplished by executing $U = Montgomery\_modmul(X, 1)$. When α is set to the value so that $\alpha n = \lceil \frac{n}{2} \rceil$, either of these transformations can be completed theoretically in half the time required by the Montgomery method without the need for precomputed constants.

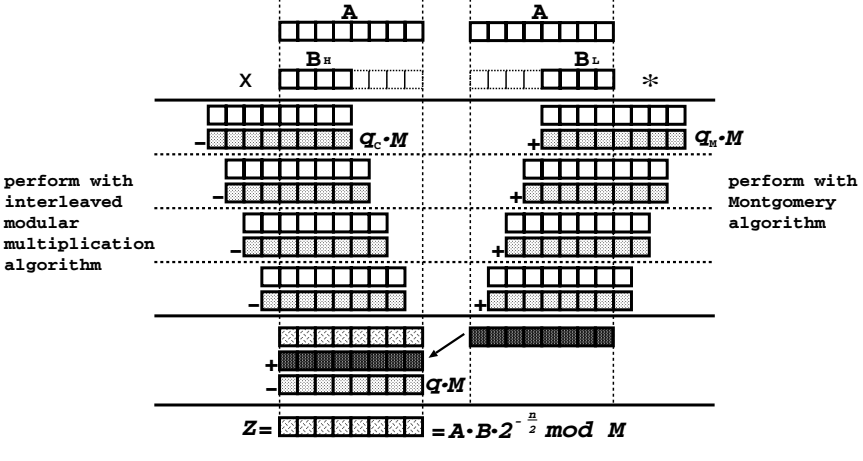


Fig. 2. New Modular Multiplication with $\alpha = 1/2$

4 Hardware Implementation

A modular multiplier based on the algorithm presented in the previous section consists of six registers, an interleaved modular multiplier, a digit-serial Montgomery multiplier, a modular adder, and a multiplexor. The registers are: A, which stores the multiplicand; B_H and B_L which are shift registers and store the upper and lower parts of the multiplier respectively; M, which stores the modulus M; and, S and T, which store the partial results. A block diagram of this hardware is shown in Fig. 3.

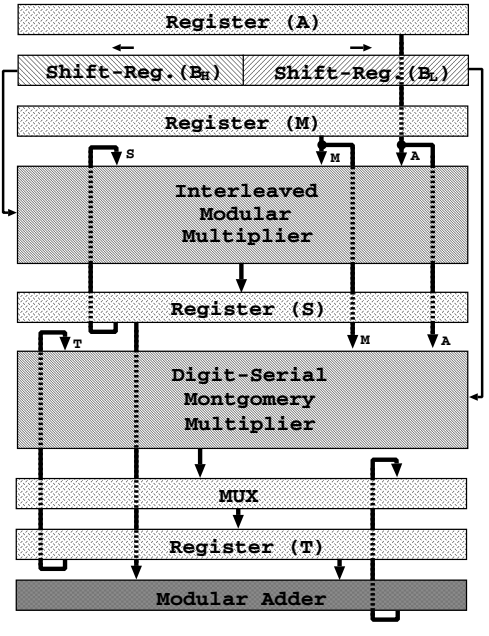


Fig. 3. Block diagram of a multiplier

Various implementations of the interleaved modular multiplier and the Montgomery multiplier are possible depending on the techniques used for speeding up the calculation. Most of these techniques use redundant representation and increase the radix, and the different combinations of the multipliers allow for a wide range of trade-offs between speed and hardware requirements.

When a radix-r interleaved modular multiplier is jointly used with a radix-r Montgomery multiplier with similar critical path delays, and n is even, the parameter α can be set to the value $1/2$ and registers of equal length can be used for B_H and B_L, thus halving the processing time compared to an individual execution of the interleaved modular multiplier or the Montgomery multiplier with the unsplit operands.

Transformation from the ordinary integer set to the new residue class representation can be performed with the same hardware provided that the hardware module which computes the interleaved modular multiplication iterates one extra cycle compared to that required for modular multiplication. Inverse transformation from the new residue class representation back to the original integer set can be performed using the hardware module that computes the Montgomery multiplication.

The value of the parameter α can be displaced around $1/2$ enabling the use of multipliers of different performance. In this case, the multiplier is then split into two unequal parts that can be stored in two registers B_H and B_L of different length. The value of α can be determined from the difference of performance between the multipliers. For example, if a radix-2 interleaved modular multiplier is used with a radix-4 Montgomery multiplier with similar critical path delays, then α can be set to a number where $\alpha n = \lceil \frac{2}{3} n \rceil$. Then, modular multiplication can be accomplished in about $\lceil \frac{n}{3} \rceil$ clock cycles. Transformation from the original integer set to the new residue system, can be performed with the same hardware by executing the interleaved modular multiplication module twice. In the first execution, an integer U is multiplied by $2^{\lceil \frac{n}{3} \rceil}$. In the second execution, the result of the first execution is multiplied by $2^{\lceil \frac{2}{3} n \rceil - \lceil \frac{n}{3} \rceil}$. Inverse transformation from the new residue system to the original integer set can be performed by the same hardware by executing the Montgomery multiplication module once.

The amount of hardware of the proposed multiplier is proportional to n. Compared to an individual interleaved modular multiplier or a Montgomery multiplier, the new modular multiplier requires an extra digit modular multiplier, an extra register, a modular adder and related multiplexors.

The space and time trade-offs for high radix modular multiplications based on the classical interleaved algorithm and the Montgomery algorithm are detailed in [14]. For both algorithms, increasing k, i.e. the number of bits of the radix, to values greater than $log(n)$, where n is the number of words, results in a penalty in time for producing the quotient bits q_C or q_M for the next modular reduction in time that makes this approach unattractive. By contrast, by using our algorithm, a speedup can be achieved for such values of radices, since the multiplication and the modular reduction for the split multiplier can be performed by two separate high-radix digit-serial multipliers processing in parallel. Thus, the number of iteration is reduced without increasing the time requirements for each cycle.

Furthermore, as there is no need to increase the radix, the design can remain relatively simple compared to hardware designed using higher radix.

In cryptographic applications, such as in RSA, this approach is much more attractive than implementing two modular multiplier processors separately because it has the advantage of producing the outputs sequentially.

5 Concluding Remarks

In this paper, we have presented a fast method for computing modular multiplication. We have defined a new residue class representation which enables the

splitting of the multiplier into two parts which can be processed by using the interleaved modular multiplication algorithm and the Montgomery algorithm in parallel, potentially doubling the speed. Transformations back and forth between the original integer set and the new residue system can be performed at a maximum of twice the speed of the Montgomery method without the need for precomputed constants. The dual processing makes it suitable for software implementation in a multiprocessor environment as well as for hardware implementation as discussed in Section 4. Finally, although this paper is focused on application in the integer field, the technique used to speed up the calculation in the proposed method can also easily be adapted for accelerating modular multiplication in the binary extended field $GF(2^m)$.

Acknowledgments

The authors would like to thank Associate Professor Kazuyoshi Takagi for his valuable comments and discussions and to Miss Grith Christensen for her help in preparing this paper.

References

1. ANSI X9.30. Public Key Cryptography for the Financial Services Industry: Part 1: The Digital Signature Algorithm (DSA). American National Standard Institute. American Bankers Association. 1997.
2. G. R. Blakley, "A Computer Algorithm for Calculating the Product *AB* Modulo *M*," *IEEE Trans. Computers*, vol. C-32, no. 5, pp. 497–500, May 1983.
3. E. F. Brickell, "A fast modular multiplication algorithm with application to two key cryptography," in *Advances in Cryptology, Proc. CRYPTO'82*, D. Chaum *et al.*, Eds. New York: Plenum, 1983, pp. 51–60.
4. W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Trans. Information Theory*, vol. 22, no. 11, pp. 644–654, Nov. 1976.
5. T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Information Theory*, vol. IT-31, no. 4, pp. 469–472, July 1985.
6. P. Kornerup, "High-Radix Modular Multiplication for Cryptosystems," *Proc. 11th IEEE Symp. Computer Arithmetic*, G. Jullien, M. J. Irwin, and E. Swartzlander, eds., pp. 277-283, Windsor, Canada, 1993.
7. P. L. Montgomery, "Modular Multiplication without Trial Division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, Apr. 1985.
8. H. Morita, "A fast modular multiplication algorithm with application to two key cryptography," in *Lecture Notes in Computer Science*, vol. 435, G. Brassard Ed., *Advances in Cryptology – CRYPTO'89 Proc.* Berlin, Germany: Springer-Verlag, 1990, pp. 387–399.
9. H. Orup, "Simplifying quotient determination in high-radix modular multiplication," *Proc. 12th IEEE Symp. Computer Arithmetic*, S. Knowles and W. H. McAllister, eds., pp. 193-199, Bath, England, 1995.
10. R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120-126, Feb. 1978.

11. K. R. Sloan, "Comments on 'A Computer Algorithm for Calculating the Product AB Modulo M'," *IEEE Trans. Computers*, vol. C-34, no. 3, pp. 290–292, March 1985.
12. N. Takagi, "A radix-4 modular multiplication hardware algorithm for modular exponentiation," *IEEE Trans. Comput.*, vol. 41, no. 8, pp. 949–956, Aug. 1990.
13. A. F. Tenca, G. Todorov, and Ç. K. Koç, "High-Radix Design of a Scalable Modular Multiplier," *Cryptographic Hardware and Embedded Systems - CHES 2001*, Ç. K. Koç, D. Naccache, C. Paar, eds., pp. 185–201, Paris, France, 2001.
14. C. D. Walter, "Space/Time Trade-offs for Higher Radix Modular Multiplication using Repeated Addition," *IEEE Trans. Computers*, vol. 46, no. 2, pp. 139–141, Feb. 1997.
15. C. D. Walter, "Systolic Modular Multiplication," *IEEE Trans. Computers*, vol. 42, no. 3, pp. 376–378, Mar. 1993.

Fast Truncated Multiplication for Cryptographic Applications

Laszlo Hars

Seagate Research, 1251 Waterfront Place,
Pittsburgh, PA 15222, USA
Laszlo@Hars.US

Abstract. The *Truncated Multiplication* computes a *truncated product,* a contiguous subsequence of the digits of the product of 2 integers. A few truncated polynomial multiplication algorithms are presented and adapted to integers. They are based on the most often used n-digit full multiplication algorithms of time complexity $O(n^\alpha)$, with $1<\alpha\le 2$, but a constant times faster. For example, the least significant half products with Karatsuba multiplication need only 80% of the full multiplication time. The faster the multiplication, the less relative time saving can be achieved.

Keywords: Computer Arithmetic, Short product, Truncated product, Cryptography, RSA cryptosystem, Modular multiplication, Montgomery multiplication, Karatsuba multiplication, Barrett multiplication, Optimization.

1 Notations

- Long integers are denoted by $A=\{a_{n-1}\dots a_1, a_0\}=a_{n-1\dots 0}=\sum d^i a_i$ in a d-ary number system, where a_i, $0\le a_i\le d-1$ are *digits* (usually 16 or 32 bits: $d=2^{16}=65{,}536$; $d=2^{32}=4{,}294{,}967{,}296$)
- $|A|$ or $|A|_d$ denotes the number of digits, the length of a d-ary number. $|\{a_{n-1}\dots a_1a_0\}|=n$
- $A\,\|\,B$ the number of the concatenated digit-sequence $\{a_{n-1}...a_0,b_{m-1}...b_0\}$; $|A|=n$, $|B|=m$.
- $[x]$ denotes the integer part of x, and $0\le\{x\}<1$ is the fractional part, such that $x=[x]+\{x\}$
- $\lg n=\log_2 n=\log n/\log 2$
- LS stands for **L**east **S**ignificant, the low order bit/s or digit/s of a number
- MS stands for **M**ost **S**ignificant, the high order bit/s or digit/s of a number
- (*Grammar*) *School multiplication, division*: the digit-by-digit multiplication and division algorithms, as taught in elementary schools
- $A\ltimes B$, $A\rtimes B$ denote the MS or LS half of the digit-sequence of $A\times B$ (or $A\cdot B$), respectively
- $A\otimes B$ denotes the middle third of the digit-sequence of $A\times B$
- $M_\alpha(n)$ the time complexity of the Toom-Cook type full multiplication, $O(n^\alpha)$, with $1<\alpha\le 2$
- γ_α = the speedup factor of the half multiplication, relative to $M_\alpha(n)$
- δ_α = the speedup factor of the middle-third product, relative to $M_\alpha(n)$

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 211 – 225, 2005.

2 Introduction

Many cryptographic algorithms are based on modular arithmetic operations. The most time critical one is multiplication. For example, exponentiation, the fundamental building block of RSA-, ElGamal- or Elliptic Curve - cryptosystems or the Diffie-Hellman key exchange protocol [17], is performed by a chain of modular multiplications. For modular reduction division is used, which can be performed via multiplication with the reciprocal of the divisor, so fast reciprocal calculation is also important. Modular multiplications can be performed with reciprocals and regular multiplications, and in some of these calculations truncated products are sufficient.

We present new speedup techniques for these and other basic arithmetic operations.

For operand sizes of cryptographic applications school multiplication is used the most often, requiring simple control structure. Speed improvements can be achieved with Karatsuba's method and the Toom-Cook 3- or 4-way multiplication, but the asymptotically faster algorithms are slower for these operand lengths [9], [14]. In this paper we consider digit-serial multiplication algorithms of time complexity $O(n^{\alpha})$, $1< \alpha \leq 2$, similar to microprocessor software, that is, no massive-parallel- or discrete Fourier transform based multiplications, which require different optimization methods [3].

3 Truncated Products

The *Truncated Multiplication* computes a *Truncated Product*, a contiguous subsequence of the digits of the product of 2 integers. If they consist of the LS or MS half of the digits, they are sometimes called short products or half products. These are the most often used truncated products together with the computation of the middle third of the product-digits, also called middle product.

No exact speedup factor is known for truncated multiplications, which are based on full multiplications faster than the school multiplication. For half products computed by Fourier or Nussbaumer transform based multiplications no constant time speedup is known.

One way to calculate truncated products is implied by covering the corresponding area in the multiplication square (Fig. 1) with polygons (like smaller squares or triangles), which correspond to partial products of known complexity. Fig. 3 shows a covering of a triangle, corresponding to the MS half product, proposed by Mulders [19] for polynomials. Covered areas, which don't belong to the truncated product to be computed, get ignored when the final digit-sequence is generated. They don't cause extra costs beyond the work needed to calculate them (except handling carries), but the resulting shapes might correspond to products, which can be faster calculated. (A full square, for example, has many dependencies among the digit-products contained, so full multiplications could be faster than truncated multiplications of similar size, corresponding to narrow and long regions, without that many dependencies.)

The covering shapes may extend out from the full product square, like in Fig. 10. The products in the excess area are calculated with 0-padding the multiplicands, not affecting the result. If the covering polygons overlap, the involved area has to be processed again and the corresponding digits subtracted from the result digit-sequence one fewer times than they were covered.

4 Time Complexity

Multiplication is more expensive (slower and/or more hardware consuming) even on single digits, than addition or store/load operations. Many computing platforms perform additive- and data movement operations parallel to multiplications, so they don't take extra time. In order to obtain general results and to avoid complications from architecture dependent constants we measure the time complexity of the algorithms with the ***number of digit-multiplications*** performed.

For the commonly used multiplication algorithms, even for moderate operand lengths the number of digit-multiplications is well approximated by n^{α}, where α is listed in the table below.

School	Karatsuba	Toom-Cook-3	Toom-Cook-4
2	$^{\log 3}/_{\log 2} = 1.5850$	$^{\log 5}/_{\log 3} = 1.4650$	$^{\log 7}/_{\log 4} = 1.4037$

On shorter operands asymptotically slower algorithms could be faster, when architecture dependent minor terms are not yet negligible. (We cannot compare different multiplication algorithms, running in different computing environments, without knowing all these factors.) For example, when multiplying linear combinations of partial results or operands, a significant number of non-multiplicative digit operations are executed, that might not be possible to perform in parallel to the digit-multiplications. They affect some minor terms in the complexity expressions and could affect the speed relations for shorter operands. To avoid this problem, when we look for speedups for certain multiplication algorithms, when not all of their product digits are needed, we only consider algorithms performing ***no more auxiliary digit operations than what the corresponding full multiplication performs***. When each member of a family of algorithms under this assumption uses internally one kind of black-box multiplication method (School, Karatsuba, Toom-Cook-k), the speed ratios among them are about the same as that of the black-box multiplications. Consequently, if on a given computational platform and operand length one particular multiplication algorithm is found to be the best, say it is Karatsuba, then, within a small margin, the fastest algorithm discussed in this paper is also the one, which uses Karatsuba multiplication.

4.1 Complexity Paradox

We don't consider algorithms with excessive operand mixing, like many linear combinations of partial results. They might speedup calculations asymptotically, but at certain operand lengths the speed relations among the algorithms could be changed, due to the hidden minor terms in the time complexities. For example, consider the complexity of truncated multiplications using Karatsuba's method. Let us partition the multiplicands into $k > 2$ digit-blocks and perform one k-way Toom-Cook multiplication step. The digit-blocks are multiplied with Karatsuba's algorithm of time complexity $n^{\lg 3}$. The product (an extreme truncated product) takes asymptotically $(2k-1)(n/k)^{\lg 3} < n^{\lg 3}$ digit-products to

calculate. Since the Toom-Cook multiplication algorithm performs more non-multiplicative operations than Karatsuba (long additions and multiplications / divisions with small constants), some minor terms become significant (left out from our asymptotical complexity expression), and so we might paradoxically conclude that Karatsuba's multiplication is faster than itself.

5 School Multiplication

In multiplications C = A·B, the product digits are calculated as $c_k = \Sigma_{i+j=k}\, a_i b_j$ for indices when a_i and b_j exist. It is convenient to write the digits of the multiplicands on the top and left side of a rectangle, and the digit-products inside, where the corresponding rows and columns meet. The digits of the product are then calculated by summing up the digit-products inside the rectangle along the diagonals, starting with the single entry diagonal at the top right corner, moving left and downward. The MS digit is the result only of the last carry: Fig. 1. (We can think this square as a "straightened up" product parallelogram, taught in the school for arranging the partial products.)

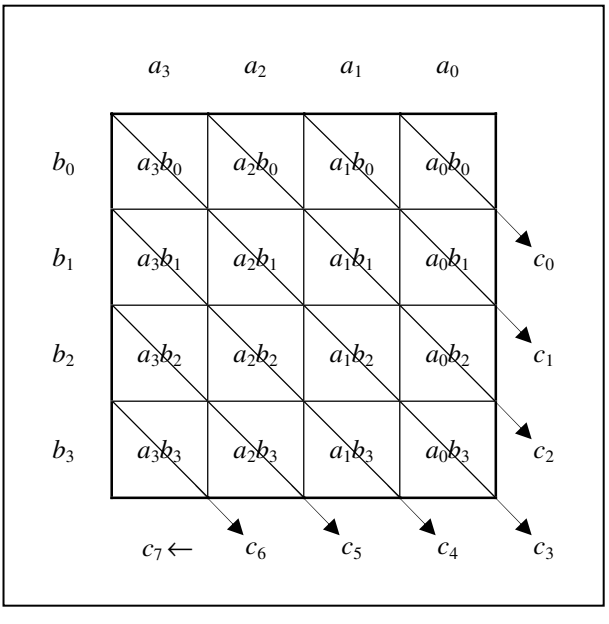


Fig. 1. School multiplication

6 Carry

The digit-products can be 1 or 2 digits long. When calculating the digits of the (truncated) product these digit-products are added, so there is usually a carry,

propagating to more significant digits. The carry is the largest when all digits of both multiplicands are $d-1$. The diagonal i, $i = 0\ldots n-1$, in the triangle I in Fig. 2 contributes to the total $(i+1)(d-1)^2 d^i$. Summing them up from 0 to k (e.g. with the help of the differential of the generator function $G(x) = (d-1)^2 \sum_i x^{i+1}$, a geometric series) gives the total of the first $k+1$ diagonals (see also [16]):

$$s_k = kd^{k+2} + (d-k-2)d^{k+1} + 1.$$

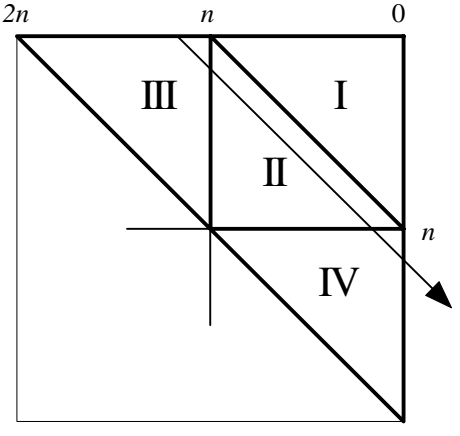


Fig. 2. Carry in MS half

If $k = n\ldots 2n-2$, (triangle II in Fig. 2) we embed the multiplication square in a double size one, and sum the diagonals there. The products belonging to parts of the diagonals, which lie in the triangles III and IV are then subtracted:

$$\begin{aligned} \boldsymbol{s_k - 2d^n s_{k-n}} &= (2n-k-1)d^{k+2} + (k-2n+2)d^{k+1} - 2d^n + 1 \\ &= (2n-k-2)d^{k+2} + (d-2n+k+1)d^{k+1} + (d^{k+1} - 2d^n) + 1 \end{aligned}$$

Lemma C. The carry is maximal at the main diagonal: s_{n-1}.

Proof. The above equations show, that the carry s_k / d^k in the upper right triangle is increasing, in the lower right triangle, decreasing. The larger of the two middle ones is the maximum. Compare $d \cdot s_{n-1} = (n-1)d^{n+2} + (d-n-1)d^{n+1} + d$ and $s_n - 2d^n s_0 = (n-2)d^{n+2} + (d-n+1)d^{n+1} + (d^{n+1} - 2d^n) + 1$. The coefficient of the dominant d^{n+2} term is larger at the former one, proving the proposition for $d > 2$. For $d = 2$ direct substitution proves it. □

6.1 Guard Digits

The most often used truncated products of n-digit multiplicands contain either the MS half of the product digits $(c_{2n-1}, c_{2n-2}, \ldots, c_n)$, or the LS half $(c_{n-1}, \ldots, c_1, c_0)$. There is no problem with the carry at the LS half, but the MS half product is affected by the carry

from the LS half. This is the same problem as proper rounding of floating point multiplications [16].

Other truncated products without the LS digits have also problems with the carry. From Lemma C it follows that the largest carry is caused by the LS half products, and it can be as large as $s_{n-1} = (n-1)d^{n+1} + (d-n-1)d^n + 1$. That is, the rightmost 2 digits of the MS half product can be affected. We can calculate 2 more digits than requested, called "guard" digits, with the carry they generate. There is, however, still a chance of carry propagation from further to the right. The LS digits all, up to c_{n-3} can contribute $s_{n-3} = (n-3)d^{n-1} + (d-n+1)d^{n-2} + 1$. If the first guard digit c_{n-1} was $d-n+3$ or larger, the left out LS product could cause a carry propagating even to the digit c_n.

In the middle of cryptographic calculations partial results look uniformly random. In this sense, having ignored the LS $n-2$ product digits, the chance of a possible carry to c_n is $(n-3)/d$. It is small at usual settings (0.1% at $d = 2^{16}$, $n = 64$; and 6.7×10^{-9} at $d = 2^{32}$, $n = 32$ – corresponding to RSA-1024). We can see the danger of ineffective guard digits. If they happen to be too large we calculate a third guard digit, too. A carry could only be propagating, if $c_{n-1} = d-1$ and $c_{n-2} \geq d-n+4$. The chance of this is very small: 1.4×10^{-8} at $d = 2^{16}$, $n = 64$; and 1.5×10^{-18} at $d = 2^{32}$, $n = 32$. In Newton iterations or in the Barrett multiplication an occasional error of 1 does not matter, but in other situations we ***cannot allow wrong results*** even with this low probability.

Keeping calculating more guard digits when they turn out to be large, the expected amount of work to get the ***exact carry*** for the MS half product is barely larger than $2n$, but the worst case complexity is $O(n^2)$. Stopping at the 2nd or 3rd guard digit and calculate the full product when there is a possible carry propagation, gives slightly more expected work, but in the worst case only one extra multiplication is performed. It is better at sub-quadratic multiplication algorithms.

Let us denote by T_1 and T_0 the average time complexity of truncated multiplications, providing the product digits $\{c_k, c_{k+1} \ldots c_m\}$, $k \geq m$, with or without proper rounding, respectively. Since the main diagonal is the longest, the reasoning above covers the worst case carry, proving

Lemma G. $T_1 < T_0 + c \cdot m$, with $c \approx 2$. []

Consequently, the LS and MS half products are of equal complexity (within an additive linear term). In the sequel we don't distinguish between truncated products with or without proper rounding, unless the linear term of the time complexity is in question.

7 Half Products

With school multiplication half products can be calculated in half of the time of the full multiplication, simply by not calculating the digit-products, which are not needed.

Let $M_\alpha(n) = n^\alpha$ denote the time complexity of the full n-digit multiplication, $1 < \alpha \leq 2$; and $H_\alpha(n)$ denote the time needed for a half product. The construction of Fig. 3 leads to the inequality:

$$H_\alpha(n) \leq M_\alpha(n-k) + 2H_\alpha(k).$$

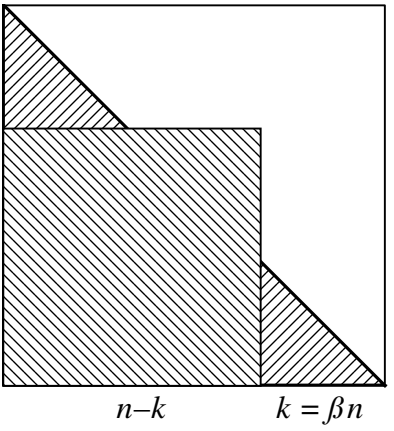


Fig. 3. Half product computation

Note. If we chose $k = n/2$, such that there is no excess area (the small square does not cross the diagonal), we get $H_\alpha(n) \le M_\alpha(n/2) + 2H_\alpha(n/2)$. Denoting the complexity of the resulting algorithm by $S'_\alpha(n)$, we get the recursive equation $S'_\alpha(n) = M_\alpha(n/2) + 2\,S'_\alpha(n/2)$. Unfortunately, the solution shows $S'_\alpha(n)$ larger than the full multiplication time $M_\alpha(n)$ if $\alpha < \lg 3$, and when $\alpha = \lg 3$ (Karatsuba), any initial speed advantage (for small n values) diminishes very fast with increasing n, giving $S'_{\lg 3}(n) \approx M_{\lg 3}(n)$. See [16].

Let us denote the time complexity of the half multiplication algorithm by $S_\alpha(n)$, derived from the covering shown in Fig. 3, with the underlying multiplication complexity $M_\alpha(n) = n^\alpha$.

Looking for a constant speedup ratio, let's chose appropriate β and γ factors: $S_\alpha(n) = \gamma n^\alpha$, and $k = \beta n$. The recurrence relation $S_\alpha(n) = M_\alpha(n-k) + 2S_\alpha(k)$ becomes

$$\gamma n^\alpha = n^\alpha(1-\beta)^\alpha + 2\gamma n^\alpha \beta^\alpha \tag{1}$$

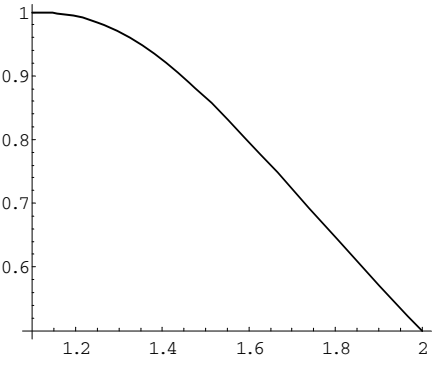


Fig. 4. Half product speedup γ vs. α

Theorem H. The time complexity of the half multiplication, based on multiplications of time complexity $M_\alpha(n) = n^\alpha$, $1 < \alpha \le 2$, is at most $\gamma_\alpha M_\alpha(n)$, with

$$\gamma = (1-\beta)^\alpha / (1-2\beta^\alpha) \text{ and } \beta = 2^{-1/(\alpha-1)}.$$

Proof. The value of β, which minimizes γ is found by differentiation of (1). []

For shorter operands numerical calculations or simulations find the exact speedup factors [28]. They were found close to the asymptotic values graphed in Fig. 4. The numerical values of these asymptotical speedup factors and the corresponding splitting ratios are tabulated below.

	Exponent α	Speed γ_α	Split at β_α
School	2	0.5	0.5
Karatsuba	1.585	0.8078	0.3058
Toom-Cook-3	1.465	0.8881	0.2252
Toom-Cook-4	1.404	0.9232	0.1796

There are many more ways to cover a product triangle. For example:

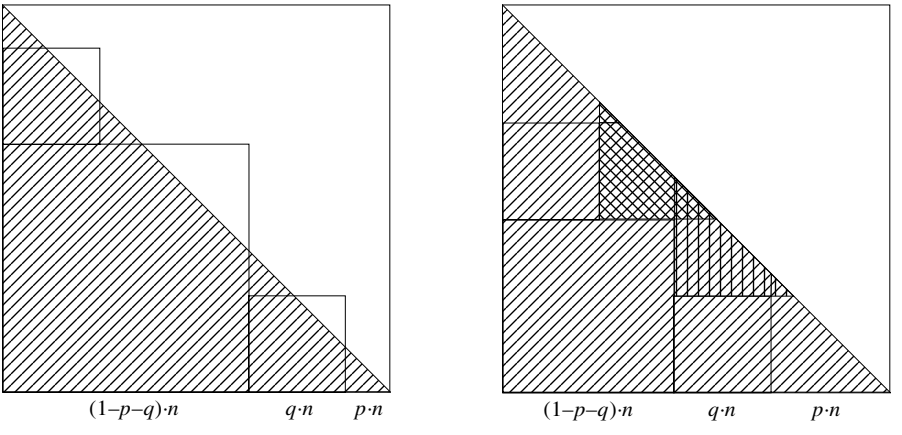


Fig. 5. Alternative coverings for half products

The left hand side configuration on Fig. 5 is optimal if $q+p = \beta_\alpha$, when it becomes the same as Fig. 3, recursively applied in the small triangles. The right hand side configuration is actually worse, because of the overlapping triangles.

8 LS and MS Products

In the geometric constructions of the half product calculations in the previous section there were large squares, with not needed product digits corresponding to their upper

right corner. These represent MS truncated products. A natural question is: How much must we cut off, such that the truncated product corresponding to the remaining part is faster to compute than the whole product?

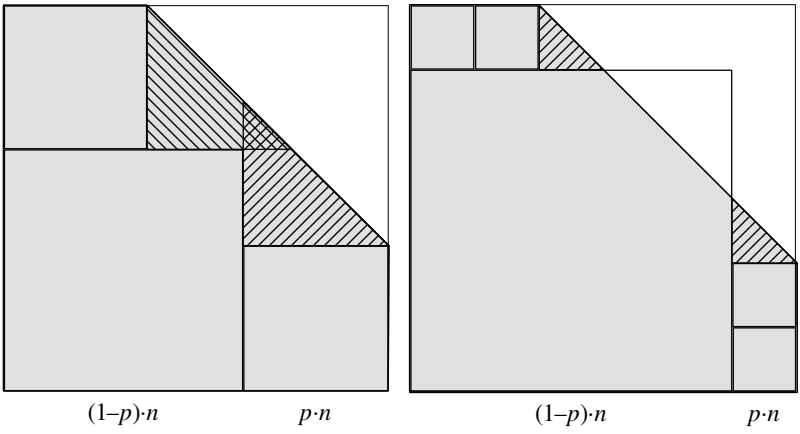


Fig. 6. Cut corner product constructions

The constructions of Fig. 6 give lower bounds on the size of the largest MS truncated products faster computed than the full products. With varying number of the small squares on the sides numerical calculations give the following optimum p values:

		Karatsuba	Toom-Cook-3	Toom-Cook-4
□	Fastest product	0.9402	0.9744	0.9860
	Max p	0.2174	**0.1225**	**0.0791**
□□	Fastest product	0.9706	0.9895	0.9950
	Max $2p$	**0.2176**	0.1020	0.0575
□□□	Fastest product	0.9822	0.9944	0.9976
	Max $3p$	0.1982	0.0817	0.0419

The "Fastest product" entries indicate the maximum speedup for MS truncated products over full products at the best p value with the corresponding number of little side-squares (Fig. 6). With this kind of constructions the gain is not large, at most 6% (Karatsuba with 1 small square at the side and a triangle stacked up).

Proposition S. The MS and LS truncated multiplication of two n-digit numbers is faster than the full multiplication if k of the $2n$ product-digits are computed, $k < 61\%$ of $2n$ with Karatsuba multiplication: $(1+kp)/2 = 0.6088$, $k < 56\%$ with Toom-Cook-3 and $k < 54\%$ with Toom-Cook-4. []

These MS products, which can be faster calculated than the full product, are not long enough to improve half multiplications, needing speed improvement at $½/(1-\beta)$.

9 Middle-Third Products

Some crypto algorithms could use products containing the middle n digits $a \otimes b$ of the $3n$ digits of the product $a \times b$, when one of the multiplicands is twice longer ($2n$-digit) than the other. The corresponding digit-products are contained in the shaded parallelogram in Fig. 7. The simplest way to achieve faster multiplications is to split it in half, and combine the 2 half products corresponding to the left and right triangle, with time complexity $2\gamma_\alpha \mathrm{M}_\alpha(n)$.

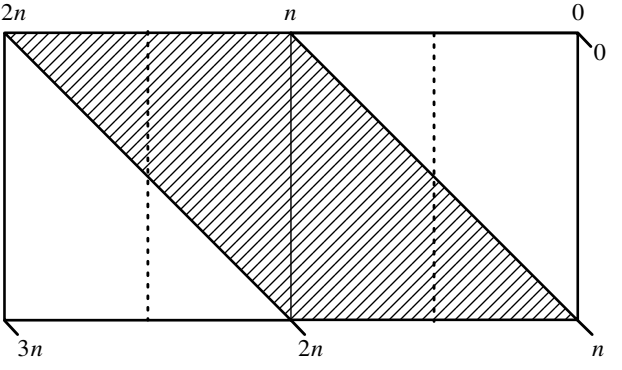


Fig. 7. Middle-third product

Alternatively, an $n \times n$ square covers the center of the parallelogram (dotted lines in Fig. 7). When we process separately the two small triangles left out: the corresponding complexity is better (for the Karatsuba case it is 1.5385):

$$2\gamma_\alpha \mathrm{M}_\alpha(n) > \mathrm{M}_\alpha(n) + 2\gamma_\alpha \mathrm{M}_\alpha(n/2) = (1+2^{1-\alpha}\gamma_\alpha)\mathrm{M}_\alpha(n).$$

In [11] a clever speedup was presented for the Karatsuba case. It dissects the operands and calculates products of sums, but uses only as many extra addition/load operations as the multiplication, so it does not change speed relations. The resulting complexity is the same as that of the Karatsuba multiplication. Unfortunately, the idea does not directly generalize to asymptotically faster multiplications, so only the second entry is improved in the table below for the middle product speedup factors, but the Karatsuba case is the most important one in practice.

The idea is to cut the middle product parallelogram into 4 congruent pieces, as shown in Fig. 8. Each smaller parallelogram represents a middle third product, as seen, e.g. PURV in the rectangle PQRS. With the notation $\mathrm{A}_{ij} = \{a_{jn/2-1} \ldots a_{in/2}\}$ the 4 middle products are $\mathrm{MP}(\mathrm{A}_{13},\mathrm{B}_0)$, $\mathrm{MP}(\mathrm{A}_{24},\mathrm{B}_0)$, $\mathrm{MP}(\mathrm{A}_{13},\mathrm{B}_1)$, $\mathrm{MP}(\mathrm{A}_{02},\mathrm{B}_1)$. Calculate α, β and γ:

$$\alpha \leftarrow \mathrm{MP}(\mathrm{A}_{02}+\mathrm{A}_{13},\mathrm{B}_1)$$
$$\beta \leftarrow \mathrm{MP}(\mathrm{A}_{13}, \mathrm{B}_0-\mathrm{B}_1)$$
$$\gamma \leftarrow \mathrm{MP}(\mathrm{A}_{13}+\mathrm{A}_{24},\mathrm{B}_0)$$

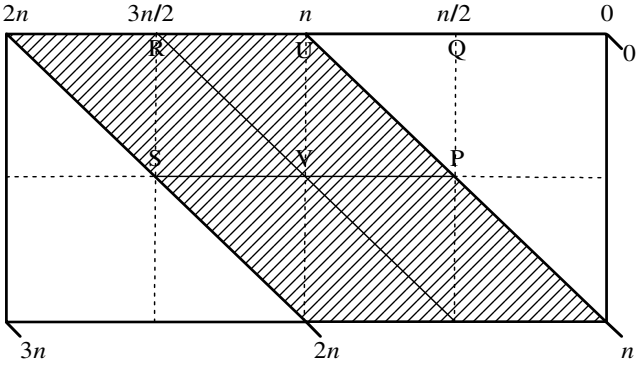


Fig. 8. Karatsuba Middle-third product

The desired result is $(\alpha+\beta) \parallel (\gamma-\beta)$, with carry propagation. That is, the middle third product is calculated with 3 half size middle third products and 5 additions, giving the same recursion (for power of 2 operand lengths) as at the Karatsuba multiplication. (Other lengths are slightly slower.) Let δ_α denote the speedup factor relative to $M_\alpha(n)$. It is summarized in the following table. (The Toom-Cook entries represent new results.)

School	Karatsuba	Toom-Cook-3	Toom-Cook-4
1	1	***1.6434***	***1.6979***

10 Third-Quarter Products

The third quarter of the product digits $(c_{3n/2-1} \ldots c_n)$ contains $\frac{3}{8}$ of the digit-products. Accordingly, the school multiplier takes $\frac{3}{8} n^2$ time. It is of the worst shape discussed

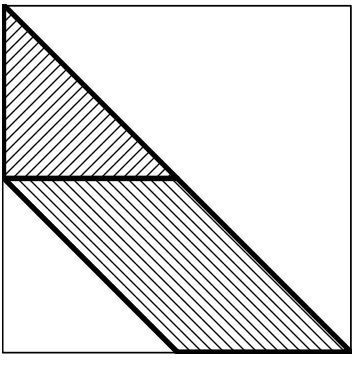


Fig. 9. Third-quarter product computation

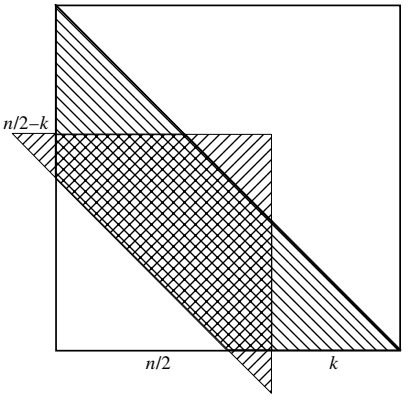


Fig. 10. Alternative computation for third-quarter products

here. Narrow, and long, so no much room is left to find dependencies among digit-products, which would allow faster multiplication algorithms.

The shaded trapezoid can be cut into a parallelogram and a triangle, as shown in Fig. 9, and the corresponding time complexity is

$$(\delta_\alpha+\gamma_\alpha)\mathrm{M}_\alpha(n/2) = (\delta_\alpha+\gamma_\alpha)2^{-\alpha}\mathrm{M}_\alpha(n).$$

It gives the following coefficients: 0.375, 0.6026, 0.9170 and 0.9907. The school and the Karatsuba multiplication case are faster than calculating the full triangle, but at the Toom-Cook multiplications we are better off taking the containing whole half product. (One could try to cover the trapezoid with triangles, which overhang the multiplication square, as in Fig. 10. Simple calculations show, that it does not help. The complexity is minimal at $k = n/2$, that is, where the cover is exact.)

11 Squaring

All the known multiplication algorithms with time complexity $O(n^\alpha)$ speed up almost twofold for squaring, which is the case for small operands and the recursion these algorithms follow keeps this ratio. Additions and processing overhead in the algorithms reduce the achievable speedup ratio to 0.55…0.65. See [28]. (A speedup below 0.5 cannot be achieved, since it would yield faster general multiplications with the identity $4ab = (a+b)^2-(a-b)^2$ and the corresponding algorithm gives 0.5 multiplication time for squaring.)

This is not a truncated product, being represented by a triangle with its hypotenuse perpendicular to the main diagonal of the multiplication square. At general products the upper-left and lower-right triangles contain different digit-products, but the corresponding digit-sequences can be calculated faster than full products, broadening the choice of building blocks for truncated products. In this paper we don't use this tool.

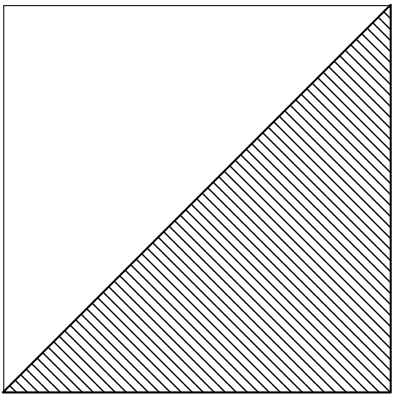


Fig. 11. Squaring as half product

In the **school multiplication** the terms $a_i b_j$ and $a_j b_i$ become the same, $a_i a_j$. All products are repeated, except the squares $a_i a_i$, which reduces the number of the necessary digit-multiplications (and so the time complexity) from n^2 to $n(n+1)/2$, almost twofold, as seen on Fig. 11.

Karatsuba multiplication cuts the operands into halves, and it does 5 additions and 3 multiplications on them to get the product. It has a recurrence relation $M(2n) = 3M(n) + 5cn$ (see [9], [14]), where the term $5cn$ comes from 5 additions of n-digit numbers. Squaring has a similar recurrence equation $S(2n) = 3S(n) + 4cn$. Unfolding the recurrence equation for the multiplication gives:

$$M(n) = 3M(n/2) + {}^5/_2\, cn = 9M(n/4) + {}^5/_2\, cn(1 + {}^3/_2) = \ldots$$
$$= 3^{\lg n} M(1) + 5cn(3^{\lg n}/2^{\lg n} - 1) \approx n^{\lg 3} M(1)$$

Similarly, $S(n) \approx n^{\lg 3} S(1)$. If school multiplication is performed below a certain operand length, like 8 digits, the recursion is followed only to this length. There the ratio $S(n)/M(n)$ is close to ½, which will be preserved for large n values. (Practical values are $S(32)/M(32) \approx 0.6$ and $S(64)/M(64) \approx 0.62$, with huge n values the ratio remaining below 0.65. [28])

Toom-Cook multiplications ([9], [14]) are similar. They, too, cut the operands into smaller pieces and build the product from them. At the bottom of the recursion asymptotically slower multiplication methods get faster, so we turn to them. Their relative speedup for squaring is approximately maintained to large operands.

12 Summary

The presented fast truncated multiplication algorithms help improving the performance of several cryptographic algorithms. (See the accompanying paper *Applications of Fast Truncated Multiplication in Cryptography*.) The most important results in this paper:

- Carry estimation and exact rounding algorithms for truncated products
- Proof of equivalent complexity of the LS and MS half products, within a linear term
- Fast truncated long integer multiplication algorithms (half products, middle third products, third quarter products) for Toom-Cook type multiplications.
- Finding the lengths of MS and LS truncated products, which can be faster computed than the full product
- Close to double speed squaring algorithms

References

[1] J.-C. Bajard, L.-S. Didier, and P. Kornerup. *An RNS Montgomery multiplication algorithm*. In 13th IEEE Symposium on Computer Arithmetic (ARITH 13), pp. 234–239, IEEE Press, 1997.

[2] P. D. Barrett, *Implementing the Rivest Shamir Adleman public key encryption algorithm on standard digital signal processor*, In Advances in Cryptology-Crypto'86, Springer-Verlag, 1987, pp.311-323.

[3] D. J. Bernstein, *Fast Multiplication and its Applications*, http://cr.yp.to/papers.html#multapps
[4] Bosselaers, R. Govaerts and J. Vandewalle, *Comparison of three modular reduction functions*, In Advances in Cryptology-Crypto'93, LNCS 773, Springer-Verlag, 1994, pp.175-186.
[5] E.F. Brickell. *A Survey of Hardware Implementations of RSA*. Proceedings of Crypto'89, Lecture Notes in Computer Science, Springer-Verlag, 1990.
[6] C. Burnikel, J. Ziegler, *Fast recursive division*, MPI research report I-98-1-022
[7] B. Chevallier-Mames, M. Joye, and P. Paillier. *Faster Double-Size Modular Multiplication from Euclidean Multipliers*, Cryptographic Hardware and Embedded Systems – CHES 2003, vol. 2779 of Lecture Notes in Computer Science,, pp. 214–227, Springer-Verlag, 2003.
[8] J.-F. Dhem, J.-J. Quisquater, *Recent results on modular multiplications for smart cards*, Proceedings of Cardis 1998, Volume 1820 of Lecture Notes in Computer Security, pp 350-366, Springer-Verlag, 2000
[9] GNU Multiple Precision Arithmetic Library manual http://www.swox.com/gmp/gmp-man-4.1.2.pdf
[10] W. Fischer and J.-P. Seifert. *Increasing the bitlength of crypto-coprocessors via smart hardware/software co-design*. Cryptographic Hardware and Embedded Systems – CHES 2002, vol. 2523 of Lecture Notes in Computer Science, pp. 71–81, Springer-Verlag, 2002.
[11] G. Hanrot, M. Quercia, P. Zimmermann, *The Middle Product Algorithm, I.* Rapport de recherche No. 4664, Dec 2, 2002 http://www.inria.fr/rrrt/rr-4664.html
[12] K. Hensel*, Theorie der algebraische Zahlen*. Leipzig, 1908
[13] J. Jedwab and C. J. Mitchell. *Minimum weight modified signed-digit representations and fast exponentiation*. Electronics Letters, 25(17):1171-1172, 17. August 1989.
[14] A. H. Karp, P. Markstein. *High precision division and square root*. ACM Transaction on Mathematical Software, Vol. 23, n. 4, 1997, pp 561-589.
[15] D. E. Knuth. *The Art of Computer Programming*. Volume 2. Seminumerical Algorithms. Addison-Wesley, 1981. Algorithm 4.3.3R
[16] W. Krandick, J.R. Johnson, *Efficient Multiprecision Floating Point Multiplication with Exact Rounding*, Tech. Rep. 93-76, RISC-Linz, Johannes Kepler University, Linz, Austria, 1993.
[17] A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.
[18] P.L. Montgomery, "*Modular Multiplication without Trial Division*," Mathematics of Computation, Vol. 44, No. 170, 1985, pp. 519-521.
[19] T. Mulders. *On computing short products*. Tech Report No. 276, Dept of CS, ETH Zurich, Nov. 1997 http://www.inf.ethz.ch/research/publications/data/tech-reports/2xx/276.pdf
[20] P. Paillier. *Low-cost double-size modular exponentiation or how to stretch your cryptoprocessor.* Public-Key Cryptography, vol. 1560 of Lecture Notes in Computer Science, pp. 223–234, Springer-Verlag, 1999.
[21] K. C. Posh and R. Posh. *Modulo reduction in Residue Number Systems*. IEEE Transactions on Parallel and Distributed Systems, vol. 6, no. 5, pp. 449–454, 1995.
[22] J.-J. Quisquater, *Fast modular exponentiation without division*. Rump session of Eurocrypt'90, Arhus, Denmark, 1990.
[23] R. L. Rivest; A. Shamir, and L. Adleman. 1978. *A method for obtaining digital signatures and public key cryptosystems*. Communications of the ACM 21(2):120--126

[24] J. Schwemmlein, K.C. Posh and R. Posh. *RNS modulo reduction upon a restricted base value set and its applicability to RSA cryptography*. Computer & Security, vol. 17, no. 7, pp. 637–650, 1998.
[25] SNIA OSD Technical Work Group http://www.snia.org/tech_activities/workgroups/osd/
[26] C.D. Walter, "*Faster modular multiplication by operand scaling*," Advances in Cryptology, Proc. Crypto'91, LNCS 576, J. Feigenbaum, Ed., Springer-Verlag, 1992, pp. 313—323
[27] L. Hars, "*Long Modular Multiplication for Cryptographic Applications*", CHES 2004, (Misprinted in LNCS 3156, pp 44-61. Springer, 2004.) http://eprint.iacr.org/2004/198/
[28] L. Hars, "*Finding the Fastest Multiplication for Cryptographic Operand Lengths: Analytic and Experimental Comparisons*" manuscript.

Using an RSA Accelerator for Modular Inversion

Martin Seysen

Giesecke & Devrient GmbH, Prinzregentenstraße 159,
D-81677 Munich, Germany
Martin.Seysen@de.gi-de.com

Abstract. We present a very simple new algorithm for modular inversion. Modular inversion can be done by the extended Euclidean algorithm. We substitute the extended Euclidean algorithm by a standard (non-extended) Euclidean algorithm that works on integers of approximately double the length of the modulus. This substitution can be very useful on smart card coprocessors, since in some cases computations with longer numbers than necessary can be done at no extra cost. Many smart card coprocessors have been designed for the RSA algorithm of, say, 1024 bits length. On the other hand, elliptic curve algorithms work with much smaller numbers, and modular inversion is a much more important primitive in elliptic curve cryptography than in RSA cryptography. On one smart card coprocessor the new algorithm is more than twice as fast as the classical algorithm.

Keywords: Smart card coprocessor, modular inversion, Euclidean algorithm.

1 Introduction

When public key cryptography was first used in low-cost devices such as smart cards, it turned out that the standard CPU of such a device is too slow to perform the necessary cryptographic operations. At that time algorithms that work in the multiplicative group modulo a long integer such as RSA or Diffie-Hellman were the most popular public key algorithms. Several coprocessors for performing operations on long integers have been designed. Most of them have been optimised for modular multiplications of $512, \ldots, 1024$ bit length, as required for public key algorithms in the early nineties of the last century. A variety of different modular multiplication techniques has been used in such coprocessors, see [12,14,15,19]. For an overview of modular multiplication techniques see [8].

Depending on the architecture, such a coprocessor performs either operations modulo numbers of a fixed (maximum) bit length, or the bit length is limited only by memory resources. Since the required key size for public key systems increases during the years (see e.g. [10]), coprocessors must deal with larger numbers than they have been designed for. Solutions for this problem have been given in [4,13,2].

When elliptic curve cryptography became more popular on smart cards in the late nineties, implementers had to deal with the opposite problem. Here the key

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 226–236, 2005.

size is typically in the range of $160, \ldots, 256$ bits, which is much smaller than the key size in the RSA algorithm. This means that coprocessors have to deal with much smaller numbers than they have been designed for. Another problem is that some coprocessors are optimised for modular multiplication, as required for the RSA algorithm. But other operations such as modular addition, subtraction and inversion are more important in EC operations than in the RSA algorithm. Here we mainly consider elliptic curves over GF(p).

This paper deals with the modular inversion on smart card coprocessors. The oldest algorithm for computing the modular inverse is the extended Euclidean algorithm for computing the greatest common divisor (gcd), see e.g. [8]. A binary algorithm for computing the gcd has been proposed in [18]. Modular inversion techniques have been investigated under a variety of aspects. The classical gcd algorithms have been improved and optimised for large numbers on a CPU with a fixed word length, see e.g. [6,9,16,17]. A hardware-optimised modular inversion algorithm has been proposed in [11]. Modular inversion algorithms that avoid the computation of the greatest common divisor have been proposed in [7].

While modular multiplication is hardware supported by a coprocessor on many smart cards, modular inversion usually has to be coded in software. On some CPUs a modular inversion may be about 100 times slower than a modular multiplication.

Therefore it is crucial to use as little modular inversions as possible in EC operations on smart cards. Techniques for reducing the number of modular inversions in EC operations are well-known, see e.g. [1] or [5] for an overview. Here the basic idea is to avoid modular inversions by representing the points on a curve in projective or Jacobian projective co-ordinates instead of affine co-ordinates. We can also used mixed co-ordinate systems to reduce the number of arithmetic operations. These techniques can be combined with windowing methods to reduce the number of EC operations. Depending on the details of the implementation, more inversions may be necessary in this case, see [3].

One of the most well-known EC algorithms is the ECDSA algorithm. Note that apart from the elliptic curve operations, the ECDSA signing algorithm requires another modular inversion operation modulo the group order.

When running the ECDSA signing algorithm with $160, \ldots, 192$ bit length on a smart card coprocessor, then up to about 20 percent of the run time may be spent for modular inversions.

Therefore any significant speedup of the modular inversion algorithm on a smart card coprocessor has a non-negligible effect on the run time of the ECDSA signing algorithm.

We propose a very simple new modular inversion algorithm that is specially suited for a coprocessor optimised for RSA cryptography. The new algorithm performs about half the number of operations compared to the classical extended Euclidean algorithm, but it works with double-length numbers. As we shall see, the speed of the new algorithm is more than doubled compared to the standard implementation of the modular inversion on the Infineon SLE66CX322P CPU. The main reason for this speedup is the fact that basic double-length integer op-

erations such as addition, subtraction and shifting, are not much more expensive than single-length operations on this CPU for key sizes of $160, \ldots, 256$ bits.

The new algorithm may also be useful on other smart card CPUs, since many of these CPUs combine a highly optimised coprocessor for long-integer arithmetic with a main processor of much less performance. Therefore instructions such as register switches, loop control, pointer management or data transfer between main processor and coprocessor, (called glue instructions in [7],) may take up a considerable part of the run time of a modular inversion algorithm. Obviously, less glue instructions are necessary, when fewer operations are performed on longer integers.

2 Modular Inversion with a Non-extended Euclidean Algorithm

In this paper we will show the correctness of the following modular inversion algorithm:

Algorithm NINV. *(Modular inversion with a non-extended Euclidean algorithm)*

`Input`	*Integers* $u \geq 0$, $v > 1$, *and an arbitrary extension factor* f *with* $f > 2v$.
`Output`	*Modular inverse* $x = u^{-1} \pmod{v}$ *with* $-v < x < v$, *or an error if* u *is not invertible modulo* v.
[1]	`Put` $U = fu + 1, V = fv$.
[2]	`While` $V \geq f + v$ `do` $\{T = V, V = U \bmod V, U = T\}$.
[3]	`If` $V > f - v$ `then return` $V - f$ `and stop` , `else return` *"error"* `and stop` .

Throughout the paper we write $u \bmod v$ for the integer x satisfying $x = u \pmod{v}$, $0 \leq x < v$.

For the analysis of Algorithm NINV we may rephrase Step 2 as follows:

$$U_0 = U = fu + 1\ ,\ V_0 = V = fv\ ;$$
$$U_{i+1} = V_i\ , \quad V_{i+1} = U_i \bmod V_i\ ; \quad \text{for } i \geq 0\ ,\ V_i > 0\ . \tag{1}$$

Note that this is exactly the process of the Euclidean algorithm applied to U and V. The number of Euclidean steps in Algorithm NINV is the about the same as the number of Euclidean steps in the standard Euclidean algorithms applied to u and v. See [8] for an analysis of the number of Euclidean steps required in the Euclidean algorithm. The extension factor f can be chosen as a power of two such that the bit length of the numbers U and V is about twice the bit length of $\max(u, v)$. So Algorithm NINV requires roughly the same number of operations as the standard (non-extended) Euclidean algorithm, but it has to work with

integers of double length. On the other hand the overhead for the 'extended' part of the Euclidean algorithm is saved. Depending on the details of the smart card CPU, this may lead to considerable saving of glue instructions.

Before showing the correctness of the algorithm, we first give some motivation why the algorithm is expected to work. The Euclidean algorithm, when applied to u and v returns a number $d = \gcd(u, v)$. It is well known that the algorithm can be extended to compute integers λ, μ, $|\lambda| < v$, with $d = \lambda u - \mu v$. In case $d = 1$ the number λ is just the modular inverse we are looking for. When we apply the Euclidean process to $\tilde{U} = fu$ and $V = fv$ instead of u and v, we obtain exactly the same sequences of quotients in the modular reduction operations in both cases, and we also obtain a linear combination $\gcd(\tilde{U}, V) = fd = \lambda\tilde{U} - \mu V$ with the same values λ, μ as above. Now we introduce a small perturbation by replacing $\tilde{U}$ by $U = \tilde{U} + 1$. Assume for a moment that this perturbation does not change the sequence of quotients in the Euclidean process. Then a remainder $\lambda U - \mu V = fd + \lambda$ would occur in the sequence (V_i) of remainders in the Euclidean process, and in case $d = 1$ we could simply read the modular inverse λ from a remainder of size $\approx f$. Although the above assumption is in general not true, the construction of Algorithm NINV is based on this idea. More specifically, the following theorem proves the correctness of Algorithm NINV.

Theorem 1. *In case* $\gcd(u, v) = 1$ *there is an integer* i *such that* $V_{i-1} > 2f - v$, $f + v > V_i > f - v$ *and* $(V_i - f) \cdot u = 1 \pmod{v}$ *hold. Otherwise every* V_i *satisfies either* $V_i > 2f - v$ *or* $V_i \leq \frac{v}{2}$.

The proof of the theorem uses continued fractions. Some basic facts about continued fractions are stated in the next section. Theorem 1 is proved in section 4.

The relation between the standard extended Euclidean algorithm and Algorithm NINV can be stated as follows. The standard algorithm computes integers $u_i, v_i, \lambda_i, \lambda'_i$ with $\lambda_i u \equiv u_i$, $\lambda'_i u \equiv v_i \pmod{v}$, starting with $u_0 = u$, $v_0 = v$, $\lambda_0 = 1$, $\lambda'_0 = 0$; and it stops when arriving at some v_i with $v_i = \gcd(u, v)$. Algorithm NINV stores the numbers $U_i = fu_i + \lambda_i$ and $V_i = fv_i + \lambda'_i$ in two double-length variables instead. Note that some of the perturbations λ_i, λ'_i can be negative and may spoil the Euclidean quotients. This means that we may have $\lfloor U_i/V_i \rfloor \neq \lfloor u_i/v_i \rfloor$ in some cases, regardless of the size of f. Therefore we have to modify the analysis of the standard extended Euclidean algorithm to obtain a proof of Theorem 1.

3 Continued Fractions

Algorithm NINV can most easily be analysed in terms of continued fractions. We take notation for and standard facts about continued fractions from [8]. A continued fraction $/\!/x_1, x_2, \ldots, x_{n-1}, x_n/\!/$ is defined by:

$$/\!/x_1, x_2, \ldots, x_{n-1}, x_n/\!/ = 1/(x_1 + 1/(x_2 + 1/(\ldots(x_{n-1} + 1/x_n)\ldots))).$$

Continued fractions are closely related to the so-called continuant polynomials $K_n(x_1, \ldots, x_n)$ defined by:

$$K_n(x_1, \ldots, x_n) = \begin{cases} 1 & \text{If } n = 0; \\ x_1 & \text{If } n = 1; \\ x_1 K_{n-1}(x_2, \ldots, x_n) + K_{n-2}(x_3, \ldots, x_n) & \text{If } n > 1. \end{cases}$$

Then the following explicit formulas hold for continued fractions:

$$/\!/x_1, x_2, \ldots, x_n/\!/ = \frac{K_{n-1}(x_2, \ldots, x_n)}{K_n(x_1, x_2, \ldots, x_n)} \tag{2}$$

$$x_0 + /\!/x_1, x_2, \ldots, x_n/\!/ = \frac{K_{n+1}(x_0, x_1, \ldots, x_n)}{K_n(x_1, x_2, \ldots, x_n)} \tag{3}$$

Every real number X has a (regular) continued fraction expansion defined as follows: Let $A_0 = \lfloor X \rfloor, X_0 = X - A_0$, and for all $n \geq 0$ such that $X_n \neq 0$ define:

$$A_{n+1} = \lfloor 1/X_n \rfloor, X_{n+1} = 1/X_n - A_{n+1}. \tag{4}$$

For a rational number X there is an integer n such that A_{n+1} is not defined; and the continued fraction expansion of X is:

$$X = A_0 + /\!/A_1, \ldots, A_n/\!/. \tag{5}$$

For an irrational number X, an infinite continued fraction expansion can be defined.

For a given X, we define quantities p_i, q_i by:

$$\begin{aligned} p_{-1} = 1, q_{-1} = 0, p_0 = A_0, q_0 = 1, \\ p_{i+1} = A_i p_i + p_{i-1},\ q_{i+1} = A_i q_i + q_{i-1} \quad ;\ i = 0, \ldots, n-1\ . \end{aligned} \tag{6}$$

By induction over i we easily obtain:

$$p_i = K_{i+1}(A_0, \ldots, A_i) \quad ;\ i = 0, \ldots, n\ ; \tag{7}$$

$$q_i = K_i(A_1, \ldots, A_i) \quad ;\ i = 0, \ldots, n\ ; \tag{8}$$

$$\frac{p_i}{q_i} = A_0 + /\!/A_1, \ldots, A_i/\!/ \quad ;\ i = 1, \ldots, n\ . \tag{9}$$

For a number X, assume that in its continued fraction expansion $A_0 + /\!/A_1, A_2, \ldots/\!/$ the term A_{i+1} exists. Then the following facts are well-known, see e.g. [8].

The number X lies between p_i/q_i and p_{i+1}/q_{i+1}, and we have:

$$q_{i+1} > q_i \quad ; \quad |p_i q_{i+1} - p_{i+1} q_i| = 1\ ; \tag{10}$$

$$\left| X - \frac{p_i}{q_i} \right| \leq \frac{1}{q_i q_{i+1}} < \frac{1}{q_i^2}\ . \tag{11}$$

The last fact means that the rational numbers p_i/q_i are very close rational approximations for the number X. They are called the convergents of (the regular continued fraction expansion for) X.

The following classical result is due to A. M. Legendre:

Theorem 2. *If a rational number p/q with $p \in \mathbb{Z}, q \in \mathbb{N}$, satisfies*

$$\left| X - \frac{p}{q} \right| \leq \frac{1}{2q^2} ,$$

then p/q is a convergent of the regular continued fraction expansion for X.

We still need another fact about continued fractions:

Proposition 3. *Assume that μ_1/λ_1 and μ_2/λ_2 are two convergents of the regular continued fraction expansion for X satisfying $|\mu_1\lambda_2 - \mu_2/\lambda_1| = 1$. Then these two convergents are either adjacent in the sequence of convergents for X, or there is exactly one more convergent $(\mu_2 - \mu_1)/(\lambda_2 - \lambda_1)$ between them.*

Sketch Proof. W.l.o.g. we assume $\lambda_1 < \lambda_2$. The Diophantine equation $|\mu_2 x - \lambda_2 y| = 1$ has exactly two solutions (x, y) satisfying $0 \leq x < \lambda_2$, and we easily identify these two solutions as (λ_1, μ_1) and (λ_3, μ_3), with $\lambda_3 = \lambda_2 - \lambda_1$, $\mu_3 = \mu_2 - \mu_1$. Assume that μ_1/λ_1 and μ_2/λ_2 are not adjacent. Then by (10), μ_3/λ_3 is the convergent preceding μ_2/λ_2. Now let μ_4/λ_4 be the convergent preceding μ_3/λ_3. By (6) there is an integer $A \geq 1$ with $\lambda_2 = A\lambda_3 + \lambda_4$. Hence

$$\lambda_4 \leq \lambda_2 - \lambda_3 = \lambda_1 .$$

But since by assumption μ_4/λ_4 does not precede μ_1/λ_1 in the series of convergents, these two quantities are equal. □

4 Proof of Theorem 1

Applying the Euclidean Algorithm to U and V as stated in (1) corresponds in a natural way to the regular continued fraction expansion of the rational number $X = U/V$.

Define:

$$\bar{A}_i = \lfloor U_i/V_i \rfloor ,$$
$$\bar{X}_i = U_i/V_i - \bar{A}_i = (U_i \bmod V_i)/V_i = V_{i+1}/U_{i+1} \; ; \quad i \geq 0, V_i > 0 . \tag{12}$$

Comparing (4) with (1) and (12) we see that the quantities $\bar{A}_i, \bar{X}_i$ defined in (12) are exactly the numbers A_i, X_i that appear in the continued fraction expansion of $X = U/V$.

Regarding the quantities p_i and q_i defined by (6), and using induction over i, we obtain from (1) and (12):

$$q_i U - p_i V = (-1)^i V_{i+1} \quad ; \; i \geq -1 . \tag{13}$$

Define $d = \gcd(u, v)$. There are coprime integers λ, μ with

$$\lambda u - \mu v = d . \tag{14}$$

We now assume that $u \neq 0 \pmod{v}$ holds in Algorithm NINV. Otherwise we have $U \bmod V = 1$ and the algorithm terminates with an error as expected.

Since $v \neq 0, u \neq 0 \pmod{v}$, we have $0 < d < v$. Hence $\lambda \neq 0 \bmod v$ holds and λ and μ cannot have opposite signs. So we have $\mu/\lambda \geq 0$ in all cases. From (1) we obtain:

$$\left|\frac{|\mu|}{|\lambda|} - \frac{U}{V}\right| = \left|\frac{\mu}{\lambda} - \frac{U}{V}\right| = \left|\frac{fd+\lambda}{f\lambda v}\right| \quad ; \ \lambda \neq 0 \pmod{v} \, ; \ \frac{\mu}{\lambda} \geq 0 \, . \tag{15}$$

In (14) we may replace λ and μ by $\lambda + kv/d$ and $\mu + ku/d$ for any integer k.

Our goal is now to find values λ and μ with $|\mu/\lambda - U/V| < 1/(2\lambda^2)$ so that we may apply Theorem 2. A sufficient condition for this is:

$$-\frac{1}{2}df + \frac{1}{2}\sqrt{(df)^2 - 2fv} < \lambda < -\frac{1}{2}df + \frac{1}{2}\sqrt{(df)^2 + 2fv} \quad ; \ f \geq 2v \, . \tag{16}$$

We easily check that length of the feasible interval for λ given by (16) is always greater than v/d. So we can always find integers $\lambda, \mu, \mu/\lambda \geq 0$ satisfying (14) and $|\mu/\lambda - U/V| < 1/(2\lambda^2)$. Thus by Theorem 2 we obtain:

Lemma 4. *There exist coprime integers λ, μ satisfying $\lambda u - \mu v = \gcd(u, v)$ such that $|\mu/\lambda|$ is a convergent of the regular continued fraction expansion of U/V.*

Let λ_{max} be the highest possible value for λ that satisfies Lemma 4. We want to calculate an upper and a lower bound for λ_{max}. By (11) and (15), λ_{max} must satisfy:

$$\left|\frac{fd + \lambda_{max}}{f\lambda_{max} v}\right| < \frac{1}{\lambda_{max}^2} \, .$$

This implies:

$$\lambda_{max} < v/d \, . \tag{17}$$

On the other hand, every interval of length at least v/d must contain an integer solution λ to the Dipohantine equation $\lambda u - \mu v = d$. Since the feasible interval for λ given in (16) is sufficiently long, we conclude:

$$\lambda_{max} > \left(-\frac{1}{2}df + \frac{1}{2}\sqrt{(df)^2 + 2fv}\right) - \frac{v}{d} \ > \ -\frac{v}{2d} - \frac{v^2}{4d^3 f} \ > \ -\frac{5v}{8d} \, . \tag{18}$$

Now we choose once and for all the convergent μ/λ satisfying Lemma 4 such that λ is maximal, i.e. $\lambda = \lambda_{max}$.

We have $|u/v - U/V| = 1/V < 1/(2v^2)$, so that by Theorem 2 the rational number $(u/d)/(v/d)$ is a convergent for U/V too. Furthermore, we have $\lambda u/d - \mu v/d = 1$. Thus Proposition 3 implies:

Lemma 5. *Either $\frac{|\mu|}{|\lambda|}, \frac{u/d}{v/d}$ or $\frac{|\mu|}{|\lambda|}, \frac{u/d - |\mu|}{v/d - |\lambda|}, \frac{u/d}{v/d}$ are adjacent convergents of the regular continued fraction expansion of U/V.*

For the subsequent convergents p_i/q_i given by Lemma 5, we can now use (13) to calculate the corresponding values V_{i+1}; and we will use (17) and (18) to obtain upper and lower bounds for the values V_{i+1}. These values are part of the sequence V_i defined by (1). The result is given in Table 1. Note that the second entry in Table 1 corresponding to the convergent $\frac{u/d-|\mu|}{v/d-|\lambda|}$ may or may not be present in the sequence of convergents.

Table 1. Bounds for the intermediate results $V_{i+1} = (-1)^i(q_iU - p_iV)$ depending on the convergents p_i/q_i

numerator p_i	denominator q_i	V_{i+1}	lower bound for V_{i+1}	upper bound for V_{i+1}
$\|\mu\|$	$\|\lambda\|$	$fd + \lambda$	$fd - \frac{5}{8}v/d$	$fd + v/d$
$u/d - \|\mu\|$	$v/d - \|\lambda\|$	$fd + \lambda - v/d$	$fd - \frac{13}{8}v/d$	fd
u/d	v/d	v/d	v/d	v/d

Proof of Theorem 1

Case 1: $d = \gcd(u, v) > 1$

In this case Table 1 shows that no V_i with $\frac{v}{2} \le V_i \le 2f - \frac{13}{16}v$ exists.

Case 2: $d = 1$

The first entry of Table 1 shows that a V_{i+1} with $f - \frac{5}{8}v < V_{i+1} < f + v$ and $V_{i+1} = f + \lambda$ exists. From (14) we conclude that λ is a modular inverse of u modulo v. Let V_j be the entry of the sequence $V_{i+1}, i = 0, 1, \ldots$ corresponding to the first entry of the table. We still have to show that the entry V_{j-1} preceding V_j is sufficiently large. By (1) there is an integer $A \ge 1$ such that we have:

$$V_{j-1} = AV_j + V_{j+1} \; . \tag{19}$$

Here V_{j+1} corresponds either to entry two or three of Table 1.

Case 2a: V_{j+1} corresponds to entry two of Table 1.

Then there is exactly one more convergent between the convergents $|\mu/\lambda|$ and u/v of U/V. So we conclude from (13) that $|\lambda|U - |\mu|V$ and $vU - uV = v > 0$ have the same sign. By (1) and (14) we have $\lambda U - \mu V = f + \lambda > 0$. Thus λ must be positive in this case. Then by (19) we have:

$$V_{j-1} \ge V_j + V_{j+1} = 2f + 2\lambda - v > 2f - v.$$

Case 2b: V_{j+1} corresponds to entry three of Table 1 and $A > 1$ holds in (19).

Then we have:

$$V_{j-1} \geq 2V_j + V_{j+1} \geq 2f + 2\lambda + v > 2f - v \; .$$

Case 2c: V_{j+1} corresponds to entry three of Table 1 and $A = 1$ holds in (19).

In this case we will show that the algorithm outputs V_{j-1} and that V_{j-2} is sufficiently large. Since $|\mu/\lambda|$ and u/v are adjacent convergents of U/V, we conclude from (13) that $|\lambda|U - |\mu|V$ and $vU - uV = v > 0$ have opposite sign. By (1) and (14) we have $\lambda U - \mu V = f + \lambda > 0$. Thus λ must be negative in this case. Now we compute:

$$V_{j-1} = V_j + V_{j+1} = f + \lambda + v < f + v \; ;$$
$$V_{j-2} \geq V_{j-1} + V_j = 2f + 2\lambda + v > 2f - v \; .$$

From this we see that $V_{j-1} - f = \lambda + v$ is a modular inverse of u modulo v of appropriate size, and that the predecessor V_{j-2} of V_{j-1} is sufficiently large. □

5 Implementation Results

Algorithm NINV has been implemented on the Infineon SLE66CX322P CPU. There the Advanced Crypto Engine (ACE), an arithmetic coprocessor for long integer arithmetic, is used for elliptic curve operations. The coprocessor provides an arithmetic unit that operates with numbers of up to 1120 bit length in long mode and 560 bit in short mode. In principle, the coprocessor always performs elementary operations such as additions, subtractions or shifts on full-length registers. In elliptic curve cryptography, we hardly ever use numbers of more than 512 bit length, so that the overhead for operating with double-length numbers is marginal.

In the Table 2 we compare the run time of Algorithm NINV with the run time of the standard extended Euclidean algorithm for modular inversion as provided by the manufacturer. In both cases the PLL of the CPU runs in an asynchronous mode with maximum possible frequency.

The table shows that Algorithm NINV is more than twice as fast as the standard modular inversion algorithm on the SLE66CX322P CPU.

Our implementation of Algorithm NINV chooses an extension factor $f = 3 \cdot 2^k$, for some integer k with $2^k > v$. Then we have $2f - v > 2^{k+2} > f + v$ and $f - v > 2^{k+1}$, so that it suffices to check the bit length of V in steps 2 and 3 of the algorithm. This trick saves some overhead on the coprocessor.

Table 2. Run time of the standard extended Euclidean algorithm and of Algorithm NINV in milliseconds for various bit lengths

Modular inversion algorithm	160 bit	192 bit	256 bit	320 bit
Extended Euclidean	4.80 ms	5.73 ms	7.46 ms	9.16 ms
Algorithm NINV	2.09 ms	2.43 ms	3.16 ms	4.45 ms

Algorithm NINV does not specify a modular reduction method. Both implementations listed above perform an integer division by using a simple binary subtract-and-correct method. They discard the quotient and keep the remainder. This is adequate here, since the ACE coprocessor can do long integer additions, subtractions and shifts very fast, and the Euclidean quotients are usually quite small, see [8].

We have not implemented any of the optimised gcd algorithms [6,9,16] on the SLE66CX322P CPU, since they have been designed for long integers that do not fit into a single CPU register.

In principle, Lehmer's variant [9] of the Euclidean algorithm for long integers can be used to compute the modular inverse in the same way as in Algorithm NINV, since it produces the same quotients as the original Euclidean algorithm, see [8].

6 Conclusion

In practice, many elliptic curve implementations are running on smart card coprocessors that have been designed for RSA cryptography. Taking into account this special design of the Infineon Advanced Crypto Engine (ACE), we have more than doubled the speed of modular inversions used in EC cryptography. This speedup is so significant that it has an observable effect on the speed of some EC algorithms, such as the ECDSA signing procedure.

References

1. I.F. Blake, G. Seroussi, N.P. Smart, *Elliptic Curves in Cryptography,* vol. 265 of *London Mathematical Society Lecture Note Series.* Cambridge University Press, 1999.
2. B. Chevallier-Mames, N. Joye, P. Paillier, Faster Double-Size Modular Multiplication from Euclidean Multipliers. *C.D. Walter, Ç.K. Koç, and C. Paar (Eds.): Cryptographic Hardware and Embedded Systems CHES 2003*, Springer LNCS vol. 2779, (2003), pp. 214-227.
3. H. Cohen, A. Miyaji, T. Ono, Efficient Elliptic Curve Exponentiation using Mixed Coordinates. *K. Ohta and D. Pei (Eds.): Advances in Cryptology - ASIACRYPT '98*, Springer LNCS vol. 1514 (1998), pp. 51-65.
4. W. Fischer, J.-P. Seifert, Increasing the Bitlength of a Crypto-coprocessor. *B.S. Kaliski Jr, Ç.K. Koç, and C. Paar (Eds.): Cryptographic Hardware and Embedded Systems CHES 2002*, Springer LNCS vol. 2523 (2002), pp. 71-81.
5. D. Hankerson, A. Menezes, S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer Verlag, 2004.
6. T. Jebelean, A Generalization of the Binary GCD Algorithm. *M. Bronstein (Ed.), 1993 ACM International Symposium on Symbolic and Algebraic Computation, Kiev, Ukraine*, ACM Press (1993), pp. 111-116.
7. M. Joye, P. Paillier, GCD-Free Algorithms for Computing Modular Inverses. *C.D. Walter, Ç.K. Koç, and C. Paar (Eds.): Cryptographic Hardware and Embedded Systems CHES 2003*, Springer LNCS vol. 2779, (2003), pp. 243-253.

8. D.E. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms,* 3rd ed., Addison-Wesley, 1997.
9. D.H. Lehmer, Euclid's Algorithm for Large Numbers. *American Mathematical Monthly 45*, (1938) pp. 227-233.
10. A.K. Lenstra, E.R. Verheul, Selecting Cryptographic Key Sizes. *J. Cryptology 14 No 4*, (2001) pp. 255-293.
11. R. Lórencz, New Algorithm for Classical Modular Inverse. In *B.S. Kaliski Jr, Ç.K. Koç, and C. Paar (Eds.): Cryptographic Hardware and Embedded Systems CHES 2002* Springer LNCS vol. 2523, (2003), pp. 57-70.
12. K. Nakada, *Data Processor and Microcomputer,* US Patent No. 5,961,578, Oct. 5, 1999.
13. P. Paillier, Low-Cost Double-Size Modular Exponentiation or How to Stretch Your Cryptoprocessor. *H. Imai, Y. Zheng (Eds.): Public-Key Cryptography*, Springer LNCS vol. 1560 (1999), pp. 223-234.
14. J.-J. Quisquater, *Encoding System according to the so-called RSA Method, by means of a Microcontroller and Arrangement Implementing this System.* US Patent No. 5,166,978, Nov 24,1992.
15. H. Sedlak, The RSA Cryptography Processor. *Proc. of EUROCRYPT '87*, Springer LNCS vol. 293 (1987), pp. 95-105.
16. J.P. Sorenson, Two Fast GCD Algorithms. *Journal of Algorithms, 16*, (1994), pp. 110-144.
17. J.P. Sorenson, An Analysis of the Generalized Binary GCD Algorithm. `http://euclid.butler.edu/~sorenson/papers/genbin.pdf`.
18. J. Stein, Computational Problems Associated with Racah Algebra. *Journal of Computational Physics 1*, (1967), pp. 397-405.
19. D. Symes, D.J. Seal, *A System for Performing Modular Multiplication.* UK Patent GB 2352309 A, Jan 24, 2001.

Comparison of Bit and Word Level Algorithms for Evaluating Unstructured Functions over Finite Rings

B. Sunar* and D. Cyganski

Department of Electrical & Computer Engineering,
Worcester Polytechnic Institute,
Worcester, Massachusetts 01609, USA
{sunar, cyganski}@wpi.edu

Abstract. We study the problem of implementing multivariate functions defined over finite rings or fields as parallel circuits. Such functions are essential for building cryptographic substitution boxes and hash functions. We present a modification to Horner's algorithm for evaluating arbitrary n-variate functions defined over finite rings and fields. Our modification is based on eliminating redundancies in the multivariate version of Horner's algorithm which occur when the evaluation takes place over a small finite mathematical structure and may be considered as a generalization of Shannon's lower bound and Muller's algorithm to word level circuits. If the domain is a finite field $GF(p)$ the complexity of multivariate Horner polynomial evaluation is improved from $O(p^n)$ to $O(\frac{p^n}{2n})$. We prove the optimality of the presented algorithm. Our comparison of the bit level approach to the optimized word level approach yields an interesting result. The bit level algorithm is more efficient in both area consumption and time delay. This suggests that unstructured functions over finite rings or fields should be implemented using the bit-level approach and not the commonly used word level implementation style.

Keywords: Horner's method, polynomial evaluation, multivariate polynomials, word level, finite fields.

1 Introduction

Essentially all secret and public key schemes are based on arbitrary looking and highly nonlinear logic function families which are indexed by a fixed length key. In fact, it is expected that a proper cryptographic function is practically indistinguishable from a function that is randomly picked from the set of all functions of certain degree. This arbitrariness requires cryptographers to propose highly complex and structureless functions. A few examples are round subfunctions of

* Berk Sunar is funded by National Science Foundation research grants CAREER award ITR-ANI-0133297 and ITR-ANI-0112889.

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 237–249, 2005.

hash algorithms (e.g. SHA-1 [6], MD5 [11] etc.), and substitution boxes in block and stream ciphers (e.g. DES [7], AES [8] etc.), or public-key schemes defined over polynomial rings or finite fields. Earlier constructions were given in terms of lookup tables which were built by experimentation and extensive statistical testing. A good example is DES where the substitution boxes have no publicly known structure. Therefore, one is left with a choice of either using costly lookup tables or a direct logic implementation. Although less regular and more difficult to design, the latter choice tends to be much more efficient. Recently proposed algorithms (e.g. AES, and the 3GPP Standard's Kasumi block cipher [13]) have substitution boxes that may be expressed as algebraic functions (e.g. as inversion in a binary finite field). This makes it possible to use the algebraic structure (e.g. use composite or tower field representation) to have more efficient substitution computations. Another good example is algebra on elliptic curves defined over optimal extension fields (e.g. $GF(p^k)$ where typically p fits into a word and is of pseudo-Mersenne form). Such arithmetic is commonly implemented in two levels: polynomial arithmetic to implement the operations in the field extension, and then $GF(p)$ arithmetic to support the coefficient operations.

In this context we pose the following question: Assume we are given a multivariate function (or polynomial) f of fixed degree defined over a finite ring Z_n (or field $GF(p)$) randomly picked[1] from the set of all such functions. What is the best choice for a parallel circuit implementation of f:

1. a binary logic realization of the function f (bit-level approach), or
2. a realization of f over Z_n (or $GF(p)$) (word-level approach)?

Intuitively, the later (word level) approach is attractive since it allows one to use the present mathematical structure in the implementation. However, since any circuit contructed from optimal binary logic implementations of word level function blocks can be re-optimized at the binary operator level as a whole, the potential of significant size advantage of the latter may overwhelm the design convenience of the former. In this paper we will obtain a measure of the relative merits of optimal binary versus optimal word level implementations for unstructured functions defined over a finite field or ring.

The remainder of this paper is as follows. We present a brief background on multivariate function (or polynomial) evaluation in the next section and introduce related notation and the multivariate formulation of Horner's algorithm. Then a generalization of Horner's algorithm to higher characteristic fields and rings is developed. We also present specialized optimization techniques based on the field/ring structure. The paper concludes by comparing the bit-level algorithm with the word-level approach in terms of bit-complexities.

[1] It is important to note that, the functions we use in cryptography are not always random and thus may permit more efficient computation. The work in this paper focuses on arbitrarily chosen functions and hence may be more suitable for the implementation of substitution boxes defined over finite rings or fields.

2 Background

The problem of function (or polynomial) evaluation has a rich history. The prominent polynomial evaluation algorithm attributed to Horner [3] has found its way into many applications due to its simplicity and efficiency. The univariate Horner's algorithm was shown by Ostrowski [9] to be optimal in the number of additions in the straight line algorithm in 1954. More than a decade later in 1966 Pan [10] proved its optimality in the number of multiplications as well. Furthermore, in [1] Borodin proved the uniqueness of Horner's algorithm, i.e. that all algorithms of similar complexity reduce to Horner's algorithm, as initially conjectured by Ostrowski.

Although Horner's algorithm is optimal for evaluating polynomials with arbitrary coefficients there are more efficient algorithms for evaluating polynomials by allowing precomputation on the coefficients. For example, the polynomial $p(x) = x^n$ can be computed by using only $\log_2 n$ multiplications and no additions. Similar algorithms can be derived for polynomials with less structure. In fact, when precomputation is allowed the evaluation can be achieved using only about $\frac{n}{2}$ multiplications [4].

Using the multivariate version of Horner's method it is possible to efficiently implement boolean functions. In an early result Shannon proved [12] a lower bound as $O(\frac{2^n}{n})$ on the size of circuits implementing arbitrary boolean functions of n variables. Optimally solving this problem, Muller gave an explicit construction based on a modification on Horner's method which satisfies the lower bound[5].

3 Preliminaries

We consider the problem of evaluating multivariate polynomials over Z_m using Horner's method. In the univariate case a polynomial of degree $r-1$ over Z_m is represented as

$$u(x) = u_0 + u_1 x + u_2 x^2 + \ldots + u_{r-1} x^{r-1} \quad , \quad u_i \in Z_m \ .$$

The most general polynomial one need consider is such that $r = order(Z_m)$ as any other polynomial may be reduced to this. A naive evaluation of $u(x)$ would require $r-1$ additions and $2r-3$ multiplications in Z_m. By the application of Horner's method, however, the following paranthesization is obtained.

$$u(x) = u_0 + x(u_1 + x(u_2 + x(u_3 + \ldots + x(u_{r-2} + xu_{r-1}))\ldots)$$

Now the polynomial can be evaluated by computing only $r-1$ additions and $r-1$ multiplications without using any temporary storage. A parallel implementation introduces a delay of

$$T = (r-1)T_A + (r-1)T_M$$

where T_A and T_M denote the delay of two input addition and multiplication operations in Z_m.

4 The Multivariate Case

A multivariate polynomial of n variables may be represented in sum of products representation as follows

$$u(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{r^n} u_i \prod_{j=1}^{n} x_j^{i_{j-1}}$$

where i_j denotes the j-th digit in the base r representation of i. Here again, the most general such polynomial involves each literal, x_j up to only the power $r-1$ as any other polynomial is reducible to this case. It is for this reason that a natural description of the "degree" of such polynomials is the maximum degree of any literal versus the commonly used measure of a polynomial's total degree. We set $u^{(1)}(x_1, x_2, \ldots, x_n) = u(x_1, x_2, \ldots, x_n)$ and expand it in powers of x_1 as follows

$$u^{(1)}(x_1, x_2, \ldots, x_n) = \sum_{i=0}^{r-1} u^{(2)}(x_2, \ldots, x_n) x_1^i \ .$$

With the expansion we can now treat the summation as an $(r-1)$-st degree univariate polynomial of an indeterminate x_1. This enables us to use Horner's method as introduced earlier to evaluate the polynomial using $r-1$ additions and $r-1$ multiplications assuming the values of the coefficients are available. Note that the coefficients are still polynomials, however, the x_1 indeterminate is eliminated. The same expansion can now be applied on the r coefficients $u^{(2)}(x_2, \ldots, x_n)$ with x_2 as the indeterminate. We repeatedly expand the coefficients in the same fashion until polynomials in only x_n are obtained. The expansion will be repeated n times and in each step a level is obtained with one less variable in which the number of coefficients will grow by a factor of r as shown in Table 1. The process is recursively iterated as follows.

$$u^{(k)}(x_k, \ldots, x_n) = \sum_{i=0}^{r-1} u^{(k+1)}(x_{k+1}, \ldots, x_n) x_k^i \quad , \quad k = 1, 2, \ldots, n \ .$$

The total number of additions or multiplications is found as

$$C = \sum_{i=1}^{n} (r-1) r^{i-1} = r^n - 1 \ .$$

We summarize this result in the following theorem.

Theorem 1 (Multivariate Horner). *The evaluation of an n-variate polynomial over Z_m of maximum degree $(r-1)$ in all variables requires at most $r^n - 1$ additions and $r^n - 1$ multiplications in Z_m.*

A parallel implementation of n levels creates a total delay of

$$T = n(r-1)T_A + n(r-1)T_M \ .$$

Table 1. Number of coefficient polynomials introduced in each level

Level	#Coefficient Polynomials	#Mult or #Add
1	r	$(r-1)$
2	r^2	$(r-1)r$
3	r^3	$(r-1)r^2$
$\vdots$	$\vdots$	$\vdots$
n	r^n	$(r-1)r^{n-1}$

5 Our Contribution

We follow the same strategy as in Horner's algorithm, however, we make a key observation. In the last level in Table 1 the number of coefficients is given as r^n. These coefficients are univariate polynomials in x_n with maximum degree of $r-1$. However, the number of unique polynomials of this degree in Z_m is m^r. If $r^n > m^r$ then many of the coefficients which we counted as distinct are redundant. In the worst case the number of distinct coefficients for level n is therefore m^r. The same argument can be made for level $n-1$. In this level, the coefficients are polynomials in x_{n-1} with r coefficients that are polynomials in x_n. The number of unique polynomials in this level is therefore m^{2r}. The number of unique polynomials for each level is shown in Table 2. In the table we observe that the number of coefficient polynomials increases while the number of unique polynomials decreases with increasing levels. This suggests an optimization strategy which would compute the recursion using Horner's method until the number of coefficient polynomials exceeds the number of unique polynomials. Then for the remaining levels we compute *all* unique polynomials. Before continuing our analysis we find it instructive to illustrate our optimization strategy on a simple example:

Example 1. Let $Z_m = Z_2$ and $f = f(x_1, x_2, x_3, x_4)$ represent a multivariate polynomial $f : (Z_2)^4 \mapsto Z_2$ explicitly given as

$$f = x_1x_2x_3x_4 + x_1x_2x_3 + x_1x_2x_4 + x_2x_3x_4 + x_1x_3 + x_3x_4 + x_2x_4 + x_3x_4 + x_3 + x_2 + x_1 + 1 \ .$$

Applying Horner's algorithm we convert the polynomial into the following representation[2]

$$f = 1x_1\,[1x_2\{1x_3(1x_4+1)+(1x_4+0)\}+\{1x_3(0x_4+1)+(1x_4+1)\}] + [1x_2\{1x_3(1x_4+0)+(1x_4+1)\}+\{1x_3(1x_4+1)+(0x_4+1)\}]$$

We make out point by simply focusing on the last level of computation. Now note that in the last level we have 8 polynomial evaluations of the form $ax_4 + b$

[2] We use different kinds of parantheses to hint the computations that take place in each level.

where $a, b \in Z_2$. However, there can be only 2^2 such polynomials. Hence, a blind implementation of Horner's algorithm will be redundant. Since our algorithm is generic (and therefore should not depend on the particular choice of polynomial coefficients) in the last level we compute all possible polynomials in x_4, and simply wire the outputs to as many locations as required in the last level of the circuit evaluating f.

To compute all unique polynomials in level n, in which all polynomials are univariate over x_n, we use $r-1$ multiplications and $r-1$ additions per polynomial evaluation and $(r-1)m^r$ in total. Similarly in level $n-1$, all polynomials are now (since all polynomials over x_n are already computed) univariate over x_{n-1}. There are r coefficients with m^r choices for each coefficient. Hence there are $(m^r)^r = m^{r^2}$ polynomials requiring $(r-1)m^{r^2}$ additions and multiplications. This process is repeated until the first level is reached. The resulting complexities for each level are shown in Table 2.

To find the level k in which the number of coefficients exceeds the number of unique polynomials we need to find the smallest value of k satisfying the following inequality

$$r^k \geq m^{r^{n-k+1}} \ . \tag{1}$$

By taking the logarithm of both sides the following inequality is obtained

$$kr^k \geq r^{n+1} \log_r m \ . \tag{2}$$

Let the right-hand-side be called c. Taking the logarithm of both sides with respect to base r and solving for equality we obtain

$$k = \log_r c - \log_r k \ .$$

Now we may substitute the value of k in the logarithm on the right-hand-side.

$$k = \log_r c - \log_r(\log_r c - \log_r k) \ .$$

We may continue in the same fashion substituting infinitely many times.

$$k = \log_r c - \log_r(\log_r c - \log_r k(\log_r c - \log_r k(\log_r c - \log_r(\ldots)\ldots) \ .$$

Table 2. Number of coefficient polnomials and unique polynomials at each level

Level	#Coefficient Polynomials	#Mult or #Add	#Unique Polynomials	#Mult or #Add
1	r	$(r-1)$	m^{nr}	$(r-1)m^{r^n}$
2	r^2	$(r-1)r$	$m^{(n-1)r}$	$(r-1)m^{r^{n-1}}$
3	r^3	$(r-1)r^2$	$m^{(n-2)r}$	$(r-1)m^{r^{n-2}}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n-2$	r^{n-2}	$(r-1)r^{n-3}$	m^{3r}	$(r-1)m^{r^3}$
$n-1$	r^{n-1}	$(r-1)r^{n-2}$	m^{2r}	$(r-1)m^{r^2}$
n	r^n	$(r-1)r^{n-1}$	m^r	$(r-1)m^r$

Note that by each new term the value of k becomes more precise. At the same time the contribution of these terms shrink logarithmically. Since we are interested in only integer values of k it suffices to approximate k by neglecting the terms after only two levels of substitution.

$$k \approx \log_r c - \log_r(\log_r c) \ . \tag{3}$$

The exact solution of (2) is defined in terms of the Lambert-W function [2]

$$k \geq W(\log r \frac{r^{n+1}}{\log_m r}) / \log r \tag{4}$$

where $W(x)$ is defined as the inverse of the map $x \rightarrow xe^x$.

Now we can compute the total number of operations by simply summing the entries in the third column from the first level through level k and the entries in the last column from level $k+1$ through level n in Table 2.

$$\begin{aligned} C &= \sum_{i=1}^{k}(r-1)r^{i-1} + \sum_{i=1}^{n-k}(r-1)m^{r^i} \\ &= (r^k - 1) + (r-1)(m^r + m^{r^2} + m^{r^3} + \ldots + m^{r^{n-k}}). \end{aligned} \tag{5}$$

Ignoring the smaller order terms in the super-exponential summation the complexity may be approximated as follows

$$C \approx r^k + rm^{r^{n-k}} \ .$$

Using (3) and r^{n-k} directly obtained from (2) the complexity is further simplified as

$$\begin{aligned} C &= \frac{c}{\log_r c} + rm^{\frac{n \log_m r}{r}} \\ &= \frac{r^{n+1} \log_r m}{(n+1) + \log_r(\log_r m)} + r^{\frac{n}{r}+1} \end{aligned} \tag{6}$$

Hence, both the addition and multiplication complexities grow by $O(\frac{r^n}{n})$.

Theorem 2 (Modified Horner). *Given $r^n > m^r$ the evaluation of an n-variate polynomial over Z_m of maximum degree $n(r-1)$ requires at most $O(\frac{r^n}{n})$ additions and multiplications in Z_m.*

In the improved algorithm a parallel implementation of n levels creates a total delay of

$$T = k(r-1)(T_A + T_M) + (n-k)(r-1)(T_{A,const} + T_{M,const}) \ .$$

Here $T_{A,const}$ and $T_{M,const}$ denote delays of *constant* addition and multiplication in Z_m, respectively.

The structure that results is depicted in Figure 1. The product and the summation symbols indicate blocks implementing multiplication and addition in Z_m, respectively. Note that, in each level up to the $k-1$ level the fan-out of the polynomial implementing arithmetic blocks is unity while for higher levels (i.e. $k, k+1, \ldots, n$) the fan-out may be greater than one.

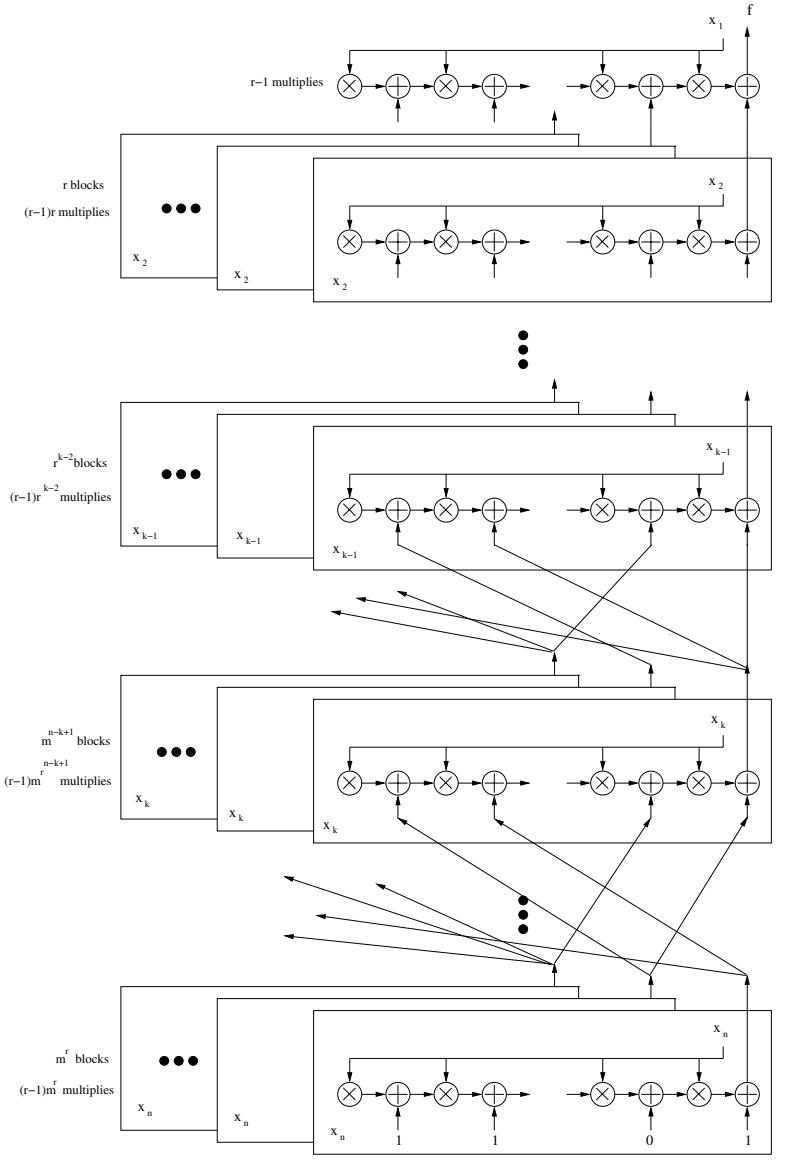


Fig. 1. Block diagram of a generic circuit implementing the modified Horner algorithm

6 Polynomials over Prime Fields

For polynomials over prime fields $GF(p)$ the polynomial degree is bounded by $p-1$ as any polynomial of higher degree may be reduced using Fermat's Theorem:

$x^{p-1} = 1 \bmod p$. Substituting $m = p$ and $r = p$ in (6) we obtain the total number of additions and multiplications as [3]

$$C \approx \frac{p^{n+1}}{n+1} \tag{7}$$

Hence, both the addition and multiplication complexities grow by $O(\frac{p^n}{n})$. On the other hand, using $k = (n+1) - \log_p(n+1)$ derived from (3) the time complexity simplifies as follows

$$T = ((n+1)-\log_p(n+1))(p-1)(T_A+T_M)+(\log_p(n+1)-1)(p-1)(T_{A,const}+T_{M,const}) .$$

Theorem 3 (Modified Horner over $GF(p)$). *Given $n > p$ the evaluation of an n-variate polynomial over $GF(p)$ requires at most $O(\frac{p^n}{n})$ additions and multiplications in $GF(p)$ with a delay of $O((p-1)(n-\log_p n))$.*

7 Optimality

Consider a parallel implementation of a function in $GF(p)$ of size s which denotes the total number of addition and multiplication components used in the circuit. The total number of circuits that can be built using s components is

$$(((s+p+n)^2)^s)$$

since each of the s components can have either the output of another component (s choices) a constant (p choices) or a literal (n choices) connected as input. To build an arbitary function, the number of circuits must exceed the number of n-variate functions over $GF(p)$. This leads to the following inequality.

$$((s+p+n)^2)^s \geq p^{p^n}$$

Substituting $s = \frac{p^n}{2n}$ we see that (although close) the inequality is still not satisfied

$$(\frac{p^n}{2n}+p+n)^{\frac{p^n}{2n}} \approx \frac{p^{p^n}}{2n} \not\geq p^{p^n}$$

Hence we found a lower bound on the circuit complexity any circuit must satisfy to evaluate an arbitrary n-variate function over $GF(p)$.

Theorem 4 (Lower Bound on Circuit Size). *Any circuit evaluating an arbitrary n-variate polynomial over $GF(p)$ requires at least $\Omega(\frac{p^n}{2n})$ adders and multipliers in $GF(p)$.*

The bound can be made tighter by more careful analysis but if suffices for our purposes. With Theorems 3 and 4 it directly follows that the presented modification to Horner's algorithm for multivariate polynomials over $GF(p)$ is asymptotically optimal.

[3] In the $p = 2$ case, the multiplications in the second summation disappear since they are constant multiplications by either 0 or 1.

8 Comparison to Muller's Algorithm

The Muller construction [5] gives a method for evaluating arbitrary n-variate polynomials over $GF(2)$ with $O(\frac{2^{n+1}}{n+1})$ complexity (see (7)). We may consider the task of evaluating an n-variate polynomial over $GF(p)$ as equivalent to evaluating $(\log_2 p)$ polynomials of $(n \log_2 p)$-variables over $GF(2)$. In this case the bit-level algorithm implementing a polynomial evaluation over $GF(p)$ has bit-complexity

$$C_B = O\left((\log_2 p)\frac{2^{n \log_2 p+1}}{n \log_2 p + 1}\right) = O\left(\frac{2p^n}{n}\right) .$$

On the other hand the complexity equation (7) derived in this paper may be expressed in bit operations rather than operations in $GF(p)$. Then assuming a $GF(p)$ multiplication operation takes $(\log_2 p)^2$ bit operations we obtain the bit complexity as follows

$$C_W = O\left(\frac{p^{n+1}}{n+1}(\log_2 p)^2\right) .$$

Interestingly, the bit-level algorithm seems to be a constant $\frac{p}{2}(\log_2 p)^2$ times more area efficient[4]. Note that in the $GF(p)$ case we are limiting our algorithms to operate on groups of $\log_2 p$ bits whereas Muller's algorithm operates on individual bits. Due to the fine grained approach Muller's algorithm has more opportunity for optimization.

We see a similar picture in the time complexities. We may assume both the $GF(p)$ multiplication and the addition circuits compute the result in $O(\log_2 \log_2 p)$ two-input gate delays where $p > 2$ using a fast addition circuit. Thus ignoring the constant operations the overall computation takes

$$T_W = O((p-1)(\log_2 \log_2 p)(n - \log_p n)).$$

gate delays in the word-level approach. The bit-level approach yields a time complexity of

$$T_B = O(n \log_2 p - \log_2(n \log_2 p)).$$

gate delays. The bit-level algorithm seems to be roughly $\frac{(p-1)(\log_2 \log_2 p)}{\log_2 p}$ times faster than the $GF(p)$ algorithm.

9 Further Optimizations

Up until now we have not used any special properties of the ring structure. One strategy that comes to mind is to use Euler's Theorem to reduce the polynomial degree r. For relatively prime integers a and m Euler's Theorem is simply stated as $a^{\phi(m)} = 1 \pmod m$. When the degree r of $u(x)$ is larger than $\phi(m)$, then by restricting $x_1, x_2, \ldots, x_n$ to integers that are relatively prime to m we obtain the following strategy:

[4] This figure may be reduced by employing fast (FFT based) methods to realize $GF(p)$ multiplications.

– First compute

$$u'(x_1, x_2, \ldots, x_n) = u(x_1, x_2, \ldots, x_n) \bmod (x_1^{\phi(m)}, x_2^{\phi(m)}, \ldots, x_n^{\phi(m)})$$

offline.
– Evaluate $u(x_1, x_2, \ldots, x_n)$ by evaluating $u'(x_1, x_2, \ldots, x_n)$.

With this strategy the direct application of the modified Horner's algorithm has complexity $O(\frac{\phi(m)^n}{n})$.

It is possible to obtain further improvements by using the factorization of the modulus m to use efficient residue arithmetic. For instance, if m is factorized into distinct prime powers as $m = p_1^{e_1} p_2^{e_2} \cdots p_t^{e_t}$, then we may achieve the evaluation in two steps:

– Evaluate $u(x)$ with respect to moduli $p_1^{e_1}, p_2^{e_2}, \ldots, p_t^{e_t}$.
– Use the Chinese Remainder Theorem (CRT) to assemble the result w.r.t. modulus m.

Note that this evaluation procedure may provide more than the standard speedup obtained by the CRT. If we know that $x_1, x_2, \ldots, x_n$ are not divisible by any prime factor of m then the t polynomial evaluations may be performed by evaluating $u'(x) = u(x) \bmod x^{\phi(p_i^{e_i})}$ for $i = 1, 2, \ldots, t$. Then the total complexity becomes

$$C = \frac{1}{n} \sum_{i=1..t} \phi(p_i^{e_i})^n = \frac{1}{n} \sum_{i=1..t} (p_i^{e_i} - p_i^{e_i - 1})^n$$

To gain more insight we assume roughly equal sized partitions, i.e. $p_i^{e_i} \approx m/t$ and simplify the complexity further to

$$C \approx \frac{1}{n} \sum_{i=1..t} (m/t)^n = \frac{m^n}{n t^{n-1}}$$

Note that this complexity figure gives the number of addition and multiplication operations carried out in rings roughly of size m/t which is much smaller in size than Z_m. Hence there is additional improvement in the bit-complexities. Nevertheless, we observe that the complexity $O(\frac{m^n}{nt^{n-1}})$ is exponentially improved by growing t. The complexity of residue computations and the CRT re-construction are not included in this partial result. The complexity of these additional operations is a strong function of the prime power decomposition of the modulus. However, for a large modulus the result, i.e. C, is expected to dominate the overall complexity.

10 Conclusion

We presented a means of improving the parallel implementation complexity of evaluating unstructed n-variate polynomials over finite rings and fields. Our modification is based on eliminating redundancies in the multivariate version

of Horner's algorithm which occur when the evaluation takes place over a small finite mathematical structure and may be considered as a generalization of Shannon's lower bound and Muller's algorithm to word level circuits.

We presented two strategies for further improving the multivariate version of Horner's algorithm which utilize the ring structure by employing residue arithmetic via the Chinese Remainder Theorem. It turns out that by restricting the inputs to integers relatively prime to m, exponential improvement can be obtained. Of course, this statement is based on the assumption that m is highly composite.

If the domain is a finite field $GF(p)$ the complexity of multivariate Horner polynomial evaluation is improved from $O(p^n)$ to $O(\frac{p^n}{2n})$. We prove the optimality of the presented algorithm and show that the bit-level algorithm provides a constant times better time and space complexities than the word-level approach. The lesson taught by this exercise is that the currently popular implementation style which favors the word-level approach diverges from optimality as the order of the finite field increases. We have shown that the bit-level approach provides significant time and area savings provided that the function is chosen arbitrarily, which is the case for substitution boxes in cryptographic applications. We should point out that our result will not apply to highly structured specialized functions since there is significantly more opportunity for optimization by using the special structure of the function.

References

1. A. Borodin. Horners Rule is Uniquely Optimal. In Z. Kohavi and A. Paz, editors, *Proceedings of an International Symposium on the Theory of Machines and Computations*, pages 45–57. Academic Press, 1971.
2. R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the Lambert W Function. *Advances in Computational Mathematics*, 5:329–359, 1996.
3. W. G. Horner. A new method of solving numerical equations of all orders by continuous approximation. *Philos. Trans. Roy. Soc. London*, 109:308–335, 1819.
4. D. E. Knuth. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, Massachusetts, USA, 2nd edition, 1981.
5. D. E. Muller. Complexity in Electronic Switching Circuits. *IRE Transactions on Electronic Circuits*, (5):15–19, 1956.
6. NIST FIPS PUB 180-1. *Secure Hash Standard*. Federal Information Processing Standards, National Bureau of Standards, U.S. Department of Commerce, April 1995.
7. NIST FIPS PUB 46-3. *Data Encryption Standard*. Federal Information Processing Standards, National Bureau of Standards, U.S. Department of Commerce, 1977.
8. U.S. Department of Commerce/National Institute of Standard and Technology. *Advanced Encryption Standard (AES)*, November 2001.
9. A. M. Ostrowski. On two problems in abstract algebra connected with Horner's rule. pages 40–48. Academic Press, 1954. presented to Richard von Mises.
10. V. Ya. Pan. Methods for Computing Values of Polynomials. *Russian Mathematical Surveys*, 21(1):105–136, 1966.

11. R.L. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm*. Corporation for National Research Initiatives, Internet Engineering Task Force, Network Working Group, April 1992.
12. C. E. Shannon. The Synthesis of Two-terminal Switching Circuits. *Bell System Technical Journal*, 28(1):59–98, 1949.
13. ETSI/SAGE Specification. Specification of the 3GPP confidentiality and integrity algorithms; part 2: KASUMI specification. 3GPP TS 35.202, European Telecommunications Standards Institute, Sophia-Antipolis Cedex, France, November 1999. Draft.

EM Analysis of Rijndael and ECC on a Wireless Java-Based PDA

Catherine H. Gebotys, Simon Ho, and C.C. Tiu

Department of Electrical and Computer Engineering, University of Waterloo, N2L 3G1 Waterloo, Canada
cgebotys@uwaterloo.ca

Abstract. Although many wireless portable devices offer more resistance to bus probing and power analysis due to their compact size, susceptibility to electromagnetic (EM) attacks must be analyzed. This paper demonstrates, for the first time, a real EM-based attack on a PDA running Rijndael and elliptic curve cryptography. A new frequency-based differential EM analysis, which computes the spectrogram, is presented. Additionally a low energy countermeasure for symmetric key cryptography is presented which avoids large overheads of table regeneration or excessive storage. Unlike previous research the new differential analysis does not require perfect alignment of EM traces, thus supporting attacks on real embedded systems. This research is important for future wireless embedded systems which will increasingly demand higher levels of security.

1 Introduction

As more security applications migrate to the wireless device, resistance to attacks on the PDA or cellphone will become a necessity. These attacks may not only arise from device theft or loss but also during everyday use where unintentional electromagnetic (EM) waves radiated from the wireless device during cryptographic computations may leak confidential data to a nearby attacker. Researchers have already demonstrated that this EM attack is viable[7,10] on an 8-bit processor running at 4MHz in a smartcard. For example an attack may be successful in obtaining the secret keys stored in confidential memory in a wireless device. This attack may be possible through loss or theft of the device, or alternatively through temporary access to the device by monitoring the EM waves emanating from the device while performing cryptographic computations. In the latter case the attack may be able to extract the encryption keys, making future wireless communications insecure. Nevertheless large overheads in energy to achieve resistance to these attacks may not be practical for wireless embedded systems. Outside of smartcard research (which in the past has typically been limited to cheaper 8-bit or 16 bit processors) [5,12,1,4], few researchers have examined secure implementations of cryptographic software (such as Rijndael[6] which has become a popular new standard) under the threat of EM attacks on 32-bit processors. The cryptographic algorithms which are essential for these applications are typically run by embedded processors in these wireless devices. Unfor-

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 250 – 264, 2005.

tunately cryptographic algorithms are already known to consume a significant amount of energy [2]. Even worse, cryptographic algorithms which are resistant to attacks are known to have latency overheads up to 1.96 times[3]. Although these attack resistant algorithms have been developed for smartcard applications (where energy dissipation is not viewed as important), there is an important need to study EM attacks and energy optimized countermeasures on wireless portable devices, such as PDAs, cellphones, etc.

2 Previous Research

Typically in symmetric encryption the plaintext and key are exclusive or'd together and then indexed into a table, as in the table method of the Rijndael advanced encryption standard[6]. The attacker may have control over the plaintext and by guessing the 8-bit key value, can partition EM or power traces according to a bit in the data at the output of the table. By taking the difference of the average of the partitioned traces and by recording the height of the differential for each key guess, the attacker can determine the correct key (since it will have the highest differential value). In elliptic curve cryptography (ECC) the data at the output of a double operation can be similarly partitioned according to the guess of the scalar key bit as described in [19]. Although EM attacks on smart cards have been investigated [10,8], EM attacks on other embedded systems have not been widely researched, apart from far field EM emanations from a Palm-Pilot and SSL accelerator[29,30]. Previous research studied the correlation of EM variation with data values being manipulated (known as differential EM analysis, DEMA, or DPA for differential power analysis [1]) and instruction sequencing (known as simple EM analysis or SEMA). In the former case, DEMA, the DES encryption[10] was analyzed. Differential EM attacks on embedded low power processors have not been reported in the literature. Higher order (n^{th} order) differential attacks[16] are an extension of the 1st order differential analysis which involve using joint statistics on multiple (n) points within power traces. These higher order differential analyses have been shown to provide more security, since they require more EM or power traces[16,10]. For example research with real EM measurements using a 8-bit processor running at 4MHz in a smart card, demonstrated 2nd order DEMA [10] on a 2-way exclusive-or-based secret sharing scheme using 500 EM traces. However no known results using real EM measurements have been obtained for an attack on ECC. In most cases, good EM or power trace alignment of the attack point is required since most previous differential analyses are performed in the time domain. The exception to this is [17] where the fast fourier transform is calculated, however transformation back into the time domain occurs before differential analysis is performed. This paper will use the terms DEMA or DPA to denote time domain differential analysis.

Previously researched countermeasures have been suggested such as random sequencing of instructions (desynchronization), secret splitting[9], duplication method[14], multiplicative masking[15] and random masking[3]. Secret splitting involves splitting the secret data into smaller pieces and combining them with random data [9]. To attack the splitting method, a k^{th} order differential attack is required[9]. The duplication method[14] was used to support secure computations with multiple split variables for input to the S-box. These researchers also used table duplication

such that one table contained a randomly-chosen secret transformation on x, $A[x]$, and the alternate table contained $A[x]+S[x]$, where + represents exclusive or operation. Multiplicative masking was also defeated by a DPA attack[15]. In the masking countermeasure, each secret piece of data is exclusive-or'd with a random data value (called a mask). To thwart a DPA attack the random data value must be changed periodically. However this involves remasking the tables (or exclusive-or the complete table data with a mask) within the algorithm. Some researchers have investigated storing a limited number of masked tables [14] (called the 'fixed-value' masking). However results using real EM or power measurements were not performed. Countermeasures for power or EM analysis of ECC include i) indistinguishable formulas for point operations, ii) identical operation sequences regardless of key bits, and iii) random addition chains. In general the first approach, i), restricts the ECC to specifically chosen curves [24,18] and is generally vulnerable to differential attacks [22]. The second approach, ii), includes algorithms like the double-and-always add algorithm [19] and others [18, 21, 31], which thwarts simple analysis attacks but can be attacked using differential analysis. The random addition chains approach, iii), utilizes sequences of additions, subtractions, and doublings that can mutate randomly[26]. Other general approaches to resisting differential analysis of ECC include randomizing the base point (such as point blinding [19]) and randomizing the scalar [19,25]. Countermeasures designed for thwarting differential analysis of sliding window implementations of ECC include [23]. In general there are few publications reporting differential analysis of ECC using real power or EM.

Unlike previous research, this paper presents results of EM analysis on a real embedded system, a wireless Java-based PDA. Rijndael and elliptic curve scalar multiplication are used to demonstrate a new differential attack based solely upon analysis in the frequency domain. Furthermore analysis is performed in the presence of a countermeasure for table-based symmetric key cryptography suitable for PDA like devices. Comparison to previously researched attacks show that frequency based analysis is crucial for real embedded systems, since previous time domain analysis were not successful in finding the correct key in all cases. A most-significant-bit differential attack is found to be stronger for ECC than attacking any bit as previously described in [19]. The EM analysis attack is demonstrated for the first time on ECC running on a PDA. The next section will describe the proposed frequency based differential analyses techniques and the following section will present the experimental results.

3 Differential Analysis in the Frequency Domain

This paper proposes an extension of the existing differential side channel attack, where instead of performing analysis in the time domain, the frequency domain is used. In general instead of computing the differential signals in the time domain (as in [4] and in almost all previous research), the computation is performed in the frequency domain. Analyzing signals captured in the frequency domain solves the problem of misalignment (or time-shifts) in traces since fast fourier transform (FFT) analysis is time-shift invariant. Frequency analysis is important for attacking real embedded systems where uncorrelated temporal misalignment (or time-shifting) of

traces (typically caused by the triggering signals or the Java operating system) is a big concern. Additionally, frequency analysis may reveal loops and other repeating structures in an algorithm that is not possible with time domain analysis. However there are two problems with using frequency domain signals in differential analysis. First, it reveals no information of when data-dependant operations occur. This timing information is very useful as it helps an adversary focus the signal analysis on these data-dependant operations. Secondly, any peaks in frequency domain due to an event that occurs in a short duration may be discernable if the acquisition duration is a lot longer. The solution of these problems is to use spectrogram, DSA, which is a time dependant frequency analysis.

The following terminology is used to describe the algorithms which follow, specifically *SPECGRAM, DSA*: $i\varepsilon\{0,\ldots,n-1\}$ is the trace number; $b\varepsilon\{0,1\}$ is the set number; T_i^b is the EM signal of set b and trace i; $t\varepsilon\{0,\ldots,m-1\}$ is time; $s\varepsilon\{0,\ldots,p-1\}$ is the frame number in spectrogram; w is window size; $f\varepsilon\{0,\ldots,wp/2-1\}=\{0,\ldots,m/2-1\}$.

SPECGRAM*(T)*

1 : for each b, $b \in \{0,1\}$ and for each i, $i \in \{0,\ldots,n-1\}$: for each s, $s \in \{0,\ldots,p\}$:

2 : $F \leftarrow \mathrm{FFT}(T_i^b(s*w:(s+1)*w-1))$

3 : $V_i^b(s*\frac{w}{2}:(s+1)*\frac{w}{2}-1) \leftarrow F \bullet \mathrm{HAMMING}(\frac{w}{2})$

4 : return V

DSA(T^0, T^1, κ)

1 : $P \leftarrow SPECGRAM(T), R \leftarrow \mathrm{SD\_DOM}(P^0, P^1), D \leftarrow \mathrm{Mean}(P^1) - \mathrm{Mean}(P^0), s \leftarrow 0$

2 : for each f, $f \in \{0,\ldots,\frac{wp}{2}-1\}$:

3 : if (abs$(D(f)) > \kappa * R(f)$)

4 : $s \leftarrow s + (\mathrm{abs}(D(f)) - \kappa * R(f))$

5 : return s

There are two main components of creating a spectrogram for each window in a time domain trace. The first component is taking the FFT, which results in a frequency domain signal. The second component is taking a dot product between the frequency signal and a Hamming window. The application of the Hamming function suppresses the Gibbs' phenomena in spectral windowing. The creation of the spectrogram is detailed in the DSA algorithm above. When only one window whose width is equivalent to the duration of the time domain trace is used, the power spectral density can be used to perform differential analysis in the frequency domain[28]. Each signal trace is measured over an interval of m time points. A spectrogram applied on p time windows with w time points would have $w/2$ frequency points in each window (assuming w is an even number) and $wp/2$ points in total, according to the Nyquist criterion.

An important part of the differential analysis is locating the significant peaks in a differential signal. The routine SD_DOM is the standard deviation of the difference of means. The differential peaks that exceed a constant multiple κ of SD_DOM are considered to be significant. The analysis methodology involves first using DSA on a time frame to locate possible areas of attack. Next the attacker can focus in on smaller time frames which show interest. For ECC attacks, the most-significant-bit is chosen to perform the partitioning into the two sets (V^b, P^b, T^b).

The next section will present the results of the DSA, proposed countermeasure (see appendix), and previous attack techniques on Rijndael and ECC running on a wireless PDA.

4 Experimental Results

A high sample rate oscilloscope, a 1-cm loop EM probe, wide band amplifier, and a PDA (which was opened to expose the packaged chip over which the probe was placed) were used to acquire EM traces. Figure 1 is a photograph of a single loop EM probe over the chip in the PDA. The Rijndael (de)encryption algorithm (implemented using the table-based method of [6]) was used to illustrate the EM attack and countermeasure verification. Additionally analysis of an Elliptic Curve point multiplication (using 192bit prime field with Jacobi projective coordinates as standardized in FIPS 186-2[27]) was performed using the new differential analysis techniques, DSA. Both the Rijndael and ECC code were written in Java and loaded onto the PDA device. A trigger signal was generated from the PDA using the Java code to turn the light emitting diode (LED) on and off. The voltage across the terminals of the LED was used to trigger the scope.

Fig. 1. EM Probe over chip in PDA device

Figure 2 illustrates the difference between capturing EM signals from a single processor evaluation board (where assembly code can be written and executed directly on the processor) in a) and EM signals from a real embedded device with operating system and executing Java code in b). In the PDA experiment, Rijndael was run with the same input data for capturing of over 1000 EM traces. The PDA captured the end of the Rijndael algorithm, whereas the evaluation board captured a different part of the algorithm. It is clear that there is little difference between the average signal and the raw EM signal in figure 2a). However in figure 2b) it is clear that there is significant difference between EM signals running on a Java-based PDA device (where the averaged signal appears as a straight line at zero).

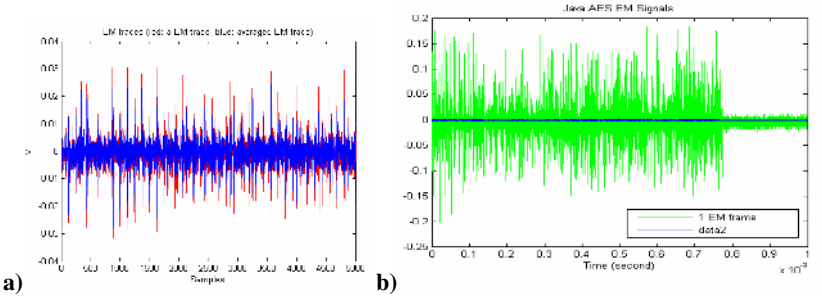


Fig. 2. Averaged (blue) versus raw EM signal (red/green) from single processor evaluation board in a) and Java-based PDA device in b)

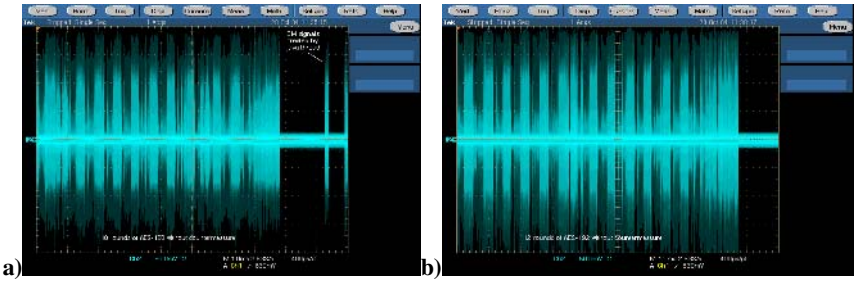


Fig. 3. SEMA attack on Rijndael in PDA device with 10 rounds a), and 12 rounds in b)

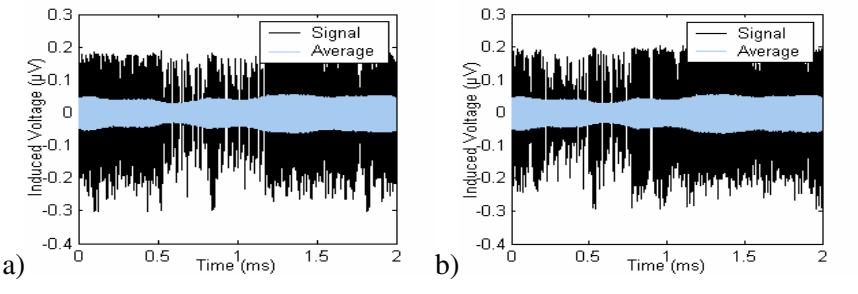


Fig. 4. EM signal of ECC on PDA over iteration 10 and 30 in a),b) respectively

Figure 3a) presents a single EM trace of 128 bit Rijndael captured by the scope from the EM signals emanating from the chip in the PDA device. Each of the 10 rounds can be seen in the figure, thus illustrating a SEMA attack on the device. The EM signals at the end are created by the timer interrupt. A thread is created to call the Rijndael encryption algorithm. After Rijndael's execution has completed, the thread is programmed to sleep. Therefore, there are minimum EM signals as shown in the graph when the thread is in the sleep mode. However the timer interrupt occurs every few milliseconds to check if a thread needs to be activated (evident from the last EM

burst, which is not part of the Rijndael algorithm). Figure 3b) illustrates scope capture of EM signals from PDA running a 192 bit Rijndael, where 12 rounds are evident.

The EM trace and EM average of elliptic curve cryptography running on the PDA is shown in figure 4. For example different iterations of the same point double routine in the ECC algorithm are shown in figure 4a) and figure 4b). Clearly there are significant differences in the EM trace (since the averaged EM trace has a much lower amplitude).

4.1 Differential EM Results for Rijndael

EM traces were acquired from the PDA while it was executing a Java-based implementation of the table-based method of the Rijndael algorithm[6]. Results are presented for the proposed DSA and previously researched DEMA. Additionally the proposed split mask countermeasure (detailed in the appendix) is also analyzed. Differential traces and plots of all keys versus peak height are provided.

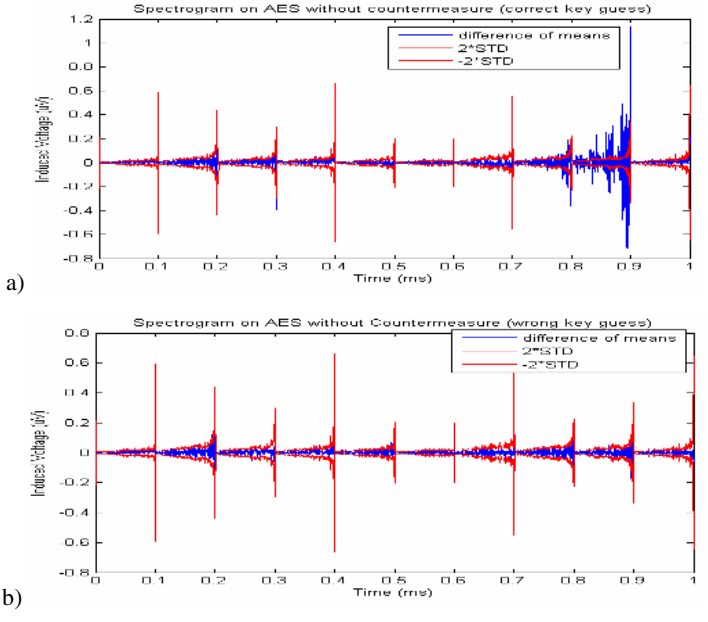


Fig. 5. DSA on Rijndael for correct key in a) and incorrect key in b) over 1ms time period

According to section 3's DSA algorithm, the spectrogram is calculated for each EM trace. Then the set of spectrograms is partitioned into two groups based upon a key guess. The mean of each group of spectrograms is calculated. Finally the difference of these two means will be called the differential spectrogram trace. The differential spectrogram trace, is illustrated in figure 5a) for a correct key guess partitioning of EM traces (where partitioning was based upon the least significant bit at the output of the first Sbox table). Time intervals of 0.1ms were used to create this spectrogram. In between the 0.1 time intervals is the plot of the differential signal over a range of

frequencies. A total of 1303 EM traces were acquired with the scope set at 25M samples/sec. Each EM trace had 25,000 sample points. Figure 5b) illustrates the analysis of the same set of EM traces, however partitioned for an incorrect key guess. Plus or minus two standard deviations are shown as red in the figures and the actual differential spectrogram trace is shown in blue. Clearly Figure 5a) indicates significant differentials over the region of 0.7 to 0.9 ms.

Figure 6a) illustrates the results of the previously researched DEMA applied to the same set of acquired EM traces. This differential trace, which is computed solely in the time domain, does not show any significant spike outside of the two standard deviations. These insignificant results were further confirmed by computing the DEMA for all possible key guesses, and plotting the maximum absolute peak value of the differentials for each key, as shown in figure 6b), where the correct key is not evident.

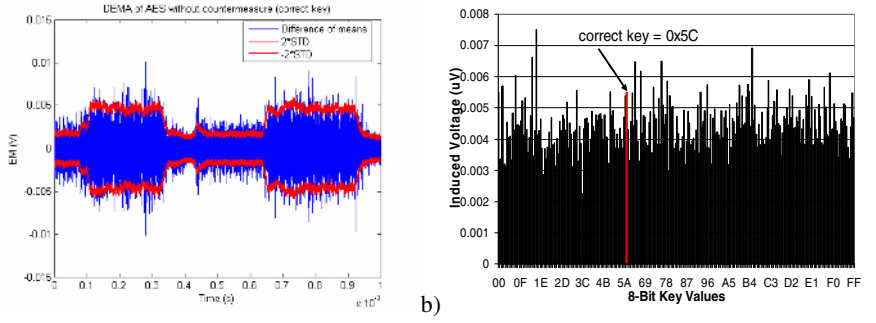


Fig. 6. DEMA of Rijndael for correct key over 1 ms and all keys plot in b)

The DSA algorithm as detailed in section 3, returns the sum of the absolute part of the differential trace which was outside of the two standard deviations across the frequency spectrum (in each time region) of the spectrogram. The plot of this value versus key value is shown in figure 7a). The correct key is clearly evident using DSA since it has the highest value. Figure 7b) provides the same plot but obtained from analyzing the EM traces acquired from Rijndael with the proposed split mask countermeasure (see appendix). Results show that the correct key is not even close to highest peaks in figure 7b), hence the countermeasure is effective against this differential spectrogram analysis. The previously researched second order analysis techniques in [17], which are also known to be time-shift invariant, were used to attack the countermeasure. However they were also not successful in attacking the countermeasure even though over 2000 EM traces were used.

It was not possible to accurately measure the energy of the PDA device while it was computing the cryptographic algorithm because the power pins of the chip were too small to access, and the fraction of battery energy was also too small to measure. Hence to obtain the energy measurements, a separate processor evaluation board was used which contained a 32-bit ARM7TDMI RISC processor core on one chip separate from the memory. The proposed countermeasure, implemented with one extra mask table, provided 1.7 times increase in energy over Rijndael with no countermeasure and 5.2 times less energy than the countermeasure in [3] where table regeneration is re-

quired when a new mask is applied. When two mask tables were used in the proposed split mask countermeasure, the energy dissipation was 2.4 times more than the Rijndael without a countermeasure, and 3.7 times less energy than the countermeasure in [3]. Since these results did not represent the energy dissipation of the memory, and since it is well known that memory energy dissipation is significant and often dominates, an analysis was performed with static RAM models from [11]. For example with $(1/5)^{th}$ the number of masks, [13] dissipates up to 10.4 times more energy than the proposed countermeasure. The proposed countermeasure, with one or two extra mask tables, requires significantly less memory when supporting the same number of masks.

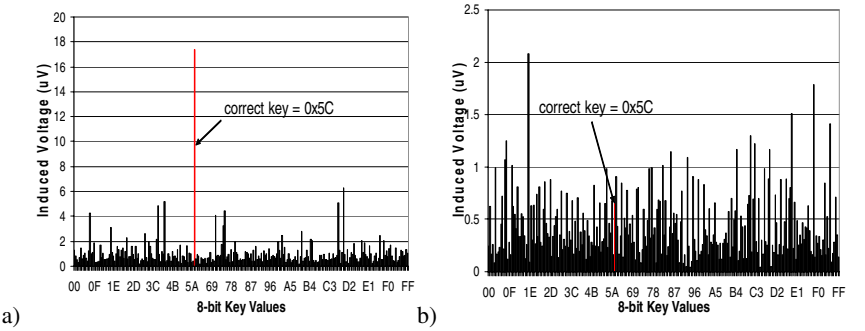


Fig. 7. All keys plot for Rijndael using DSA in a) with split mask countermeasure in b)

4.2 Differential EM Results for Elliptic Curve Cryptography

Over 1300 EM traces of the 192bit prime field (projective coordinates) elliptic curve point double operation were captured. The DSA and DEMA differential traces are presented for both correct and incorrect scalar key bit guesses. All differential results presented in this section were obtained from partitioning based upon the most significant bit of the x-coordinate of the input point of the point double operation, unless otherwise stated. Each EM trace of the elliptic curve point double routine contained 25K sample points over 2ms.

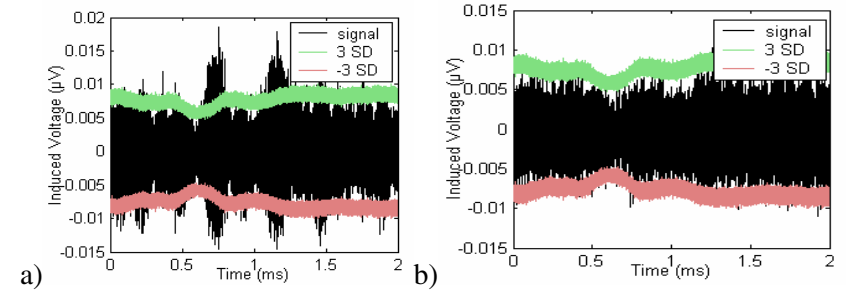


Fig. 8. DEMA of ECC double with correct a), incorrect b) scalar key bit guess

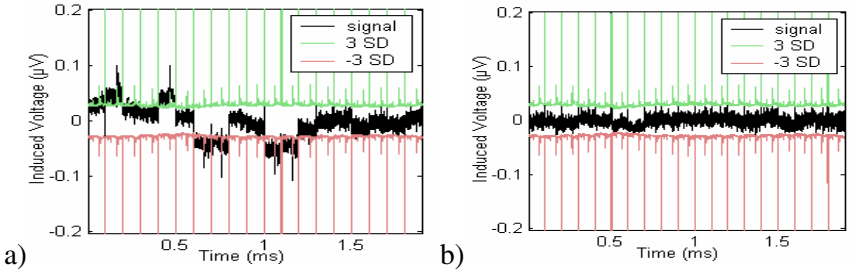


Fig. 9. DSA of ECC double with correct a), incorrect b) scalar key bit guess

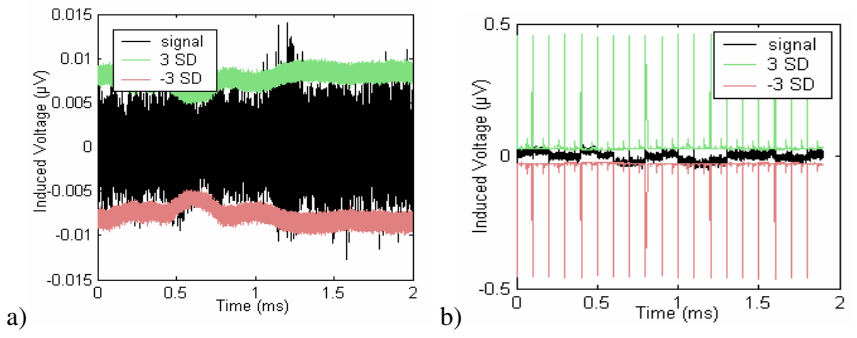


Fig. 10. DEMA a), DSA b) for correct scalar key bit partitioning on 2nd MSB

Figure 8 shows the previously researched DEMA generated with our EM traces when a correct and incorrect scalar bit is chosen in a) and b) respectively. The three standard deviations (SD) were chosen to encompass the incorrect scalar key bit. The differential signals (in black) above and below the 3 standard deviations were considered to be significant. Figure 8a), which shows the DEMA for a correct scalar key bit, features multiple significant peaks. The peaks are likely corresponding to the time of finite field computations on the x-coordinate of the input point.

Figure 9 is the differential EM spectrogram, DSA, for ECC double operation with a correct and incorrect scalar bit guess in a) and b) respectively. Again the 3 standard deviations are chosen to encompass the incorrect scalar key bit differential in figure 9b). The standard deviation of difference of means always peaked at 0 frequency, indicating that there is considerable fluctuation of EM signals over different traces. This is indicated in the figures by the long vertical green and red standard deviation lines at the 0.1ms time intervals (0,0.1,0.2, ..etc). Clearly, figure 9a) features many significant peaks above and below the 3 standard deviation curves. Furthermore, peaks in figure 9a) correlate with differential EM peaks in figure 8a), such as those that appear at 0.7ms and 1.1ms. This is expected as the differential EM signal and differential EM spectrogram are simply two different perspectives of looking at the same events unfolding on the PDA device.

Figure 10a) shows the results of performing the differential EM analysis with correct bit partitioning using second most significant bit (MSB), instead of the first most

significant bit as used with previous results. Clearly, the amplitude of the peaks in the differential signals diminishes significantly (compared to results using the most significant bit in figure 8a). Results using the third most significant bit with correct scalar key bit partitioning did not reveal any significant differential signals at all. This is because the chance of overflow is not as high if the 2nd or 3rd MSB is one, hence the probability of overflow does not correlate as closely to bits other than MSB. This demonstrates the impact of overflow in sub-operations within point doubling on the resulting EM signals. Clearly, MSB of the input coordinates works better than other bits. Figure 10b) shows a similar situation using the spectrogram. The 2nd MSB with correct scalar key bit partitioning in figure 13b) indicates some diminished significance at 0.7 ms and 1.1 ms.

5 Discussions and Conclusions

This paper presents for the first time a new differential frequency analysis approach using real EM measurements of a real embedded system, a Java-based wireless PDA. Additionally a countermeasure suitable for wireless embedded devices was presented. In the Rijndael analysis, the previously researched DEMA was not successful in obtaining the correct key on the PDA, likely since DEMA requires good EM trace alignment for DEMA to work (which is not present on the PDA since it is a larger embedded device running Java with an operating system). If differential spikes are slightly out of alignment in time, they may cancel out rather than reinforce when averaging. The spikes analyzed in DPA or DEMA can be as small as 5 sample points wide, so a misalignment of 1 or 2 sample points can already cause significant loss of information when traces are averaged together. DSA, which is less vulnerable to the effects of time-shift, was successful in obtaining the correct key from the PDA device. A countermeasure is shown to defeat the first order DEMA and DSA attacks in Rijndael. Similar to [3], the proposed countermeasure has the potential for attack through a 2nd order DEMA, however unlike [3], the proposed countermeasure can increase the security by increasing the number of extra tables. The split mask countermeasure can trade off memory for security, thus thwarting a n^{th} order DPA for (n) extra tables ($n >= 3$). A previously researched 2nd order analysis technique [17], which is also known to be time-shift invariant, was not successful in attacking the countermeasure after using over 2000 EM traces. In this case it is possible that a larger number of EM traces would be required. For the first time an attack of an elliptic curve algorithm was presented using real EM traces. Both the DEMA and DSA attacks were successful on the elliptic curve algorithm only when the 1st and 2nd most significant bits of the elliptic point data were used for partitioning. This attack worked since the MSB's were correlated with EM activity from the overflow or underflow computations (ie. those of modular reduction, etc). This is unlike previous research such as [19] where time alignment of traces was required to correlate any bit of the elliptic point data to the simulated power traces [19]. It is likely that the time misalignment was too large or the number of EM traces too small to successfully apply DSA to attack the data value (or correlate the EM trace with any bit of the elliptic point data as in [19]). It is interesting to note that a possible countermeasure for the MSB differential analysis attack of ECC is to use the same algorithm for finite field computations

regardless of whether an overflow occurs or not. As discussed earlier, spectrogram could pinpoint a time segment where there are more significant spikes which is important for determining where the activity of interest is in the traces. Differential analysis using spectrogram may be unsuccessful if the window size and the fraction of window overlap are unsuitably chosen and thus should be determined experimentally. Similar to previous research [17], DSA requires a Fast Fourier Transform, however unlike [17], this paper proposes a 1st order analysis where all computations are done in the frequency domain. The extension of DSA to higher order analysis is the subject of future work, however the 2nd order technique in [17] with over 2000 EM traces was not successful, hence supporting the belief that higher order analysis attacks are very difficult to launch. In summary the new proposed analysis techniques were successful in obtaining the correct key from both Rijndael (a symmetric key encryption standard) as well as elliptic curve cryptography (a public key cryptographic standard). They are general and applicable to other cryptographic algorithms, power as well as EM, and other embedded systems.

Using real EM measurements from a PDA device executing Java-based cryptography, a new frequency-based (time-shift invariant) differential analysis was demonstrated. Previous differential analysis techniques requiring alignment of traces in the time domain were not successful in correlating EM signals to bits of the data. Results show that a low energy countermeasure for Rijndael supporting scalable security without large overheads of table regeneration or excessive storage was able to thwart the new differential techniques, but could not successfully be attacked by higher order techniques [17] with over 2000 EM traces. This research is crucial for supporting low energy security for embedded systems which will be prevalent in wireless embedded devices of the future. This research was supported in part by grants from NSERC and CITO.

References

1. P. Kocher “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems”, LNCS 1109 (1996) 104-113
2. S. Ravi, et al. “Securing Wireless Data: System architecture challenges”, ISSS (2002) 195-200.
3. T. Messerges, “Securing the Rijndael finalists against power analysis attacks” LNCS 1978, (2001) 150-164
4. P. Kocher, J. Jaffe, B. Jun “Differential Power Analysis” Crypto’99, LNCS 1666 (1999) 388-397
5. J-J. Quisquater, et al. “a new tool for non-intrusive analysis of smartcards based on EM emissions”, Rump Session, Eurocrypt (2000)
6. Dr. Brian Gladman, “A Specification for Rijndael, the AES Algorithm”, at fp.gladman. plus.com/ cryptography_technology/ rijndael/aes.spec.311.pdf (2003)
7. D. Agrawal et al. “The EM side-channel(s)” CHES 2002 (2002) 29-45
8. K. Gandolfi et al. “Electromagnetic Analysis: concrete results” CHES 2001, LNCS 2162, (2001) 251-261
9. S. Chari, et al. “Towards sound approaches to counteract power-analysis attacks”, LNCS 1666 (1999) 398-412

10. D. Agrawal, et al. “The EM side-channel methodologies” at http://www.research.ibm.com/intsec/emf.html
11. W. Liao et al. “leakage power modeling and reduction with data retention”, IEEE ICCAD (2002) 714-719
12. M. Akkar, et al. “Power analysis, what is now possible…”, LNCS 1976 (2000) 489-502.
13. K. Itoh et al. “DPA countermeasure based on the masking method”, LNCS 2288 (2002) 440-456
14. L. Goubin, J. Patarin “DES and Differential power analysis- the duplication method” CHES (2001) 158-172
15. J. Golic “Multiplicative Masking and power analysis of Rijndael”, CHES (2002) 1-10
16. T. Messerges “Using 2nd order power analysis to attack DPA resistant software”, LNCS 1965 (2000) 238-251
17. J. Waddle, D.Wagner “Towards efficient second-order power analysis” CHES 2004, LNCS 3156, (2004) 1-15
18. E. Brier and M. Joye, “Weierstraβ Elliptic Curves and Side-Channel Attacks", *PKC 2002*, LNCS 2274, Springer-Verlag (2002) 335-345
19. J. Coron, “Resistance against differential power analysis for elliptic curve cryptosystems”, *CHES 1999*, LNCS 1717 (1999) 292-302
20. T. Izu, B. Moller, and T. Takagi, “Improved Elliptic Curve Multiplication Methods Resistant against Side Channel Attacks”, *Indocrypt 2002*, LNCS 2551, Springer-Verlag (2002) 296–313
21. T. Izu and T. Takagi, “A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks", Technical Report CORR 2002-03, University of Waterloo, 2002. Available from http://www.cacr.math.uwaterloo.ca/.
22. T. Izu and T. Takagi, “On the Security of Brier-Joye's Addition Formula for Weierstrass-form Elliptic Curves", TR No. TI-3/02, Technische University Darmstadt, 2002. Available from http://www.informatik.tu-darmstadt.de/TI/.
23. K. Itoh, J. Yajima, M. Takenaka, and N. Torii, “DPA Countermeasures by improving the Window Method", *CHES 2002*, LNCS 2523, (2002) 303-317
24. M. Joye and J. Quisquater, “Hessian elliptic curves and side-channel attacks”, *CHES 2001*, LNCS 2162, (2001) 402-410.
25. M. Joye and C. Tymen, “Protections against differential analysis for elliptic curve cryptography”, *CHES 2001*, LNCS 2162, (2001) 377-390
26. E. Oswald, M. Aigner, “Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks”, CHES 2001, LNCS 2162, (2001) 39-50
27. National Institute of Standards and Technology, Digital Signature Standard, FIPS Publication 186-2, February (2000)
28. C. Gebotys, A. Tiu, X. Chen “A Countermeasure for EM attack of a Wireless PDA” IEEE International Conference on Information Technology – Special Session on Embedded Cryptographic Systems, Las Vegas, AZ, (2005) 544-549
29. D. Agrawal, et al. “Advances in Side-Channel Cryptanalysis EM analysis and template attacks” RSA Cryptobytes, Vol6 No1 (2003) 20-32
30. D. Agrawal, et al “Power, EM and all that: is your crypto device really secure?” presentation ECC workshop http://www.cacr.math.uwaterloo.ca/conferences/2003/ecc2003/rohatgi.ppt (2003)
31. C. Gebotys, R. Gebotys “Secure Elliptic Curve Implementations: An analysis of resistance to power-attacks in a DSP Processor” Proceedings of Cryptographic Hardware and Embedded Systems, Redwood City, CA , LNCS 2523, Springer-Verlag (2002) 114-128.

Appendix: Split Mask Countermeasure

This section will describe a proposed countermeasure which will thwart the first order analysis attack of the previous section on some symmetric key encryption algorithms, such as Rijndael, DES, CAST. The proposed split mask countermeasure stores randomly masked data in the S-box (masked S-box table, *S'[x]*). Unlike previous research [13,14], each addressed data in the table uses a different random mask (*S'[x] = S[x]+r[x],* where + is exclusive or operation , *r[x]*= random mask, which is different for each table address, *x*). A second corresponding table (called the mask table, *M[x]=m+r[x], m* is a fixed random value) is used to store a corresponding mask for each address. Since tables are generated only once, this value *m* along with mask of the round keys is used to precompute tables before the cryptographic algorithm is downloaded to the device. To avoid a first order DPA, the exclusive or of the S-box masked table and mask table, *S'[x] +M[x]=m+S[x]* , is never computed during the execution of the encryption algorithm. Figure A.1 illustrates the computations performed on the mask tables before they are merged with the masked substitution tables to avoid a 1st order DPA or DEMA or differential frequency analysis.

For example, in the table method of Rijndael (described in [6] for fast implementation on 32bit processors), all masked S-box tables are accessed and their results are exclusive-or'd together (as would normally be done for unmasked S-box tables in the original algorithm). Next all corresponding mask table outputs are exclusive-or'd together. Finally the exclusive-or result from the S-box masked tables is exclusive or'd with the exclusive-or result of the mask tables. Figure A.1 illustrates the computations required in the scheme for Rijndael, where *S1', S2', S3', S4'* and *M* are the masked S-box tables and the mask table, respectively. Note that there is only one mask table, M, which is accessed four times (so it is shown four times) in figure A.1. In the table method of Rijndael one would normally compute : $t0 = S1(\{s0\}_{b3}) + S2(\{s1\}_{b2}) + S3(\{s2\}_{b1}) + S4(\{s3\}_{b0}) + rki$. However with this countermeasure one would compute : $t0a = S1'(\{s0\}_{b3}) + S2'(\{s1\}_{b2}) + S3'(\{s2\}_{b1}) + S4'(\{s3\}_{b0})$ and $t0b = M(\{s0\}_{b3}) + M(\{s1\}_{b2}) + M(\{s2\}_{b1}) + M(\{s3\}_{b0})$ (where $\{w\}_b$ refers to byte *b* of the 32bit word *w*). Then one would merge them as : $t0 = t0a + t0b$ *and* $t0 = t0 + rki$ (where *t0* value is then combined with the masked round key for input to the next set of tables, which may also have masked inputs). A similar implementation for DES and other cryptographic algorithms is also possible. Similar to previously researched countermeasures [3,16], a 2nd order DEMA attack on this countermeasure may be possible. However a 1st order DEMA is thwarted since the data values output from the S-box tables have been decorrelated through random masking. The 2nd order DEMA could use statistical processing of EM samples of both the output of the S-box masked table and the output of the mask table to launch an attack. However unlike previous countermeasures[3,16], by increasing the number of tables, an increase in the order of the required DEMA attack occurs, hence the security of this countermeasure scales with the number of tables. For *n* mask tables (*M1, M2… , Mn)*, and one S-box masked table, *(S'*) a $(n+1)^{th}$ order DEMA attack is required (where *m= r [x]+M1[x] +M2[x]... +Mn[x]*, for all *x* , thus splitting mask *m* into *n+1* masks). For example with 2 extra mask tables (*M1,M2* where *m=r[x]+M1[x]+M2[x]*, for all *x*), a 3rd order DEMA may possibly be launched. The higher order attack typically will require many more EM traces and thus provides an increase in difficulty of launching the attack. Note that

once again, even with *n* tables, to avoid a first order DEMA, the exclusive or of the S-box masked table and mask table, specifically *S'[x]+ M1[x] +M2[x]… +Mn[x]=m+S[x]* is never computed.

The proposed split mask countermeasure is similar to the duplication method [14] however unlike [14] the second table does not hold the random mask of the S-box entry, *r[x]*. The duplication method [14] also used two tables whereas this countermeasure can increase the number of tables supporting higher order DPA. Additionally these masked tables and set of mask tables would produce a masked value unlike [14] which would produce an unmasked value.

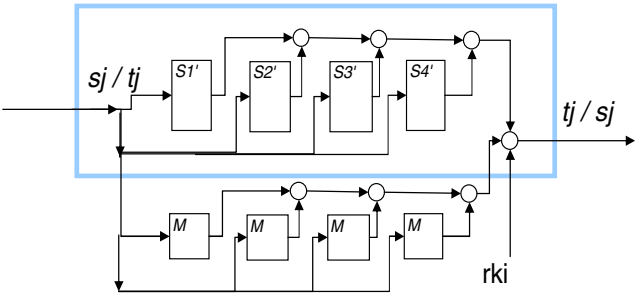


Fig. A.1. Partial Rijndael Implementation of countermeasure

Security Limits for Compromising Emanations

Markus G. Kuhn

University of Cambridge, Computer Laboratory,
15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom
http://www.cl.cam.ac.uk/ mgk25/

Abstract. Nearly half a century ago, military organizations introduced "Tempest" emission-security test standards to control information leakage from unintentional electromagnetic emanations of digital electronics. The nature of these emissions has changed with evolving technology; electromechanic devices have vanished and signal frequencies increased several orders of magnitude. Recently published eavesdropping attacks on modern flat-panel displays and cryptographic coprocessors demonstrate that the risk remains acute for applications with high protection requirements. The ultra-wideband signal processing technology needed for practical attacks finds already its way into consumer electronics. Current civilian RFI limits are entirely unsuited for emission security purposes. Only an openly available set of test standards based on published criteria will help civilian vendors and users to estimate and manage emission-security risks appropriately. This paper outlines a proposal and rationale for civilian electromagnetic emission-security limits. While the presented discussion aims specifically at far-field video eavesdropping in the VHF and UHF bands, the most easy to demonstrate risk, much of the presented approach for setting test limits could be adapted equally to address other RF emanation risks.

Keywords: Eavesdropping, emission security, Tempest, protection standards, video displays, side channels.

1 Introduction

Electronic equipment can emit unintentional radio signals from which eavesdroppers may reconstruct processed data at some distance. The civilian computer-security community became aware of the risk through van Eck's demonstration of how to eavesdrop on video displays with modified TV sets [1]. More recent studies have shown that not only are contemporary CRT monitors still vulnerable [2], but so are flat-panel displays with digital interfaces [3]. Modular exponentiation parameters in an SSL accelerator module inside a closed server have been reconstructed from emanations picked up at 5 m distance [4].

Since about 1960, NATO governments have paid considerable attention to limiting compromising emanations of computers that handle classified information. They developed test standards and procured conforming protected products. The relevant standards and their rationales are still classified documents

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 265–279, 2005.

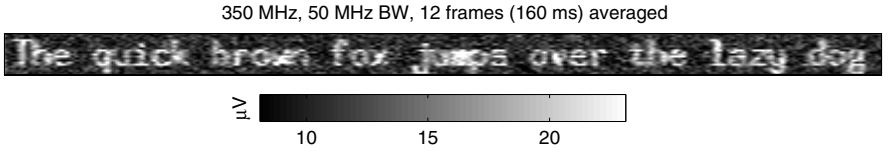


Fig. 1. Text signal received from a Toshiba 440CDX laptop at 10 m distance through two intermediate offices (3 plasterboard walls) using an omnidirectional antenna, a Dynamic Sciences R-1250 AM wideband receiver, a digital storage oscilloscope and postprocessing on a PC involving cross-correlation controlled periodic averaging. The calibration bar shows the rms voltage of a sine wave on the antenna input that would generate an equally strong signal. [3]

and conforming products remain export controlled. Although "Tempest"-certified equipment is, in principle, available to non-military customers, its use in civilian applications remains marginal.

Without open standards, emission security remains largely ignored in non-military applications, smartcard microcontrollers being a notable recent exception. There may be several reasons. Firstly, secret military specifications restrict the choice of suppliers to a small number of defense contractors with the necessary clearances and exclude the mass-market industry. In the absence of public literature, civilian product designers receive no training in emission security. Opportunities for simple low-cost countermeasures that require little more than awareness of the nature of such risks early in a design process are therefore missed. Secondly, with secret emission limits, users have no idea what level of protection is tested and how the unknown tradeoffs made in these specifications fit into their overall security concept and budget. And finally, academic researchers may simply find it less appealing to try advancing a field in which most existing work remains secret and much of the state of the art has to be reinvented first.

Recent Freedom-of-Information-Act requests to declassify the US "Tempest" standards resulted only in excerpts that describe some terminology and widely known EMC test and calibration methods [5,6,7,8,9]. The actual conformance limits and full test procedures remain unavailable, along with the literature that justifies the design of these standards.

Today, most information-processing facilities with high protection requirements use civilian off-the-shelf technology, procured according to open standards. This calls, in my opinion, for a new generation of emission-security test standards that is based entirely on published data and experimental techniques. Their development should follow the established procedures of international standardization. Any underlying data should be open to scrutiny by academic peer review, to prevent that any tradeoffs that have to be made could be influenced by conflicting concerns of the signal-intelligence community. A model for such an effort could be the work that led to international standards on electromagnetic compatibility [14].

Designing a comprehensive family of open emission-security standards will require a substantial interdisciplinary research effort. While similar principles can be applied to a wide range of information-leaking channels, the specific parameters, test procedures, and appropriate economic tradeoffs can vary by orders of magnitude across different applications, countermeasures and signal types.

In the space of this paper, we will have to restrict the discussion, as an example, to one single class of signal, namely the radiated RF leakage of video signals. This remains a particularly easy to demonstrate risk, especially because of the redundancy offered by a repetetive signal, and because the eavesdropper needs to find only a small number of parameters to exploit the signal (namely, the rough pixel-clock frequency and the precise line and frame frequencies).

2 Existing Public Standards

No public emission-security standards exist today. Two types of electromagnetic-emission limits for information technology have been widely accepted by the market already, but – as this section will show – neither was designed to reduce the risk of information-carrying emanations, nor is any of them even remotely suited to do so.

2.1 Ergonomic Standards

Since about 1992, "low radiation" CRT computer monitors with improved electromagnetic shielding have been on the market. They confirm to standards aimed at reducing the exposure of humans to electromagnetic fields, to address fears about their potential biological effects [11,12]. The TCO'92 specification developed by the Swedish Confederation of Professional Employees (TCO) limits only low-frequency fields below 400 kHz, which are those generated by CRT deflection coils. Compromising emanations are typically significantly weaker and occur at much higher frequencies in the HF/VHF/UHF bands (3 MHz–3 GHz). Therefore, a TCO'92 conformance test will provide no information about emission-security properties of a video-display system.

2.2 Radio-Frequency Interference Standards

The second class of civilian electromagnetic emanation standards is aimed at minimizing interference with radio communication services. The international specification CISPR 22 [13] is today a legal requirement in most industrialised countries. This standard imposes the following radiated emission limits for "Class B" devices:

- Electric fields must not exceed 30 dBμV/m at 10 m distance in any 120 kHz passband in the frequency range 30–230 MHz.
- Electric fields must not exceed 37 dBμV/m at 10 m distance in any 120 kHz passband in the frequency range 230–1000 MHz.

The field strength is determined with a special AM measurement receiver with a *quasi-peak* detector specified in [14]; the output of such a detector rises with a time constant of only 1 ms and falls with a time constant of 550 ms. Less strict "Class A" limits are defined for devices that are only used in industrial environments. The standard also limits emissions below 30 MHz via power and communications cables.

A brief look at the motivation and design of the radio-interference test standards helps to understand why they are not suited for emission security purposes. Radio broadcasters aim at ensuring a minimum field strength of about 50–60 dBμV/m throughout their primary reception area [16]. The CISPR limits were selected about 20 dB below that level to ensure that, at 10 m distance, the interference from a device will not limit the received signal-to-noise ratio to less than 20 dB.

The quasi-peak detector is used as a psychophysical estimation tool. It provides a measure of the approximate annoyance level that impulses of various strengths and repetition frequencies cause for human users of analog audio and television receivers. Strong disturbance impulses are tolerated if they occur sufficiently rarely, and even weak disturbances can be annoying at high repetition rates.

3 Considerations for Emission Security Limits

Eavesdroppers can work with significantly lower signal levels than what might cause interference with radio and TV reception. They are concerned about how the emitted compromising signal compares in strength to the background noise, not to a broadcasting station. They can be expected to

- use high-gain antennas directed towards the emitting target device,
- look for the broadband impulses from digital signals in a quiet part of the spectrum, without interference from broadcasting stations,
- use notch filters to suppress strong narrow-band sources that interfere with the eavesdropped signal,
- use signal-processing techniques such as periodic averaging, cross-correlation, digital demodulation, and maximum-likelihood symbol detection, in order to separate the wanted information-carrying signal from unwanted background noise.

The emission limits, therefore, have to be based on an understanding of reasonable best-case assumptions of

- the minimal background noise that the eavesdropper faces even under good receiving conditions,
- the gain from antenna types that can be used covertly,
- the gain from the use of suitable detection and signal-processing methods for the signal of interest,
- the closest distance between antenna and target device for which protection is needed.

The goal of an emission-security test standard is to provide an upper bound for the signal-to-noise ratio S/N that a radio-frequency eavesdropper could achieve in practice. We then need a model that relates such a value to the outcome of practically repeatable measurements that can be performed as part of a security evaluation in a controlled test environment. If we combine the major factors that attenuate a compromising signal for an eavesdropper, compared to a laboratory measurement, with the major factors by which an eavesdropper can boost the signal, we obtain such a model in form of the formula

$$S/N = \frac{\hat{E}_B \cdot G_\mathrm{a} \cdot G_\mathrm{p}}{a_\mathrm{d} \cdot a_\mathrm{w} \cdot E_{\mathrm{n},B} \cdot f_\mathrm{r}}, \tag{1}$$

where

$\hat{E}_B$	is the maximum field strength that the test standard permits,
B	is the impulse bandwidth [14,17] of the receiver used in the test,
a_d	is the free-space path loss caused by placing the eavesdropper's antenna at distance d from the target device, instead of the antenna distance $\hat{d}$ used during the test,
a_w	is any additional attenuation in the radiation path (e.g. building walls),
G_a	is the gain of the best directional antenna that is feasible for use by the eavesdropper,
G_p	is the processing gain that can be achieved with signal processing,
$E_{\mathrm{n},B}$	is the field strength of natural and man-made radio noise at the location of the eavesdropping antenna within a quiet band of width B,
f_r	is the noise factor of the eavesdropper's receiver.

The expected noise levels and attenuation values in the above equation are random variables, which, in the absence of better data, have to be modeled as being normally distributed with some mean and variance determined from the statistical evaluation of a large number of measurements in various environments. For the other parameters, reasonable estimates based on practical demonstrations have to be made, so that an emission limit $\hat{E}_B$ can be selected that will keep the eavesdropper's signal-to-noise ratio below an acceptable level with sufficient probability. Different types of target signal are located in different frequency bands and permit different processing gains. Therefore, the above parameters will have to be estimated separately for each signal type of interest. General emission limits would have to consider for each frequency band the lowest acceptable source signal strength $\hat{E}_B$ for all types of compromising signals.

3.1 Radio Noise

A standard survey-data reference for the noise levels to be expected in various environments throughout the radio spectrum exists in the form of ITU-R Recommendation P.372 [18], which summarizes the results from numerous noise intensity measurements and categorizes their origin.

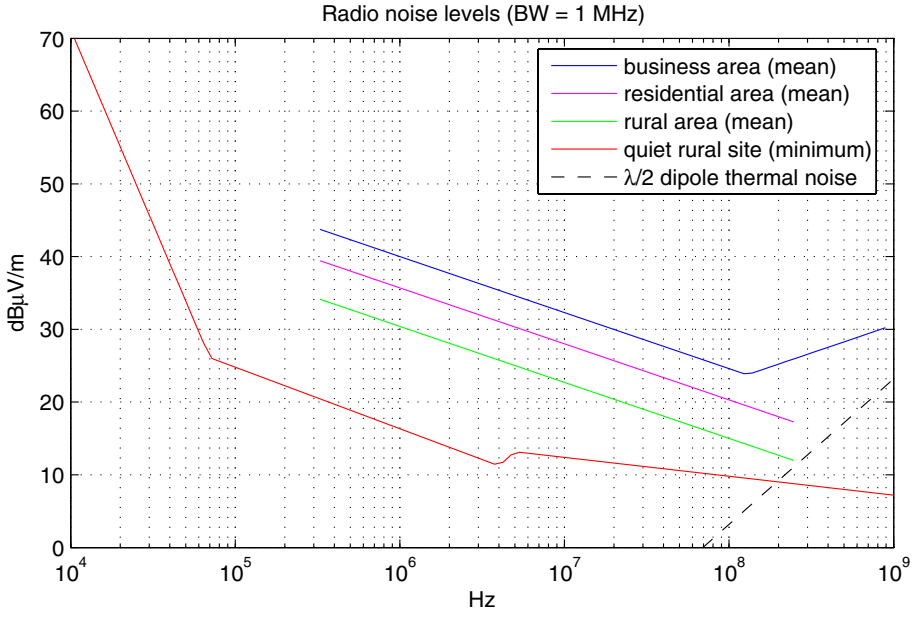


Fig. 2. Expected electric field noise levels $E_{\mathrm{n,1\ MHz}}$ excluding transmitter stations based on data from ITU-R P.372 [18]. The curves are for a receiver bandwidth of 1 MHz. Add 3/7/10/13/17/20 dB for bandwidths of 2/5/10/20/50/100 MHz, respectively. Subtract 37/20/9 dB for the 0.22/9/120 kHz bandwidth used in CISPR 16 measurements.

The mean noise levels are provided in ITU-R P.372 in form of an *external noise factor* $f_{\mathrm{a}} = P_{\mathrm{n},B}/kT_0B$ which compares the noise energy picked up over a bandwidth B by an antenna with the thermal noise energy from a resistor at room temperature T_0 (k is Boltzmann's constant).

Using the signal-power to field-strength relationship of an omnidirectional antenna (for details see [2, p. 91]), we can convert these noise figures into equivalent field strengths, which is the unit commonly used in electromagnetic emission standards.

Figure 2 uses these values, and others from [18], to estimate electric field levels at both quiet rural sites and business districts for 1 MHz bandwidth. In measurements, the receiver bandwidth will have to be smaller than the center frequency, therefore the shown curves are directly applicable only for frequencies of about 2 MHz and higher. For lower frequencies, lower bandwidths will have to be used. For noise, the received power increases proportional to the bandwidth of the receiver (10× larger bandwidth leads to 10 dB higher field strength).

It is worth noting that these are out-door levels and that this background noise might be attenuated if the eavesdropper and the target device are both located in the same building.

3.2 Radio Signal Attenuation

In free space (vacuum, dry air, etc.) the power flux density (power per area) of a radio signal drops with the square of the distance from a point source, because preservation of energy requires that the power flux density remains constant when it is integrated over the closed surface of a volume that contains the transmitter. The power flux density is proportional to the square of the electric field strength, therefore, the electric field strength will at distance d be reduced by a factor

$$a_{\mathrm{d}} = d/\hat{d} \tag{2}$$

compared to the value at a reference distance $\hat{d}$. In other words, increasing the distance to a transmitter in free space by a factor of 10 will reduce the signal strength by 20 dB.

Compared with the available references on out-door radio noise, somewhat less clear data is available in the literature on the in-door radio signal attenuation by building materials. Two survey publications [19,20] provide data for the frequency range of 900 MHz to 100 GHz. However, this data shows only a few trends and mostly documents a significant variability between different buildings.

A model in [19] suggests for 900 MHz a path loss of $a_{\mathrm{d}} \cdot a_{\mathrm{w}} = (d/\hat{d})^{1.65}$ for typical wall spacings and materials for an office environment on the same floor. An additional loss factor of 0/9/19/24 dB has to be added if the two antennas are 0/1/2/3 floors apart in a building. In large open rooms, attenuation is dominated by free-space loss ($a_{\mathrm{d}} \cdot a_{\mathrm{w}} = (d/\hat{d})^1$). In corridors it is slightly lower ($a_{\mathrm{d}} \cdot a_{\mathrm{w}} = (d/\hat{d})^{0.9}$), because walls can reflect the signal or act like a wave guide. The literature survey in [20] lists a number of alternative, but similar, models that have been used to describe attenuation in buildings, e.g. where the exponent applied to $d/\hat{d}$ increases from 1 to 6 as the distance increases from 1 m to 40 m. It also features measurements of individual building components, e.g. 3.8 dB for a double plasterboard wall or 7 dB for a 200 mm concrete block wall. Less data is available for VHF frequencies (30–300 MHz), for example [21], which suggests that building attenuation a_{w} is mostly in the range 5–45 dB. This reference also notes that VHF field strength can vary inside buildings by as much as 20 dB within a meter, which agrees with my own experience from eavesdropping demonstrations with handheld antennas inside office buildings.

Given the wide variability of radio-signal attenuation encountered in buildings, it seems not prudent to base a protection standard on any higher value for a_{w} than what is encountered in the lower decile of the available statistics, namely in the region of $a_{\mathrm{w}} = 5$ dB, a typical attenuation provided by 1–2 walls.

3.3 Antenna Gain

The compact broadband antennas that are commonly used for radio-interference compliance measurements, such as biconical, log-periodic, log-spiral, or double-ridged-horn designs, have only little directional gain G_{a}, typically about 2–6 dBi ("dBi" refers to a decibel gain compared to an isotropic antenna).

One of the most practical families of high-gain antennas for the UHF and higher VHF frequency range is the Yagi-Uda type, some forms of which are well known through their widespread use for domestic terrestrial UHF TV reception. Such an antenna is half as wide as the wavelength λ and can be designed such that its gain is

$$G_{\mathrm{a}} = 7.8\ \mathrm{dB} \cdot \log_{10} \frac{l}{\lambda} + 11.3\ \mathrm{dBi}, \tag{3}$$

where l is the length of the antenna [23, p. 458]. Increased gain and length of an antenna comes with reduced bandwidth. For the frequencies (200–400 MHz) and bandwidths (50 MHz) that are often best suited for video-signal eavesdropping, Yagi antennas with four elements seem to be an acceptable compromise, with a gain of 8.6 dBi and a length $l \approx \lambda/2$. Further gain can be achieved by connecting a group of Yagi antennas together, and each doubling of their number will in practice increase the directional gain by 2.5–2.8 dB.

As a practical example, a 2×3 group of six Yagi antennas with four elements, each tuned for the 350 MHz center frequency from Fig. 1, would be $0.43 \times 1.3 \times 1.1$ m large [2, p. 95] and could be hidden and handled quite easily behind a window or inside a suitable vehicle. It would provide a gain of about $G_{\mathrm{a}} = 16$ dBi. Doubling the reception frequency roughly quadruples the number of dipoles that can fit into the same space, leading to 5–6 dB more gain.

3.4 Processing Gain

Averaging is a practical and highly effective technique for increasing the signal-to-noise ratio of a periodic signal, such as that generated by the image-refresh circuitry in a video display system.

If X_i are independent random variables, then the variance of their sum will be the sum of their variances: $\mathrm{Var}(\sum_i X_i) = \sum_i \mathrm{Var}(X_i)$. The variance of a radio-signal voltage corresponds to its average power. If we add two sine waves with a random phase relationship together, the expected power of the result is the sum of the powers of each input signal. However, if we add two identical sine waves together, their voltage will add up and thus their power quadruple.

Similarly, adding two recorded segments of independent noise together will double the power of the noise and increase its root-mean-square voltage by a factor of $\sqrt{2} = 3$ dB. On the other hand, adding two phase-aligned repetitions of the same waveform together will increase its voltage by a factor of $2 = 6$ dB (and will therefore quadruple its power). When two recorded signals contain both independent noise and a wanted phase-aligned signal, then adding the two together will increase the signal-to-noise ratio by 3 dB. (Dividing by the number of added signals, to complete the average calculation, will not affect the SNR.)

This can be generalized to a processing gain of

$$G_{\mathrm{p}} = \sqrt{N} = 3\ \mathrm{dB} \cdot \log_2 N = 10\ \mathrm{dB} \cdot \log_{10} N \tag{4}$$

when N repetitions of a signal can be observed and added up with correct phase alignment.

How many frames of a video signal can be averaged in practice depends on a number of factors:

- When the screen content is stable for a time period T, then obviously up to $f_{\mathrm{v}}T$ frames can be received, where f_{v} is the frame rate or vertical deflection frequency of the screen. T can range in practice, depending on user behavior, from a few seconds to many minutes or longer, limiting N to about 10^2–10^6.
- Periodic averaging of a video signal can only be successful if the refresh frequency f_{v} can be determined with a relative error of less than $[2x_{\mathrm{t}}y_{\mathrm{t}}(N-1)]^{-1}$, where $x_{\mathrm{t}}y_{\mathrm{t}}$ is the total number of pixels including those representing the blanking intervals, if we demand that pixel intervals in the first and last averaged frame overlap by at least half a pixel time [2]. The frequency of crystal oscillators used in graphics cards wanders out of such a tight frequency tolerance within a small number of seconds, limiting N to less than 10^3 for averaging based on a manually adjusted vertical sync signal generated in an independent oscillator.
- An alternative to reconstructing the sync signal and averaging in real-time is to record the receiver output and then search for peaks in the auto-correlation of this signal. This way, the precise repetition frequency can be determined in a more compute-intensive post-processing step (as was done in Fig. 1). The number of frames will here be limited by the available acquisition memory. Storage oscilloscopes offer today 16–64 MB, which limits N to 10^1–10^2 frames. The available memory and processing power can be expected to grow further with Moore's law. They are limited in purpose-built hardware only by the eavesdropper's budget. With enough signal processing power available, this auto-correlation – which will only be evaluated for peaks near the expected frame time f_{v}^{-1} – could even be performed in real-time, leading not only to unlimited integration time but also to real-time monitoring of the result.

3.5 Bandwidth

A further consideration is the receiver bandwidth. We can expect an eavesdropper to work with a bandwidth B somewhere near the pixel frequency f_{p} of the video mode, as this is necessary to separate the impulses received from individual pixels and reconstruct the full video bandwidth. For larger fonts, text might remain intelligible with somewhat lower bandwidths.

In general, independent of the bit rate or pixel frequency of an eavesdropped digital signal, a higher bandwidth will lead to an improved signal-to-noise ratio. This is, because the compromising emanations of digital waveforms are in the form of switching impulses, which are inherently ultra-wideband signals. The frequency components of an impulse are correlated and therefore received impulse voltages will grow linearly with B (20 dB for every ten-fold increase in bandwidth), whereas thermal noise and narrow-band background signals are usually not correlated and their voltage will therefore only grow proportional to $\sqrt{B}$. As a result, in the best case, the signal-to-noise ratio can grow proportional to $\sqrt{B}$.

In practice, the ratio will grow somewhat less, because some of the unwanted background noise (for example emissions from other nearby computers) also has the form of broadband impulses. A reason for keeping the bandwidth small in an eavesdropping receiver is that this will make it more likely to find a quiet window in the radio spectrum that is not used continuously by powerful narrow-band transmitters such as radio and TV broadcast stations. Overall, a bandwidth in the order of $B = 50$ MHz would be a typical practical compromise for a readable video signal.

3.6 Signal-to-Noise Ratio

For the reconstruction of human readable text, a signal-to-noise ratio of at least 10 dB is necessary [2]. This requirement could perhaps – especially with much larger fonts – be reduced by a few dB when a symbol detector is used to automatically recognize characters, but the additional gain achievable here depends a lot on the font and will be significantly below the square-root of the number of pixels per character, as many characters such as `i,l,I,1` or `o,c,e` differ only in a few pixels. Therefore, a SNR of not more than 0 dB seems to be a reasonable security requirement.

4 Suggested Emission Limits

Based on all these considerations, we can now bring together possible values for equation (1). A calculation like the following example illustrates how a rationale for the emission limits in a compromising emanations test standard for video signals could look like:

- We measure field strength in the laboratory tests at a distance of $\hat{d} = 1$ m, which is already common practice in military EMC standards, e.g. [15].
- We assume our eavesdropper uses a directional Yagi antenna array like the one described in section 3.3 with $G_\mathrm{a} = 16$ dBi.
- We assume that an attacker will not get closer than $d = 30$ m with this type of antenna, therefore $a_\mathrm{d} = 30$ dB. In a quiet rural site, securing the area 30 m around a device should be feasible, whereas in an urban environment, space is typically more confined, but noise levels are also 10 dB higher, providing the same protection against attackers at 10 m distance. If the threat model includes attackers in nearby rooms in the same building ($d = 3$ m), the resulting test limits will have to be lowered 10 dB further.
- We want to ensure protection even for rooms whose attenuation by building materials is located in the lowest decile of the available statistics and therefore use $a_\mathrm{w} = 5$ dB.
- We assume the attacker uses a receiver with a noise figure of $f_\mathrm{r} = 10$ dB (the value given for the Dynamic Sciences R-1250 wideband Tempest receiver)
- We assume a receiver (impulse) bandwidth of $B = 50$ MHz.
- We assume that an attacker will in practice have difficulties with aligning the antenna, tuning to a suitable center frequency, and synchronizing to the

exact frame rate if there is no visibly usable signal after averaging $N = 32$ frames and therefore we assume a possible processing gain of $G_\mathrm{p} = 15$ dB.
- From Fig. 2 we can see that for the HF and VHF frequency range of 3 MHz to 300 MHz the background noise level $E_{\mathrm{n},1\ \mathrm{MHz}}$ remains above about 10 dBμV/m, even at a quiet receiver site. Above 200 MHz thermal noise from the antenna itself becomes the limiting factor, increasing with 20 dB per decade. After adjusting the bandwidth, we get $E_{\mathrm{n},50\ \mathrm{MHz}} = 27$ dBμV/m.

When we require $S/N \leq 0$ dB and use all of the just listed parameters in (1), we end up with

$$\begin{aligned}\hat{E}_{50\ \mathrm{MHz}} &\leq \frac{S/N \cdot a_\mathrm{d} \cdot a_\mathrm{w} \cdot E_{\mathrm{n},50\ \mathrm{MHz}} \cdot f_\mathrm{r}}{G_\mathrm{a} \cdot G_\mathrm{p}} \\ &= 0\ \mathrm{dB} + 30\ \mathrm{dB} + 5\ \mathrm{dB} + 27\ \mathrm{dB\mu V/m} + 10\ \mathrm{dB} - 16\ \mathrm{dBi} - 15\ \mathrm{dB} \\ &= 41\ \mathrm{dB\mu V/m}.\end{aligned}$$

4.1 Feasibility of Verification

It would be desirable if the suggested limits were verifiable with off-the-shelf EMC measurement equipment, such as a normal spectrum analyzer. Unlike a sophisticated eavesdropper, a spectrum analyzer will not be able to utilize the processing gain offered by periodic averaging. Therefore, eavesdroppers can still use signals that are not visible on a spectrum analyzer. To compensate for this, we have to bring during spectrum analyzer tests the antenna as close as possible to the equipment under test, that is $\hat{d} = 1$ m, even if that means that we might encounter some near-field effects.

Like the eavesdropper, the test procedure should work at as high a bandwidth as possible to make use of the fact that impulse voltage will grow proportional to B, whereas noise voltage grows proportional with $\sqrt{B}$. A wide-band receiver suitable for video eavesdropping uses extra-high intermediate frequencies in order to provide large bandwidths of 50 MHz and more. Measurements at such bandwidths are not possible with commonly used spectrum analyzers, whose intermediate frequencies only allow a maximum impulse bandwidth of 5 or 1 MHz. The corresponding limits would be 20 or 34 dB lower, respectively:

$$\begin{aligned}\hat{E}_{5\ \mathrm{MHz}} &= 21\ \mathrm{dB\mu V/m} \\ \hat{E}_{1\ \mathrm{MHz}} &= 7\ \mathrm{dB\mu V/m}\end{aligned}$$

The limits have to be above the receiver noise floor of a spectrum analyzer to be verifiable. For example, the Agilent E4402B spectrum analyzer with preamplifier has, according to manufacturer claims [24, p. 235], at a resolution bandwidth of 1 kHz a noise level of −133 dBm. This corresponds to 4 dBμV at 1 MHz and 11 dBμV at 5 MHz. The antenna factor (ratio between field strength in V/m and voltage) for a typical passive measurement antenna at frequencies up to 100 MHz is not more than 10 dB. Therefore, the noise floor of the above spectrum analyzer corresponds to field strengths of 14 dBμV/m at 1 MHz and 21 dBμV at

5 MHz bandwidth. This makes spectrum analyzer verification of the electric field strength limits problematic at a bandwidth of 1 MHz, and just about feasible at 5 MHz, at least with passive antennas. Active antennas designed for use in anechoic chambers[1] might therefore have to be used instead, which offer a 20 dB lower antenna factor and a sensitivity of 6 dBμV/m at 1 MHz.

Growing antenna factors make the limit of $\hat{E}_{5\ \mathrm{MHz}} = 21$ dBμV/m also problematic to verify for frequencies above 100 MHz, but the eavesdropper would experience thermal noise as well here at the assumed quiet receiver site. Therefore it seems acceptable to increase the limit proportional to the frequency, starting at 100 MHz, up to at least 1 GHz. For higher frequencies, the eavesdropper can use parabolic antennas with higher gain to overcome the thermal noise.

4.2 Comparison with Limits in Other Standards

In order to compare these proposed limits with those defined in CISPR 22 Class B, we have to take into account the different bandwidths and antenna distances. To increase the impulse bandwidth from 120 kHz to 5 MHz, we have to raise the permitted field strength by 32 dB, in order to keep the equivalent spectral density constant. The limits have to be raised further by 20 dB to convert the measurement distance from 10 m to 1 m.

This way, we can compare the emission security test limits proposed here with the established EMC emission limits. Radiated VHF field strength has to be 61 dB lower than allowed in CISPR 22. This corresponds to a reduction of the maximum tolerated eavesdropping distance by a factor of 10^3. In other words, the CISPR 22 EMC limits do not prevent devices from emitting pulses that could, under ideal conditions, be received kilometers away.

We can conclude from this that a shielded room with an attenuation of 60 dB across the HF/VHF/UHF frequency range for radiated emissions should provide adequate protection, if everything operated inside it complies with the CISPR 22 Class B limits, as can be expected from all currently available office equipment. It is worth stressing that this entire analysis concentrates on VHF video emanations. It should be applicable to similar high-speed digital signals such as those from system busses, but it does not take into account any low-frequency magnetic or acoustic emanations from electromechanic equipment, as they might have been a concern, for example, with some historic printers.

If the protection provided by the shielded room must be effective immediately outside the room at a very quiet site ($d = 0.3$ m), and not only in 30 m distance, then another 40 dB attenuation is required, leading to the 100 dB attenuation defined in the NSA specification for shielded enclosures [10].

The US military EMC standard MIL-STD-461E [15] provides in its requirement R102 for mobile Army and Navy equipment radiated electric limits field limits similar to those suggested here, namely measured at 1 m distance not more than 24 dBμV/m from 2 to 100 MHz, and then increasing with 20 dB per decade up to 18 GHz. However, the measurement bandwidth is with 10 kHz

[1] e.g., Rohde & Schwarz AM524 low-noise active antenna system.

up to 30 MHz, 100 kHz in the 30–1000 MHz range and 1 MHz for frequencies above 1 GHz comparable to what CISPR 22 uses, and therefore still 37 dB less sensitive for broadband impulse signals than the limits proposed here.

The different emission-control standards can also be compared via the spectral density of the strongest radiated impulse that they permit (measured at 100 MHz and 1 m distance). It has 68 dBμV/(m · MHz) under CISPR 22 Class B, 44 dBμV/(m · MHz) with MIL-STD-461E/R102, and 7 dBμV/(m · MHz) for the emission-security limit proposed here.

Dropping the logarithmic scales and the dependence on a measurement distance, we can also compare the radiated impulse emission limits in terms of the peak effective isotropic radiated power (EIRP) permitted within a given bandwidth. For a 50 MHz wide band, this would be about 0.5 mW under CISPR 22 Class B, 2 μW under MIL-STD-461E/R102, and 0.3 nW under the limit proposed here. For comparison, the peak EIRPs observed during clearly readable eavesdropping demonstrations in [2] were in the range 10–240 nW. The 10 dB stricter limit to protect even against an eavesdropper in a neighbor room in an urban environment would be 30 pW.

4.3 Other Considerations

The general measurement procedure and setup (use of a wood table over a ground plane, arrangement of cables and impedance stabilization networks, etc.) in an emission security standard could be adopted largely from existing EMC specifications such as CISPR 22 or MIL-STD-461E. Some changes that would have to be made include:

- CISPR 16-1 suggests that the ambient noise levels at a test site should be 6 dB below the measurement limits, for perfect results even 20 dB below. While CISPR 22 describes the use of an open area test site, this will hardly be feasible with the emission security limits proposed here. A well-shielded anechoic chamber will be required instead as a measurement site, to remain at least 6 dB below the measurement limits.
- While the CISPR 22 limits are for a quasi-peak detector, emission security tests should be performed with peak detectors, because rather than perceived annoyance, the separability from noise is the concern. The video bandwidth should not be limited in spectrum analyzer measurements.

The HF/VHF/UHF emission limits suggested here are justified based on video signal eavesdropping, but they are also likely to provide adequate protection against other forms of radiated compromising emanations above about 5 MHz. It seems unlikely that other emanations will offer substantially higher processing gain than video signals, except perhaps carefully encoded intentional broadcasts by malicious software. In addition, the suggested limits are already very close to what seems technically feasible in the form of generic limits on spectral energy as it can be measured with EMC broadband antennas and spectrum analyzers in anechoic chambers.

Different measurement techniques can be used in order to apply the same limits to specific signals that are suspected of being emitted. In the case of a periodic signal, such as the output of a video refresh circuit, it is possible to use a wide-band receiver, just as an eavesdropper would, together with a suitable storage oscilloscope that is triggered from the vertical sync signal. Averaging $N = 1024$ frames (about 12 s of video signal) will lead to a processing gain of 30 dB. If all 1024 lines of the test image are identical and averaged as well, this will lead to another 30 dB gain, making the measurement in principle better than one that took place inside a shielded room with 60 dB attenuation.

The limits discussed here are aimed at products in which emission security is achieved by signal suppression and shielding. Other eavesdropping countermeasures, such as jamming, would require rather different protection standards.

5 Conclusions

We outlined design considerations for a security standard on radiated compromising emanations from video systems. Due to the redundancy of a periodic signal and due to the ultra-wideband nature of compromising emanations from digital baseband signals, meaningful emission limits end up near the performance limits of modern spectrum analyzers. If the protection is to be achieved by shielding and attenuation, the permitted signal power must be several million times lower than what civilian radio-interference standards permit.

References

1. Wim van Eck: Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk? Computers & Security, Vol. 4, pp. 269–286, 1985.
2. Markus G. Kuhn: Compromising emanations: eavesdropping risks of computer displays. Technical Report UCAM-CL-TR-577, University of Cambridge, Computer Laboratory, December 2003. `http://www.cl.cam.ac.uk/TechReports/`
3. Markus G. Kuhn: Electromagnetic Eavesdropping Risks of Flat-Panel Displays. 4th Workshop on Privacy Enhancing Technologies, 23–25 May 2004, Toronto, LNCS 3424, Springer.
4. Suresh Chari, Josyula R. Rao, Pankaj Rohatgi: Template Attacks. 4th International Workshop on Cryptographic Hardware and Embedded Systems, LNCS 2523, Springer, 2002, pp. 13–28.
5. National Security Telecommunications and Information Systems Security Advisory Memorandum NSTISSAM TEMPEST/1-92: Compromising Emanations Laboratory Test Requirements, Electromagnetics. National Security Agency, Fort George G. Meade, Maryland, 15 December 1992. Partially declassified transcript: `http://cryptome.org/nsa-tempest.htm`
6. NACSIM 5000: Tempest Fundamentals. National Security Agency, Fort George G. Meade, Maryland, February 1982. Partially declassified transcript: `http://cryptome.org/nacsim-5000.htm`
7. National Security Telecommunications and Information Systems Security Advisory Memorandum NSTISSAM TEMPEST/2-95: RED/BLACK Installation Guidance. National Security Agency, Fort George G. Meade, Maryland, 12 December 1995. Transcript: `http://cryptome.org/tempest-2-95.htm`

8. National COMSEC/EMSEC Information Memorandum NACSEM-5112: NONSTOP Evaluation Techniques. National Security Agency, Fort George G. Meade, Maryland, April 1975. Partially declassified transcript: http://cryptome.org/nacsem-5112.htm
9. National Security Telecommunications and Information Systems Security Instruction NSTISSI No. 7000: TEMPEST Countermeasures for Facilities. National Security Agency, Fort George G. Meade, Maryland, 29 November 1993. Partially declassified transcript: http://cryptome.org/nstissi-7000.htm
10. Specification NSA No. 94-106: Specification for Shielded Enclosures. National Security Agency, Fort George G. Meade, Maryland, 24 October 1994. Transcript: http://cryptome.org/nsa-94-106.htm
11. TCO'99 – Mandatory and recommended requirements for CRT-type Visual Display Units (VDUs). Swedish Confederation of Professional Employees (TCO), 1999. http://www.tcodevelopment.com/
12. Procedure for Measurement of Emissions of Electric and Magnetic Fields from VDUs from 5 Hz to 400 kHz. European Computer Manufacturers Association, Standard ECMA-172, June 1992. (also IEEE Std 1140-1994)
13. Information technology equipment – Radio disturbance characteristics – Limits and methods of measurement. CISPR 22, International Electrotechnical Commission (IEC), Geneva, 1997. (also EN 55022)
14. Specification for radio disturbance and immunity measuring apparatus and methods. CISPR 16, International Electrotechnical Commission (IEC), Geneva, 2000.
15. Requirements for the Control of Electromagnetic Interference Characteristics of Subsystems and Equipment. MIL-STD-461E, US Department of Defense, Interface Standard, 20 August 1999.
16. Reinaldo Perez (ed.): Handbook of Electromagnetic Compatibility. Academic Press, 1995.
17. IEEE Standard for the Measurement of Impulse Strength and Impulse Bandwidth, ANSI/IEEE Std 376-1975.
18. Radio noise. Recommendation ITU-R P.372-7, International Telecommunication Union, Geneva, 2001.
19. Propagation data and prediction methods for the planning of indoor radiocommunication systems and radio local area networks in the frequency range 900 MHz to 100 GHz. Recommendation ITU-R P.1238-2, International Telecommunication Union, Geneva, 2001.
20. Homayoun Hashemi: The Indoor Radio Propagation Channel. Proceedings of the IEEE, Vol. 81, No. 7, July 1993, pp. 943–968.
21. L. P. Rice: Radio Transmission into Buildings at 35 and 150 mc [MHz]. Bell System Technical Journal, Vol. 38, No. 1, January 1959, pp. 197–210.
22. M. Zimmermann, K. Dostert: A Multipath Model for the Powerline Channel. IEEE Transactions on Communications, Vol. 50, No. 4, April 2002, pp. 553–559.
23. Karl Rothammel, Alois Krischke: Rothammels Antennenbuch. Franckh-Kosmos, Stuttgart, 1995.
24. Test & Measurement Catalog 2001, Agilent Technologies, USA.

Security Evaluation Against Electromagnetic Analysis at Design Time

Huiyun Li, A. Theodore Markettos, and Simon Moore

Computer Laboratory, University of Cambridge, JJ Thomson Avenue, Cambridge CB3 0FD, UK
Huiyun.Li@cl.cam.ac.uk

Abstract. Electromagnetic analysis (EMA) can be used to compromise secret information by analysing the electric and/or magnetic fields emanating from a device. It follows differential power analysis (DPA) becoming an important side channel cryptanalysis attack on many cryptographic implementations, so that constitutes a real threat to smart card security. A systematic simulation methodology is proposed to identify and assess electromagnetic (EM) leakage characteristics of secure processors at design time. This EM simulation methodology involves current flow simulation, chip layout parasitics extraction, then data processing to simulate direct EM emissions or modulated emissions. Tests implemented on synchronous and asynchronous processors indicates that the synchronous processor has data dependent EM emission, while the asynchronous processor has data dependent timing which is visible in differential EM analysis (DEMA). In particular, DEMA of amplitude demodulated emissions reveals greater leakage compared to DEMA of direct emissions and DPA. The proposed simulation methodology can be easily employed in the framework of an integrated circuit (IC) design flow to perform a systematic EM characteristics analysis.

Keywords: EM side-channel analysis; smart card; design time security evaluation.

1 Introduction

Smart cards are widely used for authentication and securing transactions. Their cryptographic operations are based on symmetrical or asymmetrical cryptographic algorithms such as triple DES, AES or RSA. But even if the cryptographic algorithms and the protocols are secure, information about secret data may leak through side-channels such as timing of computation [1], power consumption [2], as well as electromagnetic radiation [3]. In the EM side-channel, a smart card emits different amounts of EM emission during the computation depending on the instructions and data being executed. Some sophisticated statistical techniques such as differential electromagnetic analysis (DEMA) [3, 4, 5] can detect variations in EM emission so small that individual key bits can be identified. This means secret key information can be recovered from the secure devices.

To keep these devices secure against the EM side-channel attacks, a huge amount of research has been undertaken. However, in common industrial practise, the security evaluation of the secure device designs is only performed after chips are manufactured.

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 280–292, 2005.

This post-manufacture analysis is time consuming, error prone and very expensive. This has driven the study of the design-time security evaluation which aims to examine data-dependent EM characteristics of secure processors, so as to assess their security level against EM side-channel analysis attacks.

The most straightforward way to simulate EM waves propagating in a circuit is to use a 3D or planar EM simulator, which involves solving Maxwell's equations for the electric and magnetic vector fields in either the frequency or time domain. However a full-wave 3D simulator incorporating characterised nonlinear[1] semiconductor devices is too time consuming to be practical for chip-level analysis. Various types of field sensors, namely electric or magnetic field sensor measuring in near or far field, used by attackers also increase the challenges in EMA simulation. Different types of sensors measure different types of field, so they require different simulation methods. Furthermore, the modulated EM emissions [4] have begun arousing attention in the cryptanalysis community as well as the direct EM emissions that are normally exploited in EM analysis attacks [5]. Modulated emissions occur when a data signal modulates carrier signals which then generate EM emissions propagating into the space. Different modulation mechanisms require different demodulation manners.

In this paper, we present a design time security evaluation methodology for EM side-channel analysis. It first partitions an electronic system under test into two parts: the chip and the package. The package is simulated in an EM simulator and modelled with lumped parameters R, L and C. The chip incorporating the package lumped parameters is then simulated in circuit simulators. This mixed-level simulation obtains current consumption of the system under test accurately and swiftly. Next, the security evaluation methodology involves a procedure of data processing on the current consumption to simulate EM emissions. Different methods of data processing are required to target corresponding types of sensors. Furthermore, to simulate modulated EM emissions, demodulation in amplitude or angle is incorporated into the simulation flow.

The rest of the paper is organised as follows. In Section 2, we present our simulation methodology of system partitioning and simulation procedures incorporating different types of EM emissions and different field sensors. In Section 3, we demonstrate simulation results for two processors on our test chip from which data dependent EM characteristic is successfully identified and verified by measurement results. Section 4 presents a brief conclusion.

2 Simulation Methodology for EM Analysis

2.1 System Partitioning

As described in Section 1, a 3D full-wave field simulator incorporating large number of semiconductor devices is too time consuming to be practical for chip-level analysis. Our simulation approach is to partition an electronic system into two parts. The first part is the chip, simulated in **circuit simulators** like SPICE, which is fundamentally flawed because wave coupling is not accurately represented even if transmission lines are used for the interconnects. However, the chip dimensions are small enough (compared to the

[1] Some examples of nonlinear components are Diode, BJT and MOSFET.

wavelength) to tolerate the errors[2]. The second part is the package and even the printed circuit board (PCB), which can be accurately simulated by a (3D or planar) **EM simulator** and be modelled with lumped components (R, L and C). The lumped elements will then be incorporated into the same circuit simulator to achieve the response of the entire system.

2.2 Simulation Procedure

The procedure to perform an EMA simulation on a chip design is shown in Figure 1. The EM analysis simulation flow is similar to that of power analysis which measures the global current of a device. However EM analysis may focus on a smaller block such as the ALU or the memory. In this case, a Verilog/SPICE co-simulation can be used where the partitioning function provides an easy means to select the desired block(s) to test. With Verilog/SPICE co-simulation, various instructions are easily executed and modified through testbench files written in Verilog. Accurate simulation of current consumption is achieved in the SPICE-like simulation. Once the current data $Idd(t)$ for the desired block(s) or a whole processor is collected, it is passed to MATLAB™ and is processed to implement DEMA according to the sensor types and emission types.

The data process procedure for EM analysis is shown in the shadowed box in Figure 2. It includes synchronising and re-sampling of two sets of current consumption data when the processor under test is computing with different operands. We perform signal processing on each set of current consumption data according to the types of EM emissions to be measured and according to the types of field sensors to measure the EM emissions.

Direct vs Modulated EM Emissions. EM emissions can be generally categorised into two types: direct emissions and modulated emissions [4]. *Direct emissions* are caused directly by current flow with sharp rising/falling edges. To measure direct emissions from a signal source isolated from interference from other signal sources, one uses tiny field probes positioned very close to the signal source and special filters to minimise interference. To get good results may require decapsulating the chip.

Modulated emissions occur when a data signal modulates carrier signals which then generate EM emissions propagating into the space. A strong source of carrier signals are the harmonic-rich square-wave signals such as a clock, which may then be modulated in amplitude, phase or some other manner. The recovery of the data signals requires a receiver tuned to the carrier frequency with a corresponding demodulator.

- Amplitude Modulation

  In a circuit, the data signal may couple to a carrier signal (e.g. clock harmonics) due to E field capacitive coupling or H field magnetic coupling, which generates a sum of the data signal and the carrier signal. Once these two coupled signals go through a square-law device (e.g. a transistor), the product of the two signals is generated.

[2] The velocity of electromagnetic propagation is limited by the laws of nature, and in silicon-dioxide it is approximately 1.5×10^8 m/s . Fast signal edges in smart card chips with an edge rate of under 1ns have to be considered as "high speed" only when the longest chip dimension is beyond 50mm, as a rule of thumb.

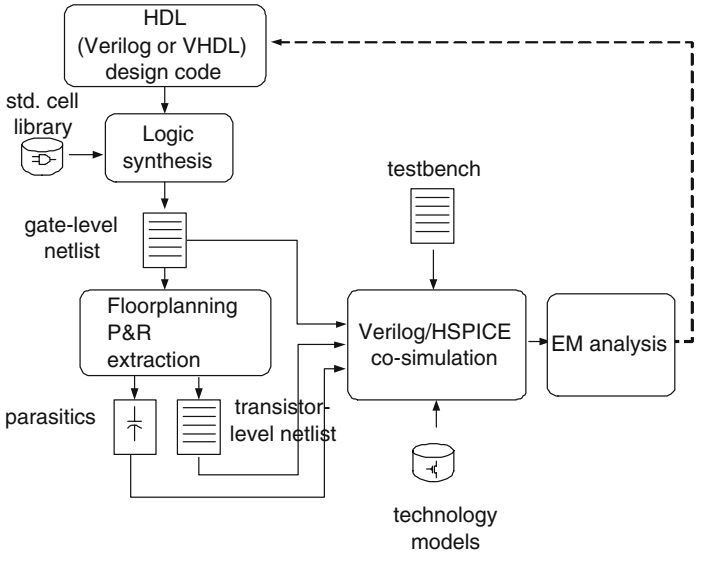


Fig. 1. Digital design flow with EM analysis

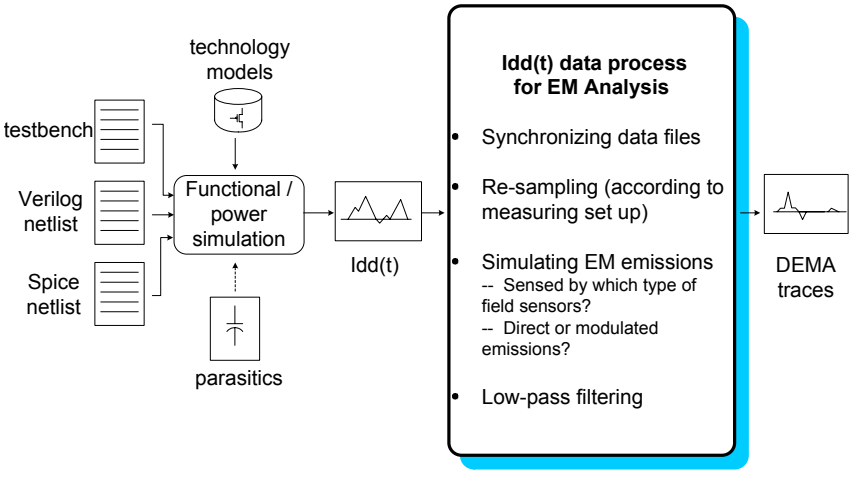


Fig. 2. EM analysis simulation procedure

For instance, an n-channel transistor operates in the saturation region when $V_{DS} > V_{GS} - V_{Tn}$, its drain current remains approximately constant as: $I_{DSn(sat)}$ [6]:

$$I_{DSn(sat)} = \frac{\beta_n}{2}(V_{GS} - V_{Tn})^2 \tag{1}$$

where the constant β_n denotes the n-channel transistor gain factor, V_{GS} denotes the gate-source voltage and V_{Tn} denotes the threshold voltage.

If the input V_{GS} is a clock signal (V_{clock}) coupled with a data signal (V_{data}): $V_{GS} = V_{data} + V_{clock}$, in which the square-wave clock signal V_{clock} can be represented as a Fourier series with the fundamental frequency f and all the odd harmonics: $\sum_{n=1,3,5...}^{\infty} \frac{4}{n\pi} \sin(2\pi nft)$.

The saturation current $I_{DSn(sat)}$ becomes:

$$I_{DSn(sat)} = \frac{\beta_n}{2}\left[V_{data} + \left(\sum_{n=1,3,5...}^{\infty} \frac{4}{n\pi}\sin(2\pi nft)\right) - V_{Tn}\right]^2 \quad (2)$$

Expanded, $I_{DSn(sat)}$ contains items of interest as the product of sinusoidal signals and the coupled data signal: $\beta_n \sum_{n=1,3,5...}^{\infty} \frac{4}{n\pi}\sin(2\pi nft)V_{data}$.

This process is amplitude modulation (AM), where the coupled data signal V_{data} modulates clock harmonics with diminishing magnitude. If the current $I_{DSn(sat)}$ is picked up by an EM sensor and fed into a bandpass filter tuned to a certain clock harmonic frequency, the signal V_{data} can be recovered. This process is amplitude demodulation.

Amplitude modulation can also occur in a transistor when the digital gate input V_{GS} is itself a square-wave, harmonic-rich signal. For example, in one cryptographic execution run, the input V_{GS1} is 00111100..., while in another run, the input at the same gate becomes V_{GS2} as 01010101.... Then V_{GS1} and V_{GS2} have Fourier series expressions different at some carrier frequencies. If a demodulator is tuned to one of these carrier frequencies, the difference of the coefficients in the Fourier series can be detected and viewed as a manifestation of the difference in V_{GS1} and V_{GS2}. This type of AM modulation mechanism is dominant for deep-submicron technologies[3]. In deep submicron processes, the dependence of saturation drain current $I_{DSn(sat)}$ on gate source voltage V_{GS} is better modelled by a linear rather than a quadratic relationship.

- Angle Modulation (phase or frequency modulation)

  Coupling of circuits can also result in changes in the angle (frequency or the phase) of the carrier signals. If there is a coupling between a data line and the internal clock circuitry, e.g. its voltage controlled oscillator (VCO), this coupling can affect the output clock frequency by affecting the VCO control voltage. The resulted clock frequency variation may be visible as data-dependent timing in differential EM analysis.

Exploiting modulated emissions can be easier and more effective than working with direct emission [4]. Some modulated carriers could have substantially better propagation than direct emission, which may sometimes be overwhelmed by noise. The modulated emission sensing does not require any intrusive/invasive techniques or fine grained positioning of probes.

Depending on the types of EM emissions in EMA attacks: direct emissions or modulated emissions, EMA simulation may require demodulation of corresponding manners of the modulation.

[3] Gate lengths below 0.35 μm are considered to be in the deep-submicron region.

EM Field Measurement Equipment. A number of sensors can be used to detect the EM signals in EMA attacks. They are divided into those detecting electric and those detecting magnetic fields in near-field[4], or those detecting far-field EM-field. In EM analysis attacks on small devices with weak EM emissions such as a smart card, near-field sensors are more appropriate.

An example of **near-field electric field sensors** is a monopole antenna. It generally measures the near-field electric component around current-carrying conductor where electric field magnitude $E \propto I$. **Near-field magnetic field sensors** generally measure the near-field magnetic component around current-carrying conductor where magnetic field magnitude $B \propto I$.

The simplest magnetic field sensor is a loop of wire. An EM field is induced in the loop due to a change in magnetic flux through the loop caused by a changing magnetic field produced by an AC current-carrying conductor. This is the transformer effect. The induced voltage is:

$$V = -\int_S \frac{\partial \mathbf{B}}{\partial t} \cdot d\mathbf{s} \tag{3}$$

over surface S using area element $d\mathbf{s}$. We can rewrite it into the following equation, which says the measurement output is proportional to the rate of change of the current which causes the magnetic field.

$$V = M\frac{dI}{dt} \tag{4}$$

where M denotes the mutual inductance between the sensor and the concerned circuit.

This type of field sensor senses the change of magnetic flux, so we use the rate of change of the current dI/dt to track EM emission. Simulation for this type of sensor involves differential calculus on current consumption data.

There are also **far-field electromagnetic field sensors** such as log-periodic antennas. They generally measure far-field electromagnetic field and often work with other equipment to harness modulated emissions. For example, an amplitude modulation (AM) receiver tuned to a clock harmonic can perform amplitude demodulation and extract useful information leakage from electronic devices [4].

This is not an exhaustive list of field sensors, but provides a view that different types of sensors measure different types of field, so that require different approaches in EM simulations.

Low-Pass Filtering Effect of EM Sensors. The last step of data processing procedure as shown in the shadowed box in Figure 2 is the low-pass filtering. Considering the inductance in field sensors, and the load resistance from connected instruments (e.g. an amplifier or an oscilloscope), an RL low-pass filter is formed as shown in Figure 3. Its 3dB cutoff [5] frequency is calculated as $f_{cutoff} = R/2\pi L$. Due to this RL low-pass filtering effect, the two sets of processed current consumption data have to be low-pass filtered at the end of the EMA data processing procedure.

[4] Near-field refers to a distance within one sixth of the wavelength from the source ($r < \lambda/2\pi$), while far-field refers to a distance beyond it ($r > \lambda/2\pi$).

[5] The frequency at which the output voltage is 70.7% of the input voltage.

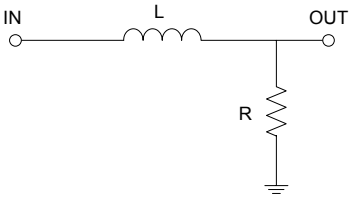


Fig. 3. RL low-pass filter

Finally, DEMA is performed by subtracting one EMA trace from another. Security weakness will be manifested as pulses in the DEMA trace, revealing data-dependent EM characteristics of the tested design. The term DEMA here (and further in this paper) refers to the variation (difference) in the EM emissions, instead of statistical treatment correlating the variation to hypothetical data being manipulated as in a real DEMA attack [3]. This is because the proposed methodology is to evaluate data-dependent EM characteristics of secure processor designs, which are the fundamental weakness a real DEMA attack exploits and can be identified with deterministic data.

3 Evaluation Results of the Simulation Methodology

3.1 EM Simulation Setup

DEMA simulation has been carried out on a test chip, fabricated in UMC 0.18μm six metal CMOS process as part of the G3Card project [7, 8]. Figure 4 shows a picture of the test chip which contains five 16-bit microcontroller processors with different design styles. This paper addresses the synchronous processor (S-XAP) on the top left corner and the dual-rail asynchronous processor (DR-XAP) in the middle.

We target simple instructions (e.g. XOR (exclusive OR), shift, load, store etc) which can give a good indication of how the hardware reacts to operations of cryptographic algorithms. A short instruction program runs twice with operands of different Hamming weight. The first run sets the I/O trigger port high by storing '1' into memory, computes '00 XOR 55', and sets the I/O trigger port low by storing '0' into memory, while the second run sets the I/O port high, computes '55 XOR 55', and sets the I/O port low.

3.2 EM Simulation of a Synchronous Processor

Figure 5 shows the EMA simulation over the S-XAP processor. We simulate direct EM emission picked up by an inductive sensor. On the graph we plot the EM traces of the processor for '00 XOR 55' and '55 XOR 55', as well as the differential EM plot of EMA1 - EMA2 (DEMA). The EM traces (EMA1 and EMA2) are superposed and appear as the top trace in Figure 5. The differential EM trace (DEMA) is shifted down from the centre by 6×10^{11} unit to clearly show its relative magnitude. The EM emission magnitude is computed through dI/dt as discussed in Section 2.2, thus has units of $\mu A/s$.

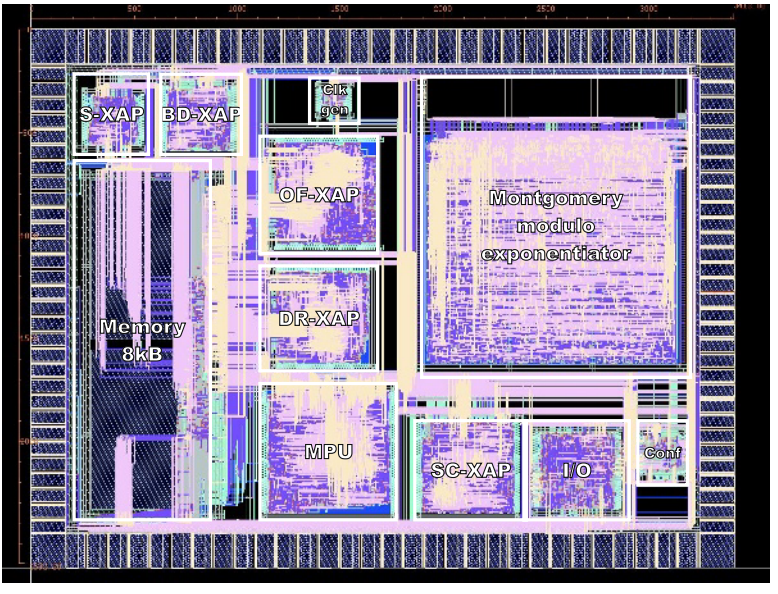


Fig. 4. The microcontroller processors (S-XAP, DR-XAP) on the chip are under EMA simulation test

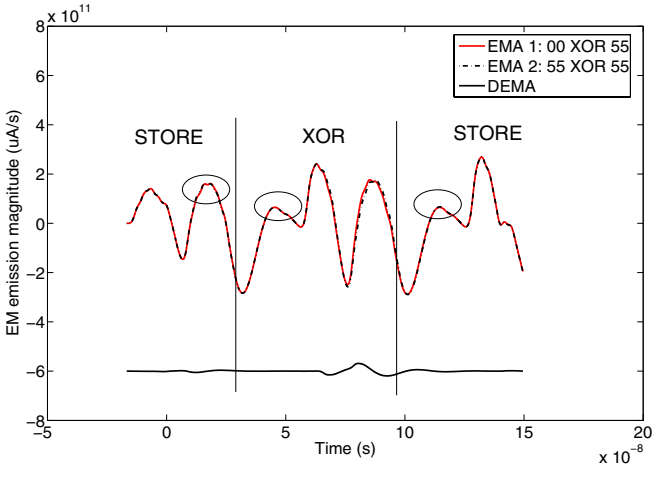


Fig. 5. EMA simulation over S-XAP processor executing `XOR` with different operands

The measurement of EM emissions on the same processor performing the same code is shown in Figure 6. The EM emissions are picked up by an inductive sensor over 5000 runs to average out the ambient noise (although 200 runs are enough), then are monitored on an oscilloscope. The inductive head in use has resistance R = 5.42Ω, inductance L = 9.16μH. When delivering power into a 4KΩ load, the 3dB cutoff is calculated as 70MHz. The measurement results demonstrate the EM traces are around

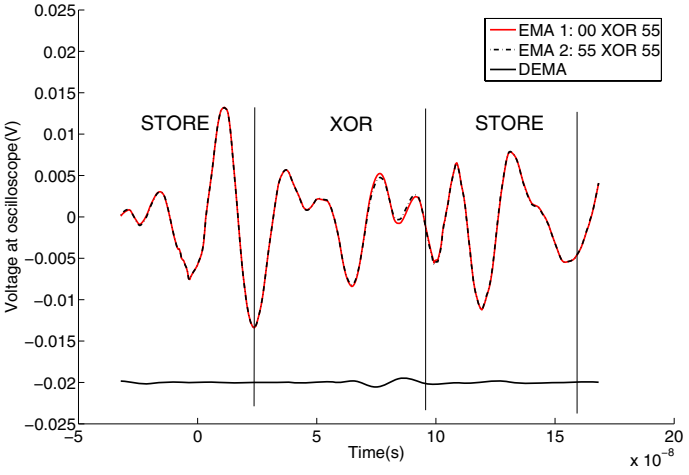


Fig. 6. EMA measurement over S-XAP processor executing `XOR` with different operands (experimental graph)

50MHz, complying to the explanation of the RL low-pass filtering effect in Section 2.2, and the parameters have been used in the EMA simulation shown in Figure 5.

Both the measurement and the simulation results observe the differential trace peaks when the processor is executing XOR logic operations. This means data dependent EM emission is leaking information related to key bits at those instances, thus means vulnerability in EMA attacks. The agreement in the measurement and the simulation results verified the validity of the proposed EMA simulation approach. The simulated EM traces in Figure 5 are lower in shape compared to those measured around the circled places, as the simulation includes no power contribution from memory accesses.

To gain a perception of the DPA attack versus the DEMA attack, Figure 7 demonstrates DPA measurement over S-XAP processor performing the same code. Although only 4 measurement runs to average out noise, data dependent power consumption can clearly identify when the processor is executing XOR logic operations. The peak to peak in the differential trace (DPA) is about 6% of the peak to peak of the original signals (Power Analysis 1 and Power Analysis 2). As a comparison, the peak to peak DEMA is about the same level of the peak to peak of the original signals (EMA 1 and EMA 2) in Figures 5 and 6, indicating the same level of information leakage in the EM side-channel and in the power channel.

3.3 EM Simulation of an Asynchronous Processor

We then perform EMA simulation on processor DR-XAP which is designed in a dual-rail asynchronous style with return-to-zero handshaking protocol. This balanced asynchronous circuitry was believed to be secure since power consumption should be data independent [8]. Figure 8 shows the EMA simulation result. On the graph we superpose the EM traces of the processor for '00 XOR 55' and '55 XOR 55', and put the DEMA

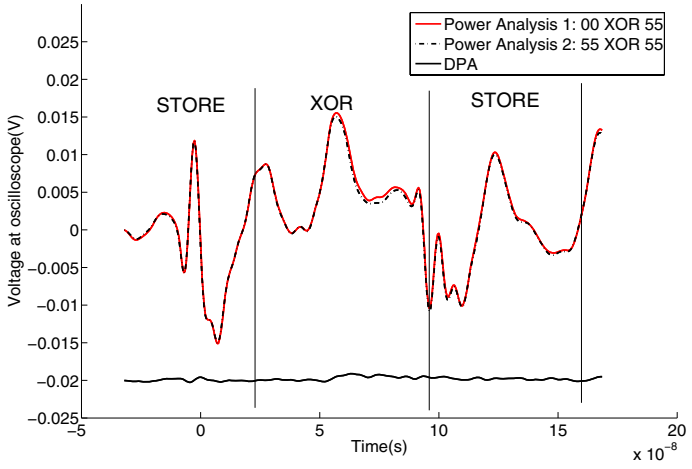


Fig. 7. DPA measurement over S-XAP processor executing `XOR` with different operands (experimental graph)

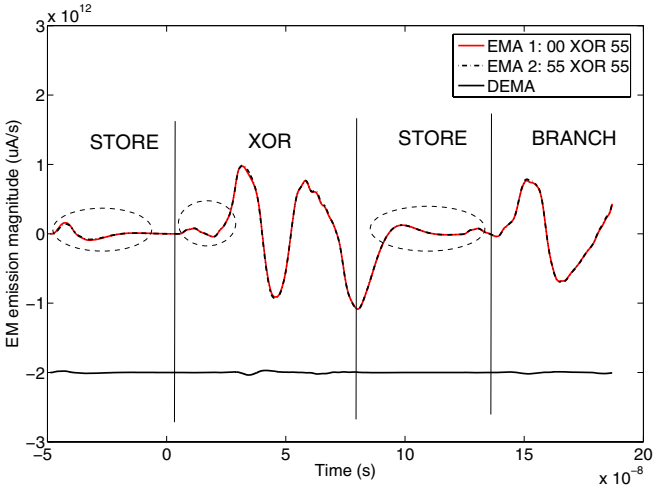


Fig. 8. EMA simulation over DR-XAP (asynchronous dual-rail) processor executing `XOR` with different operands

trace at the bottom. The DEMA trace exhibits a wobble at only about 1% magnitude of that of the original traces (EMA1 and EMA2). This matches with the projection that asynchronous design with dual-rail coding and return-to-zero handshaking is much more secure against side-channel analysis attacks.

The measurement result in Figure 9 also indicates no information leakage along the logic operation. Comparing Figure 8 and 9, we observe again lower magnitude in shape around the circled places in simulation, resulted from no memory accesses power consumption in simulation.

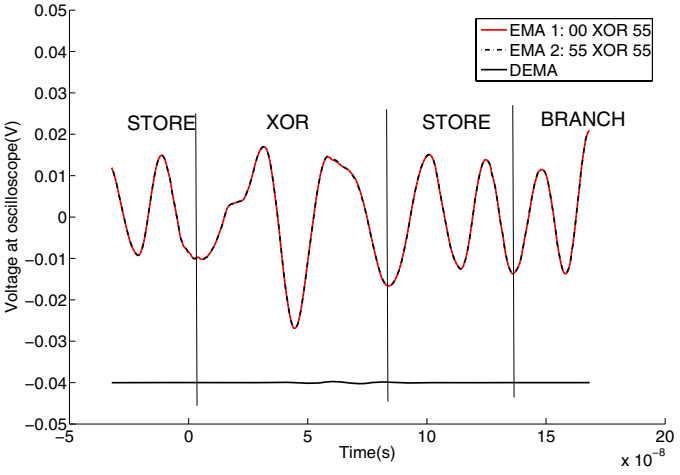


Fig. 9. EMA measurement over DR-XAP (asynchronous dual-rail) processor executing `XOR` with different operands (experimental graph)

Performing EMA simulation on *modulated emissions* on the asynchronous processor, we achieved more intriguing results. We collected the current consumption data as we did in direct emission simulation, then we processed the data with amplitude demodulation. The carrier used to demodulate the EM signal is the 17th clock harmonic (The asynchronous XAP executes at a speed around 10 to 50MHz. Here take a carrier whose fundamental is 20MHz). From the simulation results shown in Figure 10, we observed greater level of differential signals compared to Figure 8. The peak to peak of the differential trace (DEMA) is about 32% of the peak to peak of the original signals (EMA 1 and EMA 2).

The reason why the amplitude demodulated EMA reveals stronger differential signals is demonstrated in a simple example shown in Figure 11. The pulse in subplot (a) is a modulating signal. Subplot (b) shows the AM modulation with a sinusoidal carrier and its product detection based demodulation [9]. The pulse appears on the negative side of the modulation, and demodulated as a negative pulse. Subplot (c) shows the modulating signal with same magnitude and period, but time shifted a bit. Subplot (d) shows its AM modulation with the same sinusoidal carrier as in (b). The pulse appears on the positive side of the modulation, and demodulated as a positive pulse. The sign opposition in the raw traces can result in large peaks in their difference.

In a similar way, data dependent timing in the program execution caused significant peaks in the differential trace shown in Figure 10, although no obvious time shift is observed in the raw traces (AM demodulated EMA 1 and 2), because low-pass filtering has obscured the time shift. We however see higher peaks in Figure 10 around the second STORE operation, as a result of the time shift accumulated in previous operation. This data dependent timing caused EM information leakage is much higher in the tested asynchronous design than the synchronous design, as a result of the lack of clock i.e. synchronisation. The amplitude demodulated EMA simulation reveals an unexpected weakness in the tested asynchronous design against EM side-channel attacks, which

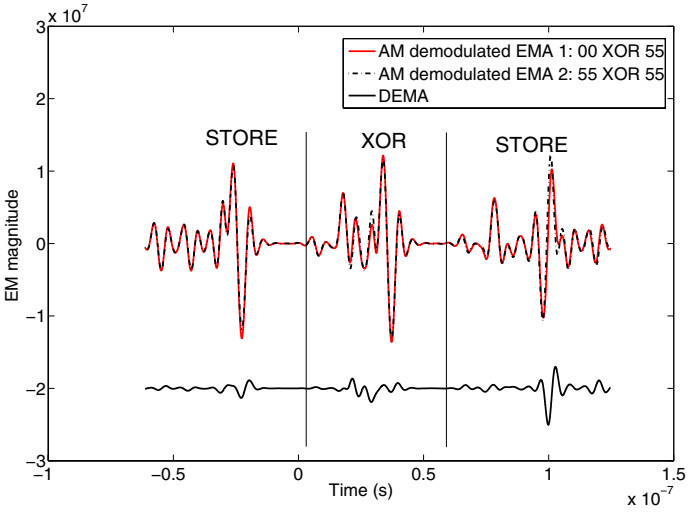
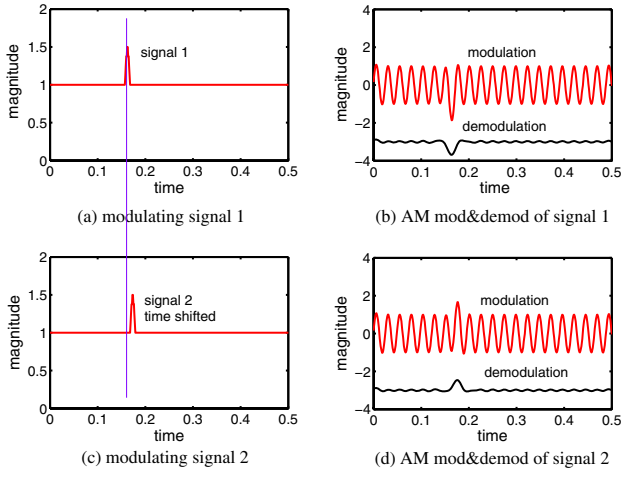


Fig. 10. EMA simulation over DR-XAP (asynchronous dual-rail) processor executing `XOR` with different operands, examining modulated emissions



(a) modulating signal 1 (b) AM mod&demod of signal 1

(c) modulating signal 2 (d) AM mod&demod of signal 2

Fig. 11. Amplitude modulation and demodulation on time shifted signals

provides a good example of usefulness of the design-time evaluation in the secure processor design flow.

4 Conclusion

A simulation methodology for EMA has been proposed on the basis of an analytical investigation of EM emissions in CMOS circuits. This simulation methodology involves simulation of current consumption with circuit simulators and extraction of IC layout

parasitics with extraction tools. Once collected, the data of current consumption is processed with MATLAB to simulate EMA.

Testing has been performed on synchronous and asynchronous processors and the results have demonstrated that DPA and DEMA of direct emissions reveal about the same level of leakage. While DEMA of amplitude demodulated emissions reveals greater leakage, suggesting better chances of success in differential EM analysis attacks. The comparison between the EMA on synchronous and asynchronous processors indicates that the synchronous processor has data dependent EM emissions, while the asynchronous processor has data dependent timing which is visible in DEMA.

The proposed simulation methodology can be easily employed in the framework of an integrated circuit design flow. To the best of our knowledge, the proposed simulation methodology for EMA is the first available assessment of EM leakage characteristics of cryptographic processors at design time. It moves one step closer to a complete security-aware design flow for cryptographic processors which aims to cover all known side-channel analysis attacks.

Acknowledgements

The authors would like to thank the Engineering and Physical Sciences Research Council (EPSRC) for funding this research project. We also thank Jacques Fournier, Scott Fairbanks, Petros Oikonomakos and Simon Hollis for their valuable comments on this paper.

References

[1] P. Kocher. Cryptanalysis of Diffe-Hellman, RSA, DSS, and other cryptosystems using timing attacks. In *Proceedings of 15th International Advances in Cryptology Conference. CRYPTO'95*, pages 171–183, 1995.

[2] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Proceedings of 19th International Advances in Cryptology Conference. CRYPTO'99*, pages 388–397, 1999.

[3] J-J. Quisquater and D. Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-smart*, pages 200–210, 2001.

[4] D. Agrawal, B. Archambeault, J. Rao, and P. Rohatgi. The EM side-channel(s). In *Proceedings of Cryptographic Hardware and Embedded Systems - CHES2002*, pages 29–45, 2002.

[5] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In *Proceedings of Cryptographic Hardware and Embedded Systems - CHES2001*, pages 251–261, 2001.

[6] M.J. Smith. *Application-Specific Integrated Circuits.* Addison-Wesley, 1997.

[7] G3Card Consortium. 3rd generation smart card project. `http://www.g3card.org/`.

[8] J. Fournier, S. Moore, H. Li, R. Mullins, and G. Taylor. Security evaluation of asynchronous circuits. In *Proceedings of Cryptographic Hardware and Embedded Systems - CHES2003*, pages 137–151, 2003.

[9] H.L. Van Trees. *Detection, Estimation, and Modulation Theory: Radar-Sonar Signal Processing and Gaussian Signals in Noise.* Krieger Publishing Co., Inc., 1992.

On Second-Order Differential Power Analysis*

Marc Joye[1,**], Pascal Paillier[2], and Berry Schoenmakers[3]

[1] CIM-PACA, Centre de Micro-électronique de Provence – George Charpak, Avenue des Anénomes, Quartier Saint Pierre, 13120 Gardanne, France
marc.joye@gemplus.com
[2] Advanced Research and Security Centre, Gemplus S.A., 34 rue Guynemer, 92447 Issy-les-Moulineaux, France
pascal.paillier@gemplus.com
[3] Dept of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Nederlands
berry@win.tue.nl

Abstract. Differential Power Analysis (DPA) is a powerful cryptanalytic technique aiming at extracting secret data from a cryptographic device by collecting power consumption traces and averaging over a series of acquisitions. In order to prevent the leakage, hardware designers and software programmers make use of masking techniques (a.k.a. data whitening methods). However, the resulting implementations may still succumb to second-order DPA. Several recent papers studied second-order DPA but, although the conclusions that are drawn are correct, the analysis is not.

This paper fills the gap by providing an exact analysis of second-order DPA as introduced by Messerges. It also considers several generalizations, including an extended analysis in the more general Hamming-distance model.

Keywords: Side-channel analysis, differential power analysis, second-order attacks.

1 Introduction

Undoubtedly, power analysis attacks constitute a cheap yet powerful cryptanalytic approach for extracting secret data from smart cards or other embedded crypto-enabled devices. Among them, Differential Power Analysis (DPA) as suggested in [8] presents the practical advantage of allowing data extraction even though the attacker has only a weak knowledge of the device being attacked. However, the original statistical technique behind DPA may require the acquisition of many power traces to average away random and computational noises. Many first-order variations of DPA, as well as other approaches such as direct

* The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.
** Seconded from Gemplus.

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 293–308, 2005.

correlation (e.g., [5]), have emerged since [8] that lead to performance improvements by lowering the number of recorded power traces.

The commonly suggested way to thwart first-order power analysis is random masking [6] a.k.a *data whitening* wherein intermediate computations are handled under a probabilistic form to defeat statistical correlation. In fact, Boolean and arithmetic maskings are certainly the most intensively used approach to protect power-sensitive cryptographic software as it appears that data randomization usually works well in practice, even when hardware countermeasures are not available. It is known, however, that masking can be defeated if the attacker knows how to correlate power consumption more than once per computation. This is known as *second-order*, or more generally *higher-order*, power analysis and was originally suggested by Messerges in [12]. These attacks are known to be more complex and delicate to carry out because they usually require the attacker to have a deeper knowledge of the device, although this might be alleviated in particular cases [16]. Investigating second-order power attacks, however, is of major importance for practitioners as it remains the only known way that is powerful enough to break real-life, DPA-protected security products.

Amazingly, second-order power analysis has remained essentially empirical so far and has never been investigated *analytically*, i.e., by the means of a direct mathematical reasoning. Second-order attacks are often described in a specific statistical setting that relies on first-order DPA one way or another, thereby leaving expected observations vague and cost estimations without a clear statement. This paper adopts a totally different approach. We formally *compute* what one expects from second-order attacking any randomized algorithm and express the amplitude of observed peaks as a function of hardware-dependent parameters. Although we essentially consider the case of Boolean masking, our results extend in several directions.

The rest of this paper is organized as follows. The next section reviews the concept of power analysis, including SPA, DPA and its higher-order generalizations. Section 3 is the core of our paper. We explain why and when second-order DPA works and carefully evaluate the height of the expected DPA peak. Next, in Section 4 we extend our main result from the Hamming-weight model to the Hamming-distance model. An experimental validation on a 1st-order protected implementation of RC6 is provided in Section 5. Finally, we conclude in Section 6.

2 Power Analysis

The power consumption of a (cryptographic) device can be monitored with an oscilloscope by inserting a resistor between the ground or Vcc pins and the actual ground. As the power consumption may vary depending on the manipulated data, some secret information may leak. This is the basic idea behind power analysis, and differential power analysis [8] in particular. In addition to power consumption, other side channels have been considered, including timing [11] and electromagnetic radiation (EM) [7,14,3].

2.1 SPA Attacks

In simple power analysis (SPA) attacks, an adversary tries to relate the power consumption to the data being handled from essentially a *single* power consumption trace (which may in turn be obtained as the average trace for a number of traces corresponding to identical data, to reduce the noise level).

To be successful in this kind of attack, however, the adversary should have (or get) the knowledge of implementation details of the system being attacked.

2.2 DPA Attacks

Differential power analysis (DPA) attacks are more powerful due to their generic nature. In this kind of attack, an adversary collects several power consumption traces for different inputs and applies statistical techniques to retrieve secret information.

As an illustration, consider the following example. Suppose that at some point, an intermediate value, say $\mathtt{I}(x, s)$, only depends on known data x and on a small portion of secret data s (i.e., small enough so that all possible values for s can be exhausted). Then for each possible value $\hat{s}$ for s, the attacker prepares two sets, $\mathfrak{S}_0(\hat{s})$ and $\mathfrak{S}_1(\hat{s})$, defined as:

$$\mathfrak{S}_b(\hat{s}) = \big\{x \mid g(\mathtt{I}(x, \hat{s})) = b\big\} \quad \text{for } b \in \{0, 1\} \tag{1}$$

where g is an appropriate *Boolean selection function* (see later). The next step consists in averaging the corresponding power consumption traces. With $\langle\cdot\rangle$ denoting the average operator and $\mathscr{C}(t)$ denoting the power consumption of the device under analysis at time period t, the adversary evaluates the *(first-order) DPA trace*

$$\Delta_1(\hat{s}, t) = \langle\mathscr{C}(t)\rangle_{x\in\mathfrak{S}_1(\hat{s})} - \langle\mathscr{C}(t)\rangle_{x\in\mathfrak{S}_0(\hat{s})} \ . \tag{2}$$

Hence, $\Delta_1(\hat{s}, t)$ is the difference of the average power consumption for sets $\mathfrak{S}_1(\hat{s})$ and $\mathfrak{S}_0(\hat{s})$, for each time period t. Assuming that (i) the intermediate data $\mathtt{I}(x, s)$ always occurs at the same time period $t = \tau$, and (ii) there are sufficiently many values for x so that $\mathtt{I}(x, \hat{s})$ is close to the uniform distribution, the DPA trace $\Delta_1(\hat{s}, t)$ exhibiting the highest peak (at time period τ) is likely the one for which $\hat{s} = s$. This way, the adversary recovers the value of secret data s.

Why does this work? Basically, the purpose of DPA is to magnify the effect of a single bit within a machine word. Suppose that a random word in a Ω-bit processor is known. Suppose further that the associated power consumption obeys the Hamming-weight model, which means that power variations are correlated to the Hamming weight (i.e., number of non-zero bits) of the manipulated data words. If the selection function $g(w)$ used to construct the sets $\mathfrak{S}_0(\hat{s})$ and $\mathfrak{S}_1(\hat{s})$ (see Eq. (1)) returns the value of a bit in the representation of word w, it follows that the words $\mathtt{I}(x, \hat{s})$ of set $\mathfrak{S}_0(\hat{s})$ have an average Hamming weight of $(\Omega - 1)/2$ whereas the words $\mathtt{I}(x, \hat{s})$ of set $\mathfrak{S}_1(\hat{s})$ have an average Hamming weight of $(\Omega + 1)/2$. As a result, the DPA trace has the property of causing a DPA peak when the selection bit, $g(\mathtt{I}(x, s))$, is handled.

2.3 Higher-Order DPA Attacks

k^{th}-order DPA attacks generalize (first-order) DPA attacks by considering simultaneously k samples — within the same power consumption trace — that correspond to k different intermediate values.

The main application of higher-order DPA attacks is to attack systems protected against first-order DPA [12]. A method commonly used to thwart (first-order) DPA attacks is the so-called *data whitening method*. Each intermediate sensitive data is XOR-ed with a random value, unknown to the adversary. Back to our illustration of Section 2.2, this means that the value of $w = \mathtt{I}(x, s)$ is XOR-ed with a random value r. Therefore, the adversary sees no longer a DPA peak in $\Delta_1(s, t)$ (cf. Eq. (2)) when $t = \tau$ and the attack fails.

However, if the adversary knows the time periods, τ_1 and τ_2 ($\tau_1 \neq \tau_2$), when the values of r and of $w \oplus r$ are manipulated, respectively, then she can evaluate

$$\overline{\Delta_2}(\hat{s}) = \left\langle |\mathscr{C}(\tau_2) - \mathscr{C}(\tau_1)| \right\rangle_{x \in \mathfrak{S}_1(\hat{s})} - \left\langle |\mathscr{C}(\tau_2) - \mathscr{C}(\tau_1)| \right\rangle_{x \in \mathfrak{S}_0(\hat{s})} \ . \tag{3}$$

The value $\hat{s}$ for which $\overline{\Delta_2}(\hat{s})$ is maximal (in absolute value) is likely $\hat{s} = s$ and again the adversary recovers the value of secret data s [12].

In case the adversary only knows the offset $\delta = \tau_2 - \tau_1$ (but not τ_1 nor τ_2), the previous attack can be extended as follows (cf. "known-offset 2DPA" of [16]). The adversary evaluates the *second-order DPA trace*

$$\Delta_2(\hat{s}, t) = \left\langle |\mathscr{C}(t + \delta) - \mathscr{C}(t)| \right\rangle_{x \in \mathfrak{S}_1(\hat{s})} - \left\langle |\mathscr{C}(t + \delta) - \mathscr{C}(t)| \right\rangle_{x \in \mathfrak{S}_0(\hat{s})} \ .$$

Again, under certain assumptions, the second-order DPA trace exhibiting the highest DPA peak will likely uncover the value of s.

3 Evaluating Second-Order DPA Peaks

3.1 Basic Result

Let $\mathsf{H}(x)$ denote the Hamming weight of x. A simple model for power leakage is the *(generalized) Hamming-weight model*. This model assumes that the (instantaneous) power consumption $\mathscr{C}$ is linearly related to Hamming weight:

$$\mathscr{C}(t) = \epsilon\, \mathsf{H}(w) + \ell\,, \tag{4}$$

for some hardware-dependent constants ϵ and ℓ, and where w is the n-bit word manipulated at time period t.

Define, for $n \geq 1$,[1]

$$E_n = 2^{-2n} \sum_{w,r \in \{0,1\}^n} |\mathsf{H}(w \oplus r) - \mathsf{H}(r)| \ .$$

[1] Note that the expression of E_n simplifies to $E_n = 2^{-2n} \sum_{w,r \in \{0,1\}^n} |\mathsf{H}(w) - \mathsf{H}(r)|$.

Then, with the notations of Section 2.3, we get

$$\begin{aligned}\left\langle |\mathscr{C}(\tau_2) - \mathscr{C}(\tau_1)| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=0}} &= |\epsilon| \left\langle |\mathsf{H}(w\oplus r) - \mathsf{H}(r)| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=0}} \\ &= |\epsilon| \left\langle |\mathsf{H}(w\oplus r) - \mathsf{H}(r)| \right\rangle_{w,r\in\{0,1\}^{n-1}} \\ &= |\epsilon|\, E_{n-1} \ . \end{aligned} \tag{5}$$

Moreover, since

$$\begin{aligned} E_n &= \frac{1}{2} \left\langle |\mathsf{H}(w\oplus r) - \mathsf{H}(r)| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=0}} + \frac{1}{2} \left\langle |\mathsf{H}(w\oplus r) - \mathsf{H}(r)| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=1}} \\ &= \frac{1}{2}\, E_{n-1} + \frac{1}{2} \left\langle |\mathsf{H}(w\oplus r) - \mathsf{H}(r)| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=1}} , \end{aligned}$$

we also get

$$\left\langle |\mathscr{C}(\tau_2) - \mathscr{C}(\tau_1)| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=1}} = |\epsilon| \,(2E_n - E_{n-1}) \ . \tag{6}$$

Subtracting Eqs (6) and (5), we obtain

$$\begin{aligned} \overline{\mathfrak{D}_2} &:= \left\langle |\mathscr{C}(\tau_2) - \mathscr{C}(\tau_1)| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=1}} - \left\langle |\mathscr{C}(\tau_2) - \mathscr{C}(\tau_1)| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=0}} \\ &= 2|\epsilon| \,(E_n - E_{n-1}) \ . \end{aligned} \tag{7}$$

It turns out that there is a nice closed formula for E_n:

Proposition 1. *For any integer $n \geq 1$, we have*

$$E_n = 2^{-2n}\, n \binom{2n}{n} \ . \tag{8}$$

Proof. From $\sum_{t=0}^{2n} \binom{2n}{t} = 2^{2n}$ and since $\sum_{t=0}^{n} \binom{2n}{t} = \sum_{t=n}^{2n} \binom{2n}{t}$, we get

$$\sum_{t=0}^{n} \tbinom{2n}{t} = \tfrac{1}{2}\big(2^{2n} + \tbinom{2n}{n}\big) = 2^{2n-1} + \tbinom{2n-1}{n} ,$$

and similarly, $\sum_{t=0}^{n-1} \binom{2n-1}{t} = \frac{1}{2} \sum_{t=0}^{2n-1} \binom{2n-1}{t} = 2^{2n-2}$. Hence, by Lemma 1 (on page 298), we have

$$\begin{aligned} 2^{2n} E_n &= \sum_{-n\leq t\leq n} |t| \tbinom{2n}{n-t} = 2 \sum_{t=0}^{n} (n-t) \tbinom{2n}{t} \\ &= 2 \sum_{t=0}^{n-1} n \tbinom{2n}{t} - 2 \sum_{t=0}^{n-1} t \tbinom{2n}{t} = 2n \sum_{t=0}^{n-1} \tbinom{2n}{t} - 4n \sum_{t=0}^{n-2} \tbinom{2n-1}{t} \\ &= 2n \left(\big[2^{2n-1} + \tbinom{2n-1}{n} - \tbinom{2n}{n}\big] - 2\big[2^{2n-2} - \tbinom{2n-1}{n-1}\big] \right) \\ &= 2n \tbinom{2n-1}{n} = n \tbinom{2n}{n} \ . \end{aligned}$$

□

Therefore $E_n - E_{n-1} = 2^{-2n+1}\binom{2n-2}{n-1}$, and we find the *exact* value of $\overline{\mathfrak{D}_2}$ and its asymptotic behavior, using equation (7):

$$\overline{\mathfrak{D}_2} = |\epsilon|\, 2^{-2n+2}\binom{2n-2}{n-1} \approx \frac{|\epsilon|}{\sqrt{\pi n}} \tag{9}$$

as asymptotic value for $\overline{\mathfrak{D}_2}$.[2] If there are sufficiently many acquisitions, it also represents an asymptotic value for $\overline{\Delta_2}(\hat{s})$ (cf. Eq. (3)). This approximation for $\overline{\mathfrak{D}_2}$ is already close for small values of n. It follows, for example, that one may expect peaks of size $\approx \frac{|\epsilon|}{\sqrt{\pi 16}} = 0.141\,|\epsilon|$, for $n = 16$ (this has to be compared with the exact value of $0.144\,|\epsilon|$, see Table 1).

3.2 Optimizing Peak Values

The higher-order DPA attacks, as described in Section 2.3, use the absolute difference between the power consumption at different time periods as the basic quantity for the analysis. This quantity fits well with the Hamming-weight model and we have shown how to determine the expected peak value in an exact way. A natural question is which other quantities can be used, and in particular, which quantities give rise to higher peak values.

In this section we analyze the peak values obtained for the following generalization of Eq. (7):

$$\overline{\mathfrak{D}_2}^{(\alpha)} = \left\langle |\mathscr{C}(\tau_2) - \mathscr{C}(\tau_1)|^\alpha \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=1}} - \left\langle |\mathscr{C}(\tau_2) - \mathscr{C}(\tau_1)|^\alpha \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=0}}$$

for arbitrary α. Extending our basic result for this case yields

$$\overline{\mathfrak{D}_2}^{(\alpha)} = 2|\epsilon|^\alpha \left(E_n^{(\alpha)} - E_{n-1}^{(\alpha)}\right), \tag{10}$$

where $E_n^{(\alpha)}$ is defined as:

$$E_n^{(\alpha)} = 2^{-2n} \sum_{w,r\in\{0,1\}^n} |\mathsf{H}(w \oplus r) - \mathsf{H}(r)|^\alpha \ . \tag{11}$$

Next we show how to find closed formulas for $E_n^{(\alpha)}$ for various values of α, from which we may then determine the corresponding expected peak values.

Lemma 1. *For any integer $n \geq 1$,*

$$\sum_{w,r\in\{0,1\}^n} |\mathsf{H}(w \oplus r) - \mathsf{H}(r)|^\alpha = \sum_{-n\leq t\leq n} |t|^\alpha \binom{2n}{n-t} \ .$$

[2] This expression corrects the analysis given in [12] and in the subsequent papers.

Proof.

$$\sum_{w,r\in\{0,1\}^n} |\mathsf{H}(w\oplus r)-\mathsf{H}(r)|^\alpha = \sum_h \sum_{\substack{r\in\{0,1\}^n\\ \mathsf{H}(r)=h}} \sum_i \sum_{\substack{w\in\{0,1\}^n\\ \mathsf{H}(w\oplus r)=i}} |\mathsf{H}(w\oplus r)-\mathsf{H}(r)|^\alpha$$
$$= \sum_{h=0}^{n}\binom{n}{h}\sum_{i=0}^{n}\binom{n}{i}|i-h|^\alpha = \sum_{-n\le t\le n}|t|^\alpha \sum_{i=0}^{n-t}\binom{n}{t+i}\binom{n}{i}$$
$$= \sum_{-n\le t\le n}|t|^\alpha\binom{2n}{n-t}\ . \qquad \square$$

Theorem 1. *For any integer n and $\alpha\ge 0$:*

$$E_n^{(\alpha)} = \begin{cases} 1\,, & \alpha = 0\,,\\ 2^{-2n}\,n\binom{2n}{n}\,, & \alpha = 1\,,\\ n\left(nE_n^{(\alpha-2)} - \left(n-\frac{1}{2}\right)E_{n-1}^{(\alpha-2)}\right)\,, & \alpha\ge 2\,. \end{cases}$$

Proof. By induction on α. Cases $\alpha=0$ and $\alpha=1$ have been proved already. For the case $\alpha\ge 2$, we have:

$$\begin{aligned} E_n^{(\alpha)} &= 2^{-2n}\sum_{-n\le t\le n}|t|^\alpha\tbinom{2n}{n-t} = 2^{-2n}\sum_{0\le t\le 2n}|n-t|^\alpha\tbinom{2n}{t}\\ &= 2^{-2n}\sum_{0\le t\le 2n}|n-t|^{\alpha-2}\,(n-t)^2\tbinom{2n}{t}\\ &= n^2\,2^{-2n}\sum_{0\le t\le 2n}|n-t|^{\alpha-2}\tbinom{2n}{t} - 2^{-2n}\sum_{0\le t\le 2n}|n-t|^{\alpha-2}(2n-t)t\tbinom{2n}{t}\\ &= n^2E_n^{(\alpha-2)} - 2^{-2n}\sum_{1\le t\le 2n-1}|n-t|^{\alpha-2}\,2n(2n-1)\tbinom{2n-2}{t-1}\\ &= n\Big(nE_n^{(\alpha-2)} - 2(2n-1)2^{-2n}\sum_{0\le u\le 2(n-1)}|n-1-u|^{\alpha-2}\tbinom{2(n-1)}{u}\Big)\\ &= n\big(nE_n^{(\alpha-2)} - (n-\tfrac{1}{2})E_{n-1}^{(\alpha-2)}\big)\ . \end{aligned}$$ $\square$

Alternatively, we have the following equivalent formulation:

Theorem 2. *For any integer n and $\beta\ge 0$:*

$$E_n^{(2\beta)} = \mathscr{P}_\beta(n) \quad \textit{and} \quad E_n^{(2\beta+1)} = \mathscr{Q}_\beta(n)\,2^{-2n}n\binom{2n}{n}$$

where

$$\mathscr{P}_\beta(n) = \begin{cases} 1\,, & \beta = 0\,,\\ n\big(n\mathscr{P}_{\beta-1}(n) - (n-\frac{1}{2})\mathscr{P}_{\beta-1}(n-1)\big)\,, & \beta\ge 1\,, \end{cases} \tag{12}$$

and

$$\mathscr{Q}_\beta(n) = \begin{cases} 1\,, & \beta = 0\,,\\ n\big(n\mathscr{Q}_{\beta-1}(n) - (n-1)\mathscr{Q}_{\beta-1}(n-1)\big)\,, & \beta\ge 1\,. \end{cases} \tag{13}$$

$\square$

Polynomials $\mathscr{Q}_\beta(n)$ (resp. $\mathscr{P}_\beta(n)$) are known as the Gandhi polynomials (resp. 'companion' Gandhi polynomials) — except that the Gandhi polynomials do not have alternating signs for the coefficients (but this difference is not essential). See [1,2] and the references therein.

For illustration, we list below the so-obtained expression for $E_n^{(\alpha)}$ for the first few values of α.

Proposition 2. *We have:*

$$\begin{aligned}
E_n^{(0)} &= \mathscr{P}_0(n) && = 1\\
E_n^{(1)} &= \mathscr{Q}_0(n)\,2^{-2n}\,n\tbinom{2n}{n} && = 2^{-2n}\,n\tbinom{2n}{n}\\
E_n^{(2)} &= \mathscr{P}_1(n) && = n/2\\
E_n^{(3)} &= \mathscr{Q}_1(n)\,2^{-2n}\,n\tbinom{2n}{n} && = 2^{-2n}\,n^2\tbinom{2n}{n}\\
E_n^{(4)} &= \mathscr{P}_2(n) && = n(3n-1)/4\\
E_n^{(5)} &= \mathscr{Q}_2(n)\,2^{-2n}\,n\tbinom{2n}{n} && = 2^{-2n}\,n^2(2n-1)\tbinom{2n}{n}\\
E_n^{(6)} &= \mathscr{P}_3(n) && = n(15n^2-15n+4)/8\\
E_n^{(7)} &= \mathscr{Q}_3(n)\,2^{-2n}\,n\tbinom{2n}{n} && = 2^{-2n}\,n^2(6n^2-8n+3)\tbinom{2n}{n}\\
E_n^{(8)} &= \mathscr{P}_4(n) && = n(105n^3-210n^2+147n-34)/16\\
E_n^{(9)} &= \mathscr{Q}_4(n)\,2^{-2n}\,n\tbinom{2n}{n} && = 2^{-2n}\,n^2(24n^3-60n^2+54n-17)\tbinom{2n}{n}\\
E_n^{(10)} &= \mathscr{P}_5(n) && = n(945n^4-3150n^3+4095n^2-2370n+496)/32\,.
\end{aligned}$$

□

As a result, we find the following peak values:

$$\begin{aligned}
\overline{\mathfrak{D}_2}^{(\alpha)} &\approx |\epsilon|\,\tfrac{1}{\sqrt{\pi}}\,n^{-1/2}, && \text{for } \alpha = 1\\
\overline{\mathfrak{D}_2}^{(\alpha)} &= |\epsilon|^2, && \text{for } \alpha = 2\\
\overline{\mathfrak{D}_2}^{(\alpha)} &\approx |\epsilon|^3\,\tfrac{1}{\sqrt{\pi}}\,n^{1/2}, && \text{for } \alpha = 3\\
\overline{\mathfrak{D}_2}^{(\alpha)} &\approx |\epsilon|^4\,\tfrac{3}{\sqrt{\pi}}\,n\,, && \text{for } \alpha = 4\,.
\end{aligned}$$

So the pattern that emerges is:

$$\overline{\mathfrak{D}_2}^{(\alpha)} \approx |\epsilon|^\alpha\, c_\alpha\, n^{(\alpha-2)/2}\,, \tag{14}$$

where c_α denotes a constant depending on α. Thus depending on the values of $|\epsilon|$ and n it may determined for which α the largest peak value is reached.

In Table 1, we tabulate the expected height, $\overline{\mathfrak{D}_2}^{(\alpha)}$, of the second-order DPA peaks for various values of n and α. We see for $\alpha = 1$ that larger values for n

Table 1. Exact values of $\overline{\mathfrak{D}_2}^{(\alpha)}$ for some common sizes of n

n	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$	$\alpha = 4$
8	$0.209\,\lvert\epsilon\rvert$	$\lvert\epsilon\rvert^2$	$4.61\,\lvert\epsilon\rvert^3$	$22\,\lvert\epsilon\rvert^4$
16	$0.144\,\lvert\epsilon\rvert$	$\lvert\epsilon\rvert^2$	$6.65\,\lvert\epsilon\rvert^3$	$46\,\lvert\epsilon\rvert^4$
32	$0.101\,\lvert\epsilon\rvert$	$\lvert\epsilon\rvert^2$	$9.49\,\lvert\epsilon\rvert^3$	$94\,\lvert\epsilon\rvert^4$
64	$0.071\,\lvert\epsilon\rvert$	$\lvert\epsilon\rvert^2$	$13.48\,\lvert\epsilon\rvert^3$	$190\,\lvert\epsilon\rvert^4$
160	$0.045\,\lvert\epsilon\rvert$	$\lvert\epsilon\rvert^2$	$21.37\,\lvert\epsilon\rvert^3$	$478\,\lvert\epsilon\rvert^4$
256	$0.035\,\lvert\epsilon\rvert$	$\lvert\epsilon\rvert^2$	$27.05\,\lvert\epsilon\rvert^3$	$766\,\lvert\epsilon\rvert^4$
512	$0.025\,\lvert\epsilon\rvert$	$\lvert\epsilon\rvert^2$	$38.28\,\lvert\epsilon\rvert^3$	$1534\,\lvert\epsilon\rvert^4$
1024	$0.018\,\lvert\epsilon\rvert$	$\lvert\epsilon\rvert^2$	$54.15\,\lvert\epsilon\rvert^3$	$3070\,\lvert\epsilon\rvert^4$

(i.e., the bit-length manipulated by the processor) yields lower (second-order) DPA peaks. More surprisingly, for $\alpha = 2$, the height of the DPA peaks does not depend on n, and for larger values of α (two last columns in Table 1) the height increases with the value of n.

The results listed in Table 1 are useful for practical purposes assuming that sufficiently many acquisitions are available. To see how fast $\overline{\mathfrak{D}_2}^{(\alpha)}$ actually converges to its expected value as a function of α, we determine the relevant signal-to-noise ratio (SNR), following, e.g., [13]. In the present paper, we are concerned with so-called algorithmic noise only, so the value of SNR tells us how many traces we need (for different values of input data x) to get a successful DPA attack.

Consider the following random variable D:

$$D := D(\hat{s}) = (2g(\hat{w}) - 1)\,|\epsilon|^\alpha\,|\mathsf{H}(w \oplus r) - \mathsf{H}(r)|^\alpha\,,$$

where r is a uniformly random n-bit string, $w = \mathtt{I}(x, s)$ with x representing the input data, and $\hat{w}$ is the outcome corresponding to the guessed secret $\hat{s}$, that is, $\hat{w} = \mathtt{I}(x, \hat{s})$. The process of evaluating quantity $\overline{\Delta_2}(\hat{s})$ (and its generalization for arbitrary α) may be viewed as sampling the random variable D.

The expected value of D is equal to $\overline{\mathfrak{D}_2}^{(\alpha)}$, and since $D^2 = |\epsilon|^{2\alpha}\,|H(w \oplus r) - H(r)|^{2\alpha}$, we obtain for the variance of D:

$$\mathrm{var}(D) = \langle D^2 \rangle - \langle D \rangle^2 = |\epsilon|^{2\alpha}\big(E_n^{(2\alpha)} - 4(E_n^{(\alpha)} - E_{n-1}^{(\alpha)})^2\big)\ .$$

Noting that the variance of D is independent of $\hat{s}$, we take as the relevant signal-to-noise ratio (for one signal):

$$\mathrm{SNR} := \frac{\langle D \rangle}{\sqrt{\mathrm{var}(D)}} = \frac{2\big(E_n^{(\alpha)} - E_{n-1}^{(\alpha)}\big)}{\sqrt{E_n^{(2\alpha)} - 4\big(E_n^{(\alpha)} - E_{n-1}^{(\alpha)}\big)^2}}\ . \tag{15}$$

From this formula for SNR, we get the following results. Firstly, SNR is independent of ϵ; this corresponds to the results found in, e.g., [13], where SNR is also independent of ϵ if there is *no* (non-algorithmic) noise. Secondly, SNR

drops off to 0 quickly as n gets larger, which was also observed in [13]. Finally, however, by evaluating SNR for fixed values of n, $n \geq 3$, searching for the optimal value of α we have found that SNR is maximized consistently at $\alpha = 3$, where SNR at $\alpha = 3$ is about 1.55 times higher than at $\alpha = 1$.

3.3 Analysis of Other Correlated Operations

So far, we have focused on the use of $\oplus$-masking, which is the basic way of implementing data whitening. Since many attacks are proposed to defeat $\oplus$-masking it is conceivable that implementors try other forms of masking, hoping to avoid such DPA attacks. For example, $w \oplus r$ may be computed using the following formula: $\overline{(w \wedge r) \vee (\bar{w} \wedge \bar{r})}$. Our calculations show that in such a case the medicine is worse than the disease, as a second-order DPA for the $\wedge$ operation yields better peaks than for $\oplus$ operation.

We illustrate this by determining $\overline{\mathfrak{D}_2}$ (cf. Eq. (7)) where τ_1 is the time period when r is manipulated and τ_2 is the time period when $w \wedge r$ is manipulated.

We have:

$$\begin{aligned}
&\left\langle |\mathscr{C}(\tau_2) - \mathscr{C}(\tau_1)| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=0}} = |\epsilon| \left\langle |\mathsf{H}(w \wedge r) - \mathsf{H}(r)| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=0}} \\
&= \frac{|\epsilon|}{2} \Big(\left\langle |\mathsf{H}(w \wedge r) - \mathsf{H}(r)| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=0, g(r)=0}} + \left\langle |\mathsf{H}(w \wedge r) - \mathsf{H}(r)| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=0, g(r)=1}} \Big) \\
&= \frac{|\epsilon|}{2} \Big(\left\langle |\mathsf{H}(w \wedge r) - \mathsf{H}(r)| \right\rangle_{w,r\in\{0,1\}^{n-1}} + \left\langle |\mathsf{H}(w \wedge r) - \mathsf{H}(r) - 1| \right\rangle_{w,r\in\{0,1\}^{n-1}} \Big) \ .
\end{aligned}$$

and

$$\begin{aligned}
\left\langle |\mathscr{C}(\tau_1) - \mathscr{C}(\tau_2)| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=1}} &= |\epsilon| \left\langle |\mathsf{H}(w \wedge r) - \mathsf{H}(r)| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=1}} \\
&= |\epsilon| \left\langle |\mathsf{H}(w \wedge r) - \mathsf{H}(r)| \right\rangle_{w,r\in\{0,1\}^{n-1}} \ .
\end{aligned}$$

Therefore, we get

$$\overline{\mathfrak{D}_2} = \frac{|\epsilon|}{2} \Big(\left\langle |\mathsf{H}(w \wedge r) - \mathsf{H}(r)| \right\rangle_{w,r\in\{0,1\}^{n-1}} - \left\langle |\mathsf{H}(w \wedge r) - \mathsf{H}(r) - 1| \right\rangle_{w,r\in\{0,1\}^{n-1}} \Big) \ .$$

To find the peak values we use the following lemma.

Lemma 2. *For any integer $n \geq 1$,*

$$2^{-2n} \sum_{w,r\in\{0,1\}^n} |\mathsf{H}(w \wedge r) - \mathsf{H}(r)| = \frac{n}{4} \, ,$$

and

$$2^{-2n} \sum_{w,r\in\{0,1\}^n} |\mathsf{H}(w \wedge r) - \mathsf{H}(r) - 1| = \frac{n}{4} + 1 \ .$$

Proof. We only prove the first part.

$$\sum_{w,r\in\{0,1\}^n} |\mathsf{H}(w\wedge r)-\mathsf{H}(r)| = \sum_h \sum_{\substack{r\in\{0,1\}^n\\ \mathsf{H}(r)=h}} \sum_i \sum_{\substack{w\in\{0,1\}^n\\ \mathsf{H}(w\wedge r)=i}} |\mathsf{H}(w\wedge r)-\mathsf{H}(r)|$$

$$= \sum_{h=0}^{n} \tbinom{n}{h} 2^{n-h} \sum_{i=0}^{h} \tbinom{h}{i} |i-h| = \sum_{h=0}^{n} \tbinom{n}{h} 2^{n-h} \sum_{i=0}^{h} \tbinom{h}{i} (h-i)$$

$$= \sum_{h=0}^{n} \tbinom{n}{h} 2^{n-h}\, h \sum_{i=0}^{h} \left[\tbinom{h}{i} - \tbinom{h-1}{i-1}\right] = \sum_{h=0}^{n} \tbinom{n}{h} 2^{n-h}\, h \sum_{i=0}^{h} \tbinom{h-1}{i}$$

$$= \sum_{h=0}^{n} \tbinom{n}{h} 2^{n-h} h\, 2^{h-1} = 2^{n-1} \sum_{h=0}^{n} \tbinom{n}{h} h = n\, 2^{2n-2} \ . \qquad \square$$

Hence, the resulting expected peak value becomes

$$\overline{\mathfrak{D}_2} = -|\epsilon|/2 \ . \tag{16}$$

This results in a higher peak value compared to the peak for $\oplus$-masking (cf. Eq. (9)).

Other binary operations such as logical or can be handled as well using basic properties such as $\mathsf{H}(w \vee r) = \mathsf{H}(w) + \mathsf{H}(r) - \mathsf{H}(w \wedge r)$.

4 Extension to the Hamming Distance Model

The basic assumption for the Hamming-weight model is that the power consumption for an operation on some data word w is linearly related to $\mathsf{H}(w)$. In terms of electronics this would mean, however, that the hardware state just prior to the moment that a data word is handled is (re)set to all-zero (or, all-one). This is the case only for a few types of electronic circuits, e.g., those that use precharged logic.

In many cases it is necessary to assume that the actual power consumption is linearly related to the Hamming *distance* to an (unknown) hardware state that existed just prior the moment when data word w is handled. For instance, each time an instruction `I` within a fixed program is executed involving w, what actually happens is that the CPU must fetch the opcode from memory (by sending the program counter over the bus and receiving the opcode in return, and decoding it). The values of the program counter and the opcode are fixed (for `I`) and therefore the power consumption incurred by transferring these values over the bus is the same each time `I` is executed. Subsequently, when w is handled, the energy required for transferring w over the bus will be related to the number of bits that need to *switched*. See [5] for more details.

So, we assume that there exist two reference values R_1 and R_2, that represent the states prior to the time periods τ_1 and τ_2, respectively. The power consumption at these time periods is therefore related to $\mathsf{H}(R_1 \oplus r)$ and $\mathsf{H}(R_2 \oplus w \oplus r)$, respectively.

We now show that the same DPA trace as before will enable us to recover the secret value, without using *any* knowledge about the values of R_1 and R_2.

Let R_1' denote R_1 but with the bit indicated by g omitted, and similarly for R_2. Let also $\delta = g(R_2) - g(R_1)$. Then, noting that for any fixed R_1 and R_2, and for any δ:

$$\begin{aligned}\left\langle \left|\mathsf{H}(R_2 \oplus w \oplus r) - \mathsf{H}(R_1 \oplus r) + \delta\right| \right\rangle &= \left\langle \left|\mathsf{H}(w \oplus r) - \mathsf{H}(r) + \delta\right| \right. \\ &= \left\langle \left|\mathsf{H}(w) - \mathsf{H}(r) + \delta\right| \right. ,\end{aligned}$$

it follows that

$$\begin{aligned}\left\langle \left|\mathscr{C}(\tau_2) - \mathscr{C}(\tau_1)\right| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=0}} &= |\epsilon| \left\langle \left|\mathsf{H}(R_2 \oplus w \oplus r) - \mathsf{H}(R_1 \oplus r)\right| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=0}} \\ &= \frac{|\epsilon|}{2}\Big(\left\langle \left|\mathsf{H}(R_2 \oplus w \oplus r) - \mathsf{H}(R_1 \oplus r)\right| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=0, g(r)=0}} + \\ &\qquad \left\langle \left|\mathsf{H}(R_2 \oplus w \oplus r) - \mathsf{H}(R_1 \oplus r)\right| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=0, g(r)=1}} \Big) \\ &= \frac{|\epsilon|}{2}\Big(\left\langle \left|\mathsf{H}(R_2' \oplus w \oplus r) - \mathsf{H}(R_1' \oplus r) + \delta\right| \right\rangle_{w,r\in\{0,1\}^{n-1}} + \\ &\qquad \left\langle \left|\mathsf{H}(R_2' \oplus w \oplus r) - \mathsf{H}(R_1' \oplus r) - \delta\right| \right\rangle_{w,r\in\{0,1\}^{n-1}} \Big) \\ &= |\epsilon| \left\langle \left|\mathsf{H}(w) - \mathsf{H}(r) + |\delta|\right| \right\rangle_{w,r\in\{0,1\}^{n-1}} .\end{aligned}$$

Similarly, letting $\delta' = 1 - g(R_2) - g(R_1)$, we have

$$\begin{aligned}\left\langle \left|\mathscr{C}(\tau_2) - \mathscr{C}(\tau_1)\right| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=1}} &= |\epsilon| \left\langle \left|\mathsf{H}(R_2 \oplus w \oplus r) - \mathsf{H}(R_1 \oplus r)\right| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=1}} \\ &= \frac{|\epsilon|}{2}\Big(\left\langle \left|\mathsf{H}(R_2 \oplus w \oplus r) - \mathsf{H}(R_1 \oplus r)\right| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=1, g(r)=0}} + \\ &\qquad \left\langle \left|\mathsf{H}(R_2 \oplus w \oplus r) - \mathsf{H}(R_1 \oplus r)\right| \right\rangle_{\substack{w,r\in\{0,1\}^n \\ g(w)=1, g(r)=1}} \Big) \\ &= \frac{|\epsilon|}{2}\Big(\left\langle \left|\mathsf{H}(R_2' \oplus w \oplus r) - \mathsf{H}(R_1' \oplus r) + \delta'\right| \right\rangle_{w,r\in\{0,1\}^{n-1}} + \\ &\qquad \left\langle \left|\mathsf{H}(R_2' \oplus w \oplus r) - \mathsf{H}(R_1' \oplus r) - \delta'\right| \right\rangle_{w,r\in\{0,1\}^{n-1}} \Big) \\ &= |\epsilon| \left\langle \left|\mathsf{H}(w) - \mathsf{H}(r) + |\delta'|\right| \right\rangle_{w,r\in\{0,1\}^{n-1}} .\end{aligned}$$

Define, for $n \geq 1$,

$$E_{n,m} = 2^{-2n} \sum_{w,r\in\{0,1\}^n} \left|\mathsf{H}(w) - \mathsf{H}(r) + m\right| . \tag{17}$$

These sums may be evaluated using the following recurrence relation:

$$\begin{aligned} f(0,m) &= |m| \\ f(n,m) &= f(n-1,m-1) + 2f(n-1,m) + f(n-1,m+1) \end{aligned}$$

where

$$f(n,m) = \sum_{w,r \in \{0,1\}^n} |\mathsf{H}(w) - \mathsf{H}(r) + m|\,,$$

for any integers n, m, $n \geq 0$.

In particular, the first few terms are:

$$f(n,0) = n\binom{2n}{n}, \quad f(n,1) = (n+1)\binom{2n}{n}, \quad f(n,2) = \frac{n^2+5n+2}{n+1}\binom{2n}{n}\,.$$

Noting that $|\delta'| = 1 - |\delta|$, we then have:

$$\overline{\mathfrak{D}_2} = |\epsilon|(E_{n-1,1-|\delta|} - E_{n-1,|\delta|})\,, \tag{18}$$

which generalizes the previous results (compare with Eq. (7)).

If $g(R_1) = g(R_2)$ we get the same result as before. If $g(R_1) = 1 - g(R_2)$ we get the same result as before, except that the sign is inverted. So we are actually able to get 1 bit of information on the values R_1 and R_2 as well.

By varying the selection function g to target other bits, all the bits of $R_1 \oplus R_2$ can be recovered this way.

5 Experimental Results and Observations

The experimental validation of our results led us to choose the RC6-$w/r/b$ block-cipher [15], in which the word size $w = n$, the number of rounds r and the length b of the encryption key in bytes are easily customizable. Besides, RC6 contains no fixed-length substitution boxes that would have limited the testing of the range $n \in [8, 1024]$. Arithmetic operations $(+, -, \times)$ are defined modulo 2^n. The encryption software takes as input a plaintext $m \in \{0,1\}^{4n}$ stored in four n-bit input registers A, B, C, D, and an extended key $S_0, \ldots, S_{2r+3}$ where $S_i \in \{0,1\}^n$.

The protection against first-order attacks relies on a data whitening technique mixing boolean and two forms of arithmetic masking. We denote by $\langle\!\langle x \rangle\!\rangle$ the variable x randomized by a boolean mask, i.e., $\langle\!\langle x \rangle\!\rangle = (x \oplus \rho, \rho)$ for some $\rho \in \{0,1\}^n$. Similarly, we note $[\![x]\!] = (x + \rho', \rho')$ an arithmetic randomization of x modulo 2^n. We make use of Goubin's conversion technique [6] to compute

1. $B = B + S_0$ and $D = D + S_1$
2. for $i = 1$ to r do
   (a) $t = (B \times (2B+1)) \lll \log_2 n$ and $u = (D \times (2D+1)) \lll \log_2 n$
   (b) $A = ((A \oplus t) \lll u) + S_{2i}$ and $C = ((C \oplus u) \lll t) + S_{2i+1}$
   (c) $(A, B, C, D) = (B, C, D, A)$
3. $A = A + S_{2r+2}$ and $C = C + S_{2r+3}$

Fig. 1. Encryption with RC6-$n/r/b$

$[\![x]\!]$ from $\langle\!\langle x \rangle\!\rangle$ securely, and on a technique due to Coron and Tchoulkine to come back to boolean masking $[\![x]\!] \mapsto \langle\!\langle x \rangle\!\rangle$ (provably) without first-order information leakage. Conversions $[\![x]\!] \mapsto \langle\!\langle x \rangle\!\rangle$ rely on a precomputed look-up table of constants embedded into the code. We chose to re-randomize the output after each conversion.

Dealing with rotations. RC6 involves circular rotations with variable offset denoted by $x \lll u$ such that $2^n - 1 \lll u = 2^n - 1$ for any $u \in \{0,1\}^n$ and $x \lll u = x \times 2^{u \bmod n} \bmod (2^n - 1)$ for any $x \neq 2^n - 1$ and any $u \in \{0,1\}^n$. We suggest a technique to compute $\langle\!\langle x \lll u \rangle\!\rangle$ from $\langle\!\langle x \rangle\!\rangle$ and $\langle\!\langle u \rangle\!\rangle$. First, we apply Goubin's conversion to $\langle\!\langle u \rangle\!\rangle$ with respect to word size $\log_2 n$, thereby computing securely $[\![u]\!]_o = (u - \alpha \bmod n, \alpha)$ where α is randomly taken from $\{0,1\}^{\log_2 n}$. Now if $\langle\!\langle x \rangle\!\rangle = (x \oplus \rho, \rho)$, we rotate the two shares twice each by $u - \alpha \bmod n$ and then α bit positions to the left:

$$\langle\!\langle x \rangle\!\rangle, [\![u]\!]_o \mapsto \langle\!\langle x \lll u \rangle\!\rangle = \langle\!\langle ((x \oplus \rho) \lll (u - \alpha)) \lll \alpha, (\rho \lll (u - \alpha)) \lll \alpha \rangle\!\rangle .$$

Description of our 1-st-order-protected RC6. We assume that the extended key is stored as $[\![S_0]\!], \ldots, [\![S_{2r+3}]\!]$ where each $S_i \in \{0,1\}^n$ is arithmetically randomized[3]. We show how to protect all the internal steps. The computation $B = B + S_0$ in Step 1 consists in randomly masking B into $[\![B]\!]$, computing $[\![B + S_0]\!]$ from $[\![S_0]\!], [\![B]\!]$ and then converting $[\![B]\!]$ into $\langle\!\langle B \rangle\!\rangle$. The same is done for D. We also randomly mask A and C into $\langle\!\langle A \rangle\!\rangle$ and $\langle\!\langle C \rangle\!\rangle$. In Step 2a, we convert $\langle\!\langle B \rangle\!\rangle \mapsto [\![B]\!]$, securely compute $[\![B \times (2B+1)]\!]$ from $[\![B]\!] = (B_1, B_0)$ as $[\![B \times (2B+1)]\!] = (B_1 \times (2B_1 + 1) - \gamma \times B_0, B_0 \times (4B_1 + 2B_0 + \gamma + 1))$ for a random $\gamma \in \{0,1\}^n$, convert this into $\langle\!\langle B \times (2B+1) \rangle\!\rangle$ and rotate each share by $\log_2 n$ bits to get (t). We apply the same process to compute u. In Step 2b, we compute $\langle\!\langle A \rangle\!\rangle, \langle\!\langle t \rangle\!\rangle \mapsto \langle\!\langle A \oplus t \rangle\!\rangle$ and convert $\langle\!\langle u \rangle\!\rangle \mapsto [\![u]\!]_o$. Then from $\langle\!\langle A \oplus t \rangle\!\rangle, [\![u]\!]_o$ we get $\langle\!\langle (A \oplus t) \lll u \rangle\!\rangle$ which we convert into $[\![(A \oplus t) \lll u]\!]$ to add it to $[\![S_{2i}]\!]$. The result is stored as $[\![A]\!]$ which we convert into $\langle\!\langle A \rangle\!\rangle$. We do the same for register C. Lastly, Step 3 consists in converting $\langle\!\langle A \rangle\!\rangle$ into $[\![A]\!]$, adding it to $[\![S_{2r+2}]\!]$ and subtracting the two shares to get A (the same applies to C). From $\langle\!\langle B \rangle\!\rangle, \langle\!\langle D \rangle\!\rangle$, we recover B and D.

Attack strategy. Assuming that we know subkeys $S_0, \ldots, S_{2j-2}, S_{2j-1}$ for some $j \in [0, r-1]$, we recover subkeys S_{2j}, S_{2j+1} involved in the j-th round. As our technique applies to boolean masking, we focus on the last computation of Step 2b that converts $[\![A]\!] \mapsto \langle\!\langle A \rangle\!\rangle$ just after S_{2j} has been added to A. Noting $\langle\!\langle A \rangle\!\rangle = (A_1, A_0)$, we know that our conversion routine handles A_1 first and then A_0 two clock cycles later. This gives us an offset $\delta = 2$ and power traces will reveal S_{2j}. We apply the same technique to the conversion $[\![C]\!] \mapsto \langle\!\langle C \rangle\!\rangle$ in the same round to get S_{2j+1}. We mention for concreteness that similar strategies apply to Steps 1, 3 to get $S_0, S_1, S_{2r+2}, S_{2r+3}$.

[3] Each $[\![S_i]\!]$ involves its own randomness. In our implementation, $[\![S_i]\!]$ is re-randomized after use and updated in volatile memory for later calls to RC6.

Data Treatment and Observations. The execution of our code is simulated with `Mathematica 5.0` and power traces generated with $\epsilon = 1$ and $\ell = 0$. We chose $r = 8$ and $j = 5$ arbitrarily, $\alpha = 3$ to maximize the SNR, and the selection function $g(w)$ is the most significant bit of w. The value of $\langle\!\langle A \rangle\!\rangle$ when converting $[\![A]\!] \mapsto \langle\!\langle A \rangle\!\rangle$ in the 5-th round is identical to the value of $\langle\!\langle B \rangle\!\rangle$ before the conversion $\langle\!\langle B \rangle\!\rangle \mapsto [\![B]\!]$ in Step 2a of round 6. The conversion routine from boolean to arithmetic masking also admits an offset of two cycles. Therefore, power traces present two attack locations and two (potential) simultaneous peaks.

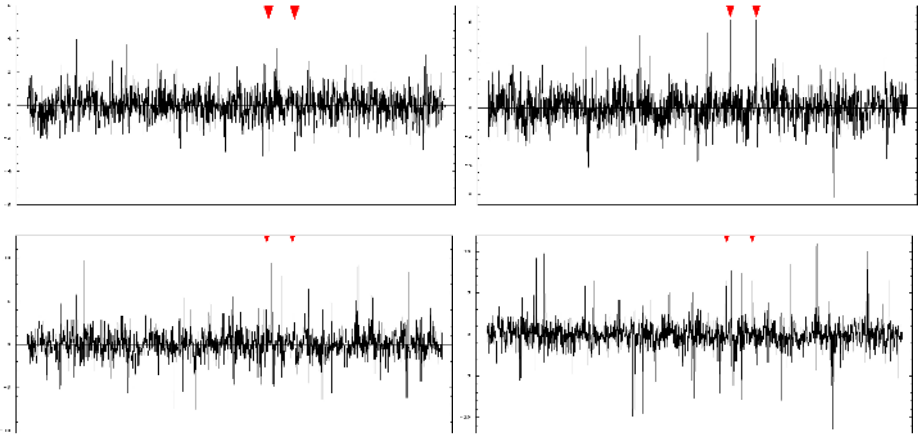


Fig. 2. Second-order DPA trace $\Delta_2(\hat{s}, t)$ with wrong guess $s \neq \hat{s}$ (left) and correct guess $s = \hat{s}$ (right). About 3000 power traces are enough to see peaks for $n = 8$ (top). More than 20000 power traces are necessary for $n = 16$ (bottom), indicating a loss of efficiency as n grows. Arrows indicate attack locations in the 5-th and 6-th rounds.

For $n = 8$, we measure peaks of height 5.52 to be compared with the theoretical value of 4.61. For $n = 16$, we observe peaks at 6.45 for an expected height of 6.65. Standard deviations are 0.82 and 1.60 respectively. This seems to confirm our theoretical results. Experiments for $n \geq 32$ show that a much larger number of power traces are needed to mount a practical attack.

6 Conclusions

We provided a formal exploration of second-order attacks by estimating the exact height of expected peaks. Our results allow to anticipate the efficiency of second-order attacks on a given hardware by providing a theoretical measurement of information leakage as a function of hardware-dependent parameters. We believe that our results constitute the theoretical basis of practical general-purpose second-order power attacks. We see many open issues for future research, in particular how to extend our results to take non-algorithmic noise into account.

References

1. ⟨`http://www.research.att.com/projects/OEIS?Anum=A036970`⟩. Triangle of coefficients of Gandhi polynomials. In *On-Line Encyclopedia of Integer Sequences*.
2. ⟨`http://www.research.att.com/projects/OEIS?Anum=A083061`⟩. Triangle of coefficients of a companion polynomial to the Gandhi polynomial. In *On-Line Encyclopedia of Integer Sequences*.
3. D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM Side-Channel(s). In *Cryptographic Hardware and Embedded Systems – CHES 2002*, vol. 2523 of *Lecture Notes in Computer Science*, pp. 29–45. Springer-Verlag, 2002.
4. G. Boros and V. Moll. *Irresistible Integrals: Symbolics, Analysis and Experiments in the Evaluation of Integrals*. Cambridge University Press, 2004.
5. É. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems – CHES 2004*, vol. 3156 of *Lecture Notes in Computer Science*, pp. 16–29. Springer-Verlag, 2004.
6. J.-S. Coron and L. Goubin. On Boolean and arithmetic masking against differential power analysis. In *Cryptographic Hardware and Embedded Systems – CHES 2000*, vol. 1965 of *Lecture Notes in Computer Science*, pp. 231–237. Springer-Verlag, 2000.
7. K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems – CHES 2001*, vol. 2162 of *Lecture Notes in Computer Science*, pp. 251–261. Springer-Verlag, 2001.
8. P.C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Advances in Cryptology – CRYPTO '99*, vol. 1666 of *Lecture Notes in Computer Science*, pp. 388–397. Springer-Verlag, 1999.
9. M. Joye. Smart-card implementations of elliptic curve cryptography and DPA-type attacks. In *Smart Card Research and Advanced Applications VI*, pp. 115–125. Kluwer Academic Publishers, 2004.
10. D.E. Knuth. *The Art of Computer Programming (Vol. 1: Fundamental Algorithms)*. Addison Wesley, 3rd edition, 1997.
11. P.C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology – CRYPTO '96*, vol. 1109 of *Lecture Notes in Computer Science*, pp. 104–113. Springer-Verlag, 1996.
12. T.S. Messerges. Using second-order power analysis to attack DPA resistant software. In *Cryptographic Hardware and Embedded Systems – CHES 2000*, vol. 1965 of *Lecture Notes in Computer Science*, pp. 238–251. Springer-Verlag, 2000.
13. T.S. Messerges, E.A. Dabbish, and R.H. Sloan. Examining smart-card security under the threat of power analysis attacks. *IEEE Transactions on Computers*, **51**(5):541 552, 2002.
14. J.-J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and couter-measures for smard cards. In *Smart Card Programming and Security (E-smart 2001)*, vol. 2140 of *Lecture Notes in Computer Science*, pp. 200–210. Springer-Verlag, 2001.
15. R.L Rivest, M.J.B. Robshaw, R. Sideney, and Y.L. Yin. The RC6 block cipher. RSA Laboratories, v1.1, August 20, 1998.
16. J. Waddle and D. Wagner. Towards efficient second-order power analysis. In *Cryptographic Hardware and Embedded Systems – CHES 2004*, vol. 3156 of *Lecture Notes in Computer Science*, pp. 1–15. Springer-Verlag, 2004.

Improved Higher-Order Side-Channel Attacks with FPGA Experiments

Eric Peeters, François-Xavier Standaert,
Nicolas Donckers, and Jean-Jacques Quisquater

UCL Crypto Group, Laboratoire de Microélectronique,
Université Catholique de Louvain, Place du Levant,
3, B-1348, Louvain-La-Neuve, Belgium
{peeters, fstandae, donckers, quisquater}@dice.ucl.ac.be

Abstract. We demonstrate that masking a block cipher implementation does not sufficiently improve its security against side-channel attacks. Under exactly the same hypotheses as in a Differential Power Analysis (DPA), we describe an improvement of the previously introduced higher-order techniques allowing us to defeat masked implementations in a low (*i.e.* practically tractable) number of measurements. The proposed technique is based on the efficient use of the statistical distributions of the power consumption in an actual design. It is confirmed both by theoretical predictions and practical experiments against FPGA devices.

Keywords: cryptographic devices, side-channel analysis, DPA, high-order power analysis, masking countermeasure, block cipher, FPGA.

1 Introduction

Since their publication in 1998 [9], power analysis attacks have attracted significant attention within the cryptographic community. Although less general than classical cryptanalysis, because they usually target one specific implementation, they have been particularly efficient to break a wide variety of devices, including smart cards, ASICs and FPGAs [12,16,20]. As a straightforward consequence, countermeasures against these attacks are of great practical interest.

In the open literature, the masking technique is among the most popular suggested ways to protect an implementation against Differential Power Analysis [1,6,7,18]. However, several works have shown that such protected devices are still sensitive to higher-order attacks, originally described in [13]. In particular, a recent advance [24] suggested that higher-order power analysis is possible, without any additional hypothesis than usually assumed for first-order attacks. They proposed a way to combine the leakages corresponding to the masked data and its mask even if their respective position within the sampled data is unknown. [21] proposed an extension of these attacks by considering a more general power consumption model. Although these papers provide indications for the practical implementation of the attack, the number of observations required to retrieve the secret key is generally large (at least significantly larger than in a

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 309–323, 2005.

first-order power analysis attack). As a consequence, masking is usually believed to improve the actual security of an implementation.

In this paper, we demonstrate that masking a block cipher implementation does not sufficiently improve its security against a side-channel opponent. Under exactly the same hypotheses as in a Differential Power Analysis, we provide strong evidence that a masked block cipher implementation can be defeated by an improved higher-order attack, using a low (*i.e.* practically tractable) number of measurements. The proposed technique is based on the efficient use of the statistical distributions of the power consumption in an actual design. Based on these distributions, we describe how to recover the secret key of a masked block cipher implementation, applying a maximum likelihood approach, as suggested in [2]. We confirm our assertions both by theoretical predictions, using the formalism of attacks introduced in [16,20], and practical experiments against real world Field Programmable Gate Array (FPGA) designs. Remark that our results focus on the extraction of information from the available power traces. For simplicity purposes, we assumed the mask and masked data to be computed in parallel and did not discuss possible synchronization issues. However, the extension to other contexts is straightforward using the techniques introduced in [24].

The rest of the paper is structured as follows. Sections 2 and 3 respectively describe the masking countermeasure and our power consumption model. The description of the improved higher-order attack is in Section 4. Simulated attacks are in Section 5 and Section 6 provides the experimental results against a masked block cipher FPGA implementation. Conclusions are in Section 7.

2 The Masking Countermeasure

The idea of masking the intermediate values inside a cryptographic algorithm is suggested in several papers as a possible countermeasure to power analysis. The technique is generally applicable if all the fundamental operations used in a given algorithm can be rewritten in the masked domain. This is easily seen to be the case in classical algorithms such as the DES [14] or AES [15]. Although these methods have been originally applied at the algorithmic level as well as at the gate level, it has been shown recently that masking at the gate level involves critical security concerns. Reference [10] notably demonstrates that the glitching activity of masked logic gates offers a previously neglected leakage source that seriously affects the security of the countermeasure. For this reason, this paper will mainly discuss the algorithmic level protection, using precomputed tables.

In the following sections, we question the security of the masking countermeasure with respect to higher-order power analysis attacks. For this purpose, we start by giving a simple description of our target implementations. An unmasked block cipher design is represented in Figure 1, where the b_is represent known input values, the k_is are the secret encryption key bits and the S blocks are non-linear substitution boxes (let N_s be the number of such S-boxes). In accordance with the structure of most present block ciphers [3,4,14,15], we do

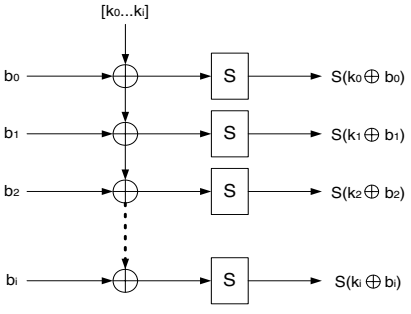


Fig. 1. Unprotected scheme

not loose in generality by focusing our attention to this combination of key additions and non-linear S-boxes. Remark that the bit-widths are not specified on the scheme.

Our protected implementation is represented in Figure 2. The masking principle is as follows. After having XORed the random mask to the initial data, both the mask and the masked data are sent through a non-linear S-box. S is the original S-box from the algorithm and S' is a precomputed table such that we have:

$$S(b \oplus k \oplus r) = S(b \oplus k) \oplus S'(r, b \oplus k \oplus r) = S(b \oplus k) \oplus q$$

As a consequence, the output values are still masked with a random mask q.

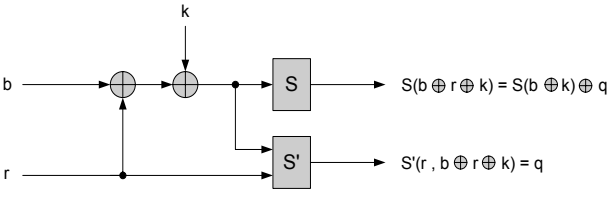


Fig. 2. Masked scheme

3 Power Consumption Model

Power analysis attacks generally target CMOS devices for which it is reasonable to assume that the main component of the power consumption is the dynamic power consumption. For a single CMOS gate, we can express it as follows [19]:

$$P_D = C_L V_{DD}^2 P_{0\to1} f \tag{1}$$

where C_L is the gate load capacitance, V_{DD} the supply voltage, $P_{0\to1}$ the probability of a $0 \to 1$ output transition and f the clock frequency. Equation (1)

specifies that the power consumption of CMOS circuits is data-dependent. As a consequence, a reasonable hypothesis for the power consumption model is: *let x and x' be two consecutive intermediate values of the running algorithm in the target device, let t be the time at which x switches into x', then the power consumption of the device at this time is proportional to the Hamming weight $W_H(x \oplus x')$.*

This hypothesis, usually denoted as the Hamming distance power consumption model, is generally true for any CMOS circuit and is specifically applicable to FPGAs. However, in certain particular contexts, more specific hypotheses hold. For example, in processors with precharged buses, the power consumption may depend on the Hamming weight of the data on the bus [5]. We note that most of our conclusions remain applicable, independently of the power consumption model and target device selected.

4 Attack Description

Let us describe the proposed technique with the single S-box scheme of Figure 3, where the inputs b, r and k are N_b-bit wide. First, we express the power

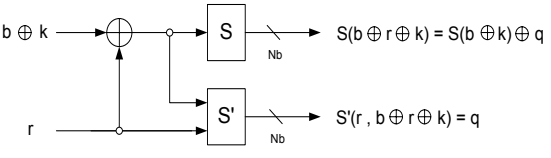


Fig. 3. Illustrative 4-bit scheme

consumption of one pair of S and S' boxes in case of a pipeline block cipher implementation and denote it as a random variable O, standing for observations. That is, we assume that the structure displayed in Figure 3 is fed with a new input at each clock cycle. As explained in the previous section, the power consumption is a function of any two consecutive values. If $b \oplus k$ switches into $b' \oplus k$ and q switches into q', we have:

$$O = WH\Big[\big(S(b \oplus k) \oplus q\big) \oplus \big(S(b' \oplus k) \oplus q'\big)\Big] + WH\Big[q \oplus q'\Big]$$

Defining the random variable $\Sigma = S(b \oplus k) \oplus S(b' \oplus k)$, where Σ stands for secret state and the random variable $R = q \oplus q'$, where R stands for random state, it is therefore possible to rewrite the observations as:

$$O(\Sigma, R) = WH\big[\Sigma \oplus R\big] + WH\big[R\big]$$

We note again that the observations could be expressed in exactly the same way in the Hamming weight power consumption model[1]. We note also that the

[1] We would find $O = WH\big[S(b \oplus k) \oplus q\big] + WH\big[q\big]$, which yields $\Sigma = S(b \oplus k)$, $R = q$.

operator used to combine the two leakage contributions is a '+' because in our analysis, the masked data and its mask are loaded on the register at the same time. But in other contexts, we may choose a '−' as in [13], or a '×' as in [21,24]. Actually, no matter what operator we use, the main point is to gather the two (or more in case of higher-order masking) statistical distributions of the power consumption so that the combined statistical distribution is key-dependent.

Indeed, while it is not possible to predict the observations, because they depend on unknown mask and key values, we can still analyze their statistical behavior. For a fixed value of the secret state $\Sigma = \sigma_i$, we can determine all the possible observations, for all the different possible random states $R = r_j$. From this analysis, it is therefore possible to derive the probability density functions $P[O = o_i|\Sigma = \sigma_i]$, for all the possible secret states.

In practice, because the observations are a sum of two Hamming weight values, they are distributed as binomials and the number of possible distributions for $P[O|\Sigma = \sigma_i]$ equals N_b+1. As a simple illustration, if $N_b = 4$, the five possible distributions of the observations are given in Figure 4.

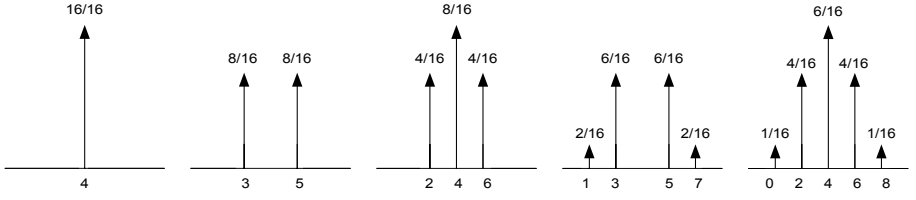


Fig. 4. Probability density functions $P[O|\Sigma = \sigma_i]$ with $N_b = 4$

The important consequence is that, knowing a secret state σ_i, we know the probability of making an observation o_i. This provides us with the tool to mount a new attack, based on a maximum likelihood approach.

Remark: The distributions $P[O|\Sigma = \sigma_i]$ all have the same mean value, $E(O|\Sigma = \sigma_i) = N_b$ and only differ in their variances. This fact allows to understand the origin of previous attacks, as the one in [24], where it is proposed to square the power consumption traces in order to obtain key-dependent measurements. The reason is that the mean of the squared power trace is a function of the mean and the variance of the initial power trace:

$$E\Big((O|\Sigma = \sigma_i)^2\Big) = E\Big((O|\Sigma = \sigma_i)\Big)^2 + V(O|\Sigma = \sigma_i)$$

It is also clear that the information contained in the expectation of the squared power trace is poor compared to what can be obtained using the complete statistical distribution of the observations.

Now, using the usual framework of side-channel attacks, we would like to find the secret key k, using a serial of observations $o_1, o_2, ..., o_n$, obtained by

feeding the encryption device with a serial of input texts $b_0, b_1, ..., b_n$ (the input transition $b_0 \rightarrow b_1$ gives rise to the observation o_1).

For this purpose, we first remark that, knowing the sequence of input texts $b_0, b_1, ..., b_n$, each key candidate $k_i \in [0, 2^{Nb} - 1]$ specifies one sequence of secret states. Therefore, we have 2^{N_b} possible chains of states denoted as:

$$\begin{aligned}
\Sigma^*(k_0) &:= \{\sigma_1(k_0), \sigma_2(k_0), ..., \sigma_n(k_0)\};\\
\Sigma^*(k_1) &:= \{\sigma_1(k_1), \sigma_2(k_1), ..., \sigma_n(k_1)\};\\
\Sigma^*(k_2) &:= \{\sigma_1(k_2), \sigma_2(k_2), ..., \sigma_n(k_2)\};\\
&\ldots
\end{aligned}$$

In practice, these state sequences cannot be observed directly, but only through the power consumption of the device, *i.e.* the sequence of observations $O^* := \{o_1, o_2, ..., o_n\}$. Then, for each possible secret state chain, we compute the probabilities $P[O^*|\Sigma^*(k_j)]$. Assuming that the observations are independent (which is reasonable since the attacker feeds the devices with random input texts), it yields:

$$\begin{aligned}
P[O^*|\Sigma^*(k_0)] = &\quad P[O = o_1|\Sigma = \sigma_1(k_0)] \times P[O = o_2|\Sigma = \sigma_2(k_0)] \times ...\\
P[O^*|\Sigma^*(k_1)] = &\quad P[O = o_1|\Sigma = \sigma_1(k_1)] \times P[O = o_2|\Sigma = \sigma_2(k_1)] \times ...\\
P[O^*|\Sigma^*(k_2)] = &\quad P[O = o_1|\Sigma = \sigma_1(k_2)] \times P[O = o_2|\Sigma = \sigma_2(k_2)] \times ...\\
&\ldots
\end{aligned}$$

The chain with the highest probability gives us the most likely key. That is, if the attack is successful, the correct key corresponds to:

$$\underset{\forall\ k_j}{argmax}\, P[O^*|\Sigma^*(k_j)]$$

We note that the proposed approach is similar to the one in [8], where it is demonstrated that Hidden Markov Models may be of great help to describe discrete time processes where a state sequence is hidden. Remark finally that, in order to keep the probabilities $P[O^*|\Sigma^*(k_j)]$ within practical boundaries (for large n's, these probabilities are smaller than the machine-ϵ), we use a step by step normalization.

5 Simulated Attacks

The previous section described a higher-order power analysis attack against a single S-box scheme, without considering any kind of noise in the measurements. However, in practice, side-channel attacks are usually affected by different kinds of noises. First, block ciphers are made of the application of several S-boxes in parallel, combined with other components such as a diffusion layer (this is typically the case of the AES Rijndael [15]). These "other components" that are not directly targeted by our attack may therefore cause additional power consumption that we denote as the "algorithmic noise". Algorithmic noise exists if these

components use different resources in the circuit, which is typically the case of parallel implementations in FPGAs. Second, real life observations are usually affected by different types of "physical noises". It includes all the possible imperfections of our model appearing during the measurement process.
In order to evaluate the efficiency of the proposed attack, this section considers attacks using "perfect measurements", without any kind of physical noise. This formalism has been introduced in [16,20] and was denoted as "attacks using simulated data". Such attacks basically use simulated measurements generated by computing the number of transitions in the targeted design. The measurements are perfect in the sense that they perfectly fit to the power consumption model. As a matter of fact, the number of measurements required to have a successful attack using simulated data lower bounds this number when real measurements are considered. Still, these simulated experiments allow us to clearly evaluate the effect of algorithmic noise and to compare our attack to a classical Correlation Power Analysis against an unprotected block cipher implementation.

Remark that, as far as noise is concerned, the probability distributions $P[O|\Sigma = \sigma_j]$ are not discrete anymore. However, the previous techniques still hold assuming that the probability density functions (or pdf's) become weighted sums of Gaussians. For example, let us assume that we target a single 4-bit S-box as in the previous section and that for a particular secret state σ_j, the pdf is represented in the left part of Figure 5. If we now consider that the target

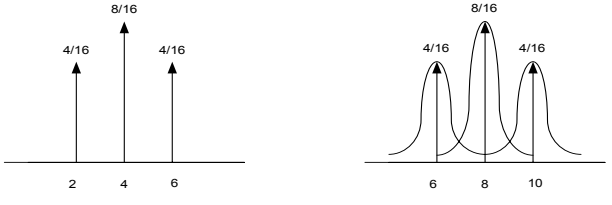


Fig. 5. Probability density functions $P[O|\Sigma = \sigma_j]$ with $N_b = 4$, without or with noise

implementation contains another masked 4-bit S-box (*i.e.* $N_s = 2$), producing algorithmic noise of mean N_b and variance $N_b/2$, we obtain the right part of the figure[2]. In general, finding the noise pdf's can simply be achieved by computing the mean and variance of the observations, as we know the signal pdf's. We now present a number of attacks using simulated data.

We define the parameters of our simulated attacks as follows. First, we use the 4-bit S-boxes of the Serpent algorithm [3] and a secret key $k = 5$. In our target implementations, we consider N_s S-boxes implemented in parallel. The number of plaintexts generated in the attacks is n and for each number of plaintexts, we observe the probabilities $P[O^*|\Sigma^*(k_i)]$, for $k_i \in [0, 15]$. The attack is considered successful when $\prod_{i=1}^{n} P[O = o_i|\Sigma = \sigma_i(k_j)]$ is maximum for $k = 5$.

[2] Note that modelling the algorithmic noise as Gaussians is reasonable since they approximate the binomial behavior of the Hamming distance values.

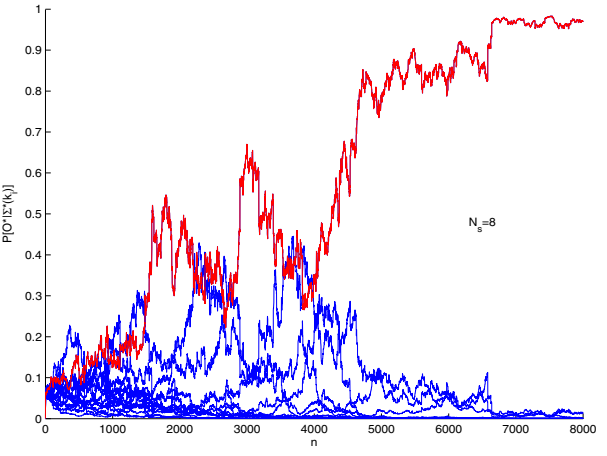


Fig. 6. A simulated higher-order attack with $N_s = 8$.

As a matter of fact, an attack against a single S-box scheme is nearly immediate: due to the discrete probabilities, a secret state such that $P[O = o_i|\Sigma = \sigma_j] = 0$ happens fast and only the correct key will remain with a non-zero probability after a few (in practice, less than 10) generated plaintexts. Much more relevant is the investigation of a simulated attack with different amounts of algorithmic noise in the design, *i.e.* different N_s values. A simulated attack with $N_s = 8$ is represented in Figure 6 and is successful after roughly 4000 generated texts. Other simulated attacks are in Appendix, Figures 8, 9, 10. From these figures, it is clear that the masked designs can be targeted by our attack with reasonable resources (*e.g.* less than 25 000 measurements), even if algorithmic noise is inserted. For comparison purposes, we also simulated first-order correlation attacks (like the ones in [16,20]) against the unprotected design of Figure 1, with the same parameters, *i.e.* same size and number of S-boxes. They are represented in Appendix, Figures 11, 12, 13, 14 and allow to measure the additional security provided by the masking. Comparisons will be discussed in the conclusions.

6 FPGA Results

We confirmed these simulated experiments with a real attack, against an FPGA implementation of the scheme in Figure 2, with $N_s = 8$ S-boxes[3]. Our target device was a Xilinx Spartan II FPGA [25] and the random mask values r_i's were generated with an on-chip LFSR.

Compared to simulated attacks, the main additional constraint was to correctly estimate the statistical characteristics (mean, variance) of the experimen-

[3] Due to area constraints, we did not target a standard algorithm such as the AES Rijndael. Indeed, as already mentioned, *e.g.* in [17,18], the hardware cost of masking a block cipher is a real concern for efficient hardware implementations.

tal signals. Indeed, in a real-world context, those values do not correspond to number of bit switches anymore, but to actual power consumption ones. That is, for example, the distance between the different gaussians in Figure 5 do not correspond to 2 bit switches anymore but to the power consumption of 2 bit switches. As a consequence, building the real pdf's $P[O|\Sigma = \sigma_j]$ from their discrete counterpart can be done with some steps and assumptions more than in Section 5.

First, we considered the measured observations to be a mixture of gaussians, as the one in the right part of Figure 5. We then assumed that the mean value of the observations E_{meas} corresponded to the mean value of the gaussian mixture.

Second, we had to determine the distance between the different gaussians, *i.e.* we had to evaluate the mean power consumption of one single bit switch in the design, denoted as m. For this purpose, we implemented a wide register inside our target FPGA and we measured the power consumption for different numbers of switching bits. Then, we derived the mean value of each gaussian in the mixture, denoted as E_i, $i \in [1, 2N_b + 1]$. It yields $E_{i+1} - E_i = m$, $\forall i \in [1, 2N_b]$.

Finally, we had to evaluate the variance of the gaussians. We assumed that all of them have the same value v, as in Section 5. Then we measured the variance of the measured observations V_{meas}, corresponding to the variance of the gaussian mixture. Knowing the different gaussian means E_i's from the previous experiments, we extracted v from [23]:

$$V_{meas} = \sum_{i=1}^{x} w_i * (v + E_i^2) - E_{meas}^2$$

where x is the total number of gaussians in the mixture (*i.e.* $2N_b + 1$), and the w_i's are weights depending on the probability of apparition of each gaussian.

We built the real pdf's for $P[O|\Sigma = \sigma_j]$ from these values and mounted a practical attack. It is represented in Figure 7, where we observe that the correct key is distinguishable after roughly 12 000 generated texts. As usually observed in side-channel attacks, practical experiments require significantly more samples than predicted because of noise and model imperfections. Still, the masked countermeasure was defeated in a remarkably low number of measurements.

In practice, we note that the attack is very sensitive to the correct evaluation of the signal mean values, for which imprecisions may lead to the selection of a wrong key candidate. The signal variance in itself does not affect the attack result, but its good evaluation allows the correct key candidate to be faster distinguished. For this purpose, we generally used a slightly larger value than the estimated v, *e.g.* $2v$ or $4v$.

Remark that the parameters m and v where naively estimated using a second programmable device. However, we also evaluated them successfully using an approach based on machine learning methods [11]. In practice, the presented attack is possible even if a second device is not available.

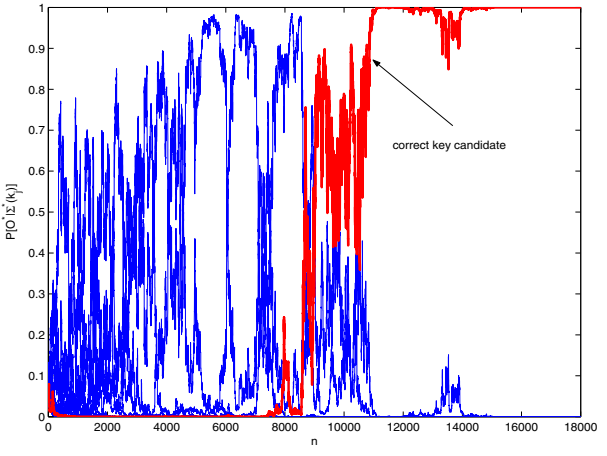


Fig. 7. A real attack against a masked FPGA design with $N_s = 8$

7 Conclusions

We proposed an improved higher-order technique to bypass the masking countermeasure. As a main result, it is demonstrated that such a countermeasure is not sufficient to protect an implementation from knowledgeable side-channel attackers. In practice, we recovered the secret key of a masked block cipher FPGA implementation in a low (*i.e.* practically tractable) number of measurements. We point out the following concluding remarks:

1. The attack was successfully applied to a parallel FPGA implementation which usually appears to be a challenging target for side-channel attacks. However, it could be straightforwardly applied to other devices, *e.g.* microprocessors. In such contexts, the algorithmic noise is usually reduced (due to the size of the buses, limited to 8 or 32 bits). For example, we estimated that an attack against an 8-bit processor would be successful after roughly 50 simulated measurements.
2. The presented attack is most fairly compared to the ones in [21,24]. For example, [21] considers a similar FPGA implementation to ours and targets a single S-box scheme (*i.e.* $N_s = 1$) in approximately 130 000 measurements. We target a $N_s = 8$ scheme in 12 000 traces.
3. Reference [13] presents experiments allowing a secret key to be recovered from a smart card implementation of the scheme in Figure 2, in about 2500 measurements. However, this attack is based on a Hamming weight power consumption model. It also requires access to the power consumption of the random mask and masked data separately, which involves these values to be computed sequentially. As the target is an 8-bit processor, it should be compared to an attack against a single S-box scheme, for which we would be successful in roughly 50 simulated measurements.

4. Compared to unprotected designs targeted by, *e.g.* a Correlation Power Analysis, a higher-order attack against a masked design still requires more traces. However, the gap between both attacks has been significantly reduced. In practice, the required number of measurements for a successful attack is not unrealistic anymore, even if large hardware implementations are considered. Note that the implementation cost of such large masked designs is another serious drawback, as mentioned in [17,18].
5. An open question is to know how much does the addition of noise affect a higher-order attack and how does it exactly compare to a first-order attack.
6. As a possible improvement, we finally suggest that a better estimation of the statistical distributions of the power consumption in the design is worth investigating. For example, the use of machine learning methods could be considered in this respect, as suggested in the previous section.

Acknowledgements. The authors would like to thank Cédric Archambeau for useful comments on previous versions of this paper.

References

1. M.L. Akkar, C. Giraud, *An Implementation of DES and AES Secure againts Some Attacks*, in the proceedings of CHES 2001, Lecture Notes in Computer Sciences, vol 2162, pp 309-318, Paris, France, May 2001, Springer-Verlag.
2. D. Agrawal, J.R. Rao, P. Rohatgi *Multi-channel Attacks*, in the proceedings of CHES 2003, Lecture Notes in Computer Sciences, vol 2779, pp 2-16, Cologne, Germany, September 2003, Springer-Verlag.
3. R. Anderson, E. Biham, L. Knudsen, *Serpent: A Flexible Block Cipher With Maximum Assurance*, in the proceedings of The First Advanced Encryption Standard Candidate Conference, Ventura, California, USA, August 1998.
4. P. Barreto, V. Rijmen, *The KHAZAD Legacy-Level Block Cipher*, Submission to NESSIE project, available from http://www.cosic.esat.kuleuven.ac.be/nessie/.
5. E. Brier, C. Clavier, F. Olivier, *Correlation Power Analysis with a Leakage Model*, in the proceedings of CHES 2004, Lecture Notes in Computer Science, vol 3156, pp 16-29, Boston, USA, August 2004.
6. S. Chari C. Jutla, J. Rao, P. Rohatgi, *Towards Sound Approaches to Counteract Power-Analysis Attacks*, in the proceedings of Crypto 1999, Lecture Notes in Computer Science, vol 1666, pp 398-412, Santa Barbara, California, USA, August 1999, Springer-Verlag.
7. L. Goubin, J. Patarin, *DES and Differential Power Analysis*, in the proceedings of CHES 1999, Lecture Notes in Computer Science, vol 1717, pp 158-172, Worcester, Massachussets, USA, August 1999, Springer-Verlag.
8. C. Karlof, D. Wagner, *Hidden Markov Model Cryptanalysis*, in the proceedings of CHES 2003, Lecture Notes in Computer Sciences, vol 2779, pp 17-30, Cologne, Germany, September 2003, Springer-Verlag.
9. P. Kocher, J. Jaffe, B. Jun, *Differential Power Analysis*, in the proceedings of CRYPTO 99, Lecture Notes in Computer Science, vol 1666, pp 398-412, Santa Barbara, USA, August 1999, Springer-Verlag.
10. S.Mangard, *Side-Channel Leakage of Masked CMOS Gates*, in the proceedings of CT-RSA 05, Lecture Notes in Computer Science, vol 3376, pp 351-365, San Fransisco, CA, USA, February 2005.

11. G. J. McLachlan and D. Peel, *Finite Mixture Models.* John Willey and Sons, New York, NY, 2000.
12. T.S. Messerges, E.A. Dabbish, R.H. Sloan, *Examining Smart-Card Security under the Threat of Power Analysis Attacks*, IEEE Transactions on Computers, vol 51, num 5, pp 541-552, May 2002.
13. T.S. Messerges, *Using Second-Order Power Analysis to Attack DPA Resistant Software*, in the proceedings of CHES 2000, Lecture Notes in Computer Sciences, vol 1965, pp 71-77, Worcester, Massachusetts, USA, August 2000, Springer-Verlag.
14. National Bureau of Standards, *FIPS PUB 46, The Data Encryption Standard*, Federal Information Processing Standard, NIST, U.S. Dept. of Commerce, Jan 1977.
15. National Bureau of Standards, *FIPS 197, Advanced Encryption Standard,* Federal Information Processing Standard, NIST, U.S. Dept. of Commerce, November 2001.
16. S.B. Ors, F. Gurkaynak, E. Oswald, B. Preneel *Power-Analysis Attack on an ASIC AES implementation*, in the proceedings of ITCC 2004, Las Vegas, April 5-7 2004.
17. E. Oswald, S. Mangard, N. Pramstaller, *Secure and Efficient Masking of AES - A Mission Impossible?*, IACR e-print archive 2004/134, http://eprint.iacr.org, 2004.
18. E. Oswald, S. Mangard, N. Pramstaller, V. Rijmen, *A Side-Channel Analysis Description of the AES S-box*, in the proceedings of FSE 2005.
19. J.M. Rabaey, *Digital Integrated Circuits*, Prentice Hall International, 1996.
20. F.-X. Standaert, S.B. Ors, B. Preneel, *Power Analysis of an FPGA Implementation of Rijndael: is Pipelining a DPA Countermeasure?*, in the proceedings of CHES 2004, Lecture Notes in Computer Science, vol 3156, pp 30-44, Boston, USA, August 2004.
21. F.-X. Standart, E. Peeters, J.-J. Quisquater, *On the Masking Countermeasure and Higher-Order Power Analysis Attacks*, in the proceedings of ITCC 2005 (vol 1), pp 562-567, Las Vegas, USA, April 2005.
22. F.-X. Standart, E. Peeters, G. Rouvroy, J.-J. Quisquater, *Power Analysis Attacks and Countermeasures of Field Programmable Gate Arrays: a Survey*, to appear in the Proceedings of the IEEE, special issue on Cryptographic Hardware and Embedded Systems, August 2005.
23. L. Trailovic, L.Y. Pao, *Variance Estimation and Ranking of Gaussian Mixture Distributions in Target Tracking Applications*, in the proceedings of the IEEE Conference on Decision and Control, pp 2195-2201, Las Vegas, NV, December 2002.
24. J. Waddle, D. Wagner, *Towards Efficient Second-Order Power Analysis*, in the proceedings of CHES 2004, Lecture Notes in Computer Science, vol 3156, pp 1-15, Boston, USA, August 2004.
25. Xilinx: *Spartan 2.5V Field Programmable Gate Arrays Data Sheet*, http://www.xilinx.com.

A Other Simulated Attacks

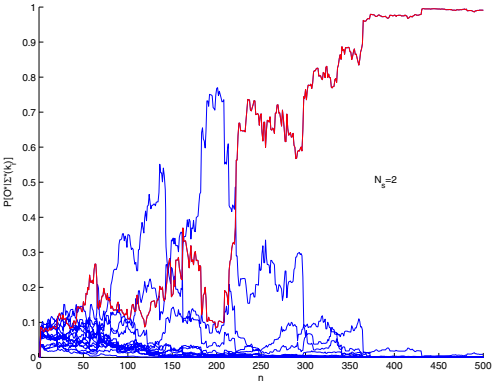


Fig. 8. A simulated higher-order attack with $N_s = 2$

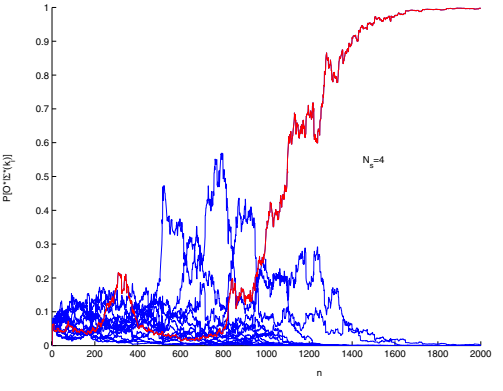


Fig. 9. A simulated higher-order attack with $N_s = 4$

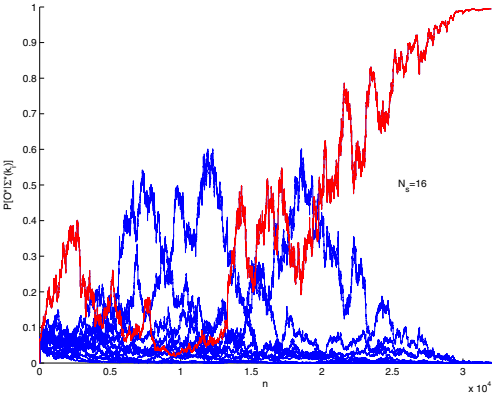


Fig. 10. A simulated higher-order attack with $N_s = 16$

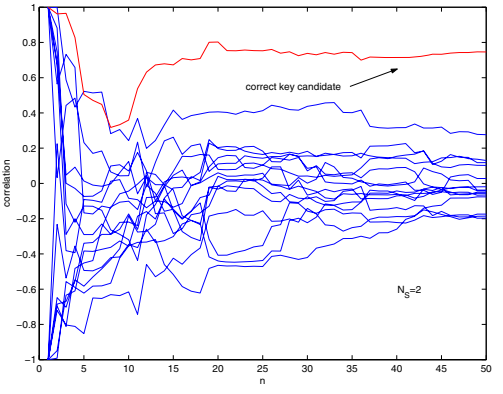


Fig. 11. A simulated correlation attack with $N_s = 2$

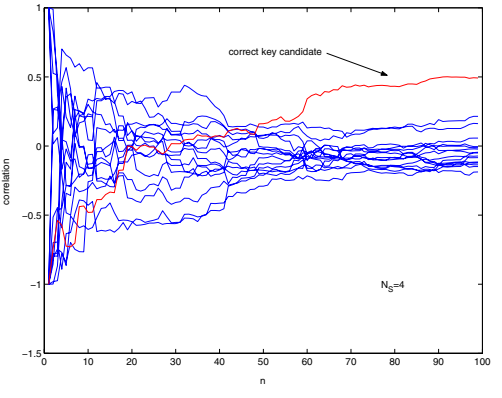


Fig. 12. A simulated correlation attack with $N_s = 4$

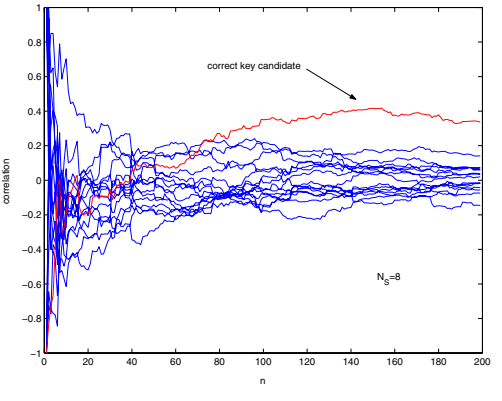


Fig. 13. A simulated correlation attack with $N_s = 8$

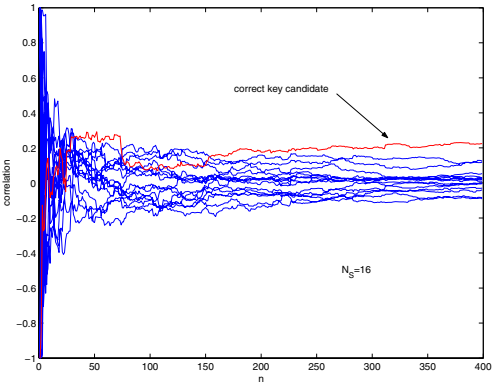


Fig. 14. A simulated correlation attack with $N_s = 16$

Secure Data Management in Trusted Computing

Ulrich Kühn[1], Klaus Kursawe[2], Stefan Lucks[3],
Ahmad-Reza Sadeghi[4], and Christian Stüble[4]

[1] Deutsche Telekom Laboratories, Technical University Berlin, Germany
ukuehn@acm.org
[2] ESAT – COSIC, KU Leuven, Belgium
klaus.kursawe@esat.kuleuven.ac.be
[3] Theoretische Informatik, University of Mannheim, Germany
lucks@th.informatik.uni-mannheim.de
[4] Horst Görtz Institute, Ruhr-University Bochum, Germany
sadeghi@crypto.rub.de
stueble@acm.org

Abstract. In this paper we identify shortcomings of the TCG specification related to the availability of sealed data during software and hardware life cycles, i.e., software update or/and hardware migration. In our view these problems are major obstacles for large-scale use of trusted computing technologies, e.g., in e-commerce, as adopters are concerned that the use of this technology might render their data inaccessible.

We propose both software and hardware solutions to resolve these problems. Our proposals could be easily integrated into the TCG specification and preserve the interests of involved parties with regard to security and availability as well as privacy.

1 Introduction

The increasing global connectivity and distributed applications both for business and personal use require IT-systems that guarantee confidentiality, authenticity, integrity, privacy, as well as availability. On the technical side, cryptographic and IT security research provide a variety of technical security measures such as encryption and strong authentication mechanisms, firewalls and so forth, to achieve these security targets. However, these measures provide only partial solutions as long as the underlying computing platforms still suffer from security problems.

These issues are addressed by a new generation of computing platforms employing both supplemental hardware and software. Concretely, these initiatives are the TCG (Trusted Computing Group), an IT-industry alliance, and Microsoft's NGSCB (Next-Generation Secure Computing Base); however, so far only the TCG has published specifications [17,16].

According to the TCG the primary goal of this architecture is to improve the security and the trustworthiness of computing platforms [4,5,12,13]. To this end, the conventional PC architecture is extended by new mechanisms to (i)

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 324–338, 2005.

protect cryptographic keys, (ii) authenticate the configuration of a platform (attestation), and (iii) cryptographically bind confidential data to a certain system configuration (sealing), i.e., the data can only be accessed (unsealed) if the corresponding system can provide the specific configuration for which the data has been sealed.

In this paper we identify shortcomings of the TCG specification related to software and hardware life cycles and propose solutions to resolve these problems. More precisely, we are concerned with the following two problem areas regarding the management of sealed data:

First, the TCG specification defines the sealing functionality in a way that an update or security patch to the trusted computing base can render sealed data inaccessible, even when keeping the same level of security. We propose possible solutions to this problem that preserve security for all involved parties, i.e. owners, users, and remote parties (e.g. content providers), in the sense of multilateral security.

Second, the TCG specification contains a (optional) protocol to partially migrate the TPM-internal data to another TPM of the same vendor (lock-in). However, to our knowledge this protocol is not implemented in any of the existing TPMs. Our proposal allows to securely migrate all data to a different platform without requiring to either tolerate loss of potentially important data or involve all remote parties while at the same time avoiding potential privacy violations.

Our proposals are based on exercising decentralized control of the update or migration procedure where a party trusted by both the system owner as well as remote parties ensures that their respective interests are served.

This paper is structured as follows: In Section 4 we give a summary of the TCG specification as far as needed for the purpose of this paper. We continue in Section 5 with a brief description of the problems regarding the handling of sealed data during software and hardware life-cycles. Section 6 describes a basic system model that we consider suitable to support our solution proposals. In Sections 7 and 8 we describe our proposals for solutions to the software update and the hardware migration problems.

2 Related Work

To our knowledge, only few articles are publicly available that discuss improvements of the TCG specification in the context of platform changes and migration.

In [15] the authors present a security measurement architecture for Linux that measures all executable content upon loading and protects the table of measurements using the TPM's functionality. Remote parties can first verify the integrity of the table of measurements using remote attestation, and can then decide if the current platform configuration is trustworthy. This is applied to remote access where a client configuration is verified before allowing it to access the network [14]. However, platform updates and platform migration is not addressed in [15,14], and, as the PCR values are employed to protect the

current list of measurements, working with sealed data seems difficult, or results in the same problems as discussed in Section 5.

Property-based attestation as proposed in [10,8] extends the remote attestation protocol such that, on a more abstract operating system layer, properties are attested instead of binary representations of platform configurations. While [10,8] focus on remote attestation, we will show in Section 7.3 that this approach is also well-suited for what we call property-based sealing.

3 Conventions

Basic Notation. A (public key) encryption scheme is denoted with the tuple $(\mathsf{Enc}(), \mathsf{Dec}())$ for encryption and decryption algorithms. The tuple (PK_X, SK_X) denotes the public and private key of a party X. Further, a digital signature scheme is denoted by a tuple $(\mathsf{Sign}(), \mathsf{Verify}())$ of signing and verification algorithms. With $\sigma \leftarrow \mathsf{Sign}_{SK_X}(m)$ we denote the signature on a message m using the signing key SK_X. The return value of the verification algorithm $ind \leftarrow \mathsf{Verify}_{PK_X}(\sigma)$ is a Boolean value $ind \in \{true, false\}$. A certificate on a quantity Q with respect to a verification key PK_X is a signature generated by applying the corresponding signing key (not something complex such as a X.509 certificate). A hash function is denoted by $\mathsf{Hash}()$.

Roles. Throughout the paper we refer to the following subjects/roles:

- *Owner*: The owner $\mathcal{O}$ of a system $\mathcal{P}$, e.g. a PC, is an entity who defines, by its security policy $\mathcal{SP}_{\mathcal{O}}$, the allowed configurations of the underlying platform, also including patches/updates. Typical examples are an enterprise represented by an administrator or an end-user owning a personal platform.
- *User*: The user $\mathcal{U}$ of a computing platform $\mathcal{P}$ is an entity interacting with $\mathcal{P}$ under the platform's security policy $\mathcal{SP}_{\mathcal{O}}$. Examples are employees using enterprise-owned hardware; user and owner might also be identical.
- *Remote party*: This refers to any party trying to assess the trustworthiness of a system, e.g. for licensing content to the system's owner. The remote party's security policy $\mathcal{SP}_{\mathcal{R}}$ defines access control rules attached to the content. An example might be a party licensing classified data to the system's owner that wants to ensure that its access policy is also enforced locally.

4 Main Aspects of the TCG Specification

In this section we briefly review the main functionalities of the trusted computing technology proposed in specifications version 1.1b [17] and 1.2 [16] of the *Trusted Computing Group* (TCG).

The main components of the TCG proposal are a hardware component *Trusted Platform Module* (TPM), a kind of (protected) pre-BIOS (Basic I/O System) called the *Core Root of Trust for Measurement* (CRTM), and a support software called *Trusted Software Stack* (TSS) which performs various functions like communicating with the rest of the platform or with other platforms.

Trusted Platform Module. A TPM is the main component of the specification providing a secure random number generator, non-volatile tamper-resistant storage, key generation algorithms, cryptographic functions like RSA encryption/decryption, and the hash function SHA-1[1]. A TPM can be abstractly described by the tuple $(EK, SRK, \mathcal{T})$: the endorsement key EK, an encryption key that uniquely identifies each TPM; the Storage Root Key SRK or Root of Trust for Storage (RTS), uniquely created inside the TPM. Its private part never leaves the TPM, and it is used to encrypt all other keys created by the TPM; the TPM state $\mathcal{T}$ contains further security-critical data shielded by the TPM (see Section 8.1).

The TPM provides a set of registers called *Platform Configuration Registers* (PCR) that can be used to store hash values. The hardware ensures that the value of a PCR register can only be modified as follows: $PCR_{i+1} \leftarrow \text{SHA1}(PCR_i|I)$, with the old register value PCR_i, the new register value PCR_{i+1}, and the input I (e.g. a SHA-1 hash value). This process is called *extending* a PCR.

There are three different types of asymmetric keys a TPM can create:

- *Migratable keys* (MK): Migratable keys are those cryptographic keys that can only be trusted by the party who generates them (e.g. the user of the platform). However, a third party has no guarantee that such key has indeed been generated on a TPM.
- *Non-migratable keys* (NMK): Contrary to a migratable key, a non-migratable key is guaranteed to be kept in a TPM-shielded location. A TPM can create a certificate stating that a key is an NMK.
- *Certified-migratable keys* (CMK): Introduced in version 1.2 of the TCG specification, this type of key allows a more flexible key-handling. Upon creation of such a key with the `TPM_CMK_CreateKey` command, a trusted *Migration Authority* (MA) can be selected, to which decisions to migrate the key are delegated. To migrate a CMK to another platform, the TPM command `TPM_CMK_CreateBlob` expects a certificate of an MA approving of the migration to the destination. Another trusted party, the *Migration Selection Authority* (MSA) may be involved in the process, which only controls the migration, but never gets in contact with any migrated data. In both cases the certificate of the CMK that is used by the owner/user to prove that it was really created by a TPM contains information about the identity of the MA resp. MSA.

Based on this functionality, the TCG specification defines four mechanisms called *integrity measurement*, *attestation*, *sealing*, and *maintenance* which are explained briefly in the following:

[1] SHA-1 [6] generates 160-bit hash values from an input of (almost) arbitrary size. One of the stated security goals for SHA-1 is: finding any collision *must* take 2^{80} units of time. Recently, Wang et. al. [18] claimed an algorithm to find such collisions in time 2^{69}, see also [3]. Though attacking SHA-1 would be challenging, SHA-1 clearly has failed its stated security goals. In contrast to some applications, full collision-resistance is essential in Trusted Computing. Hence, we anticipate revised specifications to switch to another hash function.

Integrity Measurement & Platform Configuration. Integrity measurement is done during the boot process by computing a cryptographic hash of the initial platform state. For this purpose the CRTM computes a hash of ("measures") the code and parameters of the BIOS and extends the first PCR register by this result. A chain of trust is established if an enhanced BIOS and bootloader also measure the code they are transferring control to, e.g. the operating system. The security of the chain relies strongly on explicit security assumptions about the CRTM. Thus, the PCR values $PCR_0, \ldots, PCR_n$ provide evidence of the system's state after boot. We call this state the platform's *configuration*, denoted by $S_i := (PCR_0, \ldots, PCR_n)$.

Attestation. The TCG attestation protocol is used to give assurance about the platform configuration S_i to a remote party. To guarantee integrity and freshness, this value and a fresh nonce provided by the remote party are digitally signed with an asymmetric key called *Attestation Identity Key* (AIK) that is under the sole control of the TPM. A trusted third party called *Privacy Certification Authority* (Privacy-CA) is used to guarantee the pseudonymity of the AIKs. In order to overcome the problem that this party can link transactions to a certain platform, version 1.2 of the TCG specification defines a cryptographic protocol called *Direct Anonymous Attestation* DAA [1], eliminating this CA.

Sealing. Data D can be cryptographically bound to a certain platform configuration S_0 by using the `TPM_Seal` command. Given an asymmetric key pair (SK, PK), we denote this function abstractly with $[D]_{S_0}^{PK} \leftarrow \mathsf{Seal}(S_0, PK, D)$ meaning that D is *sealed for* the configuration S_0. The `TPM_Unseal` command releases the decrypted data only if for the current configuration S_0' holds $S_0' = S_0$, or, abstractly, $D = \mathsf{Unseal}([D]_{S_i}^{PK}) \Leftrightarrow ([D]_{S_0}^{PK} \leftarrow \mathsf{Seal}(S_0, PK, D) \wedge (S_i = S_0))$.

According to the current TCG specification [16], `TPM_Seal` only accepts NMKs and it is unclear how the CMKs can be used. As we will see in Section 7 sealing using CMKs would be useful.

Maintenance. The maintenance functions can be used to migrate the SRK to another TPM: The TPM owner can encrypt the SRK under a public key of the TPM vendor using the `TPM_CreateMaintenanceArchive` command. In case of a hardware error the TPM vendor can extract the encrypted SRK from the maintenance archive, decrypt it, and load it into another TPM.

Unfortunately, the maintenance function is only optional and, to our knowledge, not implemented by currently available TPMs. Furthermore, the maintenance function works only for TPMs of the same vendor.

5 Problem Description

The integrity measurement mechanism securely stores the platform's initial configuration into the registers (PCRs) of the TPM. Any change to the measured software components results in changed PCR values, making sealed data inaccessible under the changed platform configuration. While this is desired in the case

of an untrustworthy software suite or malicious changes to the system's software, it may become a major obstacle for applying patches or software updates. Such updates do generally not change the mandatory security policy enforced by an operating system (in fact, patches *should* close an existing security weakness not included in the system specification). Nevertheless, the altered PCR values of the operating system make the sealed information unavailable under the new configuration.

We see here a major issue with the current TCG proposal, since the semantic of the sealing operation is too restrictive to efficiently support sealed information through the software life-cycle including updates / patches. The main problem is the lack of a mapping between the security properties provided by a platform configuration and its measurements. This difficulty is also pointed out in [13]: "[...] to recognize which reported PCR values were good, given the myriad platforms, operating system versions, and frequent software patches".

A further problem we see with the TCG's proposal is how to handle hardware replacements in a computing platform. Such replacements are necessary due to outdated or faulty hardware. In corporate contexts, hardware is typically replaced every few years. Any sealed data bound to a given TPM cannot directly be transferred to another TPM, because it is encrypted with a key protected by the SRK, which in turn is stored within the TPM. While this is intended to prohibit unauthorized copying, it also effectively prohibits the owner of a system to migrate the data to a replacement hardware.

In our opinion the existing mechanisms offered by the TCG specification are insufficient:

Shortcomings of Certified Migratable Keys: CMKs allow platform owners to migrate keys to another platform (see Section 4). Unfortunately, migration authorities (MA) have to explicitly certify every single key that the platform owner may want to migrate. Thus, complete migration of a TPM to another platform would require an enormous amount of certificates and thus traffic. In our opinion, it is unreasonable to assume that migration authorities can guarantee both availability and security of their services under these conditions. Another drawback is that CMKs cannot be used as encryption keys for the sealing operation. Thus, data bound to a configuration cannot be migrated using this approach.

Shortcomings of Maintenance Procedure: Although the maintenance function defined in the TCG specification protects platform owners against loss of data, its current instantiation is unsatisfying: First, this mechanism is only optional and not implemented until now. Second, the maintenance function requires interaction with the TPM vendor. In our opinion, this fact is problematic from both availability and free-enterprise perspectives: First, in case of a hardware failure all data controlled by the TPM is affected, thus TPM vendors are in the position to ask for high fees. Moreover, the TPM owner is lost if the TPM vendor does not exist anymore. Second, the maintenance function does not allow platform owners to migrate to a TPM of a different vendor.

6 Basic System Model

In this section, we propose an abstract system model that provides a practical solution to the missing link problem between the security properties offered by a platform and its configuration.

Typically, remote parties offering content do not want to limit the usability of the content to only one platform configuration. Instead, they require that their policy $\mathcal{SP}_{\mathcal{R}}$ attached to the content cannot be circumvented. Since such policies can become rather complex, we propose a three-layered security architecture to enforce them:

- **Application layer:** Remote parties can attach a piece of restricted code, a *policy checker*, to their content that decides whether the requirements of the policy $\mathcal{SP}_{\mathcal{R}}$ are fulfilled or not. Hence it is not necessary to define a general policy language be used by all remote parties. Instead, remote parties can explicitly or implicitly code the policy to be enforced into the policy checker. Note that the purpose of the policy checker is similar to what is called a *trusted viewer* [7]: The policy checker verifies that the underlying platform fulfills the necessary properties, and can enforce a complex security policy.
- **Operating-system layer:** The operating system layer performs all tasks of a usual operating system that cannot violate security policies of the involved parties. This includes resource sharing (e.g. filesystems and user interfaces) and non-critical device drivers.
- **Security-kernel layer:** As the policy checker enforces the security policies, the underlying TCB "only" has to guarantee that unauthorized entities cannot manipulate the platform such that enforcement mechanisms can be bypassed. This includes a strict separation between applications (isolation). Here the sealing mechanism provided by the underlying trusted computing hardware helps to ensure these elementary security properties. The security kernel has to be trusted by all involved parties.

Obviously existing monolithic operating systems are not capable of fulfilling these security requirements (e.g. they do not at all provide a secure isolation between processes). We therefore suggest a small Trusted Computing Base (TCB) that offers the properties of the security-kernel layer. Examples of such architectures are [2,9,11].

The advantage of this architecture is that the PCR values used by the underlying trusted computing technology (e.g. sealing, attestation) depends only on the code of the security-kernel. Applications, e.g. a web server, and the operating system layer can use more abstract services like property-based attestation [10,8] provided by the TCB. This way changes of these higher layers (e.g. due to patches) do not change the PCR values, keeping the architecture more flexible.

The reader should note that the software-based proposals in the following sections assume the existence of such a security architecture providing a trusted computing base that *securely* isolates different processes from each other. This implies that the solutions do not translate to legacy operating systems, e.g. Linux or Microsoft Windows, without weaker security guarantees.

7 Platform Updates

In our system model, updates of components outside of the TCB are easy as no PCR values are affected. However, as discussed in Section 5, a configuration change of the TCB involves more work. We consider the following security and usability requirements that must be fulfilled by a computing platform providing the sealing functionality:

- **Security.** A platform of configuration S_0 can access sealed data $[D]_{S_0}^{PK}$ only if the attached security policy $\mathcal{SP}_{\mathcal{R}}$ defines S_0 to be trustworthy. This represents the interests of the remote parties.
- **Availability.** Information sealed to a platform enforcing the security policy $\mathcal{SP}$ should be available under all platforms that are capable of enforcing $\mathcal{SP}$. Thus, a software patch should not make the information inaccessible.

In the following, we propose three solutions to the platform update problem. The first one is based on an extended software function offered by the operating system layer and allows to reuse existing TPM's. The second solution (Section 7.2) extends the TCG specification by a TPM command but allows more flexible handling of sealed data. In Section 7.3 we discuss the advantages of a property-based sealing mechanism and show how it can be implemented.

7.1 Software-Supported Updates

To guarantee availability of sealed data when the TCB's configuration S_i is changed to a S_j offering the same security properties, the TCB must provide a service we call *Update Manager* (UM). The main task of the UM is to seal the data for the new configuration S_j *before* the TCB update happens. Note that the UM has to be invoked whenever components of the TCB are to be changed.

There are several requirements on the UM to correctly update sealed data:

1. The sealed data $[D]_{S_i}^{PK}$ must be available to the update manager, i.e. in a central store, so that the UM can re-seal them. Alternatively the TCB could be implemented such that it uses only one sealed cryptographic key to encrypt all data under configuration S_i. As a consequence, only one sealed key has to be updated by the UM.
2. The PCR values for the new configuration S_j must be known to the UM. This implies that the binary representation of the module to be updated is available such that the UM can pre-calculate the expected configuration.
3. Only data that is sealed for S_i can be updated. This implies that the UM cannot update data sealed for a different configuration S_i'.
4. The update manager must have some means to ensure that the new configuration S_j offers the same security properties as the old one with respect to $\mathcal{SP}_{\mathcal{R}}$. Several solutions to this problem are imaginable. We suggest to introduce a trusted party that is responsible for certifying that two configurations S_i and S_j offer the same security properties. The identity of the trusted party could be, e.g., hard-coded into the TCB resulting in a unique

platform configuration that depends on this party. Thus, by sealing data for that configuration, remote parties explicitly agree with the TTP selected by the platform owner.
5. The whole process must be failsafe, i.e. the process must recover if it is disturbed, e.g. due to user-interruption or power-failure.

Remark 1. The underlying assumption of our model is that the TCB is small and independent enough such that updates appear only rarely. However, on a system not conforming to our system model, such as typical monolithic systems, severe problems may occur: First, system updates occur frequently, since the TCB is very complex and device drivers are part of it. Second, the UM would need to ensure that the update process does not violate any policy required by the respective remote party. Therefore, the UM has to know which security assurances are necessary for which sealed data. Currently, there is no means in the TCG proposal to specify the required assurances for sealed information.

The update process proposed above has the drawback that data can only be updated if it can be accessed under the current configuration. This complicates updates of core components like the BIOS or the bootloader, since sealed data for every possible configuration needs separate handling *before* the new component is installed. The solutions in the next two sections avoid this shortcoming.

7.2 Hardware-Supported Updates

Our second proposal is based on a new TPM command `TPM_UpdateSeal` that re-seals data for another platform configuration based on an *update certificate*. This TPM command works independently of the current platform configuration. Thus, it is possible to update all sealed data under one configuration, regardless of whether the current configuration can access the sealed data. We expect that such a command could be easily integrated into existing TPM designs.

We assume that a trusted party called *Update Certification Authority* (UCA) with a key pair (SK_{UCA}, PK_{UCA}). It issues update certificates $cert_{update} := \mathsf{Sign}_{SK_{UCA}}(S_i, S_j)$ that vouch for configurations S_i and S_j offering the same security properties.[2] Obviously, the UCA has to be trusted by all involved parties to fulfill the requirements of a secure update function. In the sense of multilateral security, both the user/owner and the remote party have to agree on an UCA *before* data is sealed. Our solution can be applied in two different ways:

– The TPM internally binds the identity $I_{UCA} = \mathsf{Hash}(PK_{UCA})$ of the UCA to the CMK key pair (SK, PK). By means of a certificate (a signature) on (PK, I_{UCA}) by the TPM, remote parties can verify that the data encrypted under this key can be updated based on certificates of that UCA. As this is based on certified migratable keys (CMKs), using a UCA instead of an MA, we require them to be usable as sealing keys.[3]

[2] A UCA could be, e.g., an existing authority already involved in software certification.
[3] Here a new type of updatable sealing key could be introduced that has exactly this functionality.

– Remote parties can define the UCA that is allowed to update data they have sealed by securely binding an identifier of the UCA to the data to be sealed (like the platform configuration under which the data can be accessed).

To provide multilateral security, the user or platform owner should be participating in defining the UCA. Otherwise, remote parties could force them to accept an untrusted one. Therefore, we prefer the first approach.

We now specify the proposed TPM command. Let the following entities and quantities be given: a TPM, a UCA with public key PK_{UCA}, and an update certificate $cert_{update} := \mathsf{Sign}_{SK_{UCA}}(S_i, S_j)$.

As prerequisites, let the user/owner have identified a UCA by $I_{UCA^*} = \mathsf{Hash}(PK_{UCA^*})$ upon creation of a new CMK key pair (SK, PK) used for sealing, and let the CMK certificate $cert_{cmk}$ state that PK is generated by a valid TPM and that it can be updated based on certificates issued by UCA^*.

Command specification (`TPM_UpdateSeal`).

Parameters: $[D]^{PK}_{S_0}$ sealed with PK bound to I_{UCA^*}, $cert_{update}$, PK_{UCA}

Command Description: The TPM

1. checks the update certificate's validity: $\mathsf{Verify}_{PK_{UCA}}(cert_{update}) \stackrel{?}{\rightarrow} \mathsf{true}$.
2. checks that $\mathsf{Hash}(PK_{UCA}) = I_{UCA^*}$.
3. checks $cert_{update}$ covers $[D]^{PK}_{S_0}$, i.e. that $S_0 = S_i$.
4. returns an error if any of the above checks fails,
5. computes and returns $[D]^{PK}_{S_j}$.

This proposed TPM command allows users/owners to control who is responsible for the creation of update certificates. Further, it is multilaterally secure with regard to remote parties who can decide if they are willing to accept the UCA.

7.3 Property-Based Sealing

While the TCG-specified trusted boot process allows to efficiently detect changes to the code, it neither allows any conclusion if a certain set of PCR values corresponds to a trustworthy system, nor does it provide any evidence if a change in the values represents a property-preserving update or an attempt to subvert the system. Since the software-supported and the hardware-supported update function are based on this so-called *binary attestation*, they both have the drawback that sealed data has to be re-sealed whenever the platform's configuration intentionally changes.

In [10,8], *property-based attestation* builds on attestation of abstract properties instead of binary representations of the platform configuration. Informally, a property of a platform describes an aspect of its behavior with respect to certain requirements, such as security-related requirements, e.g., that a platform has built-in measures for Multi-Level Security (MLS) mechanism, or built-in privacy preserving measures conforming to privacy laws; or, more suitable to our context here, a property could be the fact that the TCB guarantees the secure

execution of a policy checker (see Section 6). In general a property can be viewed as a model, and any platform complying to this model is said to provide this property.

Providing *property-based sealing* allows to bind data to properties instead of hash values of binaries. This approach has several advantages: First, if data is only bound to an abstract property, it is no longer necessary to re-seal the data during software upgrade if the underlying property does not change. Second, the UCA (see Section 7.2) would only certify that a software release provides a certain property instead of issuing update certificates between each two such releases. Third, property-based sealing allows users to use sealed data under several different platform configurations providing the same properties. Fourth, remote parties do not have to care about the concrete platform configuration of the user, since they only have to bind the data to the desired property. Fifth, remote parties are unable to discriminate certain platform configurations (e.g. Open-Source software) since the concrete configuration is kept secret.

In practice, property-based services can be provided by a small TCB (see Section 6) that depends on conventional binary attestation. As a result the large amount of applications that are using this service (e.g. a web server) do not have the problems with sealed data discussed in Section 5. In Appendix A we describe a possible realization of property-based sealing based on update certificates.

The difficulty with property-based attestation and sealing is to define the concrete semantics of a property, different remote parties may desire to bind their data to different properties, and a concrete platform configuration may provide properties that do not exactly match those desired properties. Our system model (see Section 6) moves the handling of complex property analyses to policy checker executed on the application-layer, so the only remaining property the TCB has to provide is to guarantee a secure execution of the policy checker.

8 Migrating to Another Hardware Platform

In Section 5 we discussed the shortcomings of the currently-specified maintenance mechanism. To remedy this we propose to extend the TCG specification by a multilateral-secure migration mechanism that fulfills the following requirements:

- **Completeness:** Platform owners should be able to *move* the secret state $\mathcal{T}_s$ (see Section 8.1) of a source TPM_s to a destination TPM_d. This implies that the source TPM is reliably cleared afterwards, so that only a single instance of the state exists.
- **Security:** Migration from TPM_s to TPM_d should only be possible if TPM_d has the same level of security as TPM_s.
- **Fairness:** The specification must not dictate the involved parties which TTP defines the security relations between different TPMs. Moreover, migration should be possible without the need to interact with the TPM vendors.

To fulfill these requirements, we suggest reasonable modifications and extensions of the current TCG specification, among them that all security-critical

TPM-data can be securely extracted in a non-discriminating manner. Section 8.2 describes how the idea underlying certified migratable keys (CMKs) can be used in a fail-safe protocol to securely migrate the state of a TPM to another one that provides the same security properties.

8.1 Sharing the TPM's State

Migration of a TPM's state to another TPM is only meaningful if all security-critical parts of the TPM's state can be migrated. The maintenance mechanism allows platform owners to export TPM-protected storage, including the SRK and the owner's authorization data. Unfortunately, the current TCG specification [16] does neither define mechanisms to securely export the state of the non-volatile (NV) memory, nor the values of the monotonic counters (MC). Currently, if migration is an issue, these protected resources cannot be used to store security-critical data.

Therefore we suggest to extend the TCG specification such that all resources offered by the TPM can be exported in order to make migration possible.[4]

8.2 Migration Protocol

The purpose of our migration protocol is to allow platform owners to migrate a TPM's contents while not breaking the security guarantees it provides. Our migration protocol is based on the idea of making Storage Root Keys (SRK) migratable under tightly controlled circumstances similar to Certified Migratable Keys (CMK): For this, we introduce a trusted party called *TPM Migration Authority* (TMA). Its purpose is to decide about the migration of the SRK; the TMA shall be bound to the SRK upon its creation by the `TPM_TakeOwnership` command.

The decision whether a TPM provides at least the same level of security as another TPM is highly security-critical since if TPM owners were capable of migrating their data to a less secure TPM, remote parties could not trust TPMs at all. Therefore, a TMA needs to be trusted by both the platform owner and remote parties. In practice, a TMA could be an institution that does security evaluation and certification. For privacy reasons the choice of a TMA should remain with the involved parties only. Since remote parties need to know this TMA, we further suggest to include the TMA's identity (e.g. a hash of its public key) into the AIK certificates.

The parties involved are a source $TPM_s = (EK_s, \mathcal{T}_s, SRK_s)$, a destination $TPM_d = (EK_d, \mathcal{T}_d, SRK_d)$ and a TMA identified by PK_{TMA}. We explicitly assume that the complete TPM state $\mathcal{T}$ including NV and MC can be extracted encrypted under the TPM's SRK (see Section 8.1).

[4] This would also solve another problem stemming from the NV and MC being limited resources: An operating system might allocate all these resources, so that other operating systems installed on the same system would be precluded from using them.

We assume as prerequisites:

- Upon taking ownership of TPM_s, a TMA is identified by $\mathsf{Hash}(PK_{TMA^*})$.
- The owner of TPM_s has obtained from the TMA a migration certificate $cert_{mig} = \mathsf{Sign}_{PK_{TMA}}(\mathsf{Hash}(EK_{TPM_s}), \mathsf{Hash}(EK_{TPM_d}))$ on the hashes of the endorsement keys of the TPMs. To do so, the owner proves the authenticity of both TPMs by sending the corresponding vendor certificates on TPM to the TMA.

Then the migration protocol consists of the following steps, involving a new command $\mathsf{Migrate}()$:

1. The owner extracts the encrypted TPM state $C_{\mathcal{T}_s} := \mathsf{Enc}_{PK_{SRK_s}}(\mathcal{T}_s)$ (see Sect. 8.1).
2. The owner extracts the endorsement key EK'_{TPM_d} from the destination TPM_d.
3. Upon invocation of $\mathsf{Migrate}(cert_{mig}, EK_{TPM_d}, PK_{TMA})$, TPM_s checks that
   - $cert_{mig}$ is valid, i.e. $\mathsf{Verify}_{PK_{TMA}}(cert_{mig}) \rightarrow \mathrm{OK}$,
   - the TMA issuing $cert_{mig}$ has the correct identity, i.e. $\mathsf{Hash}(PK_{TMA}) = \mathsf{Hash}(PK_{TMA^*})$,
   - the contents of $cert_{mig}$ is consistent with its endorsement key EK_{TPM_s} resp. the supplied EK_{TPM_d} using the respective hash values.
4. TPM_s encrypts SRK_s under EK_d, yielding $C_{SRK_s} := \mathsf{Enc}_{EK_d}(SRK_s)$. This is sent to the platform with TPM_d.
5. TPM_s switches into a persistent state that allows only two TPM commands: The command `TPM_ExtractMigrationData`, which returns C_{SRK_s} (the SRK encrypted under TPM_d's endorsement key), and the command `TPM_OwnerClear`, which deletes the state $\mathcal{T}_s$ and SRK_s.
6. The encrypted TPM state and SRK, i.e. $C_{\mathcal{T}_s}$ and C_{SRK_s}, are loaded into TPM_d. If an error occurs, steps 5 and 6 can be repeated.
7. After successful migration to TPM_d the owner invokes `TPM_OwnerClear` on TPM_s to clear its state and SRK.

9 Summary and Conclusion

In this paper we addressed problems arising from management of sealed data with respect to software as well as hardware life-cycles that result from the current TCG specification. In our view these problems are major obstacles for large-scale use of trusted computing technologies, e.g. in e-commerce applications.

Our proposed solutions to both problem areas are based on the principle of multilateral security and decentralized control, where only the involved parties agree on trusted parties that help to mediate between the interests of the involved parties, without a central authority like the TCG prescribing a trusted party. Furthermore, this principle protects the privacy of a system's owner and user as well as the security interests of remote parties.

For the problem of maintaining availability of sealed data after a software update we proposed a several solutions, one purely in software, a second by augmenting the TPM with an additional command, and a third one based on abstract (security) properties. While the software solution would work with current hardware, it imposes several strict requirements on the operating system design. The other solutions would open up room for advanced trusted computing concepts using abstract security properties.

Our proposal for a hardware migration method would allow to move the contents of one TPM to another one providing multilateral security. It requires some changes to the TPM, but these should be easy to integrate into the design while respecting the security interests of all involved parties.

To conclude, our proposals can resolve the identified shortcomings in the current TCG specifications regarding management of sealed data during software and hardware lifecycles. We suggest to introduce them into the TCG standardization process.

An interesting line of research might be to design protocols that employ zero-knowledge techniques so that a platform owner and remote party can agree on update and migration authorities without revealing the actual authority. Here a remote party could issue a list of authorities it trusts, and the platform owner gives a zero-knowledge proof of membership for the authority of his choice.

References

1. E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, Washington, DC, USA, Oct 2004. ACM Press.
2. P. England, B. Lampson, J. Manferdelli, M. Peinado, and B. Willman. A trusted open platform. *IEEE Computer*, 36(7):55–63, 2003.
3. A. K. Lenstra. Further progress in hashing cryptanalysis, February 2005. `http://cm.bell-labs.com/who/akl/hash.pdf`.
4. Microsoft Corporation. Building a secure platform for trustworthy computing. White paper, Microsoft Corporation, Dec. 2002.
5. C. Mundie, P. de Vries, P. Haynes, and M. Corwine. Microsoft whitepaper on trustworthy computing. Technical report, Microsoft Corporation, Oct. 2002.
6. National Institute of Standards and Technology (NIST), Computer Systems Laboratory. Secure hash standard. Federal Information Processing Standards Publication (FIPS PUB) 180-1, Apr. 1995.
7. National Research Council. *The Digital Dilemma, Intellectual Property in the Information Age*. National Academy Press, 2000.
8. J. Poritz, M. Schunter, E. Van Herreweghen, and M. Waidner. Property attestation—scalable and privacy-friendly security assessment of peer computers. Technical Report RZ 3548, IBM Research, May 2004.
9. A.-R. Sadeghi and C. Stüble. Taming "trusted computing" by operating system design. In *Information Security Applications*, volume 2908 of *Lecture Notes in Computer Science*, pages 286–302. Springer-Verlag, Berlin Germany, 2003.
10. A.-R. Sadeghi and C. Stüble. Property-based attestation for computing platforms: Caring about properties, not mechanisms. In *The 2004 New Security Paradigms Workshop*, Virginia Beach, VA, USA, Sept. 2004. ACM SIGSAC, ACM Press.

11. A.-R. Sadeghi, C. Stüble, and N. Pohlmann. European multilateral secure computing base - open trusted computing for you and me. *Datenschutz und Datensicherheit DuD, Verlag Friedrich Vieweg & Sohn, Wiesbaden*, 28(9):548–554, 2004.
12. D. Safford. Clarifying misinformation on TCPA. White paper, IBM Research, Oct. 2002.
13. D. Safford. The need for TCPA. White paper, IBM Research, Oct. 2002.
14. R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn. Attestation-based policy enforcement for remote access. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, Washington, DC, USA, Oct. 2004. ACM Press.
15. R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 11th USENIX Security Symposium*. USENIX, Aug. 2004.
16. Trusted Computing Group. TPM main specification, Version 1.2, Nov. 2003. `http://www.trustedcomputinggroup.org`.
17. Trusted Computing Platform Alliance (TCPA). Main specification, Feb. 2002. Version 1.1b.
18. X. Wang, Y. L. Yin, and H. Yu. Collision search attacks on SHA1, February 2005. `http://cryptome.org/sha-attacks.htm`.

A Property-Based Sealing Using Update Certificates

In practice, properties could be represented by a random but fixed value, while the mapping $value \rightarrow property$ defines the property assigned to that value. These values can be used in the sealing process to define the PCR values S^* of *virtual configurations* which now describe properties instead of concrete binary systems. If the `TPM_UpdateSeal` command (or an extension to `TPM_Unseal` with similar functionality) is available, it can be employed to translate between a property P_i and a concrete configuration S_i. This translation would work as follows:

- Remote parties seal data for a property P_i represented by the virtual configuration S^*, along with information which UCAs are allowed to certify that a concrete configuration actually implements the security properties, resulting in a sealed blob $[D]^{PK}_{S^*}$.
- Given a configuration S_i that actually implements the security properties of S^*, one obtains a certificate stating this fact. This certificate has the same format as the update certificates of Section 7.2, i.e. $U = \mathsf{Sign}_{SK_{UCA}}(S^*, S_0)$.
- The `TPM_updateSeal` command is used to translate $[D]^{PK}_{S^*}$ into a sealed blob $[D]^{PK}_{S_0}$ which can then directly be used.

This way, also the update problem for patched system software would just vanish, as all that is necessary to update to another concrete configuration S_j also implementing the properties of S^* is to obtain a certification of this fact. Only $[D]^{PK}_{S^*}$ would be kept in long-term storage, possibly caching $[D]^{PK}_{S_0}$.

Data Remanence in Flash Memory Devices

Sergei Skorobogatov

University of Cambridge, Computer Laboratory,
15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom
sps32@cl.cam.ac.uk

Abstract. Data remanence is the residual physical representation of data that has been erased or overwritten. In non-volatile programmable devices, such as UV EPROM, EEPROM or Flash, bits are stored as charge in the floating gate of a transistor. After each erase operation, some of this charge remains. Security protection in microcontrollers and smartcards with EEPROM/Flash memories is based on the assumption that information from the memory disappears completely after erasing. While microcontroller manufacturers successfully hardened already their designs against a range of attacks, they still have a common problem with data remanence in floating-gate transistors. Even after an erase operation, the transistor does not return fully to its initial state, thereby allowing the attacker to distinguish between previously programmed and not programmed transistors, and thus restore information from erased memory. The research in this direction is summarised here and it is shown how much information can be extracted from some microcontrollers after their memory has been 'erased'.

1 Introduction

Data remanence as a problem was first discovered in magnetic media [1,2]. Even if the information is overwritten several times on disks and tapes, it can still be possible to extract the initial data. This led to the development of special methods for reliably removing confidential information from magnetic media.

Semiconductor memory in security modules was found to have similar problems with reliable data deletion [3,4].

Data remanence affects not only SRAM, but also memory types like DRAM, UV EPROM, EEPROM and Flash [5]. As a result, there is possibility that some information still can be extracted from memory that has been erased. This could create problems with secure devices where designers assumed that all sensitive information is gone once the memory is erased.

In some smartcards and microcontrollers, a password-protected boot-loader restricts firmware updates and data access to authorised users only. Usually, the on-chip operating system erases both code and data memory before uploading new code, thus preventing any new application from accessing previously stored secrets. If the passwords or secret keys can be extracted afterwards, it could create serious problems for confidentiality of the previously encrypted information.

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 339–353, 2005.

Chip manufacturers do not publish data about remanence effects for their memory chips. The only parameter they specify is data retention – the time during which the memory content is preserved. This is almost the opposite of data remanence. Data retention time can be used roughly to estimate the data remanence effect, but this works only for devices within the same family [4].

Therefore, a series of experiments was performed to check whether it is feasible to extract information from erased EPROM, EEPROM and Flash memory devices using low-cost methods. The results should be of considerable concern to designers of embedded security applications.

2 Background

Unlike SRAM, which has only two stable logic states, EPROM, EEPROM and Flash cells store analog values in the form of a charge on the floating gate of a MOS transistor. The floating-gate charge shifts the threshold voltage of the cell transistor and this is detected with a sense amplifier when the cell is read. The maximum charge the floating gate can accumulate varies from one technology to another and normally is between 10^3 and 10^5 electrons. For standard 5 V EEPROM cells, programming causes about a 3.5 V shift in the threshold level. Some modern Flash memory devices employ multiple level detection, thus increasing the capacity of the memory [6]. There are also memory devices with a fully analog design, which store charges proportional to the input voltage [7].

Table 1. Characteristics of different memory types used in microcontrollers

	Static RAM	Mask ROM	OTP EPROM	UV EPROM	EEPROM	Flash EEPROM	NVRAM
Read time	FAST $\approx$ 10 ns	FAST $\approx$ 5 ns	MED $\approx$ 50 ns	MED $\approx$ 50 ns	MED $\approx$ 50 ns	FAST $\approx$ 20 ns	MED $\approx$ 50 ns
Write time	FAST $\approx$ 10 ns	N/A	SLOW $\approx$ 10 ms	SLOW $\approx$ 10 ms	SLOW $\approx$ 1 ms	MED $\approx$ 10 μs	FAST $\approx$ 50 ns
Data retention	$>$ 5 years (battery)	N/A	$>$ 10 years	$>$ 10 years	$>$ 40 years	$>$ 100 years	$>$ 40 years
Cell size	6T	1T	1T	1T	2T	1T	10T
Low voltage	Yes	Yes	No	No	No	No	No
Endurance (cycles)	N/A	N/A	1	100	10^3–10^6	10^4–10^6	N/A
Cost	HIGH	LOW	MED	HIGH	MED	LOW	HIGH

There are two basic processes that allow placing electrons on the floating gate – Fowler-Nordheim tunnelling and channel hot electron (CHE) injection [8]. Both processes are destructive to the very thin dielectric insulation layer between the floating gate and the channel of a transistor. This oxide layer is responsible for preserving the charge on the floating gate. As a result, the number

of possible write/erase cycles is limited, because the floating gate slowly accumulates electrons, causing a gradual increase in the storage transistor's threshold voltage and programming time. After a certain amount of program/erase cycles (typical values are represented in Table 1) it is no longer possible to erase or program the cell. Another negative effect (which is the main failure mode for Flash memory) is negative charge trapping in the gate oxide. It inhibits CHE injection and tunnelling, changes the write and erase times of the cell, and shifts its threshold voltage.

The amount of trapped charge can be detected by measuring the gate-induced drain leakage current of the cell, or its effect can be observed indirectly by measuring the threshold voltage of the cell. In older devices, which had the reference voltage for the sense amplifier tied to the device supply voltage, it was often possible to do this by varying the device supply voltage. In newer devices, it is necessary to change the parameters of the reference cell used in the read process, either by re-wiring portions of the cell circuitry or by using undocumented test modes built into the device by manufacturers.

Another relevant phenomenon is overerasing. If the erase cycle is applied to an already-erased cell, it leaves the floating gate positively charged, thus turning the memory transistor into a depletion-mode transistor. To avoid this problem, some devices, for example Intel's original ETOX [9], first program all cells to 0 before erasing them to 1. In later devices, this problem was solved by redesigning the cell to avoid excessive overerasing. However, even with this protection, there is still a noticeable threshold shift when a virgin cell is programmed and erased.

The changes in the cell threshold voltage caused by write/erase cycles are particularly apparent in virgin and freshly-programmed cells. It is possible to differentiate between programmed-and-erased and never-programmed cells, especially if the cells have only been programmed and erased once, since virgin cell characteristics will differ from the erased cell characteristics. The changes become less noticeable after ten program/erase cycles.

Programmed floating-gate memories cannot store information forever. Various processes (such as field-assisted electron emission and ionic contamination) cause the floating gate to lose the charge, and these go faster at higher temperatures. Another failure mode in the very thin tunnel oxides used in Flash memories is programming disturb, where unselected erased cells adjacent to selected cells gain charge when the selected cell is written. This is not enough to change the cell threshold sufficiently to upset a normal read operation, but could cause problems to the data retention time and should be considered during measurement of the threshold voltage of the cells for data analysis and information recovery. Typical guaranteed data retention time for EPROM, EEPROM and Flash memories are 10, 40 and 100 years, respectively.

3 Experimental Method

Obviously, in a floating gate memory cell, the floating gate itself cannot be accessed. Its voltage is controlled through capacitive coupling with the external

nodes of the device. Often, the floating-gate transistor is modelled by a capacitor equivalent circuit called the capacitor model [10]. In practice, write/erase characteristics for many EEPROM/Flash memories are close to that of a charge/discharge of a capacitor. Meanwhile there are some differences in how the charge/discharge process takes place in real memory cells. There is an initial delay between the time the voltages are applied to the cell, and the charge starting to be removed or injected. This delay is caused by the need for very high electric fields to be created inside the floating-gate transistor to start the injection or tunnelling process. Some EEPROM cells have been reported to have nonuniformity during the erase operation [11]. As a result, it might take longer to erase a half-charged cell than a fully-charged cell. In addition, an ideal capacitor discharges exponentially: $q = q_0 \cdot \mathrm{e}^{-t/\tau}$. Applied to the floating gate, that would mean that after $t = 10\tau$ the charge is totally removed from the cell. In practice this does not happen, because the parameters of the cell's transistor change as the charge is removed from its floating gate. All the above-mentioned problems could seriously affect data remanence in floating-gate memories.

The main difficulty with analysis of the floating-gate memory devices, especially EEPROM and Flash, is the variety of different designs and implementations from many semiconductor manufacturers. There are hundreds of different types of floating-gate transistor, each with its own characteristics and peculiarities. It means that for security applications where data remanence could cause problems, careful testing should be applied to the specific non-volatile memory device used in the system.

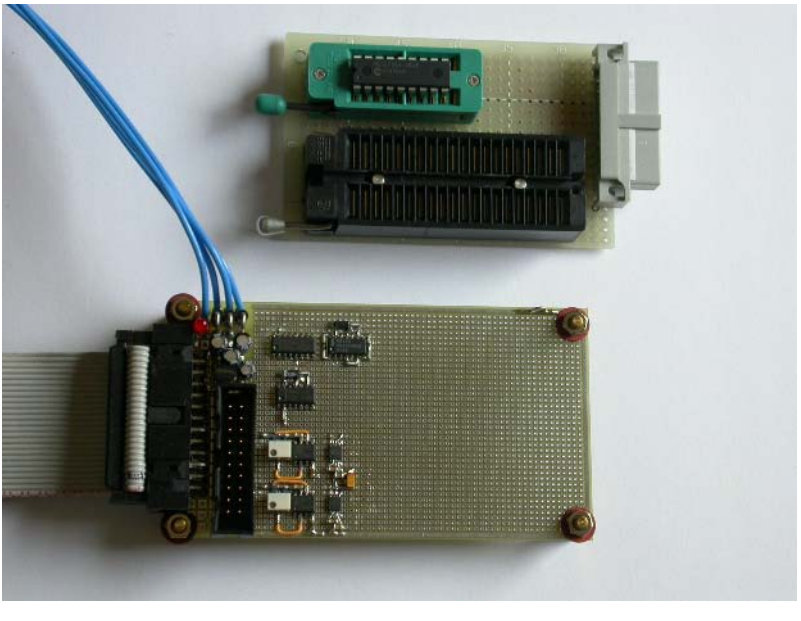


Fig. 1. The test board for data remanence evaluation

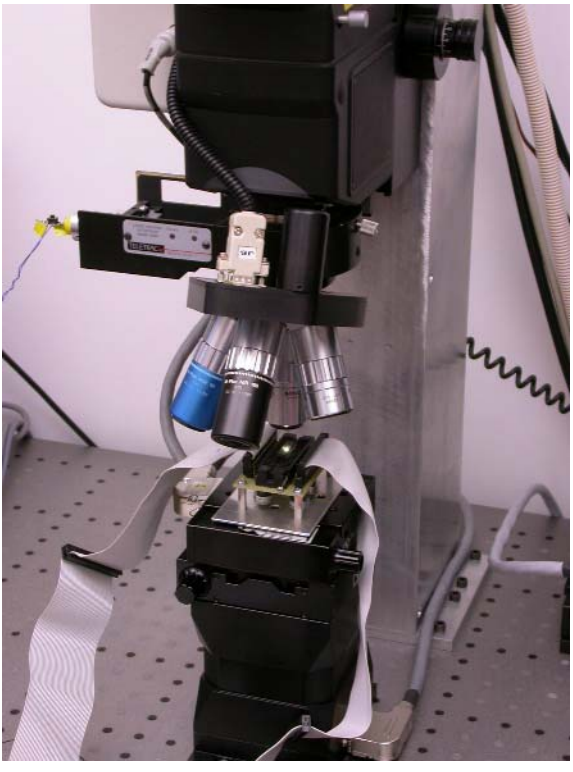


Fig. 2. Test setup for semi-invasive analysis

Some microcontrollers with different memory types to investigate the possible influence of data remanence on EPROM, EEPROM and Flash memories were tested. For that purpose I built a special test board controlled by a PC via a parallel interface (Figure 1). The board has two programmable power supplies for generating V_{DD} and V_{PP} voltages, a programming interface with bidirectional voltage level converters, and sockets for microcontroller chips. That allowed me to control the voltages applied to the chip under test with 100 μV precision and apply any signals within a 1 μs time frame.

Recently introduced semi-invasive attack methods [15] might also be helpful for testing data remanence effect in floating-gate memory devices. These methods are more effective in some applications as they do not require physical access to the internal wires inside the chip thus reducing the preparation time. However, partial or full decapsulation of the sample is required [16]. For such analysis, a low cost laser diode pointer with external power control was mounted on the autofocus module optical port of a Mitutoyo FS60Y microscope. Computer controlled Newport PM500-XYZ motorised stage was used for moving the sample under test (Figure 2). Using 100× objective on the microscope it was possible to focus the red laser beam (650 nm) down to 0.5 μm (Figure 3). Although the

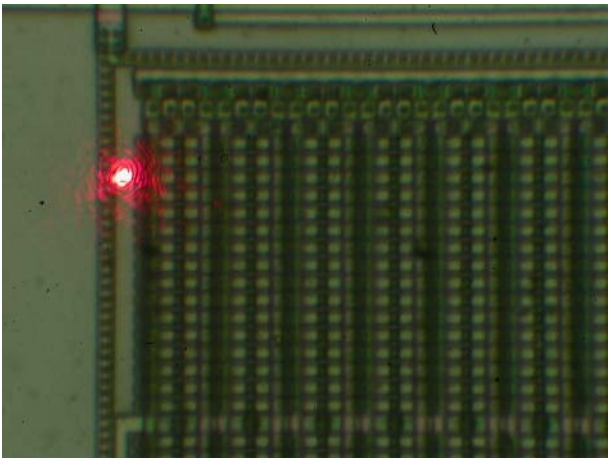


Fig. 3. Focusing the laser with a 100× objective

laser used was classified as a class 2M laser device, an ordinary digital camera mounted on the microscope was used for navigation to avoid necessity of looking at the laser beam with unprotected eyes.

4 Non-invasive Results

The first experiment was performed on the Microchip PIC12C509 microcontroller [12] with UV EPROM. The chip was programmed with all 0's (charged cell state) and exposed to UV light for different periods of time. Then it was read in the test board at different power supply voltages to estimate the threshold level for each EPROM cell in the memory array. The reference voltage was assumed to be tied to the power supply line and therefore the threshold level of the transistor is proportional to the power supply voltage $V_{\mathrm{TH}} = K \cdot V_{\mathrm{DD}}$. The fact that the exact threshold voltage of the transistor is not measured does not affect the results because an attacker is normally interested in the relative erase timing between the memory and the security protection. Once the security fuse is erased, the memory can be easily read. The same test was applied to a chip with a programmed security fuse. The results are presented in Figure 4. As can be seen from the graph, the memory gets fully erased before the security fuse is erased. However some security flaws still could exist. Although nothing could be extracted directly by reading the memory when the fuse is erased, power glitch tricks could work. For example, after seven minutes of exposure to the UV light (253 nm peak, 12 mW/cm^2) the memory content can be read non-corrupted at V_{DD} below 2.2 V, but the security fuse remains active up to 4.8 V. If the attacker works out the exact time when the data from memory is latched into the output shift register and the time when the state of the security fuse is checked, he might be able to extract the memory contents by reducing the power supply

down to 2 V for the data latching and increasing it to 5 V to make the security fuse inactive.

There is another trick that makes recovery of memory contents possible, even when there is no overlap between the erased security fuse and non-corrupted memory content at the time of erasure. For example, I found that newer samples of the same chip will start to corrupt the memory before the security fuse is erased (Figure 4). In this case a power glitch cannot be used to recover information from the memory. What can be done instead is a careful adjustment of the threshold voltage in the cell's transistor. It is possible to inject a certain portion of charge into the floating gate by carefully controlling the memory programming time. Normally, the programming of an EPROM memory is controlled by external signals and all the timings should be supplied by a programmer unit. This gives an opportunity for the attacker to inject charge into the floating gate thus shifting the threshold level enough to read the memory contents when the security fuse is inactive. Such a trick is virtually impossible to apply to modern EEPROM and Flash memory devices for several reasons. Firstly, the programming is fully controlled by the on-chip hardware circuit. Secondly, the programming of EEPROM and Flash cells is normally performed by using much faster Fowler-Nordheim tunnelling rather than CHE injection. As a result it is very hard to control the exact amount of charge being placed into the cell. Also, the temperature and the supply voltage affect this process making it even harder to control.

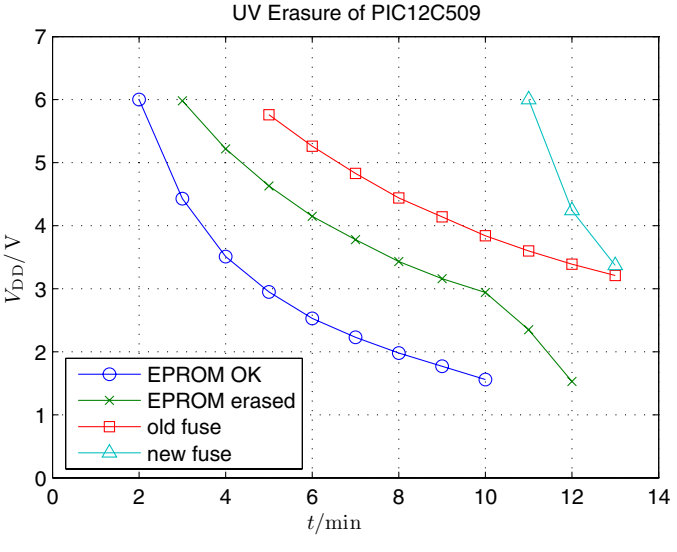


Fig. 4. Memory contents of PIC12C509 tested at different power-supply voltages after UV erasure

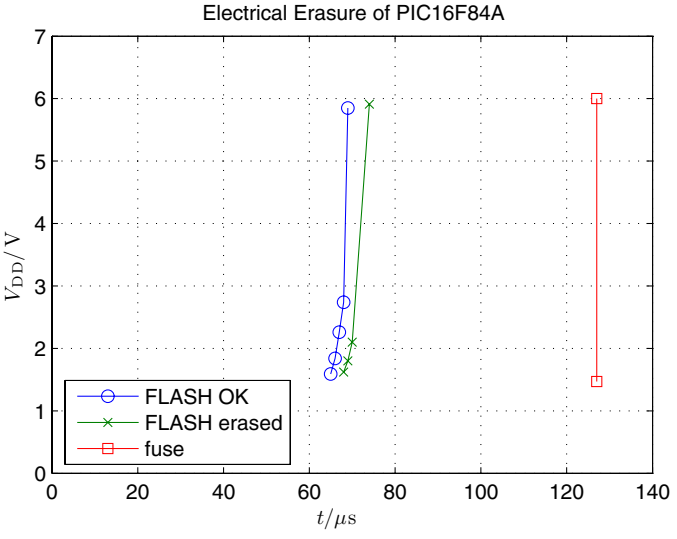


Fig. 5. Memory contents of PIC16F84A tested at different power-supply voltages after electrical erasure

The next experiment was done to the Microchip PIC16F84A microcontroller [13] which has Flash program memory and EEPROM data memory. A similar test sequence was applied with the only difference that electrical erasing was used (Figure 5). A huge difference in the memory behaviour can be observed. The memory erase starts 65 μs after the 'chip erase' command was received and by 75 μs the memory is erased. However, this time changes if the temperature or the supply voltage is changed. For example, if the chip is heated to 35 °C the memory erase starts at 60 μs and is finished by 70 μs. The security fuse requires at least 125 μs to be erased giving at least five times excess for reliable memory erase. Reducing the power supply voltage increases the erase time for both the memory and the fuse erase, so that the ratio remains practically the same. It should be mentioned that unless terminated by the hardware reset, the chip erase operation lasts for at least 1 ms. Both this fact and the fast erase time give an impression that EEPROM and Flash memories have fewer problems with data remanence and therefore should offer better security protection. I decided to investigate whether this is true or not.

In my early experiments with the security protection in PIC microcontrollers, I noticed that the same PIC16F84 chip behaves differently if it is tested right after the erase operation was completed. As this microcontroller is no longer in use and has been replaced by the PIC16F84A, the testing was applied to the new chip.

As can be seen from Figure 5, the memory is completely erased and read as all 1's well before the end of the standard 10 ms erase cycle. The threshold of the cell's transistors becomes very low after the erase and cannot be measured the same way as with UV EPROM because the chip stops functioning if the power supply drops below 1.5 V. With the power glitch technique, it is possible

to reduce the supply voltage down to 1 V for a short period of time – enough for the information from memory to be read and latched into the internal buffer. But this is still not enough to shift the reference voltage of the sense amplifier low enough to detect the threshold of the erased cells. To achieve the result another trick was used in addition to the power glitch. The threshold voltage of all the floating gate transistors inside the memory array was shifted temporarily by $V_\Delta = 0.6$–0.9 V, so that $V_{\mathrm{TH}} = K \cdot V_{\mathrm{DD}} - V_\Delta$. As a result it became possible to measure the threshold voltage of an erased cell which is close to 0 V. This was achieved by precisely controlling the memory erase operation, thus allowing the substrate and control gates to be precharged and terminating the process before the tunnelling is started. As a result, the excess charge is trapped in the substrate below the floating gate, and shifts the threshold of the transistor. The process of recombination of the trapped excess charge could take up to one second, which is enough to read the whole memory from the device. This can be repeated for different supply voltages combined with power glitches, in order to estimate the threshold of all the transistors in the memory array.

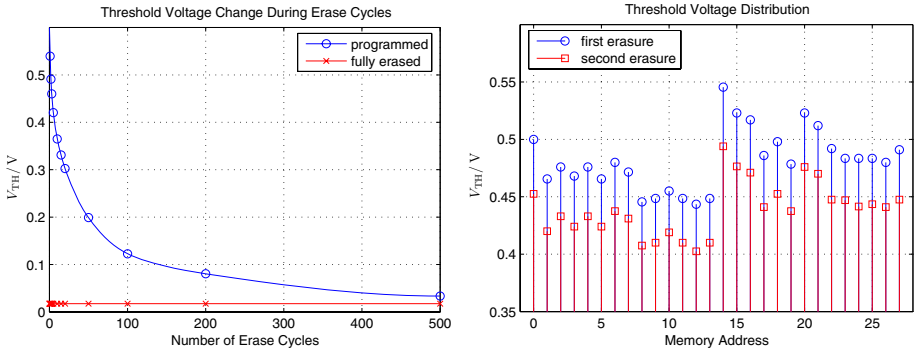


Fig. 6. Change of the threshold voltage during erasure for programmed and previously erased cells (left) and for previously programmed cells after the second erase cycle (right) in PIC16F84A

Applying the above test to differently programmed and erased chips, the diagrams for threshold voltage dependence in the Flash program memory from different factors such as the number of erased cycles (Figure 6, left) and memory address (Figure 6, right) were built. As can be seen, the charge is not entirely removed from the floating gate even after one hundred erase cycles thus making it possible for the information to be extracted from the memory. This was measured on a sample after 100 program/erase cycles to eliminate the effect of the threshold shift taking place in a virgin cell. At the same time the memory analysis and extraction is complicated by the fact that the difference in threshold voltages between the memory cells is larger than within the same cell after single erase cycle. The practical way to avoid this problem is to use the same cell as a reference and compare the measured threshold level with itself after the extra

erase operation is applied to the chip. Very similar results were received for the EEPROM data memory inside the same PIC16F84A chip. The only difference was that the threshold voltage after ten erase cycles was very close to that of the fully erased cell, thus making it almost impossible to recover the information if the erase operation was applied more than ten times.

In the next test, the chip was programmed with all 0's before applying the erase operation. As a result it was practically impossible to distinguish between previously programmed and non-programmed cells. That means that pre-programming the cells before the erase operation could be a reasonably good solution to increase the security of the on-chip memory.

One more thing should be mentioned in connection with hardware security. Some microcontrollers have an incorrectly designed security protection fuse, which gets erased earlier than the memory. As a result, if the 'chip erase' operation is terminated prematurely, information could be read from the on-chip memory in a normal way. That was the case, for example, for the Atmel AT89C51 microcontroller. When this became known in the late nineties, Atmel redesigned the chip layout and improved security to prevent this attack, so that chips manufactured since 1999 do not have this problem. Nowadays, most microcontroller manufacturers design their products so that the security fuses cannot be erased before the main memory is entirely cleared, thus preventing this low cost attack on their devices.

5 Semi-invasive Results

The first experiment was performed on the PIC16F84A microcontroller to check whether it would be possible to extract any information from previously erased memory using semi-invasive methods with the setup mentioned in Section 3.

The location of the memory was initially found under a normal optical microscope. Then, using a proprietary laser scanning setup [16], areas sensitive to the ionisation with laser radiation (bright areas) were found (Figure 7).

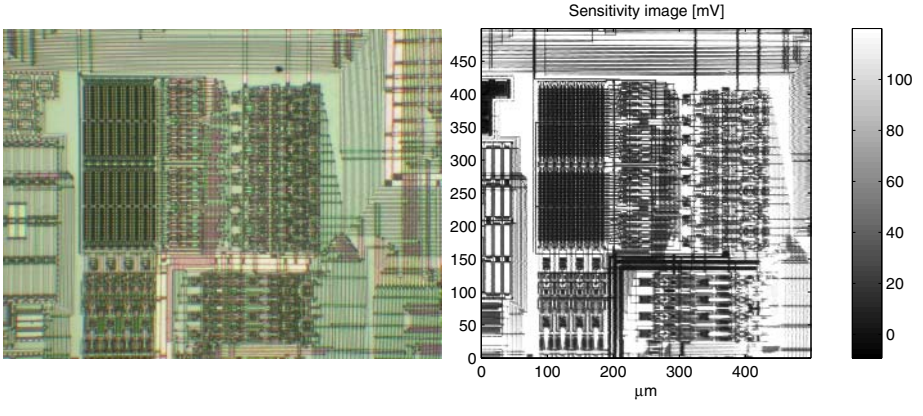


Fig. 7. Optical and laser-scanned images of the PIC16F84A EEPROM area

A standard Flash memory array consists of the current source, memory cells, row and column selectors and a sense amplifier consisting of an amplifier and a comparator to the reference cell signal which will distinguish between 0 and 1 [8]. Obviously, if we are interested in restoring the state of previously erased or discharged cell we have to either reduce the current flowing through the cell, or increase the reference voltage of the read sense amplifier, or reduce the coefficient of the amplification itself.

Because the laser can only generate the current in p-n junctions, it is not possible to manipulate the transistor in all of the desired ways. However, for most memories built with NMOS technology this will work quite well as the laser will inject current with the opposite polarity to the current sent through the memory cells.

In my experiments I erased the data EEPROM memory for the time necessary for the memory to be read back fully erased at minimum and maximum power supply voltages. Then the sample was placed under a microscope and several areas were tested with a laser pointer beam with powers ranging from 10 μW to 5 mW. Better results were received when either the area close to the column selector or the area close to the input of the sense amplifier was exposed to the laser beam. For each memory bit the value of the laser power corresponding to the change of its value from 1 to 0 was stored in the file. Due to the reason mentioned in the previous chapter it was not possible to extract the memory contents directly by adjusting the reference voltage of the sense amplifier. Therefore, after the first measurement an extra memory erase operation was performed and the next measurement was done. Comparing the results for each memory cell revealed its content because a previously programmed cell had changed its threshold value while a non-programmed cell had not.

Going back to Figure 5 it can be noticed that when more than 75 μs has elapsed since the erase command the contents of the memory cannot be read directly. Using the above technique I was able to reliably extract the information from the memory after a 150 μs erase pulse. This is still well below the standard 10 ms erase operation but is sufficient to erase the security fuse so that the attacker can perform a 'chip erase' operation and then extract the information from the memory.

The most important advantage of the semi-invasive technique is that it is independent of the power supply voltage and uses only laser power alteration to measure the threshold voltage of the memory transistors. This overcomes certain protections used in modern secure chips where either voltage monitors or voltage stabilisers are used.

The next step in my research was to test whether such a semi-invasive technique would work for modern submicron chips. As a target for my next experiments I chose the Atmel ATmega8 microcontroller [17] which employs 0.35 μm technology (Figure 8). It has three metal layers and as a result there is very little information that can be gained from direct optical observation of the chip under a microscope. To solve this problem and find the memory components on the die it was deprocessed using a wet chemical etching technique. The same die with

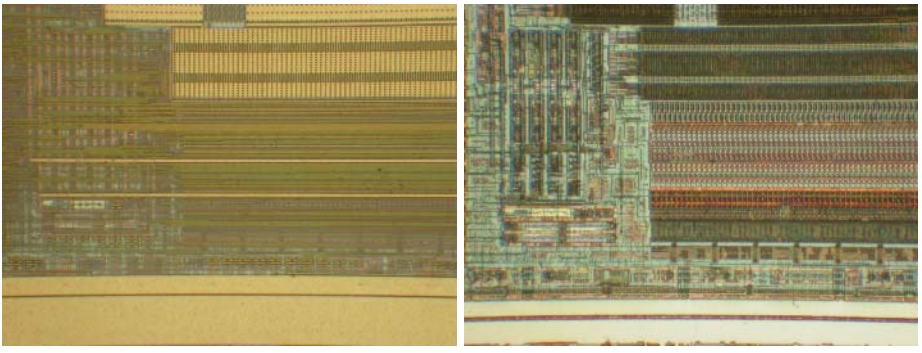


Fig. 8. Optical image of EEPROM area in the ATmega8 microcontroller before and after wet chemical etching

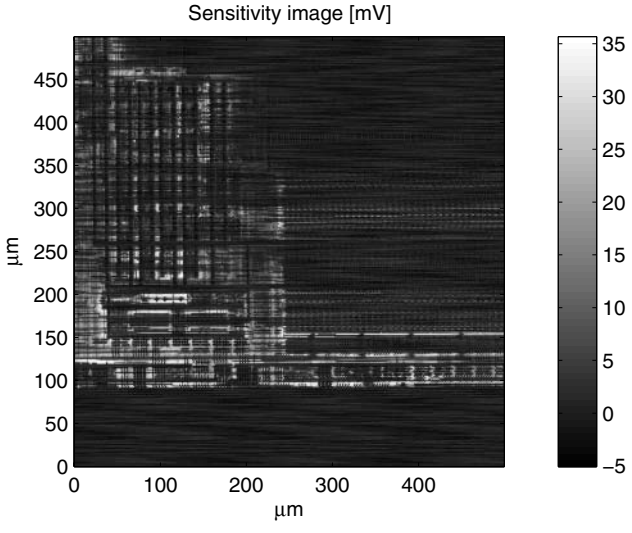


Fig. 9. Laser-scanned image of the ATmega8 EEPROM area

the top metal layer removed is shown in Figure 8. As a result of this operation all of the memory arrays located on the chip die were recognised.

To find the active areas for the laser injections, the previously mentioned laser scanning technique was used. However, as the chip was built with smaller technology and a large part of its surface is covered with metal wires, only a small part of the die was sensitive to the laser beam (Figure 9) and the injected current was significantly smaller than in case of PIC16F84A chip which has 0.9 μm technology. In addition, the chemical-mechanical polishing used in the production of ATmega8 die reduces the transparency of the layers and only a small fraction of light reaches the active area on the chip (Figure 10). All these facts made the analysis and further testing of this chip more difficult.

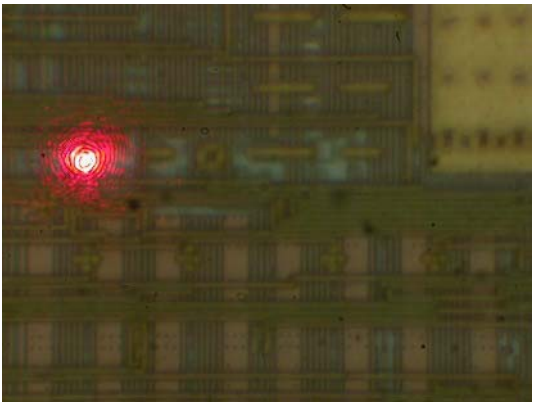


Fig. 10. Focusing the laser on the ATmega8 die using a 100× objective

The ATmega8 microcontroller employs a very reliable security protection feature which ensures that the memory is erased well before the security fuse that prevents external access to the memory. In my experiments, I was able to extract information from the erased memory only if the erase pulse was less than 100 μs long, whereas the standard 'chip erase' operation takes 10 ms. It was still impossible to read the memory contents even after a 70 μs long erase pulse at both minimum and maximum power supply voltages, but this is still not enough to overcome the security protection. However, semi-invasive methods again showed their advantages, especially because I was not able to find any non-invasive approach for extracting the information from an erased ATmega8 microcontroller.

6 Countermeasures

To avoid data remanence attacks in secure applications, the developer should follow some general design rules that help to make data recovery from semiconductor memories harder [5]:

- Cycle EEPROM/Flash cells 10–100 times with random data before writing anything sensitive to them, to eliminate any noticeable effects arising from the use of fresh cells.
- Program all EEPROM/Flash cells before erasing them to eliminate detectable effects of residual charge.
- Remember that some non-volatile memories are too intelligent, and may leave copies of sensitive data in mapped-out memory blocks after the active copy has been erased. That also applies to file systems, which normally remove the pointer to the file rather than erasing the file itself.
- Use the latest highest-density storage devices, as the newest technologies generally make data recovery more difficult.

– Using memories covered with top metal layer or built with modern deep submicron technologies helps against semi-invasive attacks because such attacks require the laser beam to reach the transistor active areas.

Using encryption, where applicable, also helps to make data recovery from erased memory more difficult. Ideally, for secure applications, each semiconductor memory device should be evaluated for data remanence.

7 Conclusions

Floating-gate memory devices, such as UV EPROM, EEPROM and Flash, have data remanence problems. From some samples, information can still be recovered after 100 erase cycles. Even if the residual charge cannot be detected with existing methods, this might be possible in the future with new technologies. Hardware designers should pay attention to the evaluation of components planned to be used in systems sensitive to data remanence.

Fortunately, the presented techniques for extracting erased memory can be applied only to a limited number of chips with EEPROM or Flash memory. Firstly, some microcontrollers, such as the Texas Instruments MSP430 family [14], have an internally stabilised supply voltage for the on-chip memory. Changing the power supply from 1.8 V to 3.6 V does not affect a memory read operation from partially erased cells. Secondly, most microcontrollers fully reset and discharge the memory control circuit if the chip is reset or the programming mode is re-entered. But still, if the memory contents do not disappear completely, this can represent a serious threat to any security based on an assumption that the information is irrecoverable after one memory erase cycle. Where non-invasive methods fail, invasive methods could still succeed. For example, the memory control circuit can be modified using a focused ion-beam workstation to directly access the reference voltage, the current source or the control gate voltage. Finally, some chips program all the memory locations before applying the erase operation. This makes it almost impossible to extract any useful information from the erased memory.

Semi-invasive methods have once again shown their use in hardware security analysis. However, they have some limitations, especially for modern deep submicron technologies, where multiple metal layers and small transistor size prevent easy and precise analysis. Further improvements to these methods might involve approaching the die from its reverse side but this requires the use of more expensive equipment.

References

1. A Guide to Understanding Data Remanence in Automated Information Systems. Version 2, September 1991, NSA/NCSC Rainbow Series
2. Peter Gutmann: Secure Deletion of Data from Magnetic and Solid-State Memory. 6th USENIX Security Symposium Proceedings, San Jose, California, July 22–25, 1996, pp. 77–89

3. Ross J. Anderson, Markus G. Kuhn: Tamper Resistance – a Cautionary Note. The Second USENIX Workshop on Electronic Commerce, Oakland, California, November 18–21, 1996
4. Sergei Skorobogatov: Low Temperature Data Remanence in Static RAM. Technical Report UCAM-CL-TR-536, University of Cambridge, Computer Laboratory, June 2002
5. Peter Gutmann: Data Remanence in Semiconductor Devices. 10th USENIX Security Symposium, Washington, D.C., August 13–17, 2001
6. Intel StrataFlash Memory (J3), 28F256J3, 28F128J3, 28F640J3, 28F320J3. ftp://download.intel.com/design/flcomp/datashts/29066719.pdf
7. P.L. Rolandi, R. Canegallo, E. Chioffi, D. Gerna, G. Guaitini, C. Issartel, A. Kramer, F. Lhermet, M. Pasotti: 1M-Cell 6b/Cell Analog Flash Memory for Digital Storage. SGS-Thomson Microelectronics, IEEE International Solid-State Circuits Conference (ISSCC), Agrate Brianza, Italy, 1998
8. William D. Brown, Joe E. Brewer: Nonvolatile Semiconductor Memory Technology: A Comprehensive Guide to Understanding and Using NVSM Devices. IEEE Press, 1997
9. Intel 28F010 and 28F020, 5 Volt Bulk Erase Flash Memory. http://www.sunmark.com/datasheets/28f010.pdf
10. Paolo Pavan, Luca Larcher, Massimiliano Cuozzo, Paola Zuliani, Antonino Conte: A Complete Model of E2PROM Memory Cells for Circuit Simulations. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 22(8), August 2003
11. H. Kume, H. Yamamoto, T. Adachi, T. Hagiwara, K. Komori, T. Nishimoto, A. Koike, S. Meguro, T. Hayashida, T. Tsukada: A Flash-erase EEPROM cell with an asymmetric source and drain structure. IEEE IEDM Technical Digest, 1987, pp. 560–563
12. Microchip PIC12C5XX Data Sheet, 8-Pin, 8-Bit CMOS Microcontrollers. http://ww1.microchip.com/downloads/en/DeviceDoc/40139e.pdf
13. Microchip PIC16F84A Data Sheet, 18-pin Enhanced Flash/EEPROM 8-bit Microcontroller. http://ww1.microchip.com/downloads/en/DeviceDoc/35007b.pdf
14. Texas Instruments, MSP430x1xx Family, User's Guide. http://focus.ti.com/lit/ug/slau049e/slau049e.pdf
15. Sergei Skorobogatov, Ross Anderson: Optical Fault Induction Attacks. Cryptographic Hardware and Embedded Systems Workshop (CHES-2002), LNCS, Vol. 2523, Springer-Verlag, 2002, pp. 2–12
16. Sergei Skorobogatov: Semi-invasive attacks – A new approach to hardware security analysis. Technical Report UCAM-CL-TR-630, University of Cambridge, Computer Laboratory, April 2005
17. Atmel ATmega8 Data Sheet, 8-bit, 8K Bytes In-System Programmable Flash Microcontroller. http://www.atmel.com/dyn/resources/prod_documents/doc2486.pdf

Prototype IC with WDDL and Differential Routing – DPA Resistance Assessment

Kris Tiri[1], David Hwang[1], Alireza Hodjat[1], Bo-Cheng Lai[1], Shenglin Yang[1], Patrick Schaumont[1], and Ingrid Verbauwhede[1,2]

[1] Electrical Engineering Dept.,
UC Los Angeles, USA
{tiri, ingrid}@ee.ucla.edu
[2] Dept. ESAT/SCD-COSIC,
K.U.Leuven, Belgium

Abstract. Wave dynamic differential logic combined with differential routing is a working, practical technique to thwart side-channel power attacks. Measurement-based experimental results show that a differential power analysis attack on a prototype IC, fabricated in 0.18µm CMOS, does not disclose the entire secret key of the AES algorithm at 1,500,000 measurement acquisitions. This makes the attack de facto infeasible. The required number of measurements is larger than the lifetime of the secret key in most practical systems.

Keywords: side-channel attack (SCA), differential power analysis (DPA), countermeasure, dual rail with precharge, wave dynamic differential logic (WDDL), differential routing, parasitic capacitance matching.

1 Introduction

A prototype IC has been fabricated in 0.18µm CMOS to demonstrate the secure digital design flow [11]. This design flow creates correct-by-construction side-channel power attack resistant integrated circuits. It starts from any HDL design and does not need custom layout, iterative design processes, or complex algorithm-specific countermeasures. It is based on employing logic cells with a single switching event per clock cycle and a place and route approach that balances the interconnect capacitance of the output wires.

Side-channel power attacks can be mounted on ASICs, FPGAs, DSPs and microprocessors because in standard CMOS technology, power is only drawn from the power supply when a 0 to 1 output transition occurs. Therefore, by measuring the power supply current during the encryption, and then performing statistical analysis of the measured power traces, the secret key can readily be determined. The secure digital design flow pursues a constant power dissipation by balancing the power consumption of the logic gate. When the power dissipation of the smallest building block is constant and independent of the signal activity, no information is leaked through the power supply. As a result, it protects against all power attacks including simple power analyses, differential power analyses and higher order power analyses.

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 354 – 365, 2005.

Two functionally identical coprocessors have been fabricated on the same die. The first 'secure' coprocessor is implemented using WDDL and differential routing. The second 'insecure' coprocessor is implemented using regular standard cells and regular routing techniques. We fabricated two functionally identical coprocessors to allow us to compare the side-channel attack resistance of a typical IC versus one with special circuit techniques. Measurement-based experimental results show that a DPA attack on the insecure coprocessor requires only 8,000 measurements to disclose the entire 128-bit secret key. The same attack on the secure coprocessor still does not disclose the entire secret key at 1,500,000 measurements.

The remainder of the paper is organized as follows. The next section describes the prototype IC. It also discusses in brief the secure digital design flow and the architecture of the AES cryptographic engine. In section 0, our measurement setup is presented and an attack is mounted on the fabricated IC to assess the increase in DPA resistance of the secure coprocessor. This section also presents area, timing and power numbers. Section 0 presents related state-of-the-art. Finally, a conclusion will be formulated.

2 Prototype IC

The prototype IC, depicted in figure 1, consists of two functionally-identical coprocessors and is fabricated on the same die using a TSMC 6M 0.18µm process. An insecure coprocessor serving as benchmark is implemented using standard cells and regular routing techniques. A secure coprocessor is implemented through the secure

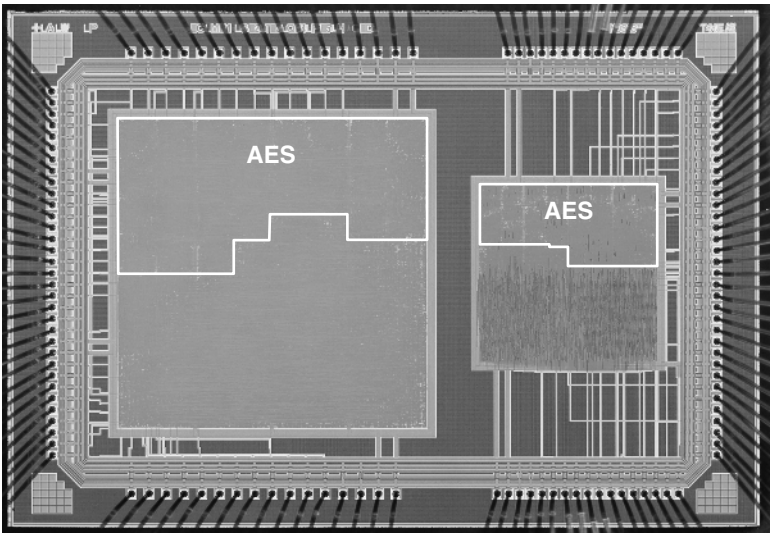


Fig. 1. IC micrograph: secure coprocessor using WDDL and differential routing (left); and insecure coprocessor using standard cells and regular routing (right)

digital design flow using WDDL and differential routing. Both coprocessors have been implemented starting from the same synthesized gate level netlist. The WDDL gates have been derived from the commercial static CMOS standard cell library used in the regular insecure design.

The IC, which is used for embedded cryptographic and biometric processing, consists of four components: an AES based cryptographic engine, a fingerprint-matching oracle, a template storage, and an interface unit. The coprocessor is part of a portable biometric and cryptographic authentication device that is called ThumbPod [5]. Architectural partitioning has been performed to divide the system into insecure (LEON SPARC V8 processor) and secure (coprocessor) modules, such that the processing and storage of all sensitive information is done on the secure module [6]. This ensures that the entire system does not need to be protected. Only the secure module must be protected for the system to remain secure.

2.1 Secure Digital Design Flow

The secure digital design flow is completely supported by mainstream EDA tools and uses a commercially available static CMOS standard cell library. The differences with a regular synchronous CMOS standard cell design flow are minor. The secure digital design flow starts from a normal design in a hardware description language (HDL) and only a few key modifications are incorporated in the backend of the design flow. A cell substitution phase and an interconnect decomposition phase parse intermediate design files. The former procedure modifies the gate level description, the latter duplicates and translates the interconnect wires. The additional steps only required six minutes of CPU time for the prototype IC.

The design flow is based on a constant power dissipating logic: in one clock cycle the power consumption of each individual logic gate is constant and independent of its input signals. In other words, 0 to 0, 0 to 1, 1 to 0, and 1 to 1 output transitions all draw the same power from the supply. Two conditions must be satisfied to have constant power dissipating logic: a logic gate must have exactly one charging event per clock cycle; and the logic gate must charge a constant capacitance in that event.

Dynamic differential logic, also known as dual rail with precharge logic, has a single charging event per cycle. The design flow uses wave dynamic differential logic to implement dynamic differential behavior using static CMOS standard cells [13]. A WDDL gate consists of a parallel combination of two positive complementary gates. In the precharge phase, both true and false inputs are set to 0. This puts the output of the gate at 0. This 0 precharge value travels as the input to the next gate, creating a precharge ‘wave’. In the evaluation phase, each input signal is differential and the WDDL gate calculates a differential output. Special registers and input converters launch the precharge value. They produce an all-zero output in the precharge phase but let the differential signal through during the evaluation phase.

Besides a 100% switching factor, it is essential that a fixed amount of capacitance is charged during the transition. Thus, the total load at the true output of the differential gate should match the total load at the false output. With shrinking channel-length of the transistors, the interconnect capacitances have become the dominant capacitance. Hence, the issue of matching the interconnect capacitances of

the signal wires is crucial. The best strategy to achieve matched interconnect capacitances is differential routing [12]. The true and false output signals are routed at all times with parallel routes in adjacent tracks of the routing grid, on the same layers, and of the same length. Independent of the placement, the two routes have the same first order parasitic effects.

2.2 AES-Based Cryptographic Engine

The cryptographic engine consists of an AES core with multiple modes of operation. The datapath is based on a single round of the AES-128 algorithm which consists of byte substitution, shift row, mix column, and key addition phases along with on-the-fly key scheduling in (see figure 2). Byte substitution is implemented using look-up tables. A full encryption of 128-bit data using a 128-bit key takes precisely eleven cycles. For a detailed discussion on the architecture, the reader is referred to [4].

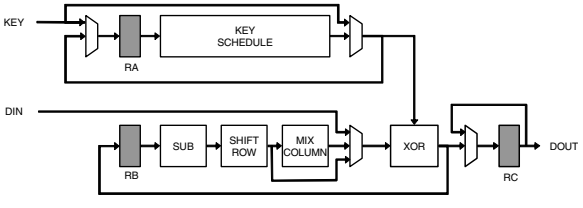


Fig. 2. Architecture of AES core

3 DPA Resistance Assessment

3.1 Measurement Setup

The measurement and analysis setup is depicted in figure 3. The core supply current is measured between the PCB decoupling capacitances and the IC. A CT1 current probe from Tektronix [10] with a 25KHz to 1GHz bandwidth measures the supply current variations. For every mA, it provides 5mV output to the HP54542C oscilloscope [1]. The oscilloscope filters the waveform transients at 500MHz and digitizes with a 2GHz sampling frequency. With a standard GPIB interface, we have made up to 400 measurements a second, including data transfer. Such a setup only requires four minutes to make 100,000 power measurements. A ‘measurement’ refers here to multiple data points which are used as one acquisition in a side-channel attack.

To facilitate the synchronization of the measurements, we have access to the encryption start signal. A clock of 50MHz is provided to the coprocessor under attack. During the attack, only the AES core processes data. This means that for the attack on the insecure processor, the other circuits and modules are quiet, while for the attack on the secure processor, the other circuits and modules are constantly charging and precharging in the same manner.

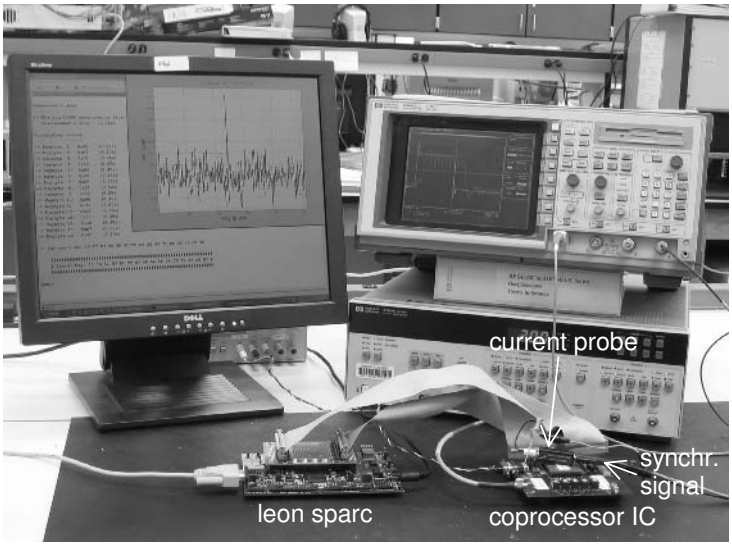


Fig. 3. DPA measurement and attack setup

Figure 4 shows the encryption start signal and the supply current of the AES coprocessors in output feedback (OFB) mode. The supply current of the insecure coprocessor exhibits large variations. It broadcasts the eleven encryption rounds and a high power peak exposes the starting point of each new encryption. The power consumption profile of the secure implementation on the other hand is invariant and does not reveal any information in a simple power analysis. In each clock cycle, nominally the same total load capacitance is charged and thus the same power is consumed, regardless of the operation been performed.

3.2 Differential Power Analysis

The DPA attack is performed as the AES core encrypts a plaintext P, using a key K, to produce a ciphertext C_{11} after eleven rounds. Note that the original K is broken up into different round keys (K_1 through K_{11}), where K_{11} is the round key for round eleven. Once K_{11} is deduced, it is easy to trace it back to find the original key K.

The influence of the datapath on the power consumption of the AES core is estimated through the Hamming distance of two successive values of register RB, (see figure 2) or in other words, through the number of changing state bits in a clock cycle. Most AES operations work with bytes and eight state bits can be calculated using a guess on one key byte. Using the same measured data, each of the sixteen bytes of the 128-bit key is cracked separately in the following manner.

We compare the estimations and the measurements with the correlation test [2]. The correct key guess is the one that results in the highest correlation coefficient between the vector of Hamming distances and the vector of representative measurements, for which we use the maximum supply current in a clock cycle.

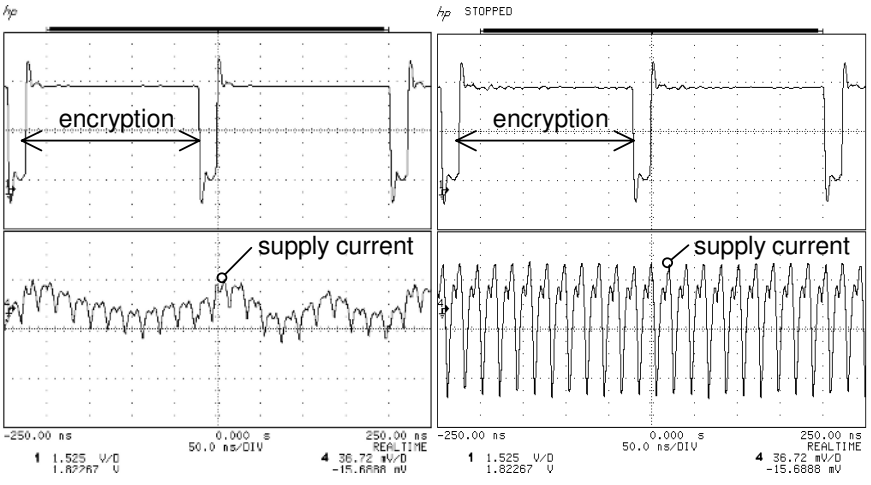


Fig. 4. Transient measurement (2 encryptions, 22 clock cycles) of encryption start signal (top) and core supply current (bottom): insecure coprocessor (left); and secure coprocessor (right)

We choose to attack register RB as it transitions from round eleven to the following round. As shown in figure 5, RB in round eleven (D_{11}) can be found by tracing back the signal obtained after xor-ing the final ciphertext (C_{11}) and a key guess (K_{11}) through both the shift row operation and the substitution box. RB in the next round, during which we perform the supply current measurement, is the known final ciphertext (C_{11}).

Each key byte (0 to 15) of K_{11} can be a value between 0x00 and 0xFF, for a total of 256 possibilities. Thus, for each key byte, there are 256 power estimations, one of which is the correct estimation. Of course the correlation may be inaccurate for only a few measurements (i.e., sets of $P_{measurement}$ and C_{11}). Hence thousands of different ($P_{measurement}$,C_{11}) pairs were measured using the same key (and hence the same K_{11}) in order to filter out the noise and provide a truthful correlation.

The correct key is found by evaluating:

$$\max_{K_{11}} f_{cost}(K_{11}) = \mathrm{corr}(P_{measurement}, P_{estimation}) \quad (1)$$

$$\textit{where} \quad P_{measurement} = \max(I_{supply,11+1})$$
$$P_{estimation} = \mathrm{HamDist}(D_{11}, C_{11})$$
$$D_{11} = \mathrm{sub}^{-1}(\mathrm{shiftrow}^{-1}(K_{11} \otimes C_{11}))$$

For the secure design, we only need to look at one round, as all signals are at 0 at the start of the evaluation phase. The number of changing bits of RB in round eleven, during which we also carry out the measurements, is the Hamming weight of RB.

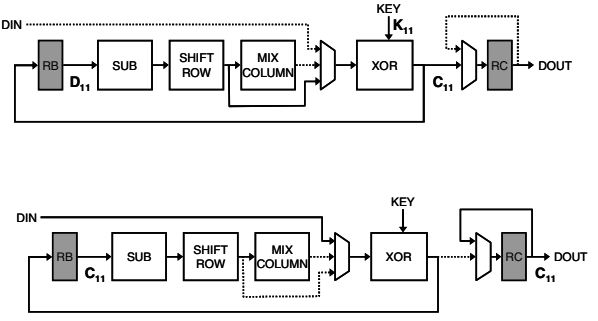


Fig. 5. AES core: round 11 (top); and round 11 + 1 (bottom)

Figure 6 shows the encryption start signal and the core supply current during the attack. The supply current of the insecure coprocessor reveals the encryption operation by showing exactly eleven peaks. The secure coprocessor has a continuous current whether or not data is being processed, either cryptographic or other. It has an identical power consumption profile in figures 4 and 6. If an attacker does not have access to the encryption start signal, it is almost impossible to know when the IC is encrypting.

For the actual attack, we only measure the round of interest. The dynamic range is set to cover the variation of the maximum current. The other irrelevant samples may be clipped. For the remainder of this manuscript, we will refer to the maximum value of one acquisition as the measurement.

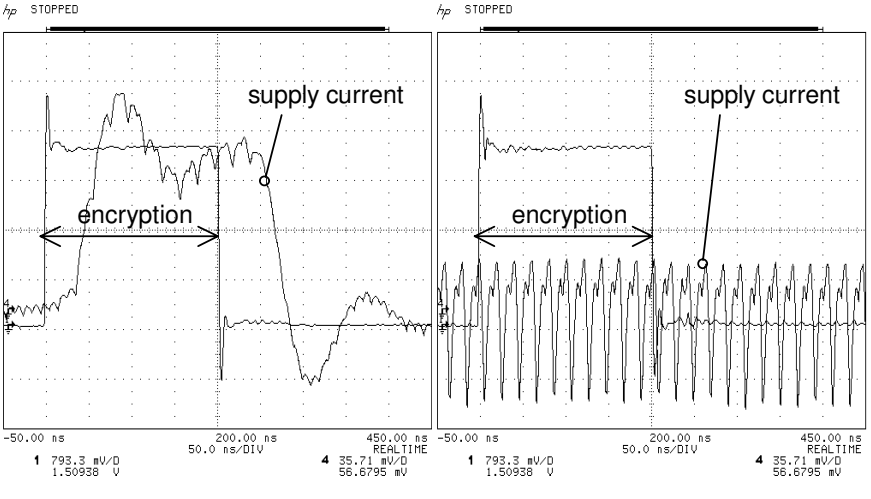


Fig. 6. Transient measurement of encryption start signal and core supply current for single encryption: insecure coprocessor (left); and secure coprocessor (right)

3.3 DPA Resistance

The resistance against DPA is quantified with the number of measurements to disclosure (MTD). This number expresses how many measurements are necessary to correctly distinguish the correct secret key from all the other wrong key guesses.

We define MTD as the cross-over point between the correlation coefficient of the correct key and the maximum correlation coefficient of all the wrong keys guesses. For both coprocessors, an example of an attack on one key byte is shown in figure 7. MTD is depicted in the 'Correlation vs. Number of Measurements' graphs as the point where the black line (correct key) crosses the grey envelope (wrong keys). The results for the other fifteen key bytes are similar. The maximum number of measurements we took is 15,000 for the insecure coprocessor and 1,500,000 for the secure coprocessor.

For the insecure implementation, the correct key bytes are found very easily. On average, 2,000 measurements are required to disclose a key byte. In one case, a mere 320 samples were sufficient to mount a successful attack. There is also a large resolution. In the top right plot of figure 7, there is no doubt about the correct key guess.

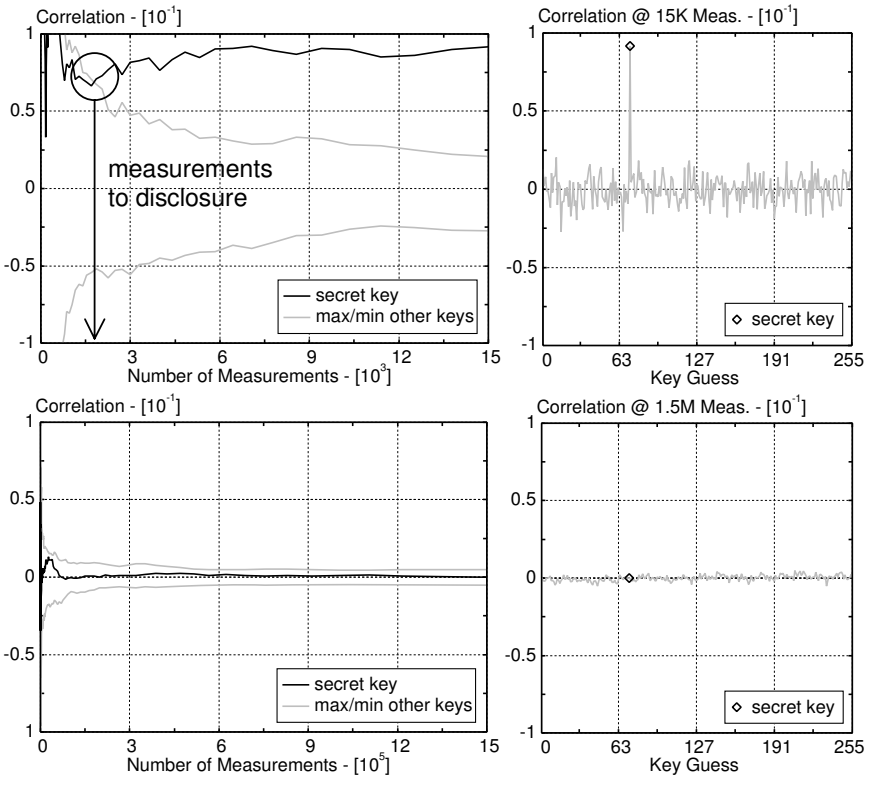


Fig. 7. Cracking the secret key: insecure coprocessor 15K measurements (top); and secure coprocessor using 1.5M measurements (bottom)

The secure coprocessor on the other hand substantially reduces this resolution of correlation, as shown by the small correlation peaks in the 'Correlation vs. Key Guess' graph in figure 7. Our measurements show that out of sixteen keys bytes, WDDL effectively protects five key bytes. One and a half million measurements are not sufficient to disclose the correct key bytes. One example is shown on the bottom of figure 7. The eleven key bytes that are found require on average 255,000 measurements, an increase of more than two orders of magnitude when compared with the insecure coprocessor. One example is shown in figure 8.

The analysis also revealed that for a dual rail design, the correlation coefficient of the correct key guess can be negative, as shown in figure 8. This means that less power is consumed as more bits change. This implies that the 0 to 1 switching of the false net uses more power than the 0 to 1 switching of the true net. The parasitic capacitances affected by the false signals are larger than the ones affected by the true signals. On the other hand, for the five bytes that have not been found, the capacitances have an almost perfect matching between the differential nets. Hence it is crucial to guarantee matched capacitances consistently for all the logic.

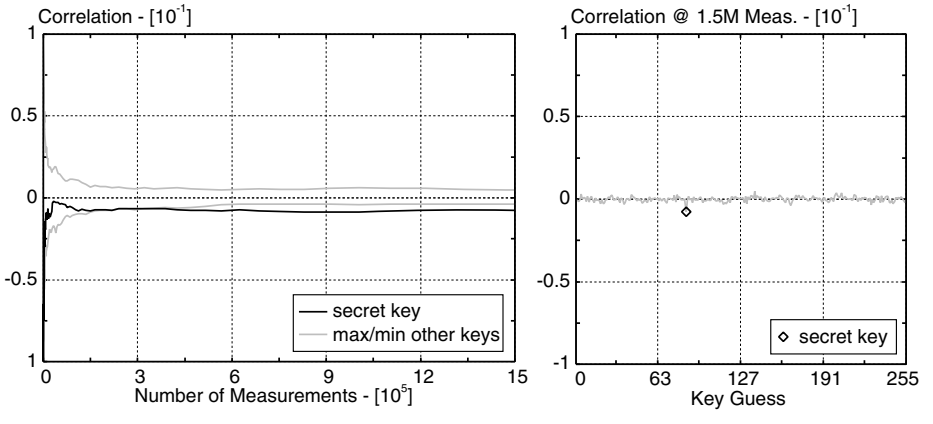


Fig. 8. Negative correlation coefficient

Shielding the differential routes on either side with a power line improves the matching. This eliminates the cross-talk to adjacent wires in the same metal layer. Alternatively, increasing the distance between different differential pairs would reduce the effect, or an improved iterative design flow could be used to identify and correct mismatches.

Table 1 summarizes the results. WDDL and differential routing is a functional technique to thwart power attacks. The trade-off is a three times increase in area, and a four times increase in power consumption and minimum clock period.

Recall from section 0 that security partitioning, the careful division of the architecture into two parts (a secure and a non-secure part) [6], minimizes the cost for complex systems. Only the relatively small part that processes sensitive information

Table 1. IC results summary

Parameter	Unprotected AES	Protected AES
Gate Count (eq. gates) [K]	79	245
Area [mm^2]	0.79	2.45
Maximum Frequency (@1.8V) [MHz]	330.0	85.5*
Maximum Throughput (@1.8V) [Gb/s]	3.84	0.99
Power Consumption (@1.8V, 50 MHz) [mW]	54	200†
Measurements to Disclosure‡		
min	320	21,185
mean	2,133	255,391
max	8,168	1,276,186
Key bytes not found (@1.5M Meas.)	n/a	5

*Duty factor of clock > 50% to guarantee precharge of all gates
†Estimation based on area ratio AES vs. Entire System
‡Based on correctly guessed key bytes

requires realization in a coprocessor with specialized logic and routing. This minimizes the area and reduces the power and time penalty. Even with these penalties, the secure coprocessor still runs orders of magnitude faster and expends less energy than a software implementation on the main processor.

The protected AES achieves a figure of merit, obtained by normalizing the throughput with the power consumption, of 2.9Gb/s/W. On the other hand, an unprotected AES implemented with C code on an embedded Sparc processor attains a mere 0.0011Gb/s/W (gcc, 1mW/MHz @120Mhz Sparc, 0.25µm CMOS). Research papers on algorithmic countermeasures unfortunately do not document the overhead in cycle count (and in byte code). Given a likely penalty of a factor 2 to 3, the figure of merit of the protected coprocessor is 4 orders of magnitude better than a software implementation of an algorithmic countermeasure on a microprocessor.

As future work, we foresee the need to explore the EMA resistance and the impact of 'noisy' regular components. Electromagnetic Analysis (EMA) is the equivalent of a power attack but instead uses the electromagnetic fields. The electromagnetic fields are generated by the (dis)charging gates. Since ideally each gate uses always the same amount of charge, a significant increase in EMA resistance is also expected. The impact of regular components, which process the insensitive data, causes a substantial increase in power variations. In figures 4 and 6, the maximum current has only minimal variations for the secure implementation while it has large variations for the regular implementation. Consequently, if both a secure and a regular component are present on an IC, the dynamic range of the measurements must be set to cover the maximum variation of the maximum current. As a result, the measurements of the side-channel information leaked by the secure module will be much less accurate and possibly within the quantization error. This analysis is possible with the prototype IC, as both the secure and the insecure processor can be operated at the same time.

4 Related Work

To our knowledge, this work is the first to deliver and demonstrate a working, practical DPA countermeasure implemented and tested in actual silicon. All other published techniques have never been implemented in silicon, or have never been measured and attacked, or did not offer any significant DPA resistance.

A dual rail asynchronous chip has been presented previously [3]. The implementation, however, did not provide a significant increase in DPA resistance. This failure has been attributed to unbalanced signal paths caused by routing differences. Note that if asynchronous logic is used to increase the DPA resistance, dual rail encoded asynchronous logic must be used. Because of the dual rail logic, there is also a factor three area increase compared with a single ended synchronous benchmark [8].

Algorithmic countermeasures are mathematically DPA resistant. In practice, however, proposed solutions actually have been insecure [7]. To the best of our knowledge, published results of algorithmic countermeasures have never been successfully demonstrated on any platform with an actual measurement-based DPA resistance assessment. We are aware of one silicon implementation of an algorithmic countermeasure [9]. Measurements and assessment of the DPA resistance, however, have not yet been performed.

5 Conclusions

We have presented a secure coprocessor that does not leak information through the power supply, which is a major and easy to access side-channel leakage source. Built in a 0.18μm CMOS technology, we believe that this is the first IC that is practically immune to DPA attacks. Its immunity has been experimentally verified and compared to a second IC that is built with a regular standard cell approach. The design approach relies on WDDL, a logic style that has a single switching event per cycle, and differential routing, a place and route technique that controls the load capacitance. An actual power attack has been mounted on the IC to experimentally assess the increase in DPA resistance. Experimental results showed that 1,500,000 acquisitions are not sufficient to fully disclose the 128-bit secret key. This makes the attack de facto infeasible. The required number of measurements is larger than the lifetime of the secret key in most practical systems.

Acknowledgements. This work was supported in part by the National Science Foundation (CCR-0098361), UC-Micro 02-079, Panasonic Foundation, SUN Microsystems, Atmel corporation and the Fannie and John Hertz Foundation.

References

1. Agilent technologies, 54542C 4 Channel 2 GSa/s Color Digitizing Oscilloscope, http://www.home.agilent.com/USeng/nav/-536894779.536881118/pd.html.
2. J. Coron, P. Kocher, and D. Naccache, “Statistics and Secret Leakage”, Financial Cryptography (FC 2000), Lecture Notes in Computer Science, vol. 1962, pp. 157–173, February 2000.

3. J. Fournier, S. Moore, H. Li, R. Mullins and G. Taylor, “Security Evaluation of Asynchronous Circuits,” Cryptographic Hardware and Embedded Systems (CHES 2003), Lecture Notes in Computer Science, vol. 2779, pp. 137–151, September 2003.
4. Hodjat, D. Hwang, B. Lai, K. Tiri, and I. Verbauwhede “A 3.84 Gbits/s AES Crypto Coprocessor with Modes of Operation in a 0.18-µm CMOS Technology”, accepted at Great Lakes Symposium on VLSI (GLSVLSI 2005), April 2005.
5. D. Hwang, P. Schaumont, Y. Fan, A. Hodjat, B. Lai, K. Sakiyama, S. Yang, and I. Verbauwhede, “Design flow for HW/SW acceleration transparency in the ThumbPod secure embedded system”, 40th Design Automation Conference (DAC 2003), pp. 60-65, June 2003.
6. D. Hwang, P. Schaumont, K. Tiri, and I. Verbauwhede, “Making embedded systems secure”, accepted at IEEE Security & Privacy Magazine.
7. S. Mangard, T. Popp, and B. Gammel, “Side-Channel Leakage of Masked CMOS Gates”, Cryptographers’ Track - RSA Conference (CT-RSA 2005), pp. 351-365, February 2005.
8. S. Moore, R. Anderson, R. Mullins, and G. Taylor, “Balanced Self-Checking Asynchronous Logic for Smart Card Applications,” Journal of Microprocessors and Microsystems, vol. 27, pp. 421–430, 2003.
9. N. Pramstaller, F. Gürkaynak, S. Häne, H. Kaeslin, N. Felber, and W. Fichtner, “Towards an AES Crypto-chip Resistant to Differential Power Analysis”, 30th European Solid-State Circuits Conference (ESSCIRC 2004), pp. 307-310, September 2004.
10. Tektronix, CT1 current probe, http://www.tek.com/site/ps/60-12572/pdfs/60W_12572.pdf.
11. K. Tiri, and I. Verbauwhede, "A VLSI Design Flow for Secure Side-Channel Attack Resistant ICs", accepted at Design, Automation and Test in Europe Conference (DATE 2005), March 2005.
12. K. Tiri, and I. Verbauwhede, "Place and Route for Secure Standard Cell Design", 6th International Conference on Smart Card Research and Advanced Applications (CARDIS 2004), pp. 143-158, August 2004.
13. K. Tiri, and I. Verbauwhede, "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation", Design, Automation and Test in Europe Conference (DATE 2004), pp. 246-251, February 2004.

DPA Leakage Models for CMOS Logic Circuits

Daisuke Suzuki[1], Minoru Saeki[1], and Tetsuya Ichikawa[2]

[1] Mitsubishi Electric Corporation, Information Technology R&D Center
{dice, rebecca}@iss.isl.melco.co.jp
[2] Mitsubishi Electric Engineering Company Limited, Kamakura Office
ichikawa@kam.mee.co.jp

Abstract. In this paper, we propose new models for directly evaluating DPA leakage from logic information in CMOS circuits. These models are based on the transition probability for each gate, and are naturally applicable to various actual devices for simulating power analysis. We also report the effectiveness of the previously known enhanced DPA on our model. Furthermore, we demonstrate the weakness of previously known hardware countermeasures for both our model and FPGA and suggest secure conditions for the hardware countermeasure.

1 Introduction

SPA (Simple Power Analysis) and DPA (Differential Power Analysis), proposed by P.Kocher, have become a threat to the security of cryptographic implementation such as SmartCard [1]. Ever since these proposals, cryptographic researchers have begun to consider not only mathematical attacks but side-channel attacks as well. This work has resulted in several proposed countermeasures, particularly against DPA. These countermeasures can be roughly classified into the following two groups:

- Algorithmic level
- Circuit level

Coron addresses countermeasures for public-key encryption algorithms [2]. Employing masked data with random numbers, Akkar uses countermeasures for block ciphers [3]. We consider the above mentioned examples to be algorithmic. On the other hand, SABL (Sense Amplifier Based Logic) [4][5] based on the DCVSL (Differential Cascode Voltage Switch Logic), SDDL (Simple Dynamic Differential Logic) based on the CMOS circuit using the SABL methodology, and WDDL (Wave Dynamic Differential Logic) [6] belong to the circuit level.

Generally, ASICs, such as microprocessors and cryptographic co-processors, are implemented based on the CMOS technology. We believe that countermeasures at the circuit level, such as WDDL and Masked-AND [7], are the most fundamental techniques because these are related to power consumption and are applicable to various cryptographic algorithms.

The manner in which the effectiveness of a countermeasure can be demonstrated is important. In this paper, we consider a methodology for the security evaluation of CMOS circuits at the outset. Some attempts have already been

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 366–382, 2005.

made to systematically analyze DPA leakage [8][9][10]. Constructing a power consumption model is an effective method for the analysis of the effectiveness of countermeasures. For instance, the model based on the analog characteristics of CMOS circuits [8], the model based on the Hamming weight [9], and the simplification model in Ref. [8] based on the transition of data registers [10] were proposed in 1999, 2000, and 2002, respectively. Each model is complex or insufficient in terms of the reason for the leakage, because the aim of the model is to simulate power consumption itself or to determine the bias of data, not the bias of power consumption. We now present new models that determines the origin of the leakage. These models are based on signal transition probability for each gate(see also [11]), and are not only more accurate than the digital model [9] but are also more easily applicable than the analog models [8][10]. We will point out that the evaluation results of some primitive logics using our models are very similar to the actual power analysis using FPGA.

Next, we discuss the relation between *enhanced DPAs* and our leakage model. Recently, various analysis technics were proposed as enhanced DPA [12][13]. The countermeasure should satisfy the requisite tolerance for these technics. In this paper, we also discuss the effectiveness of the previously known enhanced DPAs from the viewpoint of our model.

Finally, we demonstrate the weakness of previously known hardware countermeasures for both our model and FPGA and suggest secure conditions on the CMOS logic circuit.

2 Leakage Model for CMOS Circuit

The current evaluation model against DPA is constructed by simulating the power consumption of the circuit. In general, there are two approaches. One method constructs a detailed model of a characteristic of the analog device [8][10]. In this case, the power consumption can be estimated with high accuracy. However, the estimation of the power consumption is largely dependent on the device; thus, it tends to become complex. The other method roughly estimates the power consumption assuming a certain digital model; for example, it estimates the power consumption based on the Hamming weights [12]. In this approach, it is possible to construct simple models and evaluate power consumption without device dependency. However, the result might not accurately reflect the behavior of the actual device.

In the following section, we propose a more detailed model that improves on the flipping model introduced in Ref. [15] for CMOS circuits. Hereafter, we refer to this model as the *leakage model*. The primary concept of the model is mainly to evaluate only the leakage information for DPA. Power consumption itself is not considered in this model.

2.1 Leakage Model Based on Transition Probability

Power consumption in CMOS circuits is summarized by the following equation [16]:

$$P_{\text{total}} = p_{\text{t}} \cdot C_{\text{L}} \cdot V_{\text{dd}}^2 \cdot f_{\text{clk}} + p_{\text{t}} \cdot I_{\text{sc}} \cdot V_{\text{dd}} \cdot f_{\text{clk}} + I_{\text{leakage}} \cdot V_{\text{dd}}, \tag{1}$$

where C_{L} is the loading capacitance, f_{clk} is the clock frequency, V_{dd} is the supply voltage, p_{t} is the transition probability of the signal, I_{sc} is the direct-path short circuit current, and I_{leakage} is the leakage current.

The first term results from the charge/discharge of the loading capacitance. The second term depends on I_{sc}, which arises when both the NMOS and PMOS transistors are simultaneously active. The third term represents power consumption caused by the leakage current, which is primarily determined by the characteristics of the CMOS process.

DPA is an attack in which the attacker estimates the intermediate value in the encryption/decryption process, classifies the patterns of power consumption based on this estimate, and obtains the secret information from the measured differences. Here, only p_{t} is dependent on the intermediate value in Eq.(1). Other parameters are fixed when the circuit is constructed. We assume that the power difference in DPA measurements occurs because the transition probability of the signal is biased according to the intermediate value. A detailed discussion of the bias of the transition probability is presented below.

Generally, the signal transitions also depend on the delay in the transistors and the wiring in the CMOS device as well as on the logic functions of the circuits. Thus, we consider the leakage model in either of the following cases:

- *Static Model* : An ideal circuit with no delay and transient hazard in transistor and wiring.
- *Dynamic Model* : A real circuit wherein a transient hazard is generated due to the delay.

In order to clarify the discussion, we analyze the generalized circuit as shown in Fig. 1. This circuit is constructed with k gates and n inputs x_1,x_2,...,x_n and feedback paths from the combinational circuit to the registers. The transition of the output signal at the ith gate is expressed as

$$\Delta f_{(i)} = f_{(i)}(x_1 \oplus \Delta x_1, \cdots, x_n \oplus \Delta x_n) \oplus f_{(i)}(x_1, \cdots, x_n), \tag{2}$$

where Δx is a transition of the input signal and f_i is a Boolean function at the output of the ith gate. In the following section, we define the leakage model by considering the bias of the probability of $\Delta f_{(i)} = 1$ in cases where $\alpha = 0$ or $\alpha = 1$, with α being the value of the signal used by the attacker for grouping. We will refer to this signal a *selection bit*.

2.2 Static Leakage Model

We assume that x_1,x_2,...,x_n in Fig. 1 are independent variables [1]. In the static model, the expectation of the transition frequency in one clock cycle is expressed

[1] These are not strictly independent, but any variation from independence is negligible when the bias of the transition probability for each gate is discussed in a cryptographic circuit.

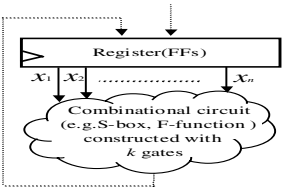


Fig. 1. General combinational circuit

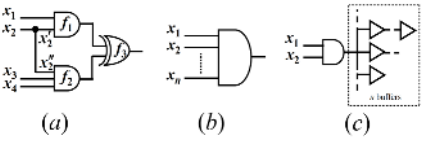


Fig. 2. Sample circuits (*a*) AND-XOR, (*b*) *n*-AND, and (*c*) 2-AND with *n*-buffer

as

$$N_{\alpha}^{\mathrm{stc}} = \sum_{i=1}^{k} p_{\alpha,(i)}^{\mathrm{stc}}, \tag{3}$$

where $p_{\alpha,(i)}^{\mathrm{stc}}$ is the transition probability at the output of the ith gate corresponding to the value of the selection bit α.

Definition 1. (Static Leakage). *Static leakage* $N_{\mathrm{diff}}^{\mathrm{stc}}$ *in the combinational circuit is*

$$N_{\mathrm{diff}}^{\mathrm{stc}} = N_{\alpha=1}^{\mathrm{stc}} - N_{\alpha=0}^{\mathrm{stc}} = \sum_{i=1}^{k} (p_{\alpha=1,(i)}^{\mathrm{stc}} - p_{\alpha=0,(i)}^{\mathrm{stc}}), \tag{4}$$

where $p_{x,(i)}^{\mathrm{stc}}$ *is the transition probability of* $\Delta f_{(i)} = 1$ *under the condition that* $\Delta x_1, ..., \Delta x_n$ *are n independent variables.*

If $N_{\mathrm{diff}}^{\mathrm{stc}} \neq 0$, it is possible that the correlation peak is observed in DPA measurements from Eq. (1). Generally, a normal nonlinear logic using a CMOS standard cell library has $N_{\mathrm{diff}}^{\mathrm{stc}} \neq 0$. Some examples are provided below.

Example 1: AND-XOR. We consider the static leakage in Fig. 2 (*a*) with random inputs. If the selection bit is x_1, we get

$$\begin{aligned}
\Delta f_{(1)} &= x_1 \cdot \Delta x_2 \oplus x_2 \cdot \Delta x_1 \oplus \Delta x_1 \cdot \Delta x_2, \\
\Delta f_{(2)} &= x_2 \cdot x_3 \cdot \Delta x_4 \oplus x_3 \cdot x_4 \cdot \Delta x_2 \oplus x_4 \cdot x_2 \cdot \Delta x_3 \oplus x_2 \cdot \Delta x_3 \cdot \Delta x_4 \\
&\quad \oplus x_3 \cdot \Delta x_4 \cdot \Delta x_2 \oplus x_4 \cdot \Delta x_2 \cdot \Delta x_3 \oplus \Delta x_2 \cdot \Delta x_3 \cdot \Delta x_4, \\
\Delta f_{(3)} &= \Delta f_{(1)} \oplus \Delta f_{(2)}.
\end{aligned} \tag{5}$$

Namely,

$$\begin{aligned}
\Delta f_{x_1=1,(1)} &= \Delta x_2 \oplus x_2 \cdot \Delta x_1 \oplus \Delta x_1 \cdot \Delta x_2, \\
\Delta f_{x_1=0,(1)} &= x_2 \cdot \Delta x_1 \oplus \Delta x_1 \cdot \Delta x_2.
\end{aligned}$$

$x_i = 1$ and $\Delta x_i = 1$ occur with a probability 1/2. Here, the input states $(x_2, \Delta x_1, \Delta x_2)$ that assume $\Delta f_{x_1=1,(1)} = 1$ are (0,0,1), (1,0,1), (1,1,0), and (1,1,1). Hence, we have

$$p^{\mathrm{stc}}_{x_1=1,(1)} = 1/2,\ p^{\mathrm{stc}}_{x_1=0,(1)} = 1/4.$$

Similarly,

$$p^{\mathrm{stc}}_{x_1=1,(2)} = 7/32,\ p^{\mathrm{stc}}_{x_1=0,(2)} = 7/32,\ p^{\mathrm{stc}}_{x_1=1,(3)} = 7/16,\ p^{\mathrm{stc}}_{x_1=0,(3)} = 5/16.$$

Thus, the static leakage in Fig.2 (a) is

$$N^{\mathrm{stc}}_{\mathrm{diff}} = 3/8.$$

The fact that AND-XOR is a basic element for S-boxes implies that a normal implementation of a block cipher necessarily has static leakage.

Example 2: n-AND. Under a condition similar to that in Example 1, the static leakage of n-input AND gates shown in Fig. 2 (b) is

$$N^{\mathrm{stc}}_{\mathrm{diff}} = (2^{n-1} - 1)/2^{2n-2},$$

where the selection bit $\alpha \in \{x_1, \cdots, x_n\}$.

Example 3: Buffer Tree. The static leakage of two-input AND gates connected to n buffers (Fig. 2 (c)) is

$$N^{\mathrm{stc}}_{\mathrm{diff}} = \frac{1}{4} \cdot n,$$

where the selection bit is x_1 or x_2. Stated simply, the static leakage at the gate with a large fan-out is amplified.

Based on Definition 1, the static leakage has the property described in the following section.

Property 1: Consecutive Static Leakage. *An equal amount of static leakage occurs both in the cycle when the selection bit appears and in the next cycle.*

Based on Eq. (2), it is evident that the transitions related to the selection bit occur in the cycle when the selection bit appears and in the next cycle as well. In cryptographic circuits, Δx_1,...,Δx_n are generally independent random variables. Thus, two static leakages of equal amounts occur for two consecutive cycles because two biased state transitions occur (random state → state dependent on α, state dependent on α → random state). This implies that two similar DPA peaks are observed for two consecutive clock cycles in the DPA measurements if the target device is ideal.

2.3 Dynamic Leakage Model

In an actual cryptographic circuit, the delay time depends on the signal route. In addition, each route tends to be non-uniform. Such non-uniformity is particularly remarkable in the circuits designed with automatic synthesis/layout.

As in Section 2.2, we consider the transition probability in Fig. 1. We assume that the transitions $\Delta x_1, ..., \Delta x_n$ of the registers reach each gate at different times. Here, the transition of the Boolean function $\Delta f_{(i)}$ occurs only when transitions of the registers reach the ith gate. Based on these facts, we can evaluate the transition probability at a certain timing by supposing that only the transition corresponding to the timing is a variable and that others are 0. We define *Dynamic Leakage* using this probability.

Definition 2. (Dynamic Leakage). *Let Δt be a time interval that an attacker can observe. Dynamic leakage $N_{\mathrm{diff}}^{\mathrm{dyc}}$ in Δt on the combinational circuit is*

$$N_{\mathrm{diff}}^{\mathrm{dyc}} = N_{\alpha=1}^{\mathrm{dyc}} - N_{\alpha=0}^{\mathrm{dyc}} = \sum_{i=1}^{k} \sum_{e \in E(i)} (p_{\alpha=1,(i)}^{\mathrm{dyc}}(e) - p_{\alpha=0,(i)}^{\mathrm{dyc}}(e)), \tag{6}$$

where $E(i)$ is the set of events with the possibility that transition occurs in the state after α appeared at the ith gate in Δt, and $p_{\alpha,(i)}^{\mathrm{dyc}}(e)$ is the probability of $\Delta f_{(i)} = 1$ under the condition that the transition of the input signal corresponding to e is a variable and the others are 0.

Here, we consider the relation between the transitions of the registers Δx_1,..., Δx_n and the event $e \in E(i)$ that depends on the selection bit α. If the circuit has not been redundantly constructed and $\Delta t \geq 2$ cycles, $E(i)$ contains at least n events corresponding to the transitions of the registers in the state wherein α appeared. This does not depend on the order of the signal transitions. It should be noted that these events are distributed between two cycles according to the delay time, which was fixed when the circuit was constructed, for each signal to propagate. Additionally, it is possible for two or more transitions to occur by the same transition of the register if the propagation route is different. In this case, the transitions corresponding to each route are treated as independent variables in Eq. (2). In the following section, we evaluate the dynamic leakage in Fig. 2.

Example 4: AND-XOR. We consider the circuit, shown in Fig. 2(a), on the dynamic model. If $\Delta t \geq 2$ cycles, we get

$$E(1) = \{e(\Delta x_1), e(\Delta x_2^{'})\}, \quad E(2) = \{e(\Delta x_2^{''}), e(\Delta x_3), e(\Delta x_4)\},$$

$$E(3) = \{e(\Delta x_1), e(\Delta x_2^{'}), e(\Delta x_2^{''}), e(\Delta x_3), e(\Delta x_4)\}.$$

Based on Eq.(6), Δf_3 at each event is

$$\Delta f_{(3)}(e(\Delta x_1)) = x_2 \cdot \Delta x_1, \quad \Delta f_{(3)}(e(\Delta x_2^{'})) = x_1 \cdot \Delta x_2^{'}, \quad \Delta f_{(3)}(e(\Delta x_2^{''})) = x_3 \cdot x_4 \cdot \Delta x_2^{''},$$

$$\Delta f_{(3)}(e(\Delta x_3)) = x_4 \cdot x_2 \cdot \Delta x_3, \quad \Delta f_{(3)}(e(\Delta x_4)) = x_2 \cdot x_4 \cdot \Delta x_3.$$

If x_1 is the selection bit, we have

$$p_{x_1=1,(3)}^{\mathrm{dyc}}(e(\Delta x_2^{'})) = 1/2, \quad p_{x_1=0,(3)}^{\mathrm{dyc}}(e(\Delta x_2^{'})) = 0.$$

Similarly, in $\Delta f_{(1)}$, we have

$$p^{\mathrm{dyc}}_{x_1=1,(1)}(e(\Delta x_2^{'})) = 1/2, \quad p^{\mathrm{dyc}}_{x_1=0,(1)}(e(\Delta x_2^{'})) = 0.$$

The dynamic leakage of Fig. 2(a) is $N^{\mathrm{dyc}}_{\mathrm{diff}} = 1$.

It should be noted that the difference between x_1 and $x_2^{'}$ at the delay time determines the timing whereby dynamic leakage occurs in the circuit. $N^{\mathrm{dyc}}_{\mathrm{diff}}$ occurs during the cycle when the predicted x_1 appears if $x_2^{'}$ is slower than x_1, and it occurs during the next cycle if the delay condition is converse.

Example 5: n-AND. Under a condition similar to that in Example 4, the dynamic leakage, shown in Fig. 2 (b), is

$$N^{\mathrm{dyc}}_{\mathrm{diff}} = (n-1)/2^{n-1},$$

where $x \in \{\ x_1\ ,...,\ x_n\ \}$.

Finally, we describe a property common to static and dynamic leakage.

Property 2. (Complementary Leakage from AND- and OR-Gate). *The static/dynamic leakages of an equal amount but of opposite polarity occur from thw AND- and OR-gate(or, the NAND- and NOR-gate) respectively, under the same input and delay time condition.*

This implies that there is the possibility that the leakage of the entire circuit is counterbalanced. In actuality, a countermeasure using this property has been proposed [17].

3 Enhanced Leakage Models

Thus far, some analysis technics that enhance standard DPA have been proposed. Here, we define the leakage model corresponding to these enhanced DPA and consider the effectiveness of each technic from the viewpoint of our model. In particular, we focus on Messerges's second-order DPA (M-2DPA) [12] and Waddle's second-order DPA (W-2DPA) [13] which are basically enhanced versions of DPA.

3.1 Standard Second-Order Attack

In standard DPA, the attacker analyzes power traces according to the average power difference at a specific time. On the other hand, in M-2DPA, the attacker analyzes power traces between two points. First, we define the leakage model of M-2DPA based on the signal transition in the following section.

Definition 3. (Leakage by Messerges's Second-Order DPA). *Let $N(t)$ be an expectation of the transition frequency at time t in the combinational circuit. Leakage by Messerges's second-order DPA $N_{\mathrm{diff}}^{\mathrm{2nd}}$ is*

$$N_{\mathrm{diff}}^{\mathrm{2nd}} = (N_{\alpha=1}(t') - N_{\alpha=1}(t)) - (N_{\alpha=0}(t') - N_{\alpha=0}(t)). \tag{7}$$

M-2DPA is an evaluation method that analyzes the correlation of the signal transition of two points. This implies that the correlation of the power consumption of two specific circuit components is evaluated. Moreover, this also implies that the correlation between cycles in the same circuit is evaluated if the combinational circuit is constructed with the loop architecture.

Next, we consider the condition to be secure against M-2DPA in CMOS logic circuits considering this leakage model. If a certain circuit is secure against standard DPA, the secure condition $N_{\mathrm{diff}} = 0$ is satisfied at any time in Eqs. (5) and (8). In this case, we have $N_{\alpha=1}(t) = N_{\alpha=0}(t)$ and $N_{\alpha=1}(t') = N_{\alpha=0}(t')$. Thus, this circuit obviously satisfies $N_{\mathrm{diff}}^{\mathrm{2nd}} = 0$. This implies that if the CMOS logic circuit is secure against standard DPA, it is also secure against M-2DPA. On the other hand, if a certain circuit is insecure against the standard DPA, we have $N_{\alpha=1}(t) - N_{\alpha=0}(t) = k_t (\neq 0)$ and $N_{\alpha=1}(t') - N_{\alpha=0}(t') = k_{t'}$ at any two points t and t', where k_t and $k_{t'}$ are the leakages against the standard DPA at each time. In this case, if $k_{t'} = k_t \neq 0$ is satisfied at any point, this circuit satisfies $N_{\mathrm{diff}}^{\mathrm{2nd}} = 0$. However, the circuit wherein equal leakage occurs at any point of time is not realistic. Namely, if the circuit is insecure against standard DPA, it is also insecure against M-2DPA in real circuits.

Taking the abovementioned facts into consideration using our models, we arrive at the following conclusion:

- Messerges's second-order DPA is an attack that is essentially equivalent to the standard DPA in CMOS logic circuits.

M-2DPA is useful only when the spike is made easily visible, the DPA trace with intuitive understanding is obtained, or the number of samples is decreased. In the construction of the hardware countermeasure, we have to consider only the the standard DPA.

3.2 Attack by Squaring Power Traces

Zero-Offset 2DPA, which was proposed by Waddle et al., is an analysis technic, which is characterized by the use of squaring power traces [13]. We will refer to this technic as the W-2DPA. In this section, we consider the effectiveness of W-2DPA from the viewpoint of our model.

Definition 4. (Leakage by Waddle's Second-Order DPA). *Let $S(t)$ be the set of transition frequencies with a possibility to occur at time t in the combinational circuit. Let $p_s(t)$ be the probability that the transition occurs in s gates at time t. The leakage by Waddle's Second-Order DPA V_{diff} is*

$$V(t) = \sum_{s \in S(t)} (s^2 \cdot p_s(t)), \tag{8}$$

$$V_{\mathrm{diff}} = V_{\alpha=1}(t) - V_{\alpha=0}(t). \tag{9}$$

Here, we compare the secure condition of W-2DPA and standard DPA. Based on Definition 4, it is necessary to satisfy the following equation for $V_{\mathrm{diff}} = 0$.

$$\sum_{s \in S(t)} (s^2 \cdot p_{\alpha=1,s}(t)) = \sum_{s \in S(t)} (s^2 \cdot p_{\alpha=0,s}(t)) \tag{10}$$

In standard DPA, on the other hand, if $\sum(s \cdot p_{\alpha=1,s}(t))$ is equal to $\sum(s \cdot p_{\alpha=0,s}(t))$, $N_{\mathrm{diff}} = 0$ is satisfied. Thus, each secure condition is obviously different. This consideration suggests the following conclusion.

– Waddle's second-order DPA can detect the bias of the distribution of the transition probability in CMOS logic circuits.

W-2DPA is an analysis technic that is essentially different from standard DPA, and we must consider this technic in the construction of the hardware countermeasure. Actually, masked CMOS logics are weak against W-2DPA, even if the static model is assumed. These results are described in Section 4.

Next, we enhance Definition 4 more effectively. Generally, it is difficult to compute $p_s(t)$ in the entire circuit when the dynamic model is assumed. Therefore, we enhance Definition 4 such that it is applicable to the actual device for simulating power analysis.

Definition 5. (Enhanced Leakage by Waddle's Second-Order DPA). *Let Δt be a time interval that an attacker can observe. Let* TC *be the transition count to occur in Δt for the combinational circuit. Let* Nm *be the number of observed samples corresponding to α. Leakage by Waddle's Second-Order DPA V'_{diff} is*

$$V'_{\mathrm{diff}} = \left(\frac{\sum \mathrm{TC}^2}{\mathrm{Nm}}\right)_{\alpha=1} - \left(\frac{\sum \mathrm{TC}^2}{\mathrm{Nm}}\right)_{\alpha=0} \tag{11}$$

Our models which are based on the transition probability can evaluate the security strength of the circuit in advance by extracting the transition of each gate from the netlist [2] for the logic simulation. In Ref. [14], we point out that the evaluation results of logic simulation using our models are very similar to the actual circuits without countermeasures.

4 Evaluation for Previously Known Countermeasures

There are two approaches to the construction of countermeasures at the circuit level. The first approach uses complementary behavior and makes power consumption independent of data. The second uses data masking in combinational circuits and renders intermediate data unpredictable. In this section, we review a typical example based on each approach and evaluate each countermeasure by using our leakage models.

[2] A netlist is a text description of the circuit connectivity. It is basically a list of connectors, a list of instances (gates). In addition, the netlist can contain delay information.

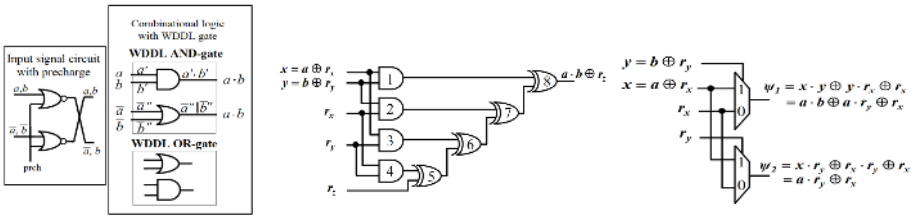


Fig. 3. Components of previously known countermeasures (*a*) WDDL, (*b*) Masked-AND, (*c*) MAND

4.1 Previously Known Countermeasures on the CMOS Circuit

Wave Dynamic Differential Logic. Tiri et al. proposed Wave Dynamic Differential Logic (WDDL) [6] which is based on dynamic and differential logic and constructed with CMOS standard cell libraries. Figure 3 (*a*) shows the basic components of WDDL. As the first step, WDDL executes a precharge at the beginning of the combinational logic. It only contains three logic gates, *i.e.*, AND, OR, and NOT. In addition, they proposed a method for the implementation of WDDL using FPGA.

Masked-AND Operation. Figure 3 (*b*) shows the Masked-AND operation as proposed by Trichina [7]. Masked-AND is a method of calculating "$(a \cdot b) \oplus r_z$" using the above 5 input data, x, y, r_x, r_y and r_z. Hence, the computations, as shown in Fig. 3 (*b*), can be performed without compromising on the bits of actual data. In addition, Blömer et al. insist that this approach is "Provably Secure" against DPA in Ref. [19].

MAND. Figure 3 (*c*) describes MAND proposed by Shimizu [18]. It is based on data masking and is characterized by the use of a dual-rail circuit.

4.2 Analysis for Complementary Logics

Based on Property 2, a complementary logic has the possibility of counterbalancing the leakage. WDDL is a method that refines this consideration. We consider the circuit shown in Fig. 3 (*a*). At the end of the precharge phase (prch = 0), all output signals of the WDDL gates are at 0. Thus, the transitions for each gate in the evaluation phase (prch = 1) are equal to the wire value. In the case of the WDDL-AND gate,

$$\Delta f_{(AND)} = a \cdot b \quad \Delta f_{(OR)} = \bar{a} \mid \bar{b} = a \cdot b \oplus 1$$

Based on this, we obtain $N_{\mathrm{diff}}^{\mathrm{stc}} = 0$ because a transition occurs only at either one of the gates without any relation to the value of a or b.

Next, we consider the dynamic model. Here, we assume that the transition Δb arrives at the gates later than transition Δa. Table 1 shows the transition probability of each CMOS gate corresponding to each selection bit in this condition.

On the basis of the data listed in Table 1, if Δt in Definition 2 is long enough and both $e(\Delta a)$ and $e(\Delta b)$ occur for that time, $N_{\mathrm{diff}}^{\mathrm{dyc}} = 0$ is satisfied without any relation to the selection bit. However, if the attacker can observe power traces in Δt that contains $e(\Delta a)$, and not contain $e(\Delta b)$ [3], he can detect the bias of the transition frequency. It should be noted that it is difficult to observe the opposite case(only containing $e(\Delta b)$) because of some capacitance in the actual device. In the example of Table 1, the dynamic leakage of the WDDL-AND gate is $N_{\mathrm{diff}}^{\mathrm{dyc}} = -1$ in evaluation phase when a is the selection bit. Although it is necessary to consider all arrival sequences of input signals when conducting a more detailed evaluation, we omit the details here. In the precharge phase, the dynamic leakage of the WDDL-AND gate is $N_{\mathrm{diff}}^{\mathrm{dyc}} = 1$ for Table 1 when b is the selection bit and Δt contains $e(\Delta a)$, and not contain $e(\Delta b)$. It is noteworthy that the polarity of the leakage changes in the evaluation and precharge phases. In addition, the reason for the leakage of W-2DPA is similar to that mentioned above.

A similar observation applies to other countermeasures using complementary logic. On the basis of this consideration, the secure condition concerning complementary gates (logic) against DPA and W-2DPA is as follows:

– All input signals reach each complementary gate (logic) simultaneously.

Generally, it is difficult to implement this condition via circuits. In particular, it is not guaranteed in the LSI designed by the automatic synthesis/layout.

4.3 Analysis for Masked CMOS Logic

Generally, in order to extract the absolutely necessary results (e.g., $a \cdot b$) from the masked operation results (e.g., $(a \oplus r_x) \cdot (b \oplus r_y)$), the DPA countermeasures based on data masking need to operate some unnecessary terms (e.g., $(a \oplus r_x) \cdot r_y$). Several methods have been proposed regarding the manner in which the operations should be divided. In this section, we will consider and evaluate Masked-AND [7] and MAND [18].

Masked-AND. When evaluating the Masked-AND circuit with Definition 1, it satisfies $N_{\mathrm{diff}}^{\mathrm{stc}} = 0$ because the wire value of each gate is randomized as mentioned in Ref. [19]. On the other hand, when evaluating the circuit with Definition 2, there is a possibility that $N_{\mathrm{diff}}^{\mathrm{dyc}} \neq 0$. In Ref. [20], we analyze the abovementioned facts in more detail.

For this example, the output of gate 6 in Fig. 3 (b) is expressed as $x \cdot r_y \oplus r_x \cdot r_y \oplus z$. Here, if we assume that the signal transition occurs in order of x, r_x and r_y,

[3] This implies that the attacker observes by using a higher sampling rate for the oscilloscope.

the transition of gate 6 caused by $e(\Delta r_y)$ can be expressed as $x \cdot \Delta r_y \oplus r_x \cdot \Delta r_y = a \cdot \Delta r_y$. It follows that $p^{\mathrm{dyc}}_{a=1,(6)}(e(\Delta r_y)) = 1/2$, $p^{\mathrm{dyc}}_{a=0,(6)}(e(\Delta r_y)) = 0$. Thus, the leakage from gate 6 brought about by $e(\Delta r_y)$ is $N^{\mathrm{dyc}}_{\mathrm{diff}} = 1/2$. Furthermore, since an XOR-gate propagates transitions of its input signal, gate 7 whose input is the output of gate 6 causes the same leakage as that of gate 6. The same is also the case with gate 8.

As mentioned above, it can be stated that the Masked-AND circuit may have the bias of signal transition according to secret information when a certain delay condition is met. The leakage in the dynamic model of Masked-AND is also analyzed in Ref. [21], where a similar result is obtained.

Next, we discuss evaluating the Masked-AND circuit with Definition 5. For simplicity, we consider only four AND-gates in Fig. 3 (*b*) here. Table 2 lists the transition counts and their event probability when the selection bit is a, where $s \in \{0, 1, 2, 3, 4\}$ is the total transition count of these AND-gates.

The following can be qualitatively inferred from Table 1: If both a and b are 0, the four AND-gates from gate 1 to gate 4 in Fig. 3 (*b*), execute the same logical operation and often exhibit similar behavior. On the other hand, they often behave differently if a or b is not 0. Actually, the event probability of $s = 4$ is $p_4 = 7/64$ when $a = 0$, while it is $p_4 = 1/32$ when $a = 1$. In a similar manner, the event probability differs depending on a predictable signal value. Quantitatively, from Table 1 and Definition 5, $V'_{\mathrm{diff}} = -5/8$ when the selection bit is a or b. Therefore, the Masked-AND circuit can be attacked with W-2DPA.

MAND. The same observation applies to MAND. Here, we describe the case that is not secure against W-DPA even if it is secure against standard DPA. We focus on the delay relation between the MUX data signals and the MUX select signals in MAND, and consider the leakage separately in the following two delay conditions:

- Condition 1 : "$\mathrm{delay}(y), \mathrm{delay}(r_y) < \mathrm{delay}(x), \mathrm{delay}(r_x)$"
  (or "$\mathrm{delay}(x), \mathrm{delay}(r_x) < \mathrm{delay}(y), \mathrm{delay}(r_y)$")

- Condition 2 : "$\mathrm{delay}(x) < \mathrm{delay}(y) < \mathrm{delay}(r_x)$" and
  "$\mathrm{delay}(x) < \mathrm{delay}(r_y) < \mathrm{delay}(r_x)$"
  (or "$\mathrm{delay}(r_x) < \mathrm{delay}(y) < \mathrm{delay}(x)$" and
  "$\mathrm{delay}(r_x) < \mathrm{delay}(r_y) < \mathrm{delay}(x)$")

Condition 1 states that transitions according to x and r_x occur at the events of the select signals. In addition, the condition in parentheses is a similar one, excluding the cycle when the transition occurs. Condition 2 states that transitions according to either x or r_x occur at the events of the select signals.

First, we evaluate the leakage in Condition 1. The transitions of ψ_1 and ψ_2 at the events of the select signals are

$$\Delta\psi_1(e(\Delta y)) = x \cdot \Delta y \oplus r_x \cdot \Delta y = a \cdot \Delta y,$$
$$\Delta\psi_2(e(\Delta r_y)) = x \cdot \Delta r_y \oplus r_x \cdot \Delta r_y = a \cdot \Delta r_y.$$

Namely, if the selection bit is a, we have

$$p^{\mathrm{dyc}}_{a=1,(\psi_1)}(e(\Delta y)) = 1/2,\ p^{\mathrm{dyc}}_{a=0,(\psi_1)}(e(\Delta y)) = 0,$$
$$p^{\mathrm{dyc}}_{a=1,(\psi_2)}(e(\Delta y)) = 1/2,\ p^{\mathrm{dyc}}_{a=0,(\psi_2)}(e(\Delta y)) = 0.$$

Thus, we evaluate the dynamic leakage of MAND as $N^{\mathrm{dyc}}_{\mathrm{diff}} = 1$. On the other hand, if the selection bit is b, it is evident that $N^{\mathrm{dyc}}_{\mathrm{diff}} = 0$.

Next, the transitions of ψ_1 and ψ_2 at the events of the select signals in Condition 2 are

$$\Delta\psi_1(e(\Delta y)) = x \cdot \Delta y \oplus r'_x,$$
$$\Delta\psi_2(e(\Delta r_y)) = x \cdot \Delta r_y \oplus r'_x,$$

where r'_x is the wire value of r_x at the previous state one cycle. In this case, it is evident that $N^{\mathrm{dyc}}_{\mathrm{diff}} = 0$ with any selection bit because random numbers are not canceled. Thus, MAND is secure against standard DPA under Condition 2.

Finally, we evaluate the leakage against W-2DPA. We show the probability distribution of MAND in the static model in Table 3. On the basis of this table, we have $V'_{\mathrm{diff}} = -1/4$. Thus, MAND is insecure against W-2DPA even if the static model is assumed.

Additionally, we consider the leakage against W-2DPA in Condition 2, which is secure against standard DPA. Although the data signal transitions influence both ψ_1 and ψ_2, the transitions of select signals influence only either one of the two. Since W-2DPA is an attack that paid attention to the distribution of the transition probability at the entire circuit, we consider the influence of the data signal transition here. The transitions of ψ_1 and ψ_2 at the events of the data signals in Condition 2 are $\Delta\psi_1(e(\Delta r_x)) = \Delta r_x \cdot y \oplus \Delta r_x, \Delta\psi_2(e(\Delta r_x)) = \Delta r_x \cdot r_y \oplus \Delta r_x$. Thus, the transition count is $s \in \{0, 2\}$ at $(e(\Delta r_x))$, assuming $b = 0$. Furthermore, each event probability is $p_0 = 3/4$ and $p_2 = 1/4$. On the other hand, the transition count is $s \in \{0, 1\}$ at $(e(\Delta r_x))$ assuming $b = 1$ and each event probability is $p_0 = 1/2$ and $p_2 = 1/2$. Therefore, since we have $V'_{\mathrm{diff}} = -1/2$, Condition 2 is insecure against W-2DPA even if it is secure against the standard DPA.

5 Experimental Results and Considerations

We evaluate the effectiveness of the previously known countermeasures by using FPGA. In this section, we show experimental results of elementary bricks of previously known countermeasures implemented on FPGA. The evaluation environment is the general one shown in Table 4. An XCV1000-6-BG560C FPGA of Xilinx Inc. is mounted on the target board. Additionally, automatic place-and-route tools were used for all layout design.

5.1 Standard DPA

Figure. 4 shows the experimental results of standard DPA for each countermeasure (i.e., normal-AND, WDDL, Masked-AND, and MAND).

Table 1. Transition probability of the WDDL-AND gate

α	CMOS gate	prch = 1		prch = 0	
		$e(\Delta a)$	$e(\Delta b)$	$e(\Delta a)$	$e(\Delta b)$
$a = 1$	AND	0	1/2	1/2	0
	OR	0	1/2	0	1/2
$a = 0$	AND	0	0	0	0
	OR	1	0	1/2	1/2
$b = 1$	AND	0	1/2	1/2	0
	OR	1/2	0	1/2	0
$b = 0$	AND	0	0	0	0
	OR	1/2	1/2	0	1

Table 2. Probability distribution of Masked-AND

Selection bit α	Transition count s	Event probability p_s
$a = 1$	0	5/32
	1	3/8
	2	5/16
	3	1/8
	4	1/32
$a = 0$	0	19/64
	1	3/16
	2	11/32
	3	1/16
	4	7/64

Table 3. Probability distribution of MAND

Selection bit α	Transition Count s	Event probability p_s
$a = 1$	0	1/4
	1	1/2
	2	1/4
$a = 0$	0	3/8
	1	1/4
	2	3/8

Table 4. Evaluation environment

Design environment	
Language	Verilog-HDL
Simulator	Verilog-XL
Logic synthesis	Synplify version 7.7
Place and Route	ISE version 6.3.03i
Measurement environment	
Target FPGA	XCV1000-6-BG560C
Oscilloscope	Tektronix TDS 7104

In Fig. 4, the average power enlarges at time t_4 and time t_5 when the WDDL circuit is activated. The first half (t_4) is an evaluation phase, and the latter half (t_5) is a precharge phase of WDDL. Figure. 5 is a magnified view of the WDDL part in Fig. 4. There appears a small downward peak at time t_4 and a small upward peak at time t_5, each of which are caused by timing differences between the input signal a and input signal b because automatic place-and-route tools were used. These peaks are in good agreement with the forecast by the evaluation based on our leakage model.

The Masked-AND circuit is activated at time t_7 where an upward peak (as shown in the considerations in Section 4.3) is observed. Since this peak is caused by transient hazards, it is relatively small as compared to that of the normal-AND.

At time t_9, the MAND circuit is activated and an upward peak appears. As mentioned above, automatic place-and-route tools were used; thus, Condition 1 and Condition 2 shown in Section 4 are mixed. Therefore, leakage from Condition 1 can be observed. Here too, as in the Masked-AND case, the peak is relatively small as compared to that of the normal-AND because it is caused by transient hazards.

In Fig. 6, the same MAND circuit as that in Fig. 4 is activated at time t_1. As evident from standard DPA traces in Fig. 6 at time t_1, leakage is observed when the selection bit is a, but it is not observed when the selection bit is b. The MAND circuit that satisfies Condition 2 is activated between time t_3 and time t_8 The MAND circuit that satisfies Condition 2 [4]. In this case, it is evident that leakage is not observed by standard DPA even if the selection bit is a. These results are in good agreement with the forecast in Section 4.

[4] The condition is created by supplying input signals one by one for every clock cycle.

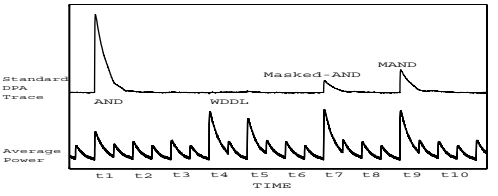


Fig. 4. Standard DPA result (200000 sample)

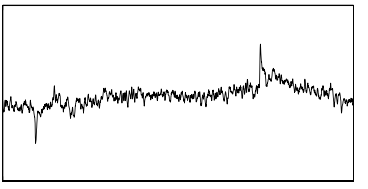


Fig. 5. Magnified view of the WDDL part in Fig.4

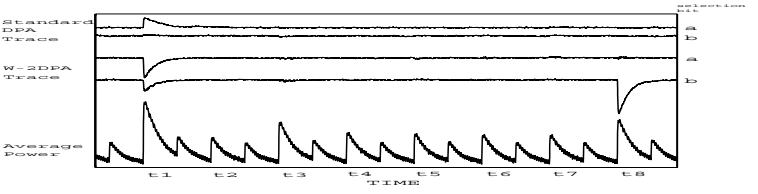


Fig. 6. Standard DPA and W-2DPA results for MAND (10000 sample)

5.2 W-2DPA

Fig. 7 shows the experimental results of W-2DPA for each countermeasure (i.e., normal-AND, WDDL, Masked-AND, and MAND). The sample data used for the analysis is the same as the one used for standard DPA (Fig. 4). Fig. 8 is a magnified view of the WDDL part in Fig. 7. Peaks in Fig. 5 and Fig. 8 look similar, as indicated in the considerations in the previous section.

It should be noted that the peaks of Masked-AND and MAND are static in this case; hence, they are as large as that of normal-AND. In the case of starndard DPA, peaks of Masked-AND and MAND are caused by transient hazards; hence, they are not so large. In Fig. 6, W-2DPA traces between time t_3 and time t_8 show the experimental results of W-2DPA for the MAND circuit that satisfies Condition 2. While standard DPA traces show no peaks at time t_8, W-2DPA traces show a downward peak if the selection bit is b. This too is in good agreement with the considerations based on our leakage model.

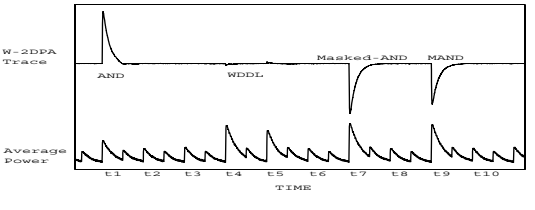


Fig. 7. W-2DPA result (200000 sample)

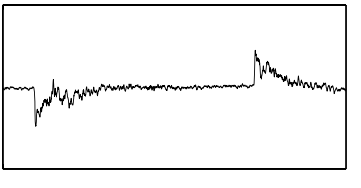


Fig. 8. Magnified view of the WDDL part in Fig.7

6 Toward a Perfect Countermeasure

Secure conditions of CMOS logic circuits against standard DPA and W-2DPA are $N_{\mathrm{diff}} = 0$ and $V'_{\mathrm{diff}} = 0$ without dependence on any selection bits.

The approach by complementary logics is very effective although the problem of the signal delay persists. Actually, the leakage of WDDL is the least in our experimental results. We predict that a manual layout strengthens WDDL.

The approach by data masking requires both main operation (e.g., $x \cdot y$) and cancel operation (e.g., $x \cdot r_y$, $y \cdot r_x$, and $r_x \cdot r_y$). When these operations are separately implemented by the CMOS logic gate, the probability distribution of the transition count in the entire circuit is different depending on sensitive information (see Section 5). A consideration of both the static model and dynamic model reveals that this fact occurs. Therefore, we suppose that it is difficult to resist various power analysis by the approach of data masking in a general CMOS gate. The solution to this is to construct a special CMOS gate, which is improved at the transistor level and satisfies secure condition. For further details of a countermeasure based on this consideration, see Ref. [11].

7 Conclusion

In this paper, we proposed leakage models of the CMOS logic circuits based on signal transition. These models are naturally applicable to various actual devices for simulating power analysis.

In addtiton, we evaluated the effectiveness of Messerges's second-order DPA (M-2DPA) and Waddle's second-order DPA (W-2DPA) from the viewpoint of our model. Thus, we demonstrated that M-2DPA is essentially equivalent to the standard DPA, and W-2DPA can detect the bias of the distribution of the transition probability in CMOS logic circuits.

Moreover, we analyzed previously known countermeasures by both our models and FPGA, and confirmed that the DPA traces on FPGA corresponded to the result obtained using our models. We emphasize the occurrence of the leakage in the previously known countermeasures. In particular, we pointed out that the masked CMOS logics have the similar weakness to standard CMOS logic without countermeasure against W-2DPA because the distribution of the transition probability are statically different.

References

1. P. Kocher, J. Jaffe and B. Jun, "Differential Power Analysis," *Crypto'99*, LNCS 1666, pp. 388-397, Springer-Verlag, 1999.
2. J.-S. Coron, "Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems," *CHES'99*, LNCS 1717, Springer-Verlag, pp. 292-302, 1999.
3. M. Akkar and C. Giraud, "An Implementation of DES and AES, Secure against Some Attacks," *CHES 2001*, LNCS 2162, pp. 309-318, Springer-Verlag, 2001.

4. K. Tiri, M. Akmal and I. Verbauwhede, "A Dynamic and Differential CMOS Logic with Signal Independent Power Consumption to Withstand Differential Power Analysis on SmartCards," Proc. of 28th European Solid-State Circuits Conference, pp.403-406, 2002.
5. K. Tiri and I. Verbauwhede, "Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology," *CHES 2003*, LNCS 2779, p.125-136, Springer-Verlag, 2003.
6. K. Tiri and I. Verbauwhede, "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation," In Proc. of Design Automation and Test in Europe Conference, pp. 246-251, 2004.
7. E. Trichina, "Combinational Logic Design for AES SubByte Transformation on Masked Data," Cryptology ePrint Archive, 2003/236, 2003.
8. S. Chari, C.S. Jutla, J.R. Rao and P. Rohatgi, "Towards Sound Approaches to Counteract Power Analysis Attacks," *Crypto'99*, LNCS 1666, pp. 398-412, Springer-Verlag, 1999.
9. C. Clavier, J.-S. Coron and N. Dabbous, "Differential Power Analysis in the Presence of Hardware Countermeasures," *CHES 2000*, LNCS 1965, pp. 252-263, Springer-Verlag, 2000.
10. R. Bevan and E. Knudsen, "Ways to Enhance Differential Power Analysis," *ICISC 2002*, LNCS 2587, pp. 327-342, Springer-Verlag, 2003.
11. D. Suzuki, M.Saeki and T.Ichikawa, "Random Switching Logic: A Countermeasure against DPA based on Transition Probability," Cryptology ePrint Archive, Report 2004/346, 2004.
12. T.S. Messerges, "Using Second-Order Power Analysis to Attack DPA Resistant Software," *CHES 2000*, LNCS 1965, pp. 238-251, Springer-Verlag, 2000.
13. J. Waddle and D. Wagner, "Towards Efficient Second-Order Power Analysis," *CHES 2004*, LNCS 3156, pp. 1-15, Springer-Verlag, 2004.
14. M. Saeki, D. Suzuki and T. Ichikawa, "Construction of DPA Leakage Model and Evaluation by Logic Simulation," ISEC2004-57, IEICE, July 2004 (in Japanese).
15. M. Akkar, R. Bevan, P. Dischamp and D. Moyart, "Power Analysis, What Is Now Posible...," *Asiacrypto 2000*, LNCS 1976, pp. 489-502, Springer-Verlag, 2000.
16. A.P. Chandrakasan, S. Sheng and R.W. Brodersen, "Low Power Digital CMOS Design," IEEE Journal of Solid State Circuits, Vol.27, N0.4. pp. 473-484, 1992.
17. Philips Electronics NV, "DATA CARRIER WITH OBSCURED POWER CONSUMPTION," Patent, WO00/026746.
18. H. Shimizu, "A Countermeasure against Side Channel Attack using Mask Logic Elements," ISEC2004-69, IEICE, September 2004 (in Japanese).
19. J. Blömer, J.G. Merchan and V. Krummel, "Provably Secure Masking of AES," Cryptology ePrint Archive, Report 2004/101, 2004.
20. T. Ichikawa, D. Suzuki and M. Saeki, "An Attack on Cryptographic Hardware Design with Masking Method," ISEC2004-58, IEICE, July 2004 (in Japanese).
21. S. Mangard, T. Popp, and B. M. Gammel, " Side-Channel Leakage of Masked CMOS Gates", *CT-RSA 2005*, LNCS 3376, pp. 361-365, Springer-Verlag, 2005

The "Backend Duplication" Method
A Leakage-Proof Place-and-Route Strategy for ASICs

Sylvain Guilley, Philippe Hoogvorst, Yves Mathieu, and Renaud Pacalet

GET/Télécom Paris, CNRS LTCI,
Département communication et électronique,
46 rue Barrault, 75634 Paris Cedex 13, France
{guilley, hoogvorst, mathieu, pacalet}@enst.fr

Abstract. Several types of logic gates suitable for leakage-proof computations have been put forward [1,2,3,4]. This paper describes a method, called "backend duplication" to assemble secured gates into leakage-proof cryptoprocessors. To the authors' knowledge, this article is the first CAD-oriented publication to address all the aspects involved in the backend design of secured hardware. The "backend duplication" method achieves the place-and-route of differential netlists. It allows for 100 % placement density and for balanced routing of dual-rail signals. Wires of every other metal layer are free to make turns. In addition, the method does not require any modification to the design rules passed to the router. The "backend duplication" method has been implemented in 0.13 μm ASIC technology and successfully tested on various ciphers. The example of the design of a DES module resistant against side-channel attacks is described into details.

Keywords: Information leakage, secured backend, differential signals.

1 Introduction: Using Differential Logic to Thwart SCA

It has been shown that sensitive information can be extracted from cryptographic hardware either by spying physical quantities or by injecting faults. The first type of attack is often referred to as "side-channel attack" (SCA [5,6,7]), whereas the second one is also known as "fault attack" (FA). Two classes of countermeasures against SCA have been put forward. The first idea is to shield the hardware at the algorithmic level: the data manipulated by the cryptoprocessor is masked or protected by secret-sharing methods. The second idea is to build the hardware using only leakage-proof gates, so as make sure that the overall cryptoprocessor is, in turn, leakage-proof.

This article focuses on the implementation of the latter class of countermeasures. Many leakage-proof logic styles have been published. The level of protection the secured gates provide depends upon their specification:

1. SABL [1] is a logic consuming a nearly constant current.
2. WDDL [2] uses dual gates pairs to ensure a constant activity, although the power consumed by each gate of the pair is not the same.

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 383–397, 2005.

3. Speed-independent (SI) logic presented in [3] features a consumption independent on the input data configuration. It also shields against the leakage of the signal transitions timing by synchronizing the inputs.
4. Refinements [4] of the previous solution also ensure that parasitic capacitances are unconditionally unloaded between two computations.

Some of those methods, for instance methods 3 and 4 above (nicknamed "SI-WDDL" in the rest of this article) can also embed an error-detection feature. The mechanism, based on an alarm propagation, is explained in [3]. Nevertheless, resistance to faults injection is not covered in this paper.

The *logical* part (coding, functionality verification, refinements for synthesis) in a design targeting FPGA or ASIC implementation is called *frontend*. The *physical* part (mainly consisting in place-and-route, but extensive description is provided in Sect. 2) is called *backend*. The common point to the secure gates listed above is the use of differential logic with a 4-phase protocol, such as "return to zero" (RTZ) or any variation [8]. It has already been stressed that the security of individual gates can extend to a netlist of gates only provided that the interconnect is kept differential [9]. Nonetheless, most articles evade the question of the implementation of a secure backend design.

Given the complexity of backend flows in sub-micron technologies, a simple way to realize the secure backend is necessary. We provide in this article a method, called "backend duplication", that integrates the secure place-and-route into any preexisting backend flow without modifying the design rules.

The rest of the article is organized as follows: the "backend duplication" is presented in Sect. 2. The method is applied to some secured gates primitives in Sect. 3. A case study, namely a DES cryptoprocessor, is provided in Sect. 4. This example was actually fabricated in HCMOS9GP 0.13 μm technology from STMicroelectronics using the method presented in this paper. This section contains an evaluation of the cost and of the security increase provided by the use of the "backend duplication". Finally, Sect. 5 concludes the article.

2 The "Backend Duplication" Method

2.1 Regular "Place-and-Route" ASIC Design Flow

In a standard cell flow, cryptographic functions are synthesized into a netlist of primitive gates. Then, the gates are placed into rows (see Fig. 1(a)). In each row, the gates are abutted, so that they share the ground (VSS) and the power (VDD) lines. When two gates are not placed side by side, a "filler" cell can be added in-between to ensure the continuity of the supply lines. In sub-micron technologies, there are enough levels of metal to allow the routing of the interconnect over the standard cell rows. For this reason, the rows are themselves abutted. Thus, the supply lines are shared between adjacent rows. This is achieved by flipping upside-down every other row: the ground (resp. the power) of one row is merged with the ground (resp. the power) of the lower (resp. the upper) row, (see Fig. 1(b)).

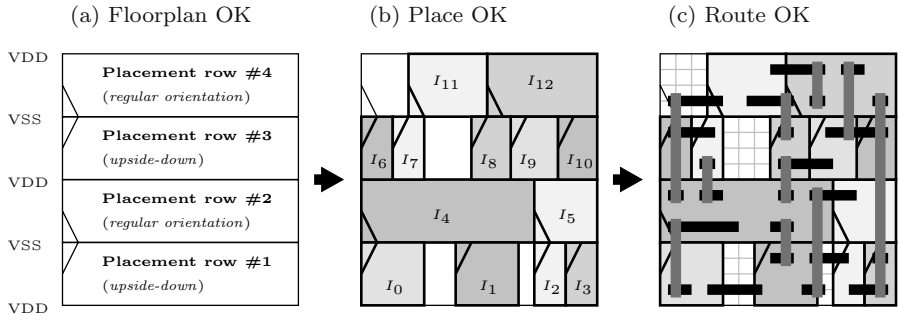


Fig. 1. Illustration of the regular (and unsecure) "Place-and-Route" ASIC design flow

Sub-micron technologies allow for 45 degree wire routes, but this feature is not yet implemented in commercial routers: currently, the routing is still Manhattan. Moreover, the most popular routers are also grid-based. Metal wires are only instantiated along a virtual routing tracks superimposed on the floorplan (see Fig. 1(c)). It is thus customary to attribute a preferred direction to every routing layer. However, routers consider the preferred direction only as a recommendation. The convention we use in this paper is that odd metal levels (metal 1, metal 3, and so forth) are preferentially routed vertically, whereas even metal levels are preferentially routed horizontally.

2.2 The "Backend Duplication" Method Overview

The "backend duplication" addresses the strengthening against SCA of sensitive ASICs (smartcards, hardwired cryptoprocessors, etc.) It consists in a single manipulation of the backend layout to ensure the security of its interconnect. However, this method shall not be confused with the tailored duplication method for software or dedicated hardware implementations [10].

The basic idea of the "backend duplication" method is to apply a regular backend flow on a single-ended (as opposed to duplicated) netlist, taking care to leave enough room on the floorplan for the duplication of the placed-and-routed netlist. The duplication basically demands that every other row be kept free, which is typically achieved by obstructing every other row for placement.

The next aspect concerned with duplication is the interconnect. To make it possible to duplicate the interconnect, the vertical wires, that connect every other row, are forced to occupy only one routing channel in two. This ensures that a simple right shift of the vertical routing by a routing pitch (i.e. the distance separating two routing tracks) does not create electrical shorts. As a consequence, vertical wires must be straight. If they were able to make turns, they would cross the adjacent routing tracks that are kept free for the duplicated vertical routes. On the contrary, wires of the "horizontal routing layers" are left free to make turns, as long as they remain in their placement row. Indeed, if the horizontal routing is confined within one row over two, the duplication of the "horizontal"

wires in the upper or the lower rows does not interfere with the wires in the current row.

The constraints imposed to the place-and-route tool summarize as follows: as the design must be translated vertically by the height of one placement row (`ROW_HEIGHT`) for placement reasons and horizontally by one routing pitch (`PITCH`) for routing reasons, the whole placed-and-routed design is scheduled to move by a $(\delta x, \delta y) = (\texttt{PITCH}, \texttt{ROW\_HEIGHT})$ vector translation. In backend taxonomy, this translation actually coincides with the minimum "placement site".

At that point, the result of the duplication is two identical netlists interleaved one into the other. Notice that the netlists cannot be "de-interleaved" because they are not independent: some signals must be exchanged locally between abutted gates. As we will see in Sect. 3, it happens for the inverter gate in SABL and WDDL (Tab. 1(b)) and for all gates in SI-WDDL (Fig. 7).

The chip finishing steps shall not delete the indistinguishability of the two netlists. For instance, the dummies generator must be constrained to add dummies (metal pieces added randomly to fulfill the minimum density design rules) only in the rows in which placement is allowed. Afterwards, dummies are duplicated and translated by a placement site: they end up in the same routing environment as initially (no short is created) since the routing was duplicated in the same manner.

2.3 The Constraints Required by the "Backend Duplication"

As mentioned above, the "backend duplication" method is implemented by (1) constraining the design and (2) duplicating the placed-and-routed design. The constraints can be generated automatically by a script setting the following obstructions:

- **placement blockages** one row over two and on the rightmost placement site of the placeable row,
- **routing blockages** of one track channel over two for vertical metals and over the rows already marked obstructed for standard cell placement for horizontal metals.

Figure 8(a) illustrates these constraints on a 16×2-site piece of floorplan. As far as the routing is concerned, these constraints are more flexible than the ones proposed in the "fat wire" method [9], since only vertical wires are forced to remain strictly straight. The metals whose preferred routing direction is horizontal are free to zigzag, provided they stay within their row. This degree of freedom is not negligible, since there are typically around 12 routing channels per row. This allows for both a more successful and a faster routing.

2.4 Insertion into an Existing Design Flow

As seen in Sect. 2.3, the "backend duplication" method need not redefine the design rules. It only relies on constraints on the CAD software. A typical backend flow includes the steps shown in Fig. 2. The insertion of the "backend duplication" consists in adding three steps (*i*, *ii* and *iii*).

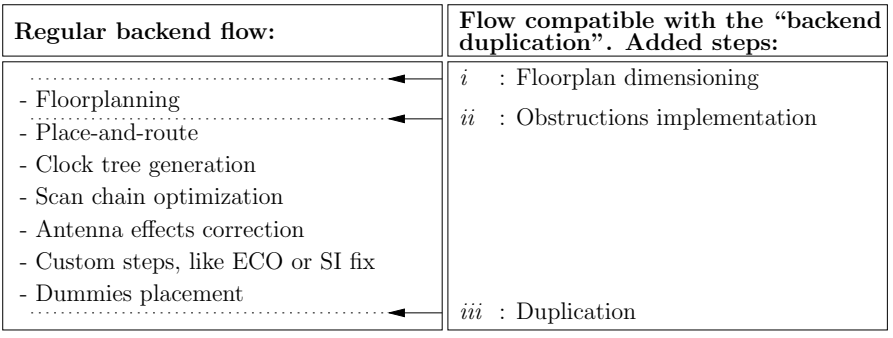


Fig. 2. Typical backend flow and modifications (steps *i*, *ii* and *iii*) to implement the "backend duplication" method

***i.* Floorplan dimensioning.** As a matter of fact, the floorplan of an design block is made up of two parts: the *core*, devoted to the standard cells placement and the *die*, that covers the core and an extra channel surrounding it. It is used for example to route a supply ring. The core horizontal dimension must be an even number of the routing `PITCH` and the vertical dimension an even number of `ROW_HEIGHT`. This condition ensures that the placement and the routing within the core do not extend out of the core after duplication.

The core can either be checked and repaired if one of the figures is odd or generated automatically. To end up with a core of density d and of aspect ratio r, the first step is to generate a core of density $d/2$ and of aspect ratio $r/2$ before duplication. Then the core dimensions (x, y) are retrieved, and a new core with the dimensions:

$$x' = \left\lceil \frac{x}{2 \times \texttt{PITCH}} \right\rceil \times 2 \times \texttt{PITCH}\ , \quad y' = \left\lceil \frac{y}{2 \times \texttt{ROW\_HEIGHT}} \right\rceil \times 2 \times \texttt{ROW\_HEIGHT}$$

is regenerated. Its density is slightly less than d and its aspect ratio roughly equal to r.

***ii.* Obstructions Instantiation.** The constraint script described previously in Sect. 2.3 can be generated automatically as soon as the floorplan dimensions are known. This script is sourced after floorplanning and before place-and-route.

***iii.* Duplication.** As far as standard cells are concerned, the duplication consists in a translation by a placement site followed by an horizontal flipping of each row.

The routing duplication is a bit more complex than a mere translation. Indeed, the design pins extend over the core to reach the die boundary. If the routing was simply translated, the duplicated design would have pins both inside and outside the die. To avoid this shortcoming, the routing extremities (u, v) of every wire undergo this transformation:

– if (u, v) belongs to the core, then $(u', v') = (u + \texttt{PITCH}, v + \texttt{ROW\_HEIGHT})$,
– otherwise $(u', v') = (u, v)$.

Additionally, to prevent shorts, the constraints described in Sect. 2.3 actually extend till to die limits and the routing channels that are entirely outside the core are obstructed. These transformations are illustrated on Fig. 8(b).

The information needed to apply the duplication is the orientation and position of standard cells and the routing coordinates. The design exchange format (DEF) typically contains all this information. Given the simplicity of the DEF syntax and the availability of parsers [11], the duplication can be implemented easily.

It is also a good idea to apply the duplication on the Verilog netlist: it consists in duplicating all wires and all leaf instances (i.e. standard cells). Verilog parsers are easy to write, even from scratch. The key benefit of generating the duplicated Verilog netlist is to enable LVS verification.

2.5 Comparison with Related Works

K. Tiri [12] noticed that the balancedness of the routing is crucial to effectively protect a differential circuit against SCA. The solution put forward in [9] is based on "fat wires" routing: a large wire is first routed and then split into two minimum-sized wires. This method implies that:

– Specific design rules must be written for the "fat wires".
– The only way for a wire to turn is to change layers.
– For the "fat wire" to access the pins of standard cells, their layout must be redefined.

The "backend duplication" implies none of these assumptions.

The experimental DPA [6] of F.G. Bouesse *et al.* [13] also showed that the weakest nodes in a differential layout correspond to unbalanced pairs. The backend correction flow described in [14] is iterative: the design is successively routed and analyzed, until every dual-rail pair is balanced. The analysis consists in the collection for every node of the sum of the parasitic elements extracted after every routing (more details in Sect. 4.2). This method requires a complex strategy to constrain the router and a non trivial algorithm to guide the iterative process towards a convergence point. On the contrary, the routing generated by "backend duplication" is balanced by design. However, the "backend duplication" only handles pairs of signals, whereas the iterative method [14] can route both dual and single-rail signals (data is dual-rail; acknowledge is single-ended.)

3 Suitability of the "Backend Duplication" Method with some Logic Styles

3.1 Backend Duplication for WDDL

The wave dynamic differential logic (WDDL, [2]) is a design style that uses standard cells by pairs, in such a way that at any step of the computation,

one and only one of the two gates has a transition. This behavior masks the fluctuations of the power consumption due to irregular activity: the activity of a WDDL circuit is constant. The computations are split into successive *precharge* and *evaluation* steps. A Boolean function $e_{i\in\{0,1,\cdots\}} \mapsto f(e_i)$ is computed using the two dual gates $f_T(e_i)$ and $f_F(e_i)$ that satisfy:

$$\begin{cases} \text{During precharge:} & \exists i, \quad f_T(e_i) = f_F(e_i)\,, \\ \text{During evaluation:} & \forall i, \quad f_T(e_i) = \overline{f_F(\overline{e_i})}\,. \end{cases} \tag{1}$$

Table 1(a) provides some examples of dual gates pairs suitable for WDDL. If the condition on the precharge in (1) cannot be met, the identity shown in Tab. 1(b) solves the problem out. The truth table of two dual gates (refer to

Table 1. Duality: definition, examples (a) and WDDL identity for the invertor (b)

(a)

	Regular gate	**Dual gate**
Definition	$f(e_i)$	$\overline{f(\overline{e_i})}$
Examples	NOT	NOT
	NAND	NOR
	$\Pi\,\Sigma\,e_i$	$\Sigma\,\Pi\,e_i$

(b)

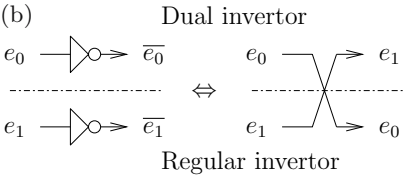
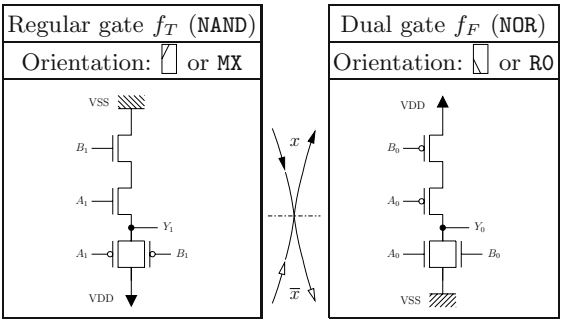


Table. 2) shows a symmetry, that can also be observed at the transistor level, as shown in Table. 3.

Table 2. Truth table of the two dual functions NAND/NOR

e_0	e_1	NAND $\overline{e_0 \cdot e_1}$	NOR $\overline{e_0 + e_1}$
0	0	1	1
0	1	1	0
1	0	1	0
1	1	0	0

Table 3. Illustration of the NAND/NOR dual gate couple symmetry $\{f_T, (\mathrm{N}, \mathrm{P}), \mathtt{MX}\} \leftrightarrow \{f_F, (\mathrm{P}, \mathrm{N}), \mathtt{RO}\}$



The symmetry illustrated in Table. 3 suggests that standard cells are ready to be used in a WDDL flow using the "backend duplication" method. This is actually only partially true: the structures in transistors indeed perfectly superimpose, but in practice, PMOS (*symbol:* ⊥) are drawn wider than NMOS

(*symbol:* .) For this reason, in a commercial standard cell library, the pins of a gate (regular orientation: or R0) and of the X-symmetric (orientation: or MX) of its dual do not match exactly. Nevertheless, as they are located on the routing grid, they usually overlap.

Fortunately, it is easy to work around this difficulty. The procedure begins with an enlargement of the pins. Then, the pins are merged considering the intersection of the enlarged pins. The routing obstructions are basically made up of the metal not included in the union of the newly created pins:

$$\begin{cases} \texttt{PIN} = \texttt{PIN}(\texttt{NAND}) \cap \texttt{PIN}(\texttt{NOR}), & (\square \text{ in Fig. 5}) \\ \texttt{OBS} = (\texttt{OBS}(\texttt{NAND}) \cup \texttt{OBS}(\texttt{NOR})) \cup (\texttt{PIN}(\texttt{NAND}) \triangle \texttt{PIN}(\texttt{NOR})) \,. & (\blacksquare \text{ in Fig. 5}) \end{cases}$$

This procedure can be applied on the sole *abstract* view of the standards cells. Thus a simple LEF parser [11] can be used turn a standard cell library into a WDDL-compliant library. Instead of describing the parser into details, a graphical example on the NAND/NOR and AND/OR gate couples is shown in Fig. 5.

As far as cell placement duplication is concerned, the method presented in step *iii* (refer to Sect. 2.4) demands that, in addition to the duplication and the flipping, the gate be replaced by its dual.

3.2 Backend Duplication for Other Logic Gates

In order to apply the "backend duplication" method to SABL or SI-WDDL, the gates must be split into two parts: one computing *true* values, the other *false* values.

The splitting is straightforward for SABL, as shown in Fig. 6.

As for SI-WDDL, the division is a bit less trivial, but is sane since it forces the symmetry of the transistor schematic to be kept in layout view. The placement of each building block of the cell along with the indication of their orientation is provided in Fig. 7.

For both SABL and SI-WDDL, the gate pins must be designed in such a way they are left unchanged in a symmetry $y \leftarrow \texttt{ROW\_HEIGHT} - y$ (or R0 $\leftrightarrow$ MX). This condition ensures that a connection to the pin of a regular gate (placed first) also arrives on a pin of the other half of the gate (placed while duplicating the backend at step *iii*). Additionally, the routing converges faster if the pins are placed on every other vertical routing track: the pins are better accessed if they are not below a vertical routing obstruction.

4 Implementing a Duplicated Netlist

4.1 The Example of a Secured DES Cryptoprocessor Design

In this section, we explain how a placed-and-routed netlist obtained by the "backend duplication" method can be embedded into a whole design. First of all, let us notice that after duplication, even global signals are duplicated: the duplicated

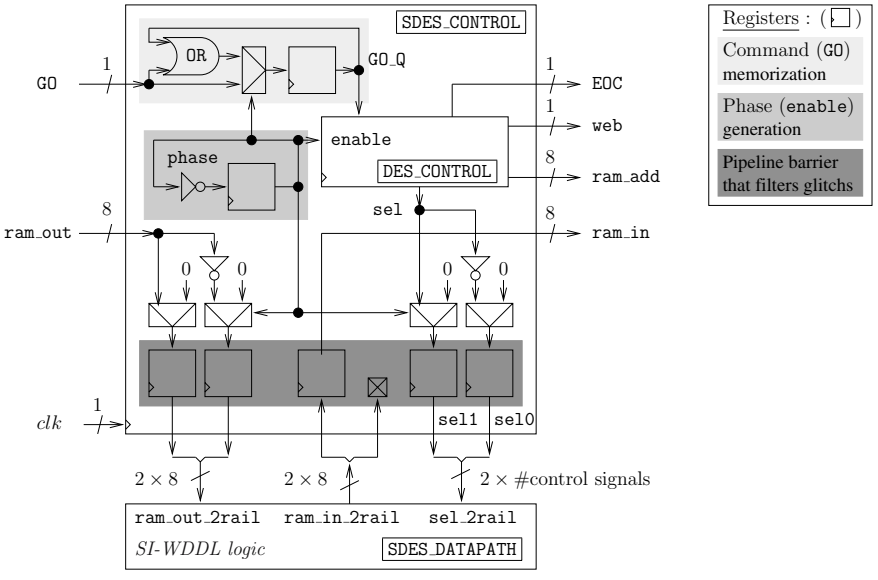


Fig. 3. Secured DES architecture. The duplicated datapath (`SDES_DATAPATH`), for example implemented in SI-WDDL logic, is obtained according to the method described in Sect. 2. The regular control (`DES_CONTROL`) is encapsulated into a wrapper (`SDES_CONTROL`) that can interface to the dual-rail datapath of DES.

backend has two clocks and two resets, that must be shorted together. The two scan chains can either be joined or be considered independently.

Most often, the whole cryptoprocessor need not be secured. The reason is that when implementing a non proprietary algorithm such as DES, the computation steps are public. As a consequence, the control leaks non confidential information. In most designs, the control (algorithm steps) can be clearly dissociated from the datapath (data processing).

It is relevant to derive the control of the duplicated datapath (dual-rail encoding, RTZ protocol) from the original control of the insecure datapath (single-ended, no RTZ): it allows to debug a single-ended control, which is easier to understand and faster to simulate. The method to update the regular control to make it compatible with the duplicated datapath requires that:

- The state machine can be frozen: it has an `enable` input. This enable forces the state machine to work twice as slow as initially to mimic RTZ.
- The control is wrapped by a converter single-to-dual rail for the datapath inputs and dual-to-single rail for its outputs. In addition to converting the control signals exchanged between the datapath and the control, the control wrapper also converts the datapath input and output data. Thus, seen from the outside, the cryptoprocessor keeps a single-ended interface. However, the internal architecture of the datapath is dual-rail RTZ secure logic obtained by "backend duplication".

When the control is disabled (`enable` = 0), all the input signals of the datapath (provided by the control wrapper) are set to the precharge state (e.g. `00`). This solution emulates the dual-rail RTZ protocol required by the duplicated architecture of the datapath. Moreover, this architecture is well suited for asynchronous gates implementations, such as SI-WDDL, because the datapath inputs (both data and control) are kept behind a register barrier, which guarantees that those signals are glitch-free. This condition is mandatory for SI-WDDL logic to work securely.

The schematic of Fig. 3 shows the secure architecture of a DES module. Let us notice that the control input signals (a simple start command, named `GO` in Fig. 3) is memorized as `GO_Q` over the two phases (precharge and evaluation), to prevent it from being discarded if it arrives when the control is disabled. The `GO` command can actually be activated at any time, because the cryptoprocessor environment is not aware of the RTZ behavior of the secured DES.

4.2 Method Cost and Security Evaluation

The method overhead is assessed below:

- The circuit frequency is unchanged, but every encryption takes twice more time to execute because of the RTZ protocol.
- The area increase of the datapath depends on which secured gates are used. If WDDL gates are chosen, `SDES_DATAPATH` is simply twice as large as `DES_DATAPTH`. If SI-WDDL gates are chosen, we obtain a 15 times area increase[1]. The overhead of the control area is 14%: the area of the module `DES_CONTROL` (resp. `SDES_CONTROL`) is 12 942 μm^2 (resp. 14 788 μm^2.)

The increase of security can be assessed by the ratio of the two dual lines routing capacitances and resistances. The capacitance "C" accounts for the power dissipation occurring at every transition: $\frac{1}{2} \times \mathrm{C} \times (\mathrm{VDD} - \mathrm{VSS})^2$. The resistance "R" is responsible for the delay $\mathrm{R} \times \mathrm{C}$ of the transition propagation. The wire pairs are all the more balanced as the ratios C(true)/C(false) and R(true)/R(false) do not spread much around 1. Figure 4 shows the repartition of those ratios for the 2 211 internal wire couples of `SDES_DATAPATH`. The three data samples correspond to a dual placed-and-routed design, obtained by the "backend duplication" method, a dual placed and regular routed design, and a regular placed-and-routed design. Both the capacitances and resistances were obtained using the RC extractor tool of Cadence SOC/Encounter. The technological information was produced by the Cadence coyote field solver.

The resistance of a "backend duplicated" circuit against EMA [7] has not been evaluated yet.

[1] The SI-WDDL gates were not optimized: a much better ratio can probably be obtained, even without any trade-off on the gate symmetry.

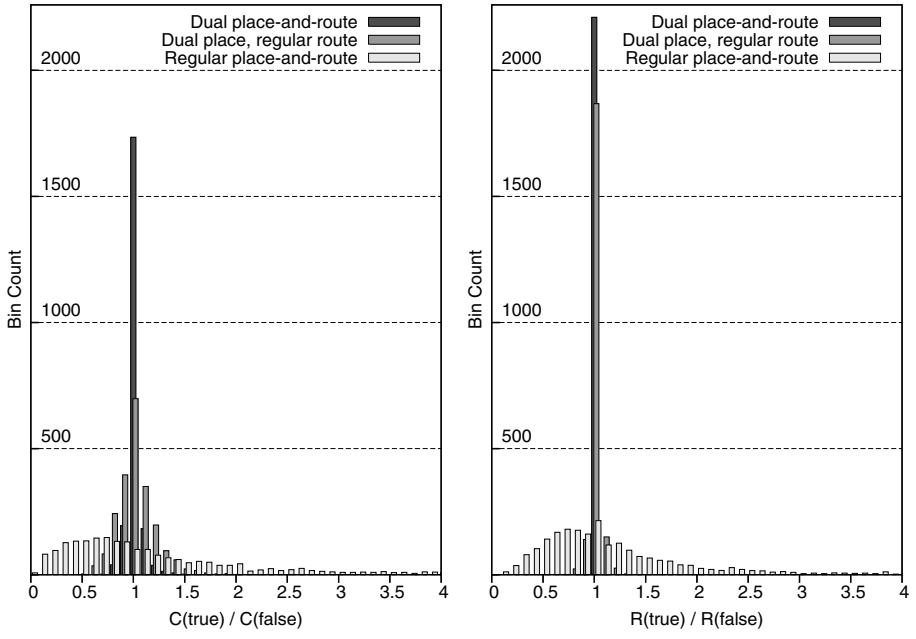


Fig. 4. Ratio of the capacitances and the resistances of `SDES_DATAPATH` dual nets

5 Conclusion

Securing a cryptoprocessor against physical attacks (either SCA or FA) can be done at the algorithmic or at the implementation level. This paper focuses on the countermeasures on the hardware implementation. Many types of primitive gates suitable for secure computation have been proposed [1,2,3,4], but the issue of building cryptoprocessor out of them is seldom addressed. To the authors' knowledge, only the "fat wire" method [9] partially tackles this problem.

We provide a complete description of a backend flow compatible with all of the above-mentioned gates. The method we describe can apply to all existing flows and requires no modification of the design rules.

The "backend duplication" method is illustrated on the example of a DES cryptoprocessor. This example also shows that the method is compatible with a secure partitioning of the design: only the datapath is duplicated. The emphasis is placed on the insertion of the duplicated datapath into the whole DES, whose interface remains unchanged. This case study proves that the hardening of a cryptoprocessor can be fully automated and that the integration of the "backend duplication" method into an existing flow is seamless.

Acknowledgements

This work has been partially funded by the "conseil régional Provence Alpes Côte d'Azur" and the Research Ministry, through ACI SI MARS. The authors

are also grateful to the AST division of STMicroelectronics (Rousset, France), for help in the design and the fabrication of the secured DES ASIC prototype.

References

1. Tiri, K., Akmal, M., Verbauwhede, I.: A Dynamic and Differential CMOS Logic with Signal Independent Power Consumption to Withstand Differential Power Analysis on Smart Cards. In: Proceedings of ESSCIRC'02. (2002) pp 403–406.
2. Tiri, K., Verbauwhede, I.: A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In: Proceedings of DATE'04. (2004) pp 246–251.
3. Moore, S., Anderson, R., Cunningham, P., Mullins, R., Taylor, G.: Improving Smart Card Security using Self-timed Circuits. In: Proceedings of ASYNC'02. (2002) pp 211–218.
4. Guilley, S., Hoogvorst, P., Mathieu, Y., Pacalet, R., Provost, J.: CMOS Structures Suitable for Secured Hardware. In: Proceedings of DATE'04. (2004) pp 1414–1415.
5. Kocher, P., Jaffe, J., Jun, B.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Proceedings of CRYPTO'96. Volume 1109 of LNCS., Springer (1996) pp 104–113.
6. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis: Leaking Secrets. In: Proceedings of CRYPTO'99. Volume 1666 of LNCS., Springer (1999) pp 388–397.
7. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic Analysis: Concrete Results. In: Proceedings of CHES'01. Volume 2162 of LNCS., Springer (2001) pp 251–261.
8. Sokolov, D., Murphy, J., Bystrov, A.: Improving the Security of Dual-Rail Circuits. In: Proceedings of CHES'04. LNCS, Springer (2004) pp 282–297.
9. Tiri, K., Verbauwhede, I.: Place and Route for Secure Standard Cell Design. In: Proceedings of CARDIS'04. (2004) pp 143–158.
10. Goubin, L., Patarin, J.: DES and Differential Power Analysis (The "Duplication" Method). In: Proceedings of CHES'99. LNCS, Springer (1999) pp 158–172.
11. LEF/DEF parsers: (website) `http://openeda.si2.org/projects/lefdef/` or `http://www.cadence.com/partners/languages/languages.aspx`.
12. Tiri, K., Verbauwhede, I.: "Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology. In LNCS, ed.: Proceedings of CHES'03. Volume 2779 of LNCS., Springer (2003) pp 125–136.
13. Bouesse, G., Renaudin, M., Robisson, B., Beigné, E., Liardet, P.Y., Prevosto, S., Sonzogni, J.: DPA on Quasi Delay Insensitive Asynchronous Circuits: Concrete Results. In: Proceedings of DCIS'04. (2004) Bordeaux, France.
14. Bouesse, G., Renaudin, M., Dumont, S., Germain, F.: DPA on Quasi Delay Insensitive Asynchronous Circuits: Formalization and Improvement. In: Proceedings of DATE'05. (2005) pp 424–429. Munich, Germany.

A Appendix: Graphical Illustrations of the "Backend Duplication" Method

Figures 5, 6 and 7 show how WDDL, SABL and SI-WDDL gates must be transformed prior to being used in the "backend duplication" design flow.

Figure 8 illustrates the "backend duplication" (steps *ii* and *iii*) on a floorplan suitable for the duplication (step *i* was already executed: the floorplan dimensions are even.)

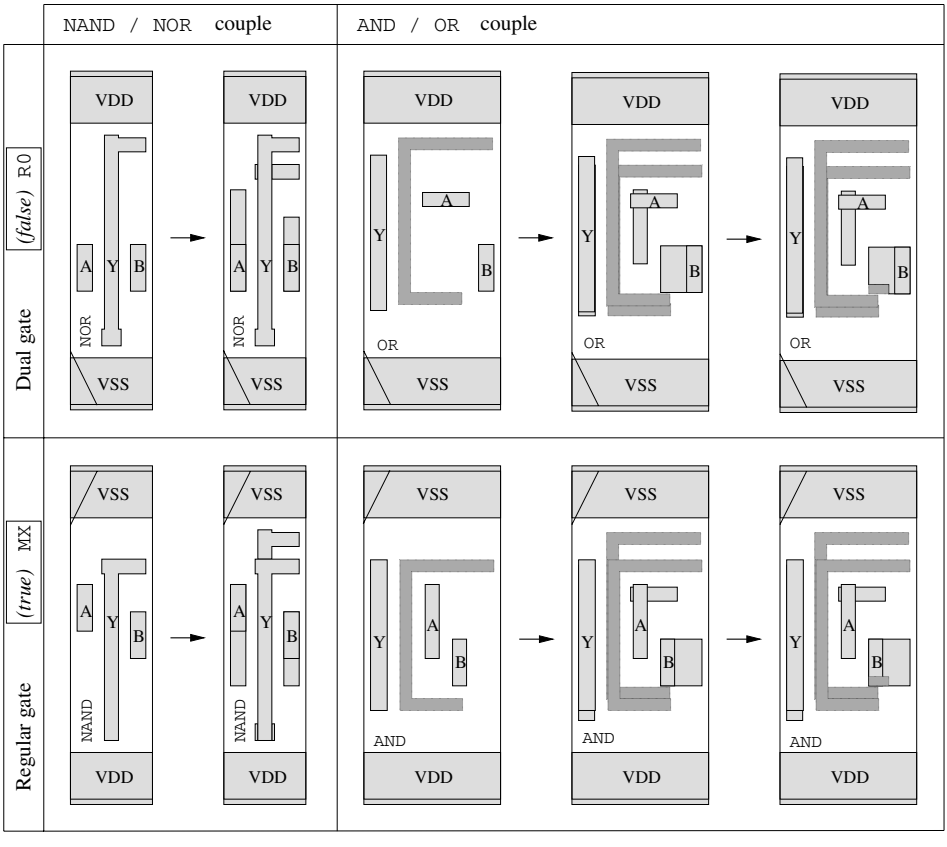


Fig. 5. Transformation on the *abstracted* views of the standard cells to make them WDDL-compliant [2]. This resulting gate couple satisfies the following condition: the abstract couples $\{f_T, (\mathrm{N}, \mathrm{P}), \texttt{MX}\}$ and $\{f_F, (\mathrm{P}, \mathrm{N}), \texttt{R0}\}$ perfectly superimpose.

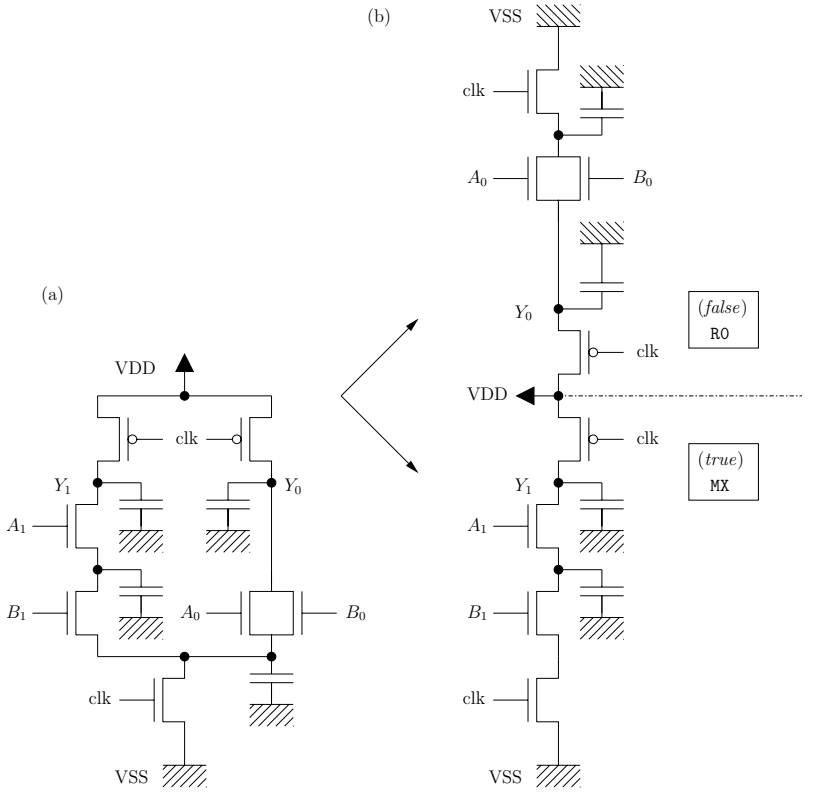


Fig. 6. Transformation of a `NAND` gate implemented in SABL [1] (a) into two dual gates (b), for subsequent use in the "backend duplication" design flow

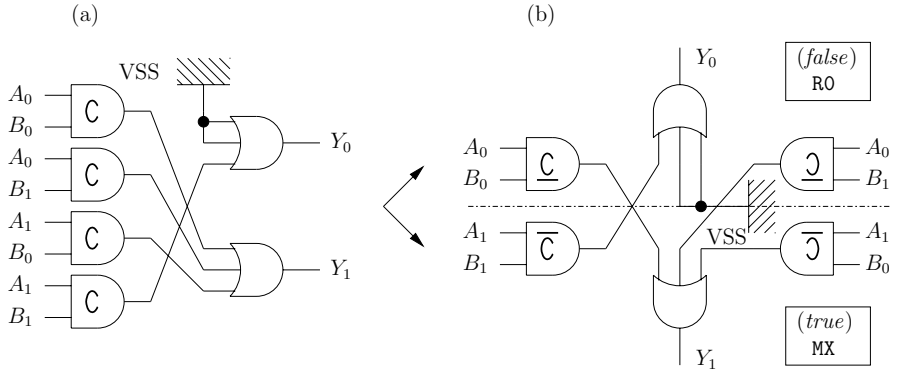


Fig. 7. Transformation of a `NAND` gate implemented in SI-WDDL [3] (a) into two dual gates (b). Notice that the two halves of the gate exchange signals.

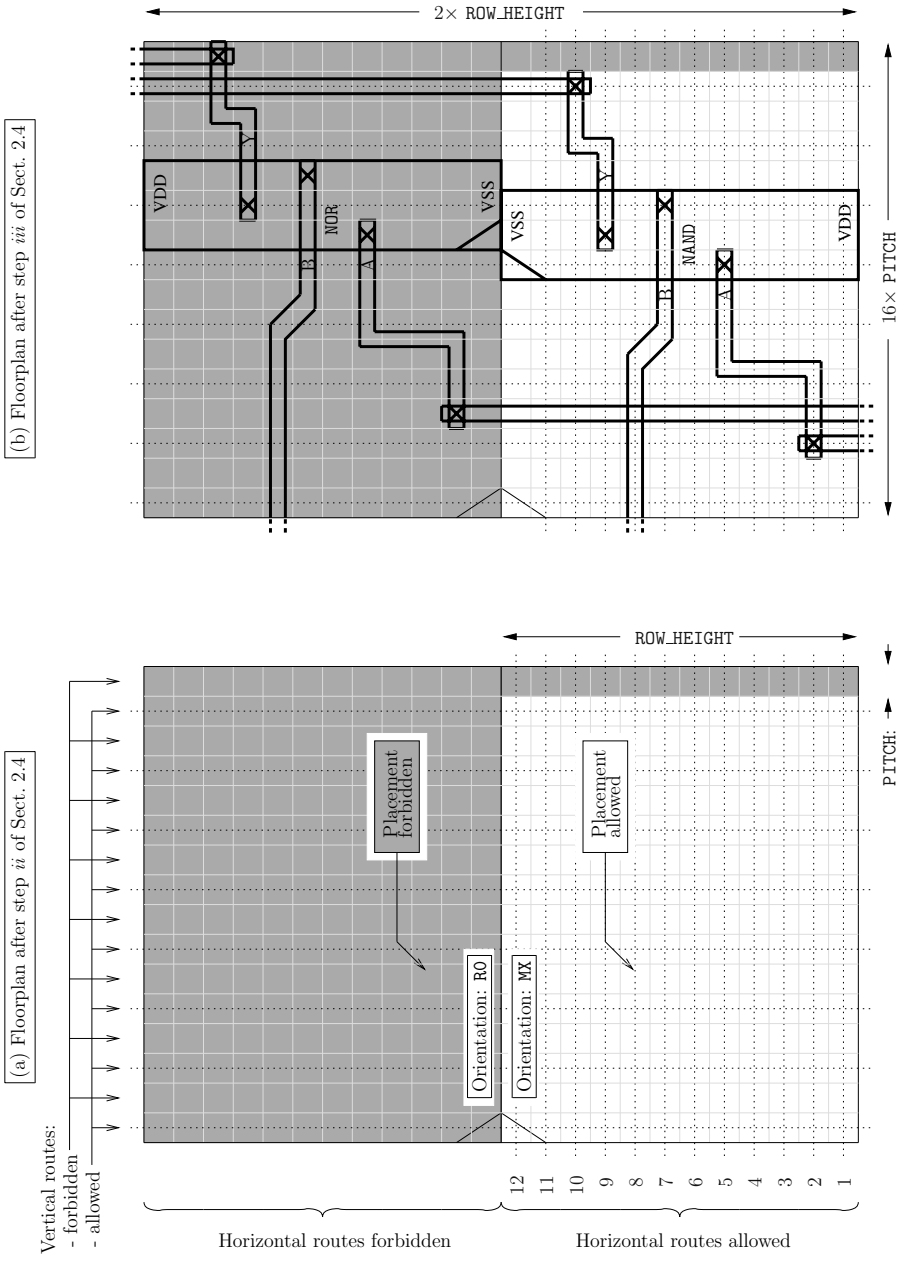


Fig. 8. (a) Place and route constraints, illustrated on a floorplan containing 16×2 placement sites. In `PITCH` units, the placement site is 1×12 and the routing grid offset is $\frac{1}{2} \times \frac{1}{2}$. (b) Final floorplan containing one single `NAND` gate (and its dual `NOR` gate). The horizontal wires can turn (wires connecting ports A, B and Y), whereas the vertical ones are straight. The vias that contact horizontal and vertical wires are noted ⊠.

Hardware Acceleration of the Tate Pairing in Characteristic Three*

P. Grabher[1] and D. Page[2]

[1] Institute for Applied, Information Processing and Communications,
Graz University of Technology, Inffeldgasse 16a,
A-8010 Graz, Austria
grabherp@sbox.tugraz.at
[2] Department of Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol, BS8 1RB, United Kingdom
page@cs.bris.ac.uk

Abstract. Although identity based cryptography offers many functional advantages over conventional public key alternatives, the computational costs are significantly greater. The core computational task is evaluation of a bilinear map, or pairing, over elliptic curves. In this paper we prototype and evaluate polynomial and normal basis field arithmetic on an FPGA device and use it to construct a hardware accelerator for pairings over fields of characteristic three. The performance of our prototype improves roughly ten-fold on previous known hardware implementations and orders of magnitude on the fastest known software implementation. As a result we reason that even on constrained devices one can usefully evaluate the pairing, a fact that gives credence to the idea that identity based cryptography is an ideal partner for identity aware smartcards.

Keywords: Identity Based Encryption, Pairing, Elliptic Curve, FPGA.

1 Introduction

The notion of identity based cryptography was first proposed by Shamir [25] in 1984. Essentially it allows a user identity, an arbitrary string, to play the role of a public key rather than have the key derived from a relationship with private information as would be the case in traditional schemes such as RSA. This can vastly reduce the amount of certification infrastructure required and generally presents a rich set of functional and security characteristics that are difficult or impossible to realise with other solutions. The first efficient Identity Based

* The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 398–411, 2005.

Encryption (IBE) scheme was presented by Boneh and Franklin [8] who followed the idea of Sakai, Ohgishi and Kasahara [23] in basing their scheme on bilinear maps, or pairings, over elliptic curves.

Although pairing and identity based cryptography has sparked a wealth of research into cryptographic schemes [7,11] and proof techniques, it has remained an ongoing task to reduce the computational cost that underpins such work. Theorists have generally worked under the gross assumption that a pairing takes around ten times as long to compute than the major computational task in elliptic curve cryptography (ECC), the point multiplication. Although in reality this ratio is significantly lower, the cost of pairing evaluation still constitutes a major hurdle. This is particularly true in constrained environments such as smart-cards which, due to their use as identity-aware tokens, seem a natural partner for identity based cryptography.

Recently, Gemplus announced that it had developed a smart-card hosted IBE implementation in partnership with the market leaders Voltage Security [27]. Although details are scarce, it seems probable that they use an existing core for $\mathbb{F}_p$ arithmetic to accelerate a software implementation of the BKLS algorithm [4]. This seems the natural decision given the increasing flexibility in parameterisation [19, 3, 5] and expertise related to implementing arithmetic in $\mathbb{F}_p$ accumulated from building conventional ECC and RSA based systems. However, in the short term at least it is attractive to consider working over fields of characteristic three since when parameterised using suitable supersingular elliptic curves, the resulting system boasts a higher security multiplier [12], given by the MOV embedding degree [20]. Additionally, there are some specialised, high-performance algorithms for computing pairings in this context: the Duursma-Lee algorithm [10], recently improved upon by Kwon [18] and Barreto et al. [2], uses a closed formula for the pairing which is efficient as long as the underlying field arithmetic in $\mathbb{F}_{3^m}$ is also efficient. To this end, previous work has considered the possibility of using polynomial [22, 6] and normal bases [13] to implement said arithmetic. However, such work has focused mainly on arithmetic performance rather than placing the designs in context to actually compute IBE related functions, the exception being Kerins, Popovici and Marnane [17] who quote estimated timings for FPGA hosted pairing hardware using a BKLS style algorithm.

In this paper, our main aims are three-fold: to evaluate the performance and cost of constructing hardware polynomial and normal basis arithmetic in $\mathbb{F}_{3^m}$; to investigate the possibility of construct a hardware accelerator that is small enough for use in constrained environments; to prove pairings over $\mathbb{F}_{3^m}$ using the closed form family of algorithms are a viable alternative to the use of $\mathbb{F}_p$ and BKLS. We prototype our work on an FPGA device and present experimental results of the performance and cost comparisons with previous work in this area. We organise our work as follows: in Section 2 we give an overview of pairings before using Section 3 to present details of arithmetic in $\mathbb{F}_{3^m}$. We then discuss the details of our accelerator architecture and present experimental results in Section 4 before concluding in Section 5.

Algorithm 1. The Duursma-Lee algorithm [10] for calculating the Tate pairing in characteristic three.

Input : point $P = (x_1, y_1)$, point $Q = (x_2, y_2)$
Output: $f_P(\phi(Q)) \in \mathbb{F}^*_{q^6}/\mathbb{F}^*_{q^3}$

$f \leftarrow 1$
for $i = 1$ **upto** m **do**
  $x_1 \leftarrow x_1^3$
  $y_1 \leftarrow y_1^3$
  $\mu \leftarrow x_1 + x_2 + b$
  $\lambda \leftarrow -y_1 y_2 \sigma - \mu^2$
  $g \leftarrow \lambda - \mu\rho - \rho^2$
  $f \leftarrow f \cdot g$
  $x_2 \leftarrow x_2^{1/3}$
  $y_2 \leftarrow y_2^{1/3}$
return f

2 An Introduction to Pairings

To provide a concrete case for discussion, we use the example of pairings where the base field is of characteristic three, i.e. $\mathbb{F}_q$ where $q = 3^m$. To allow investigation of both polynomial and normal bases we consider cases $m = 97$ and $m = 89$ respectively. Let E be an elliptic curve over a finite field $\mathbb{F}_q$, and let $\mathcal{O}$ denote the identity element of the associated group of rational points $E(\mathbb{F}_q)$. For a positive integer $l|\#E(\mathbb{F}_q)$ coprime to q, let $\mathbb{F}_{q^k}$ be the smallest extension field of $\mathbb{F}_q$ which contains the l-th roots of unity in $\overline{\mathbb{F}}_q$. Also, let $E(\mathbb{F}_q)[l]$ denote the subgroup of $E(\mathbb{F}_q)$ of all points of order dividing l, and similarly for the degree k extension of $\mathbb{F}_q$. Setting $k = 6$, we parameterise $\mathbb{F}_{q^6}$ as the quadratic extension $\mathbb{F}_{q^6} = \mathbb{F}_{q^3}[\sigma]/(\sigma^2 + 1)$. Further, we set $\mathbb{F}_{q^3} = \mathbb{F}_q[\rho]/(\rho^3 - \rho - 1)$. For efficient arithmetic in these fields, we to the work of Granger et al. [14].

Our choice of prime values for m is motivated by well known security considerations; both our choices offer an security level which is roughly equivalent to $800 - -900$-bit RSA. Using a polynomial basis with $m = 97$ provides us with a curve which is well known in the literature and hence a good reference against which to compare our results. However, one can only construct a type-two normal basis where $2m + 1$ is also prime: the most efficient type-one basis is never available. This limits our choices significantly. We settled on $m = 89$ since it is the closest choice to $m = 97$ for which affords a suitable parameterisation. For both our choices of m, we use the curve $E : Y^2 = X^3 - X + 1$. In the case of $m = 89$ this has an unattractively large cofactor [13]: this parameterisation problem alone might be viewed as a reason not to use a normal basis representation; we stress that our aim in selecting these parameters is performance and cost comparison only.

Algorithm 2. The Kwon-BGOS algorithm [18] for calculating the Tate pairing in characteristic three.

Input : point $P = (x_1, y_1)$, point $Q = (x_2, y_2)$
Output: $f_P(\phi(Q)) \in \mathbb{F}_{q^6}^*/\mathbb{F}_{q^3}^*$

$f \leftarrow 1$
$x_2 \leftarrow x_2^3$
$y_2 \leftarrow y_2^3$
$d \leftarrow mb$
for $i = 1$ **upto** m **do**
  $x_1 \leftarrow x_1^9$
  $y_1 \leftarrow y_1^9$
  $\mu \leftarrow x_1 + x_2 + d$
  $\lambda \leftarrow y_1 y_2 \sigma - \mu^2$
  $g \leftarrow \lambda - \mu\rho - \rho^2$
  $f \leftarrow f^3 \cdot g$
  $y_2 \leftarrow -y_2$
  $d \leftarrow d - b$
return f

The Reduced Tate Pairing. For a thorough treatment of the following, we refer the reader to [4] and also [12], and to [24] for an introduction to divisors. The reduced Tate pairing of order l is the map

$$e_l : E(\mathbb{F}_q)[l] \times E(\mathbb{F}_{q^k})[l] \to \mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^l,$$

given by $e_l(P, Q) = f_{P,l}(\mathcal{D})$. Here $f_{P,l}$ is a function on E whose divisor is equivalent to $l(P) - l(\mathcal{O})$, $\mathcal{D}$ is a divisor equivalent to $(Q) - (\mathcal{O})$, whose support is disjoint from the support of $f_{P,l}$, and $f_{P,l}(\mathcal{D}) = \prod_i f_{P,l}(P_i)^{a_i}$, where $\mathcal{D} = \sum_i a_i P_i$. It satisfies the following properties:

- For each $P \neq \mathcal{O}$ there exists $Q \in E(\mathbb{F}_{q^k})[l]$ such that $e_l(P, Q) \neq 1 \in \mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^l$ (*non-degeneracy*).
- For any integer n, $e_l([n]P, Q) = e_l(P, [n]Q) = e_l(P, Q)^n$ for all $P \in E(\mathbb{F}_q)[l]$ and $Q \in E(\mathbb{F}_{q^k})[l]$ (*bilinearity*).
- Let $L = hl$. Then $e_l(P, Q)^{(q^k-1)/l} = e_L(P, Q)^{(q^k-1)/L}$.
- It is efficiently computable.

The non-degeneracy condition requires that Q is not a multiple of P, i.e. that Q is in some order l subgroup of $E(\mathbb{F}_{q^k})$ disjoint from $E(\mathbb{F}_q)[l]$. When one computes $f_{P,l}(\mathcal{D})$, the value obtained belongs to the quotient group $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^l$, and not $\mathbb{F}_{q^k}^*$. In this quotient, for a and b in $\mathbb{F}_{q^k}^*$, $a \sim b$ if and only if there exists $c \in \mathbb{F}_{q^k}^*$ such that $a = bc^l$. Clearly, this is equivalent to

$$a \sim b \text{ if and only if } a^{(q^k-1)/l} = b^{(q^k-1)/l},$$

and hence one ordinarily uses this value as the canonical representative of each coset. The isomorphism between $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^l$ and the elements of order l in $\mathbb{F}_{q^k}^*$ given by this exponentiation makes it possible to compute $f_{P,l}(Q)$ rather than $f_{P,l}(\mathcal{D})$.

The Modified Tate Pairing. Duursma and Lee introduced their algorithm [10] in the context of pairings on a family of supersingular hyperelliptic curves. The performance of their method was improved upon by Kwon [18] and Barreto et al. [2] who also provide similar algorithms for other characteristics.

Let $q = 3^m$ and $E(\mathbb{F}_q) : Y^2 = X^3 - X + b$, with $b = \pm 1$, and let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be points of order l. Let $\mathbb{F}_{q^3} = \mathbb{F}_q[\rho]/(\rho^3 - \rho - b)$, with $b = \pm 1$ depending on the curve equation, and let $\mathbb{F}_{q^6} = \mathbb{F}_{q^3}[\sigma]/(\sigma^2 + 1)$. Then the modified Tate pairing on E is the mapping $f_P(\phi(Q))$ where $\phi : E(\mathbb{F}_q) \to E(\mathbb{F}_{q^6})$ is the distortion map $\phi(x_2, y_2) = (\rho - x_2, \sigma y_2)$. The methods for computing the Duursma-Lee and Kwon-BGOS algorithms are shown in Algorithm 1 and Algorithm 2 respectively. Note that the final result is powered by $q^3 - 1$ to form a compatible result with the BKLS [4] algorithm.

3 Arithmetic in $\mathbb{F}_{3^m}$

The finite field $\mathbb{F}_{3^m}$ is isomorphic to $\mathbb{F}_3[X]/(p)$ and $\mathbb{F}_3(\alpha)$ where p is an irreducible polynomial of degree m in $\mathbb{F}_3[X]$ and α is a root of p. We will identify these three fields, but our notation will be tailored toward $\mathbb{F}_3(\alpha)$. In a polynomial basis $\mathbb{F}_3(\alpha)$ is regarded as an m-dimensional vector space over $\mathbb{F}_3$ with basis

$$(\alpha^0, \alpha^1, \ldots, \alpha^{m-1}) \ .$$

For an element $\hat{a} \in \mathbb{F}_3(\alpha)$ we will simply write the elements in a polynomial, or standard basis as

$$\hat{a} = \sum_{i=0}^{m-1} \hat{a}_i \cdot \alpha^i \ .$$

Arithmetic in a polynomial basis is fairly straightforward when based on conventional polynomial arithmetic. When discussing implementation of such arithmetic, it is often useful to denote elements as a vector of coefficients such as

$$\hat{a} = (\hat{a}_0, \hat{a}_1, \hat{a}_2, \ldots, \hat{a}_{m-1}) \ ,$$

so that physical operations such as shifting and rotation of coefficients is more naturally expressed. We use the notation $\hat{a}^{(i)}$ to denote the (left) rotation of the coefficients in such a vector by distance i. That is, we write

$$\hat{a}^{(i)} = (\hat{a}_{i+0}, \hat{a}_{i+1}, \hat{a}_{i+2}, \ldots, \hat{a}_{i+m-1}).$$

where in all cases, coefficient indices are reduced modulo m. Using this notation, $\hat{a}_j^{(i)}$ represents the j-th coefficient of the rotated element $\hat{a}^{(i)}$.

In a normal basis, things are slightly more involved. Given an irreducible polynomial p of degree m and with root α, the full set of roots of p in $\mathbb{F}_3(\alpha)$ is

$$\mathbb{B} = (\alpha, \alpha^3, \alpha^{3^2}, \ldots, \alpha^{3^{m-1}}).$$

If the elements of $\mathbb{B}$ are linearly independent then the set of roots forms a basis of $\mathbb{F}_3(\alpha)$ over $\mathbb{F}_3$ and this basis, p and α are all called normal. To construct such as basis, and the matrix M which determines how the multiplication operation works, we use the techniques of Granger et. al [13] based on work by Nöcker [21]. For an element $\bar{a} \in \mathbb{F}_3(\alpha)$ we write

$$\bar{a} = \sum_{i=0}^{m-1} \bar{a}_i \cdot \alpha^{3^i}$$

but again, for brevity, we often denote a normal basis field element using the coefficient vector and rotated coefficient vector notation as described above.

When using both polynomial and normal basis representations, we hold a polynomial over $\mathbb{F}_3$ of degree m as a $2m$ length vector of bits. Two sequential bits are used to hold each coefficient so that

$$a = (a_0^L, a_0^H, a_1^L, a_1^H, \ldots, a_{m-1}^L, a_{m-1}^H)$$

where

$$\begin{aligned} a_i^L &= a_i \mod 2 \\ a_i^H &= a_i \operatorname{div} 2 . \end{aligned}$$

For concreteness, we set the defining polynomial for our polynomial basis to $\alpha^{97} + \alpha^{16} + 2$ and the normal polynomial p that defines M in our normal basis to $\alpha^{89} + \alpha^{88} + 2\alpha^{87} + \alpha^{84} + 2\alpha^{83} + 2\alpha^{82} + \alpha^{81} + \alpha^{72} + \alpha^{71} + \alpha^{70} + 2\alpha^{69} + \alpha^{66} + 2\alpha^{65} + 2\alpha^{64} + \alpha^{63} + 2\alpha^{54} + \alpha^{35} + \alpha^{34} + 2\alpha^{33} + \alpha^{30} + 2\alpha^{29} + 2\alpha^{28} + \alpha^{27} + 2\alpha^{18} + 2\alpha^{17} + 2\alpha^{16} + \alpha^{15} + 2\alpha^{12} + \alpha^{11} + \alpha^{10} + 2\alpha^9 + 1$.

3.1 Addition and Subtraction

The most basic operations on field elements are addition and subtraction. These are made reasonably straightforward because they can be performed component-wise with no interaction with other coefficients. Given that our coefficients are held using two bits, we can construct cells for the required arithmetic using simple logical operations. Following Harrison et al. [15], the addition $r_i = a_i + b_i$ of two coefficients a_i and b_i can be specified using

$$\begin{aligned} r_i^H &= (a_i^L \vee b_i^L) \oplus t \\ r_i^L &= (a_i^H \vee b_i^H) \oplus t \end{aligned}$$

where

$$t = (a_i^L \vee b_i^H) \oplus (a_i^H \vee b_i^L).$$

Subtraction, and hence multiplication by two, are equally efficient since the negation of an element a simply swaps the bits a_i^H and a_i^L over and can therefore be implemented by the same function as addition.

3.2 Cubing and Cube Roots

When working in characteristic three, cubing is an important operation since curve and pairing arithmetic is often manipulated to utilise cubing rather than a more costly multiplication. In addition, the cube root operation is important in the Duursma-Lee algorithm if pre-computation is avoided.

When using a normal basis, the cube and cube root operations are very efficient in characteristic three: both can be achieved by cyclic shifting the coefficients in an elements so that for an element $\bar{a}$

$$\begin{aligned} \bar{a}^3 &= (\bar{a}_{m-1}, \bar{a}_0, \ldots, \bar{a}_{m-3}, \bar{a}_{m-2}), \\ \sqrt[3]{\bar{a}} &= (\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_{m-1}, \bar{a}_0). \end{aligned}$$

Clearly these rotations can be easily implemented in a hardware circuit, where they reduce to wired permutation of bits with no actual computational overhead.

In a polynomial basis, cubing is a linear operation in the same way squaring is linear in characteristic two [22, 6]. That is, we have

$$(\hat{a}_i \alpha^i)^3 = \hat{a}_i^3 \alpha^{3i} = \hat{a}_i \alpha^{3i} \ .$$

Therefore, we can implement it using by simply thinning the coefficients, i.e. padding them with zeros, before performing a reduction. Cube root is somewhat more involved but since our chosen field is of the right form, we can utilise the method highlighted by Barreto [1]. Specifically, since our defining polynomial for $m = 97$ is $\alpha^{97} + \alpha^{16} + 2$ we have that $97 = 3u + 1$ and $16 = 3v + 1$ so that $u = 32$ and $v = 5$. Hence, for an element $\hat{a} = t_0 + t_1 + t_2$ where

$$\begin{aligned} t_0 &= \textstyle\sum_{i=0}^{u} \hat{a}_{3i} \alpha^i \\ t_1 &= \textstyle\sum_{i=0}^{u-1} \hat{a}_{3i+1} \alpha^i \\ t_2 &= \textstyle\sum_{i=0}^{u-1} \hat{a}_{3i+2} \alpha^i \end{aligned}$$

we have that

$$\sqrt[3]{\hat{a}} = t_0 + t_1^{\ll 2u+1} - t_1^{\ll u+v+1} + t_1^{\ll 2v+2} - 2t_2^{\ll u+1} - 2t_2^{\ll v+1}$$

given that for $t \in \mathbb{F}_{3^m}$, $t^{\ll n}$ denotes $t\alpha^n$, the value t shifted left by n coefficients and suitable reduced.

3.3 Multiplication

In addition to component-wise addition and subtraction, for normal basis multiplication we also require a component-wise multiplication of the form $r_i = a_i \cdot b_i$. This can be performed using similarly inexpensive logical operations

$$\begin{aligned} r_i^H &= (a_i^L \wedge b_i^H) \vee (a_i^H \wedge b_i^L) \\ r_i^L &= (a_i^L \wedge b_i^L) \vee (a_i^H \wedge b_i^H) \ . \end{aligned}$$

Armed with a function to perform this operation, we construct a general multiplication result of the form $\bar{c} = \bar{a} \cdot \bar{b}$ using

$$\bar{c}_k = \sum_{i=0}^{m-1} \bar{a}_{k+i} \cdot \sum_{j=0}^{m-1} M_{i,j} \cdot \bar{b}_{k+j}$$

where in all cases, coefficient indices are reduced modulo m. The sparse matrix M in this description is constructed from the normal polynomial p and essentially dictates how reduction behaves for the field. We developed a compiler that takes M and automatically produces circuitry to implement the three phases of the above formula: an addition phase to compute the terms $M_{i,j} \cdot \bar{b}_{k+j}$, keeping in mind that $M_{i,j} \in \{0, 1, 2\}$; a multiplication phase to multiply $\bar{a}_{k+i}$ by the summed terms; and accumulation phase sum all the multiplied terms and form $\bar{c}_k$. Such circuitry generates a single coefficient and hence requires m clock cycles to complete a multiply; we can place several of them working in parallel to accelerate the multiplication [13].

There has already been plenty of previous work dedicated to hardware polynomial basis multiplication methods in characteristic three [22, 6, 17]. We follow the approach of Bertoni et al. [6] in employing a digit-serial approach. In a similar way that a normal basis is scalable since we can utilise D parallel coefficient calculation circuits, a digit-serial multiplier allows us to scale the digit-size D in order to find a suitable balance between size and speed.

3.4 Inversion

Inversion is generally the most expensive operation when dealing with finite field arithmetic, so much so that in systems like ECC every effort is made to construct higher level operations so that inversion is not required. Due to the cost of constructing dedicated hardware for limited return, we implement inversion in software using our hardware for other operations in $\mathbb{F}_{3^m}$. To avoid the extra hardware cost described by Kerins et al. [17], we implement inversion using the relationship

$$a^{-1} = a^{3^m - 2}.$$

using a ternary expansion of the exponent since cubing operations are so inexpensive. In a polynomial basis this could be improved upon incrementally by using a translation of the standard binary Euclidean algorithm [15]. Since we only require inversion once in the final powering, we leave this issue for further work.

3.5 Exponentiation

Generally, we avoid exponentiation of pairing values by arbitrary exponents since one can use the bilinearity property to push the operation inside the pairing as a point multiplication which is more efficient, see the work of Granger et al. [14] for efficient methods in this area. However, we do need to consider the final powering

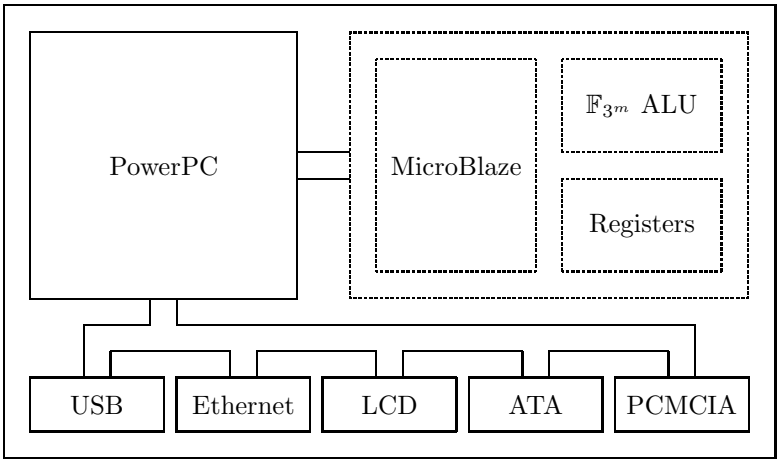


Fig. 1. A block diagram of our experimental architecture as hosted on a Xilinx ML300 prototyping device. Note that FPGA hosted elements are shown in dashed boxes while dedicated elements are shown in solid boxes.

of the pairing output by $q^3 - 1$ in order to yield a value compatible with BKLS. To power the pairing output f by the required exponent, we decompose the operation into

$$f^{3^{3m}} \cdot f^{-1}$$

the first term of which is simply three applications of the q-frobenius and the second is an inversion. Thanks to our field arithmetic, the inversion is reasonably efficient essentially because it can be done directly [14] rather than using an iterative method.

4 Architecture and Results

4.1 Architecture

Our design was realised using VHDL synthesised with a combination of Xilinx EDK 7.1 and ISE 7.1. Our experimental platform was a Xilinx ML300 prototyping board which hosts a Virtex-II PRO FPGA (XC2VP4FF672-6) device with 4928 slices. Our philosophy with this design was to treat the $\mathbb{F}_{3^m}$ arithmetic as a kind of co-processor, which is controlled by a more general purpose processor rather than hardwiring logic to directly compute the pairing. By swapping the co-processor we can provide arithmetic in either polynomial or normal bases; the FPGA size prevented making both available in one design. Since the instructions that are issued to the co-processor are executed synchronously, one might view this as a kind of instruction set extension. With this approach, we can easily implement other higher level operations based on the same field arithmetic, such as

Table 1. Cost and performance characteristics of hardware based field, point and pairing arithmetic using polynomial and normal bases, clocked at low and maximum frequencies

$\mathbb{F}_{3^{97}}$ in Polynomial Basis					
	Slices	Cycles	Instructions	Speed	
				At 16 MHz	At 150 MHz
Add	112	3	1	-	-
Subtract	112	3	1	-	-
Multiply	946	28	1	-	-
Cube	128	3	1	-	-
Cube Root	115	3	1	-	-
Point Doubling	-	220	15	13.8μs	1.5μs
Point Tripling	-	52	9	3.3μs	0.4μs
Point Addition	-	366	22	22.9μs	2.4μs
Pairing					
Duursma-Lee	-	59946	7857	3746.6μs	399.4μs
Kwon	-	64602	9409	4037.6μs	430.7μs
Powering	-	4941	397	308.8μs	32.9μs
Total	4481	-	-	-	-
$\mathbb{F}_{3^{89}}$ in Normal Basis					
	Slices	Cycles	Instructions	Speed	
				At 16 MHz	At 85 MHz
Add	102	3	1	-	-
Subtract	102	3	1	-	-
Multiply	1505	48	1	-	-
Cube	0	3	1	-	-
Cube Root	0	3	1	-	-
Point Doubling	-	360	15	22.5μs	4.2μs
Point Tripling	-	72	9	4.5μs	0.8μs
Point Addition	-	606	22	37.9μs	7.1μs
Pairing					
Duursma-Lee	-	89046	7857	5563.3μs	1047.6μs
Kwon	-	93702	9409	5856.3μs	1102.4μs
Powering	-	7941	397	496.3μs	93.4μs
Total	4233	-	-	-	-

the ECC point multiplication over $E(\mathbb{F}_{3^m})$ which is also required in most pairing based schemes.

As such, we combine our arithmetic in $\mathbb{F}_{3^m}$ with a register file, backed by BlockRAM, of 32 registers each able to store an element of $\mathbb{F}_{3^m}$ which total under 1 kilobyte for our choices of m. We control this combined data-path with a Xilinx MicroBlaze soft-core, a 32-bit, 3-stage pipelined RISC processor which interfaces to the logic using the Fast Simplex Link (FSL) interface. The MicroBlaze code

to control the co-processor was compiled using a re-targeted GCC tool-chain; we were able to achieve fast development times as a result. In short, the FPGA of our prototyping board is filled, as described by Figure 1, with what could be considered an embedded processor with a co-processor for arithmetic in $\mathbb{F}_{3^m}$. The obvious real-world analogy of this type of architecture is a smart-card with an associated co-processor.

4.2 Results

Having selected our fields for polynomial and normal bases so that they were as close as possible in size, we took the approach of utilising as equal an amount of the FPGA as possible to make comparison easier. Since our multiplier architecture in both cases allows for scalability by altering the digit-size D, we parameterised the polynomial basis multiplier with $D = 4$ and the normal basis multiplier with $D = 2$, choices that resulted in roughly the same area cost.

Table 1 shows the performance of our arithmetic and higher level functions at a modest clock speed that could be useful in a constrained environment and the fastest possible speed resulting from our synthesis results. A given arithmetic operation essentially requires $n + 2$ cycles, 1 cycle for the instruction fetch and decode, n for the execution and 1 to write-back the result into the register file. As well as cycle and wall-clock timings, we quote the number of instructions issued by the MicroBlaze core to the ALU. The area costs are inclusive of all system elements bar the instruction memory and register file which are backed by BlockRAM. The MicroBlaze core, FSL interface and debugging unit consumes roughly 1300 slices; the finite state machine to control the ALU consumes roughly 500 slices; the ALU logic consumes roughly 1700 slices depending on which elements are included. Note that our upper clock speed was bounded by 150 MHz since this was the maximum permitted by use of the MicroBlaze.

In terms of field arithmetic, we find that the polynomial basis representation is generally faster since although the cube and cube root circuits are more complex, the dominant feature was the multiplier. The critical path of the normal basis multiplier was far longer, forcing a lower clock speed, and the design much larger, meaning the polynomial multiplier could employ a larger, more efficient digit-size. Using these results and by simply looking at the algorithms, it is clear that the Duursma-Lee algorithm will be faster than that of Kwon-BGOS since although the later removes the need for a cube root in $\mathbb{F}_q$, it requires a cubing in $\mathbb{F}_{q^k}$. Thanks to the single-cycle cube root implementations, the cube in $\mathbb{F}_{q^k}$ will inevitably be slower. Table 1 confirms this by quoting results for evaluating the pairing and for the final powering: one should view a pairing as being the combination of these two if the goal is compatibility with other algorithms.

Note that although the Kwon-BGOS algorithm is marginally slower it offers an attractive trade-off since we can omit the cube root logic from our design and save the associated slices. Also note that because of the fast cube root method of Barreto [1], the perceived advantage of a normal basis in being able to perform fast cube root operations is eliminated: the multiplier is the dominant cost as a result.

4.3 Analysis

In characteristic three, given our constrained setting, an efficient way to perform point multiplication using minimal pre-computation is to use the generalised non-adjacent form (GNAF) [9, 26], to construct a signed ternary expansion of the exponent $d \pmod{l}$. Such a representation is easy to compute and reduces the average density of non-zero trits from two thirds to one half. Using $\mathcal{A}$ to denote point addition and $\mathcal{T}$ to denote point tripling, the cost of an average point multiplication is

$$\frac{\log(d)}{\log(3)}\mathcal{T} + \frac{\log(d)}{2\log(3)}\mathcal{A}.$$

The Boneh-Franklin IBE scheme [8] is perhaps the most definitive example of the use of pairings within a concrete scheme. The trust authority or TA has a public key $P_{TA} = s \cdot P$ for a master secret s. A users public key is calculated from the string ID using a hash function as $P_{ID} = H_1(ID)$. The corresponding secret key is calculated by the TA as $S_{ID} = s \cdot P_{ID}$. To encrypt the message M, one selects a random r and computes the tuple

$$C = (U, V) = (r \cdot P, M \oplus H(e(P_{ID}, P_{TA})^r)),$$

to decrypt $C = (U, V)$, one computes the result

$$M = V \oplus H(e(S_{ID}, U)).$$

Considering our faster implementation using polynomial basis and Duursma-Lee algorithm with a modest clock speed of 16 MHz, we use $\mathcal{P}$ to denote the combination of pairing and final powering, $\mathcal{M}$ a point multiplication and $\mathcal{E}$ a field exponentiation. Using this notation we see that encryption costs $2\mathcal{M} + \mathcal{P}$ while decryption costs $\mathcal{P}$. Although we do not consider it as an option, given some extra storage the pairing required for encryption can be pre-computed which results in the cost being $\mathcal{M} + \mathcal{E}$. Using these costs and our timings from Table 1, we find that using our architecture we can perform Boneh-Franklin encryption in $\approx 7ms$ and decryption in $\approx 4ms$.

This performance is easily enough for practical applications since a given scheme will typically try to minimise the number of pairings executed. Thus, one can consider making a trade-off between performance and cost to reduce the device size. For example, we can remove the cube root logic as described above and utilise the Kwon-BGOS algorithm. Additional optimisations in this direction include: reduction of the digit-size in our multiplication units; sharing a group of addition cells between the addition and multiplication operations, at the moment we place individual copies for each; improving the register allocation strategy or spilling values to the main memory so as to reduce the size of our register file containing $\mathbb{F}_q$ elements; and further turning of the MicroBlaze to eliminate the debug and RS232 logic used for development purposes only.

5 Conclusions

We have presented an accelerator for arithmetic in $\mathbb{F}_{3^m}$ and used it to implement the Tate pairing, a primitive which is of increasing importance in cryptographic schemes. Unlike previous work, we investigate both polynomial and normal basis representations of field elements and both the Duursma-Lee and Kwon-BGOS algorithms to compute the pairing. Our results demonstrate roughly a ten-fold improvement on the only other known hardware implementation [17] and orders of magnitude better than the fastest known software implementations.

The issue of size of slightly harder to quantify due to the use of FPGA as a target. Although our design is clearly still unrealistically large to place on a smart-card for example, we have demonstrated that our performance margin is so great, trade-offs that significantly reduce the area are viable. We leave the realisation of such optimisations for further work which might also include other marginal issues: acceleration of inversion in $\mathbb{F}_{3^m}$ using Euclidean techniques rather than by powering, perhaps by using extra hardware [17]; some comparison with existing, proprietary smart-card hosted implementations of the Tate pairing [27].

Acknowledgements

The authors would like to thank Rob Granger, Johann Großschädl, Elisabeth Oswald, Nigel Smart, Martijn Stam and Fré Vercauteren for invaluable help and support throughout the course of this work.

References

1. P.S.L.M. Barreto. A Note On Efficient Computation Of Cube Roots In Characteristic 3. In *Cryptology ePrint Archive*, Report 2004/305, 2004.
2. P.S.L.M. Barreto, S. Galbraith, C. O'hEigeartaigh and M. Scott. Efficient Pairing Computation on Supersingular Abelian Varieties. In *Cryptology ePrint Archive*, Report 2004/375, 2004.
3. P.S.L.M. Barreto, B. Lynn. M. Scott. Constructing Elliptic Curves with Prescribed Embedding Degree. In *Security in Communication Networks (SCN)*, Springer-Verlag LNCS 2576, 257–267, 2002.
4. P.S.L.M. Barreto, H. Kim, B. Lynn and M. Scott. Efficient Algorithms for Pairing-Based Cryptosystems. In *Advances in Cryptology (CRYPTO)*, Springer-Verlag LNCS 2442, 354–368, 2002.
5. P.S.L.M. Barreto and M. Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In *Cryptology ePrint Archive*, Report 2005/133, 2005.
6. G. Bertoni, J. Guajardo, S. Kumar, G. Orlando, C. Paar and T. Wollinger. Efficient $GF(p^m)$ Arithmetic Architectures for Cryptographic Applications. In *Topics in Cryptology (CT-RSA)*, Springer-Verlag LNCS 2612, 158–175, 2003.
7. I.F. Blake G. Seroussi and N.P. Smart. *Advances in Elliptic Curve Cryptography.* Cambridge University Press, 2004.
8. D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. In *SIAM Journal on Computing*, **32**(3), 586–615, 2003.

9. W. Clark and J. Liang. On Arithmetic Weight for a General Radix Representation of Integers. In *IEEE Transactions on Information Theory*, **19**, 823–826, 1973.
10. I. Duursma and H. Lee. Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$. In *Advances in Cryptology (ASIACRYPT)*, Springer-Verlag LNCS 2894, 111–123, 2003.
11. R. Dutta, R. Barua and P. Sarkar, Pairing-Based Cryptographic Protocols : A Survey. In *Cryptology ePrint Archive*, Report 2004/064, 2004.
12. S. Galbraith. Supersingular Curves in Cryptography. In *Advances in Cryptology (ASIACRYPT)*, Springer-Verlag LNCS 2248, 495–513, 2001.
13. R. Granger, D. Page and M. Stam. Hardware and Software Normal Basis Arithmetic for Pairing Based Cryptography in Characteristic Three. In *Cryptology ePrint Archive*, Report 2004/157, 2004.
14. R. Granger, D. Page and M. Stam. On Small Characteristic Algebraic Tori in Pairing-Based Cryptography. In *Cryptology ePrint Archive*, Report 2004/132, 2004.
15. K. Harrison, D. Page and N.P. Smart. Software Implementation of Finite Fields of Characteristic Three, for use in Pairing Based Cryptosystems. In *LMS Journal of Computation and Mathematics*, **5** (1), 181–193, London Mathematical Society, 2002.
16. T. Itoh and S. Tsujii. A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^n)$ Using Normal Bases. In *Information and Computation* **78**, 171–177, 1988.
17. T. Kerins, E. Popovici and W.P. Marnane. Algorithms and Architectures for Use in FPGA Implementations of Identity Based Encryption Schemes. In *Field Programmable Logic and Application (FPL)*, Springer-Verlag LNCS 3203, 74–83, 2004.
18. S. Kwon. Efficient Tate Pairing Computation for Supersingular Elliptic Curves over Binary Fields. In *Cryptology ePrint Archive*, Report 2004/303, 2004.
19. A. Miyaji, M. Nakabayashi and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. In *IEICE Transactions on Fundamentals*, **E84-A** (5), 1234–1243, 2001.
20. A. Menezes, T. Okamoto, and S. A. Vanstone. Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. In *IEEE Transactions on Information Theory*, **39**, 1639–1646, 1993.
21. M. Nöcker. Data Structures for Parallel Exponentiation in Finite Fields. PhD Thesis, Universität Paderborn, 2001.
22. D. Page and N.P. Smart. Hardware Implementation of Finite Fields of Characteristic Three. In *4th Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, Springer-Verlag LNCS 2523, 529–539, 2002.
23. R. Sakai, K. Ohgishi and M. Kasahara. Cryptosystems Based on Pairings. In *Symposium on Cryptography and Information Security (SCIS)*, 2000.
24. J. Silverman. *The Arithmetic of Elliptic Curves*. Springer-Verlag GTM 106, 1986.
25. A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In *Advances in Cryptology (CRYPTO)*, Springer-Verlag LNCS 196, 47–53, 1985.
26. T. Takagi, S-M. Yen and B-C. Wu. Radix-r Non-Adjacent Form. In *Information Security Conference (ISC)*, Springer-Verlag LNCS 3225, 99–110, 2004.
27. Voltage Security, Press Release, *Gemplus Develops the World's First Identity-Based Encryption for Smart Cards*. Available from `http://www.voltage.com/about/pressreleases/PR041102.htm`.

Efficient Hardware for the Tate Pairing Calculation in Characteristic Three

T. Kerins[1], W. P. Marnane[1], E. M. Popovici[2], and P.S.L.M. Barreto[3]

[1] Dept. of Electrical and Electronic Engineering,
University College Cork, Cork City, Ireland
{timk, liam}@rennes.ucc.ie
[2] Dept. of Microelectronic Engineering,
University College Cork, Cork City, Ireland
e.popovici@ucc.ie
[3] Dept. Computing and Digital Systems Engineering,
Escola Politécnica, Universidade de São Paulo,
São Paulo, Brazil
pbarreto@larc.usp.br

Abstract. In this paper the benefits of implementation of the Tate pairing computation on dedicated hardware are discussed. The main observation lies in the fact that arithmetic architectures in the extension field $GF(3^{6m})$ are good candidates for parallelization, leading to a similar calculation time in hardware as for operations over the base field $GF(3^m)$. Using this approach, an architecture for the hardware implementation of the Tate pairing calculation based on a modified Duursma-Lee algorithm is proposed.

Keywords: Tate pairing, hardware accelerator, characteristic three, tower fields.

1 Introduction

In recent years an ever increasing number of pairing based cryptosystems have appeared in the literature, see [8]. In turn this has driven research into efficient algorithms for the implementation of bilinear pairings on elliptic curves. The Tate pairing (originally introduced to cryptography by Frey and Rück in [10]) has attracted attention as an efficiently computable bilinear pairing and over supersingular elliptic curves it achieves its maximum security in characteristic three.

Until 2002 the best method of Tate pairing computation on elliptic curves was via the algorithm of Miller [21]. In 2002 the work of Galbraith *et al.* and Barreto *et al.* furthered this development so that the Tate pairing became easier to compute in practice [11,1]. As described in the BKLS/GHS algorithms, prudent choice of points, by use of a distortion map of the type discussed in [27], as well as a triple-and-add algorithm in characteristic three greatly simplifies the pairing calculation. The utilization of so called tower fields of $GF(3^m)$ for arithmetic in $GF(3^{6m})$ was proposed in [11,23].

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 412–426, 2005.

In 2003 further improvements in the implementation of the Tate pairing were described by Duursma and Lee in [9], leading to the Duursma-Lee algorithm for Tate pairing computation. Here, the pairing computation was extended to more general hyperelliptic curves. Also, the distortion map was incorporated into the operation of the algorithm itself, as well as modifying the loop of the BLKS/GHS algorithms, to yield a more efficient implementation. Further enhancements to the Duursma-Lee algorithm for supersingular elliptic curves over fields of characteristic three were described in [20,25,12]. As will be described in this paper, this modified Duursma-Lee algorithm described in [20] is an excellent candidate for implementation on dedicated hardware. Further work on even more efficient general pairing algorithms appeared recently in [3].

Despite the large body of work regarding improving the algorithmic efficiency of the Tate pairing computation, to date the hardware implementation of such algorithms, particularly over characteristic three, have received scant attention in the literature. This is somewhat surprising given the well known speed and security advantages of dedicated cryptographic hardware [24]. The main contribution of this paper is the description of how the modified Duursma-Lee algorithm in characteristic three can be efficiently implemented in hardware, and a number of conclusions are then derived about the expected calculation time of such an architecture. The architecture described in this paper has application as a hardware accelerator for pairing based cryptographic protocols in an internet server, where high speed pairing calculation is the primary design consideration.

2 Related Work

Hardware architectures for polynomial basis arithmetic in characteristic three have appeared in [23,5,16,18] while architectures for normal basis arithmetic have appeared in [13]. In hardware and indeed software, the basis representation is a significant design choice. For this paper, the polynomial basis representation of $GF(3^m) \cong GF(3)[x]/f(x)$ was chosen, where $f(x)$ is a degree m irreducible polynomial over $GF(3)$. Polynomial basis multiplication in $GF(3^m)$ is possible in $d = \lceil m/D \rceil$ clock cycles for a digit size D following the architectures outlined by Bertoni *et al.* in [5]. The coefficient serial multiplier discussed in [16,18] is a special case of this. As will be described in Section 3 the primary required operations over $GF(3^m)$ for the modified Duursma-Lee algorithm are addition, subtraction, multiplication and cubing.

It has been outlined in [5,16,18,13] that addition and subtraction (and also negation as a special case) can be efficiently performed in hardware by small combinational gate circuits, using various two bit binary encoding of $GF(3)$ elements and that the gate delay for these addition and subtraction architectures is low. This implies that additive operations in $GF(3^m)$ arithmetic hardware can be performed almost for free. Arithmetic in $GF(3)$ can be performed in two 4:1 FPGA lookup tables using 2-bit encoding [16] and will not significantly contribute to a processor's calculation time. In hardware, elements of $GF(3^m)$ can be represented in $2m$ bits, hence additive operations over $GF(3^m)$ cost $4m$

4:1 FPGA lookup tables. This is a relatively small hardware cost compared to that of multiplicative architectures (see Section 4.2).

In [5] a digit serial multiplier over $GF(3^m)$ is described. This considers multiplication over $GF(3^m)$ as a series of matrix-vector multiplications with coefficients in $GF(3)$. This can also be implemented efficiently in hardware assuming a low weight irreducible polynomial $f(x) \in GF(3)[x]$ (trinomial or pentanomial) has been used to define arithmetic in $GF(3^m)$. Under this assumption cubing circuitry in $GF(3^m)$ can also be efficiently implemented in much less hardware than general multiplication and cubing can be performed in a single clock cycle. An efficient algorithm and hardware architecture for inversion in $GF(3^m)$ in $2m$ clock cycles based on the extended Euclidean algorithm appeared recently in [16,18]. Few full hardware processor architectures for Tate pairing calculation in characteristic three have appeared in the literature. However, an FPGA implementation of a pairing based cryptosystem coprocessor architecture based on the binary BLKS/GHS algorithm appears in [19].

3 Tate Pairing Calculation by Modified Duursma-Lee Algorithm

This section presents an outline of the modified Duursam-Lee algorithm along with some observations regarding its efficient calculation in hardware.

3.1 The Tate Pairing

Following from [1,2,3] the modified Tate pairing is defined on the supersingular elliptic curve $E_\pm$ in affine coordinates defined over a Galois field $GF(3^m)$, where in practice m is generally prime

$$E_\pm : y^2 = x^3 - x \pm 1 \tag{1}$$

The set of points on $E_\pm$, along with the point at infinity $\mathcal{O}$, form a group of order $\#E_\pm$ under the well known chord-tangent law of composition [26]. The curve (1) is chosen so that it contains a large cyclic subgroup of prime order l. Also l^2 does not divide $\#E_\pm$ but l divides $3^{6m} - 1$ and not any $3^{jm} - 1$, $j < 6$. In order to resist discrete logarithm solving attacks it is recommended that l is at least 150 bits long [6].

Now $E_\pm(GF(3^m))$ contains an l-torsion group $E_\pm[l](GF(3^m))$ and similarly $E_\pm(GF(3^{6m}))$ contains an l torsion group $E_\pm[l](GF(3^{6m}))$. Following [1] for our purposes, the Tate pairing of order l is defined as a bilinear map between $E_\pm[l](GF(3^m))$ and $E_\pm[l](GF(3^{6m}))$ to an element of the multiplicative subgroup of $GF(3^{6m})$, i.e. $GF(3^{6m})^*$

$$E_\pm[l](GF(3^m)) \times E_\pm[l](GF(3^{6m})) \rightarrow GF(3^{6m})^* \tag{2}$$

It is only defined up to l^{th} powers of unity; to obtain a unique value in $GF(3^{6m})$ suitable for cryptographic applications it is necessary to raise it to the power $\epsilon = (3^{6m} - 1)/l$.

Now consider $P = (x_p, y_p), R = (x_r, y_r) \in E_{\pm}[l](GF(3^m))$, i.e. $x_p, y_p, x_r, y_r \in GF(3^m)$. The pairing is efficiently computed in practice by considering the point $\phi(R) \in E_{\pm}[l](GF(3^{6m}))$ where ϕ is a distortion map of the type introduced in [27]. The distortion map ϕ is defined as

$$\phi(R) = \phi((x_r, y_r)) = (\rho - x_r, \sigma y_r) \tag{3}$$

where $\rho, \sigma \in GF(3^{6m})$ such that $\rho^3 - \rho \mp 1 = 0$, ($\rho^3 - \rho - 1 = 0$ for E_+ (1) and $\rho^3 - \rho + 1 = 0$ for E_- (1)) and $\sigma^2 + 1 = 0$. Following [9,2,20] the modified Tate pairing is now defined on points $P, R \in E[l](GF(3^m))$ as

$$\hat{e}(P, R) = e_{3^{3m}-1}(P, \phi(R))^{\epsilon_1} = e_l(P, \phi(R))^{\epsilon} = \tau \in GF(3^{6m}) \tag{4}$$

The calculation of (4) is performed in two stages. First the modified pairing $e_{3^{3m}-1}(P, \phi(R)) = t \in GF(3^{6m})^*$ is evaluated. This is performed by the modified Duursma-Lee algorithm illustrated as Algorithm 1. Then the resulting $t \in GF(3^{6m})$ is raised to the Tate power ϵ_1, i.e. $\tau = t^{\epsilon_1}$. Tate power $\epsilon_1 = \epsilon/3^{3m} = 3^{3m} - 1$ as the Duursma-Lee algorithm benefits from the equivalence property of the Tate pairing.

Algorithm 1: The Modified Duursma-Lee Algorithm (char 3)

input: $P = (x_p, y_p), R = (x_r, y_r) \in E_{\pm}[l](GF(3^m))$
output: $t = e_{3^{3m}-1}(P, \phi(R))) \in GF(3^{6m})^*$
01 initialize : $t, \gamma, \in GF(3^{6m})$,
$\alpha = x_p, \beta = y_p, x = x_r^3, y = y_r^3, \mu = 0 \in GF(3^m)$
$d = (\pm m) \bmod 3 \in GF(3)$ (* $+m \leftrightarrow E_+$, $-m \leftrightarrow E_-$ *)
02 for i in 0 to $m - 1$ loop
03 $\alpha = \alpha^9$,$\beta = \beta^9$ (* arithmetic in $GF(3^m)$ *)
04 $\mu = \alpha + x + d$ (* arithmetic in $GF(3^m)$ *)
05 $\gamma = -\mu^2 - \beta y \sigma - \mu\rho - \rho^2$ (* arithmetic in $GF(3^{6m})$ *)
06 $t = t^3$ (* cubing in $GF(3^{6m})$ *)
07 $t = t\gamma$ (* multiplication in $GF(3^{6m})$ *)
08 $y = -y$ (* arithmetic in $GF(3^m)$ *)
09 $d = (d \mp 1) \bmod 3$ (* $d = d - 1 \leftrightarrow E_+$, $d = d + 1 \leftrightarrow E_-$ *)
10 end loop
return: t

3.2 A Tower Field Representation for $GF(3^{6m})$

As discussed in Section 2, efficient hardware architectures exist for addition, subtraction, cubing and multiplication in the base field $GF(3^m)$. However, as seen from Algorithm 1, the principal complexity in performing the modified Tate pairing (4) lies in the implementation of efficient arithmetic in $GF(3^{6m})$ as well as $GF(3^m)$. The suggestion of constructing the field $GF(3^{6m})$ as an extension field of $GF(3^m)$ appeared in [11,1] and is prudent for hardware implementation. The suggestion of the application of Karatsuba multiplication to this

arithmetic hardware appeared in [23]. In [12] much of the arithmetic developed in this section is explicitly described. Tower fields have previously been used in the implementation of Galois field arithmetic for elliptic curve cryptography in [7,14,22].

The choice of basis for construction of $GF((3^{6m}))$ from $GF(3^m)$ is motivated by a desire to simplify as much as possible the $GF(3^{6m})$ elements ρ and σ used in the distortion map ϕ (3), appearing in Step 05 of Algorithm 1. Elements of $a \in GF(3^{6m})$ are represented as $a = \sum_{i=0}^{5} a_i \zeta^i$ where $a_i \in GF(3^m)$. The basis $\{\zeta^0, \zeta^1, \zeta^2, \zeta^3, \zeta^4, \zeta^5\} = \{1, \sigma, \rho, \sigma\rho, \rho^2, \sigma\rho^2\}$ is equivalent to a tower field extension of $GF((3^m)^6) \cong GF(((3^m)^2)^3)$ where σ and ρ are zeros of $\sigma^2 + 1$ and $\rho^3 - \rho \mp 1$ as defined by the distortion map i.e.

$$GF(3^{2m}) \cong GF(3^m)[y]/g(y) \tag{5}$$

where $g(y) = y^2 + 1$ is an irreducible polynomial over $GF(3^m)$ (provided that m and 2 are coprime) and

$$GF(3^{6m}) \cong GF(3^{2m})[z]/h_{\pm}(z) \tag{6}$$

where $h_{\pm}(z) = z^3 - z \mp 1$ is an irreducible polynomial over $GF(3^{2m})$. Polynomial $h_+(z) = z^3 - z - 1$ is used for E_+ and $h_-(z) = z^3 - z + 1$ for E_- (1) provided that m and 3 are coprime.

In this basis the elements $GF(3^{6m})$ elements σ and ρ required by the distortion map so that $\sigma^2 + 1 = 0 \in GF(3^{6m})$ and $\rho^3 - \rho \mp 1 = 0 \in GF(3^{6m})$ are represented by

$$\sigma = 0\zeta^0 + 1\zeta^1 + 0\zeta^2 + 0\zeta^3 + 0\zeta^4 + 0\zeta^5 = (0, 1, 0, 0, 0, 0)$$

and

$$\rho = 0\zeta^0 + 0\zeta^1 + 1\zeta^2 + 0\zeta^3 + 0\zeta^4 + 0\zeta^5 = (0, 0, 1, 0, 0, 0)$$

The implementation of multiplication by σ and ρ in Step 05 of Algorithm 1 becomes much simpler in hardware. Consider calculation of $\gamma \in GF(3^{6m})$

$$\begin{aligned} \gamma &= -\mu^2 - \beta y\sigma - \mu\rho - \rho^2 \\ &= (-\mu^2)\zeta^0 + (-\beta y)\zeta^1 + (-\mu)\zeta^2 + (0)\zeta^3 + (-1)\zeta^4 + (0)\zeta^5 \end{aligned} \tag{7}$$

Now calculation of γ involves only two multiplications of μ^2 and βy in the $GF(3^m)$ subfield which can be carried out in parallel. The $GF(3^m)$ negation operation does not need to be clocked and can be carried out by a small amount of combinational gate circuitry. Calculation of μ from Step 04 of Algorithm 1 requires only addition over $GF(3^m)$ which can also be carried out un-clocked using a small amount of combinational logic. Multiplication of the respective $GF(3^m)$ elements by ζ in (7) can be performed by a simple rewiring in hardware. As elements of $GF(3^m)$ are represented by $2m$ bits in hardware elements of $GF(3^{6m})$ are represented in $12m$ bits.

A further advantage of using this representation from a hardware perspective is that cubing and full multiplication in $GF(3^{6m})$ (Steps 06, 07 Algorithm 1) can also be performed using only simpler cubing and multiplication operations respectively over the base field $GF(3^m)$ and similarly all these simpler operations can be carried out in parallel.

Multiplication. Consider multiplication $c = ab$ of two elements $a = \sum_{i=0}^{5} a_i \zeta^i$ and $b = \sum_{j=0}^{5} b_j \zeta^j$ of $GF(3^{6m})$ where $a_i, b_j \in GF(3^m)$. In the equivalent tower field representation from (5) and (6) elements $a \in GF(3^{6m})$ are represented as triples of elements of $GF(3^{2m})$

$$a = \underbrace{(a_0 + a_1\sigma)}_{\tilde{a}_0} + \underbrace{(a_2 + a_3\sigma)}_{\tilde{a}_1}\rho + \underbrace{(a_4 + a_5\sigma)}_{\tilde{a}_2}\rho^2$$

In this representation multiplication of $GF(3^{6m})$ elements $a = \sum_{i=0}^{2} \tilde{a}_i \rho^i$ and $b = \sum_{j=0}^{2} \tilde{b}_j \rho^j$, $\tilde{a}_i, \tilde{b}_j \in GF(3^{2m})$ is performed by Karatsuba multiplication [15] of a and b over $GF(3^{2m})$ to form a degree 4 polynomial $d = \sum_{k=0}^{4} \tilde{d}_k \rho^k$ over $GF(3^{2m})$

$$\begin{bmatrix} \tilde{d}_0 \\ \tilde{d}_1 \\ \tilde{d}_2 \\ \tilde{d}_3 \\ \tilde{d}_4 \end{bmatrix} = \begin{bmatrix} \tilde{a}_0\tilde{b}_0 \\ (\tilde{a}_1 + \tilde{a}_0)(\tilde{b}_1 + \tilde{b}_0) - \tilde{a}_1\tilde{b}_1 - \tilde{a}_0\tilde{b}_0 \\ (\tilde{a}_2 + \tilde{a}_0)(\tilde{b}_2 + \tilde{b}_0) + \tilde{a}_1\tilde{b}_1 - \tilde{a}_2\tilde{b}_2 - \tilde{a}_0\tilde{b}_0 \\ (\tilde{a}_2 + \tilde{a}_1)(\tilde{b}_2 + \tilde{b}_1) - \tilde{a}_2\tilde{b}_2 - \tilde{a}_1\tilde{b}_1 \\ \tilde{a}_2\tilde{b}_2 \end{bmatrix} \tag{8}$$

Polynomial d from (8) is then reduced modulo the irreducible polynomial $h_{\pm}(z)$ (6) over $GF(3^{2m})$ to form $c = \sum_{i=1}^{2} \tilde{c}_i \rho^i$ as illustrated in (9) for $h_+(z)$ and in (10) for $h_-(z)$

$$\begin{bmatrix} \tilde{c}_0 \\ \tilde{c}_1 \\ \tilde{c}_2 \end{bmatrix} = \begin{bmatrix} \tilde{d}_0 + \tilde{d}_3 \\ \tilde{d}_1 + \tilde{d}_3 + \tilde{d}_4 \\ \tilde{d}_2 + \tilde{d}_4 \end{bmatrix} \tag{9}$$

$$\begin{bmatrix} \tilde{c}_0 \\ \tilde{c}_1 \\ \tilde{c}_2 \end{bmatrix} = \begin{bmatrix} \tilde{d}_0 - \tilde{d}_3 \\ \tilde{d}_1 + \tilde{d}_3 - \tilde{d}_4 \\ \tilde{d}_2 + \tilde{d}_4 \end{bmatrix} \tag{10}$$

As seen from (8) the composition stage of multiplication in $GF(3^{6m})$ is performed in six multiplications, seven additions and six subtractions in $GF(3^{2m})$ while the reduction stage is performed in either five additions for $h_+(z)$ (10) or three additions and two subtractions for $h_-(z)$ in $GF(3^{2m})$. Addition and subtraction in $GF(3^{2m})$ are performed coefficient-wise so are easy and cheap to perform in hardware using arrays of simple gate circuits as previously discussed. The hardware complexity in $GF(3^{6m})$ multiplications lies in the required six multiplications in $GF(3^{2m})$. From the dataflow diagram for (8) illustrated as Figure 1 it is seen that the six required $GF(3^{2m})$ multiplications can be carried out in parallel.

Multiplication $\tilde{c} = \tilde{a}\tilde{b} \in GF(3^{2m})$ (5) of elements $\tilde{a} = a_0 + \sigma a_1$ and $\tilde{b} = b_0 + \sigma b_1$, $a_1, a_0, b_1, b_0 \in GF(3^m)$ is performed by Karatsuba multiplication in three multiplications, two additions and three subtractions in $GF(3^m)$, see (11).

$$\begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} a_0b_0 - a_1b_1 \\ (a_1 + a_0)(b_1 + b_0) - a_1b_1 - a_0b_0 \end{bmatrix} \tag{11}$$

Here both the polynomial composition and reduction steps are performed simultaneously by the observation that $\sigma^2 = -1 \in GF(3^{2m})$ from $g(y)$ in (5). Again, additive operations in $GF(3^m)$ are easily performed by simple gate circuits and

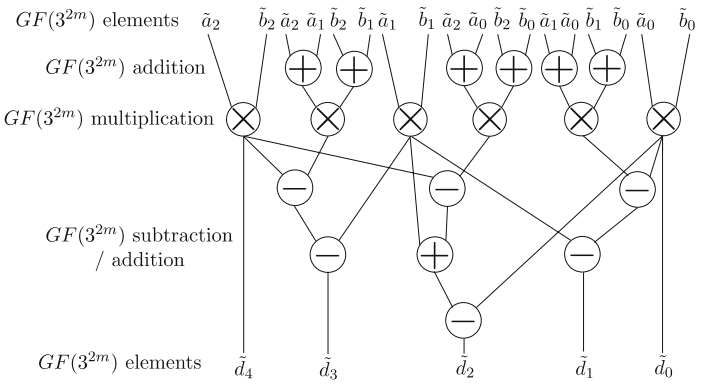


Fig. 1. Dataflow for Karatsuba composition stage of multiplication in $GF(3^{6m}) \cong GF(3^{2m})[z]/h_{\pm}(z)$

multiplication in $GF(3^m)$ can be performed as discussed in Section 2. As illustrated from Figure 2 the three required $GF(3^m)$ multiplications can be carried out in parallel.

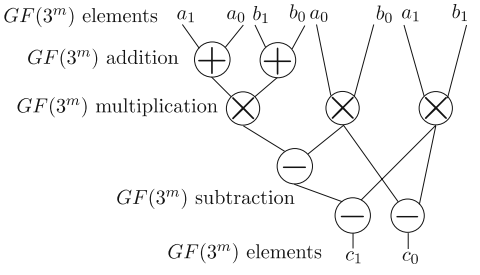


Fig. 2. Dataflow for Karatsuba multiplication in $GF(3^{2m}) \cong GF(3^m)[y]/(y^2+1)$

This implies, that by this method, multiplication in $GF(3^{6m})$ requires eighteen multiplications in the base field $GF(3^m)$ plus a number of additive operations. The advantage of implementing this operation in dedicated hardware over serial general purpose processors lies in the fact that all eighteen $GF(3^m)$ multiplications can be carried out in parallel. By parallelizing this operation the calculation time for multiplication in $GF(3^{6m})$ can be made very close to that in $GF(3^m)$. Due to the large number of $GF(3^m)$ additions/subtractions required (124 in total), it may be impractical to implement these as pure combinational logic. Depending on hardware resource usage considerations, it may be more prudent to implement a smaller number of additive gate circuits and schedule the required operations through these in an extra few clock cycles. Therefore, using the digit serial multiplier of Bertoni *et al.* [5] the hardware implementation

of multiplication in $GF(3^{6m})$ can be performed in $\lceil m/D \rceil + n_m$ clock cycles, where n_m is the relatively small number of extra clocks required for scheduling the additions/subtractions and register read/write operations. For ease of implementation these would be controlled via finite state machines.

Cubing. Cubing $c = a^3 \in GF(3^{6m}) \cong GF(3^{2m})[z]/h_{\pm}(z)$ (6) of an element $a = \sum_{i=0}^{2} \tilde{a}_i \rho^i, \tilde{a}_i \in GF(3^{2m})$ is performed by (12) for $GF(3^{6m})$ generated by polynomial $h_+(z)$ and by (13) for $GF(3^{6m})$ generated by polynomial $h_-(z)$.

$$\begin{bmatrix} \tilde{c}_0 \\ \tilde{c}_1 \\ \tilde{c}_2 \end{bmatrix} = \begin{bmatrix} \tilde{a}_0^3 + \tilde{a}_1^3 + \tilde{a}_2^3 \\ \tilde{a}_1^3 - \tilde{a}_2^3 \\ \tilde{a}_2^3 \end{bmatrix} \qquad (12) \qquad \begin{bmatrix} \tilde{c}_0 \\ \tilde{c}_1 \\ \tilde{c}_2 \end{bmatrix} = \begin{bmatrix} \tilde{a}_0^3 - \tilde{a}_1^3 + \tilde{a}_2^3 \\ \tilde{a}_1^3 + \tilde{a}_2^3 \\ \tilde{a}_2^3 \end{bmatrix} \qquad (13)$$

Each involves three cubing operations, two additions and a subtraction in $GF(3^{2m})$. As illustrated in Figures 3 and 4 in both cases the three $GF(3^{2m})$ cubing operations can be carried out in parallel.

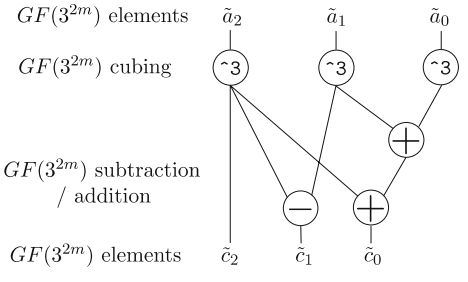


Fig. 3. Dataflow for cubing in $GF(3^{6m}) \cong GF(3^{2m})[z]/h_+(z)$

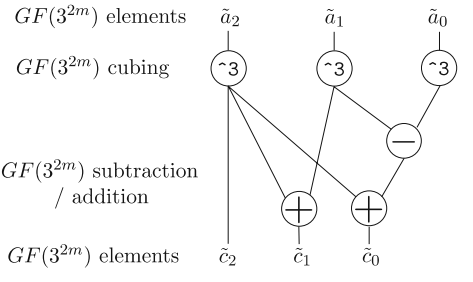


Fig. 4. Dataflow for cubing in $GF(3^{6m})[z] \cong GF(3^{2m})[z]/h_-(z)$

From (12) and (13) the main complexity of cubing in $GF(3^{6m}) \cong GF(3^{2m})[z]/h_{\pm}(z)$ lies in performing the cubing operation in the field $GF(3^{2m}) \cong GF(3^m)[y]/g(y)$ (5). Consider an element $\tilde{a} = a_0 + \sigma a_1 \in GF(3^{2m})$ generated by $g(y) = y^2 + 1$, where $a_1, a_0 \in GF(3^m)$. Now $\tilde{c} = c_0 + \sigma c_1 = \tilde{a}^3 \in GF(3^{2m})$ is calculated by

$$\begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} a_0^3 \\ -a_1^3 \end{bmatrix} \qquad (14)$$

which involves two cubing operations in $GF(3^m)$ which again can be performed in parallel. So the cubing operation in $GF(3^{6m})$ can be efficiently calculated in hardware by performing six $GF(3^m)$ cubing operations in parallel as well as three $GF(3^m)$ negation operations and six addition/subtraction operations. Following from [5] $GF(3^m)$ cubing can be performed efficiently in a single clock cycle and the additive operations can be performed by simple combinational gate circuits. Using this type of parallel cubing architecture with six $GF(3^m)$

cubing circuits $GF(3^{6m})$ cubing is performed in a single clock cycle and the six additive operations are performed by simple un-clocked gate circuits previously discussed.

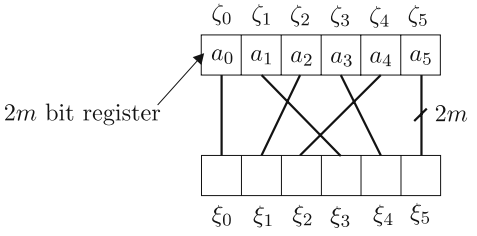


Fig. 5. Hardware rewiring for $GF(3^{6m})$ basis change from $\{\zeta_i\}$ to $\{\xi_j\}$

Raising to Tate Power. The basis $\{\zeta^0, \zeta^1, \zeta^2, \zeta^3, \zeta^4, \zeta^5\} = \{1, \sigma, \rho, \sigma\rho, \rho^2, \sigma\rho^2\}$ of $GF(3^{6m})$ over $GF(3^m)$ described by the distortion map, as previously discussed, is converted to the other basis $\{\xi, \xi^1, \xi^2, \xi^3, \xi^4, \xi^5\} = (1, \rho, \rho^2, \sigma, \sigma\rho, \sigma\rho^2)$ described by the distortion map by a simple rewiring in hardware as illustrated in Figure 5 . This is analogous to the tower field representation

$$GF(3^{3m}) \cong GF(3^m)[y]/h_{\pm}(y) \tag{15}$$

where $h_{\pm}(y) = y^3 - y \mp 1$ is an irreducible polynomial over $GF(3^m)$ ($E_+ \leftrightarrow h_+, E_- \leftrightarrow h_-$)

$$GF(3^{6m}) \cong GF(3^{3m})[z]/g(z) \tag{16}$$

where $g(z) = z^2 + 1$ is an irreducible polynomial over $GF(3^{3m})$.

In this basis $a \in GF(3^{6m})$ is represented by a pair of elements $\check{a}_0, \check{a}_1 \in GF(3^{3m})$

$$a = \underbrace{(a_0 + a_1\rho + a_2\rho^2)}_{\check{a}_0} + \underbrace{(a_3 + a_4\rho + a_5\rho^2)}_{\check{a}_1}\sigma$$

As described in [12] raising $a = \sum_{i=0}^{5} a_i\xi_i \in GF(3^{6m})$ to the Tate power $\epsilon_1 = 3^{3m} - 1$ in this basis can be performed in a much more efficient manner than typical multiply-and-accumulate methods of exponentiation by the observation that for m odd

$$a^{3^{3m}} = (\check{a}_0 + \sigma\check{a}_1)^{3^{3m}} = \check{a}_0 - \sigma\check{a}_1 \tag{17}$$

as $\sigma^2 = -1 \in GF(3^{3m})$. Thus (17) implies that $c = a^{\epsilon_1} \in GF(3^{6m})$ is calculated by

$$c = \check{c}_0 + \sigma\check{c}_1 = \frac{\check{a}_0 - \sigma\check{a}_1}{\check{a}_0 + \sigma\check{a}_1} = \left[1 + \check{a}_1^2\nu^{-1}\right] + \sigma\left[1 - (\check{a}_0 + \check{a}_1)^2\nu^{-1}\right] \tag{18}$$

where $\nu = (\check{a}_0^2 + \check{a}_1^2) \in GF(3^{3m})$. Thus raising to the Tate power ϵ_1 involves five multiplications, three additions and a subtraction and an inversion in $GF(3^{3m})$.

Multiplication in the field $GF(3^{3m})$ (15) is carried out in a similar manner to that outlined in (8),(9) and (10) except in this case the base field is $GF(3^m)$. The six required $GF(3^m)$ multiplications can be carried out in parallel and the additive operations are carried out by the gate circuits previously discussed. The calculation time for multiplication in $GF(3^{3m})$ is given as $\lceil m/D \rceil + n_m$. Inversion in $GF(3^{3m})$ is carried out by arithmetic in $GF(3^m)$ as illustrated in Appendix A. As this operation is performed only once, optimizing the calculation is not as important as for the $GF(3^{6m})$ multiplication and cubing previously discussed.

4 A Hardware Architecture for Tate Pairing Calculation Based on Duursma-Lee Algorithm

This section considers a prospective hardware implementation for Tate pairing calculation $\hat{e}(P, R) = \tau$ (4) over elliptic curves (1) based on Algorithms 1 considering the observations from Section 3.2 on the efficient calculation time achievable by parallelizing $GF(3^{6m})$ arithmetic.

4.1 Observations on the Modified Tate Pairing Calculation

It is interesting to consider the number of clock cycles required for the main iteration loop (Steps 03-09) of Algorithm 1 on a dedicated hardware architecture. Here eighteen $GF(3^m)$ digit size multipliers (digit size D, $d = \lceil m/D \rceil$) and six $GF(3^m)$ cubing circuits are available in parallel, along with a suitable amount of simpler $GF(3^m)$ arithmetic circuits for performing addition, subtraction and negation. Also required on such an architecture are $2m$ bit registers for storage of elements of $GF(3^m)$ and $12m$ bit bus lines for elements of $GF(3^{6m})$. The calculation time for an iteration of Algorithm 1 using this type of architecture is illustrated in Table 1. An extra two clock cycles are added to the calculation time of each operation for register read/write operations.

From Table 1 the modified Duursma-Lee Algorithm, Algorithm 1, can be performed on the type of dedicated hardware discussed in Section 3 in $\theta_{DL} = m(2\lceil m/D \rceil + 17 + n_m)$ clock cycles. After $e_{3^{3m}-1}(P, \phi(R)) = t \in GF(3^{6m})$

Table 1. Number of clock cycles required for an iteration of the Modified Duursma-Lee algorithm implemented on a parallel $GF(3^m)$ hardware architecture

step	operations	logic	clock cycles
03	$\alpha = \alpha^3$, $\beta = \beta^3$	$\times 2$ $GF(3^m)$ cube	1+2
	$\alpha = \alpha^3$, $\beta = \beta^3$	$\times 2$ $GF(3^m)$ cube	1+2
04	$\mu = \alpha + x + d$	combinational	0+2
05	γ see (7)	$\times 2$ $GF(3^m)$ mul	$d + 2$
06	$t = t^3$	$\times 6$ $GF(3^m)$ cube	$1 + 2$
07	$t = t\gamma$	$\times 18$ $GF(3^m)$ mul	d+n_m+2
08 09	$y = -y$, $d = d \mp 1$	combinational	0+2

has been performed, it is then necessary to raise this $GF(3^{6m})$ element to the Tate power ϵ_1 using (18). This generates the required unique result $\tau = t^{\epsilon_1} \in GF(3^{6m})$. This operation can be efficiently performed on much of the same underlying hardware as required for Algorithm 1. The only operations required are multiplication, additive operations and a single inversion in the base field $GF(3^m)$. Performing the $GF(3^m)$ multiplications as required in parallel implies that (18) can be performed in $\theta_{TP} = 9(\lceil m/D \rceil + n_m) + 2m$ clock cycles.

Assuming a worst case situation, where the register read/write operations and scheduling through the simple gate circuits take the same number of clock cycles as a multiplication operation (i.e. $n_m \approx \lceil m/D \rceil$) using this type of hardware architecture the number of clock cycles for calculation of (4) is given by

$$\begin{aligned} \theta_{TATE} &\approx \quad \theta_{DL} \quad + \quad \theta_{TP} \\ &\approx m(2\lceil m/D \rceil + 17 + n_m) + 9(\lceil m/D \rceil + n_m) + 2m \\ &\approx \quad 3m(\lceil m/D \rceil + 17) \quad + \quad 18\lceil m/D \rceil + 2m \end{aligned} \tag{19}$$

4.2 Implementation Aspects

The question remains : How practical is the parallel architecture as discussed in Section 4.1? The primary hardware complexity in this type of architecture is the implementation of the $GF(3^{6m})$ multiplier circuit using eighteen $GF(3^m)$ digit serial multipliers in parallel.

In order to gauge the feasibility of the architecture, the $GF(3^m)$ multiplier and cubing cores were captured in the VHDL hardware design language and prototyped on the Xilinx Virtex2Pro125 device [28] for the field $GF(3^{97}) \cong GF(3)[x]/x^{97} + x^{16} + 2$. The FPGA resource usage of the $GF(3^{97})$ digit serial multiplier for digit sizes $D = 1, 4, 8, 12$ is 1,006 (1% device), 1,821 (3% device), 2,655 (8% device) and 4,335 (12% device) FPGA slices, respectively. The fast $GF(3^{97})$ cubing circuitry was also implemented on this target technology and occupied 514 slices (0.5%). The $GF(3^m)$ inverter architecture of occupied 2210 (4% device) FPGA slices.

The $GF(3^{6m})$ parallel multiplier is the most complex part of the proposed architecture. It was implemented on the target technology, using eighteen $GF(3^{97})$ multipliers with a digit size of $D = 4$, and all of the additive operations were performed in parallel (multiplication in $\lceil 97/4 \rceil = 25$ clock cycles). In this case, all of the arithmetic was hardwired into the design. In total, it occupied 32,403 FPGA slices (inc. routing) which represents 58% of the target device. A post place and route clock frequency of 29.3 MHz was achieved for the $GF(3^{6m})$ multiplier and this translates into a calculation time of 0.9 μs. This preliminary result indicates that a parallel $GF(3^{97})$ multiplier using eighteen $GF(3^{97})$ multipliers with a digit size of $D = 4$ can be accommodated on the target device. The six required $GF(3^{97})$ cubing circuits and inversion circuit in total occupy approximately 7% of the device. This leaves the remaining 35% of the device for storage registers, control and arrays of gate circuits for the simple $GF(3^m)$ addition and subtraction logic.

The data path for the parallel multiplier, the principal complexity of this processor architecture, has already been fully implemented. The simpler parallel cubing logic can be implemented in a similar manner. In our experience the addition of a register bank for storage, and control via finite state machines (as in [17,19]) is a straightforward matter. Using this method of control some redesign of hardware is involved to accommodate future algorithmic improvements. However, the ability to reconfigure the target FPGA makes such possible changes easy to accommodate. Using the pessimistic (19) for the required number of clock cycles this implies that calculation of (4) on E_+ from (1) over $GF(3^{97})$ with a digit size of $D = 4$ could be performed in 12,866 clock cycles. A conservative estimated 15 MHz clock frequency for the entire processor (data path, control and memory) implies a calculation time for (4) over $E_+(GF(3^{97}))$ of 0.85 ms. This represents a considerable improvement over the calculation times of 4.05 ms and 4.33 ms reported for optimized software implementations on serial general purpose processors [12,4].

The proposed architecture could also be adapted, (with a small amount of extra control) to also perform scalar multiplication on $[k]P \in E_{\pm}(GF(3^m))$ and exponentiation $t^k \in GF(3^{6m})$ as required in most pairing based protocols. Using the formulae in [1], point cubing is performed by four cubings in $GF(3^m)$ and point addition is performed by an inversion, two multiplications and a cubing in $GF(3^m)$ (neglecting the cost of the simpler additive operations). Assuming $k \approx m = 97$ (Hasse's Theorem [6]), the base three representation of k is approximately 97 trits [11]. In general, k is chosen so that these are mostly zero (say 25% nonzero). Under these assumptions $[k]P \in E_+(GF(3^{97}))$ is performed in approximately 0.5 ms using a serial triple-and-add algorithm at the same clock frequency. Using a cube-and-multiply method for $t^k \in GF((3^{97})^6)$ this is performed using the parallel multiplier and cubing circuitry previously discussed in approximately 0.1 ms.

5 Conclusions

In this paper the suitability of the modified Duursma-Lee algorithm for implementation in dedicated hardware has been illustrated. Prudent choice of basis construction for the fields $GF(3^{6m})$ allows the efficient implementation of multiplication and cubing operations and only arithmetic in the $GF(3^m)$ subfield is required. Multiplication in $GF(3^{6m})$ can be performed by eighteen $GF(3^m)$ multipliers in parallel along with and cubing in $GF(3^{6m})$ can be performed by six $GF(3^m)$ cubing circuits in parallel, along with some combinational logic for additive operations. This leads to a low number of clock cycles for arithmetic in $GF(3^{6m})$ compared to those required on serial processors. Modern FPGA devices such as the Virtex2Pro currently have enough resources to contain an implementation of this type of parallel hardware for calculation of the modified Duursma-Lee algorithm. Assuming pessimistic operating parameters this type of dedicated parallel hardware is projected to drastically reduce the calculation time currently possible using optimized software implementations.

Acknowledgement

The authors would like to acknowledge the help of the anonymous referees, whose comments were valuable in the preparation of this paper. This research was funded as part of an Enterprise Ireland Research innovation fund project.

References

1. P. S. L. M. Barreto, H.Y. Kim, B. Lynn and M. Scott. Efficient implementation of pairing based cryptosystems. In *Advances in Cryptology - CRYPTO 2002* volume 2442 of *Lecture Notes in Computer Science*, pages 354-368. Springer-Verlag 2002.
2. P. S. L. M. Barreto. The well-tempered pairing. Bochum, Germany 2004. 8th Workshop on Elliptic Curve Cryptography - ECC 2004. Invited talk.
3. P. S. L. M. Barreto, S. Galbraith, C. O hEigeartaigh and M. Scott. Efficient Pairing Computation on Supersingular Abelian Varieties. Cryptology ePrint Archive, Report 375/2004. 2004. `http://eprint.iacr.org/2004/375`.
4. P. S. L. M. Barreto. A note on efficient computation of cube roots in characteristic 3. Cryptology ePrint Archive, Report 035/2004 2004. `http://eprint.iacr.org/2004/305`.
5. G. Bertoni, J. Guajardo, S. Kumar, G. Orlando C. Paar and T. Wollinger. Efficient $GF(p^m)$ Arithmetic Architectures for Cryptographic Applications. In *Topics in Cryptology - CT RSA 2003* volume 2612 of *Lecture Notes in Computer Science* pages 158-175. Springer-Verlag 2003.
6. I. Blake, G. Seroussi and N. Smart. *Elliptic Curves in Cryptography* volume 265 of *London Mathemtatical Lecture Note Series*, Cambridge University Press, 1999.
7. E. DeWin, A. Bosselaers, S. Vandenberghe, P. De Gersem, and J. Vandewalle. A fast software implementation for arithemtic operations in $GF(2^n)$. In *Advances in Cryptology - Asiacrypt 1996*, volume 1163 of *Lecture Notes in Computer Science*, pages 65-76. Springer-Verlag, 1996.
8. R. Dutta, R. Barua and P. Sarkar. Pairing-based cryptography : A survey. Cryptology ePrint Archive, Report 2004/064 2004. `http://eprint.iacr.org/2004/64`.
9. I. Duursma and H.-S. Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In *Advances in Cryptology - Asiacrypt 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 111-123. Springer-Verlag, 2003.
10. G. Frey and H. Rück. A remark considering m-divisibility in the divisor class group of curves. *Mathematics of Computation,* 62:865-874, 1994.
11. S. Galbraith, K. Harrison and D. Soldera. Implementing the Tate pairing. In *Algorithm Number Theory Symposium - ANTS V*, volume 2369 of *Lecture Notes in Computer Science*, pages 324-337. Springer-Verlag 2002.
12. R. Granger, D. Page and M. Stam. On Small Characteristic Algebraic Tori in Pairing-Based Cryptography. Cryptology ePrint Archive, Report 2004/132, 2004. `http://eprint.iacr.org/2004/132`
13. R. Granger, D. Page and M. Stam. Hardware and Software Normal Basis Arithemtic for Paring Based Cryptography in Characteristic Three. Cryptology ePrint Archive, Report 157/2004. 2004. `http://eprint.iacr.org/2004/157`.
14. J. Guajardo and C. Paar. Efficient Algorithms for Elliptic Curve Cryptosystems. In *Advances in Cryptology - Crypto 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 342-355. Springer-Verlag, 1997.

15. A. Karatsuba and Y. Ofman. Multiplication of Multidigit numbers on Automata. *Sov. Phys. Dokl (english translation)*, 7(7):595-596, 1963
16. T. Kerins, E. M. Popovici and W. P. Marnane. Algorithms and Architectures for use in FPGA implementations of Identity Based Encryption Schemes. In *Field Programmable Logic and Applications- FPL 2004* volume 3203 of *Lecture Notes in Computer Science*, pages 74-83, Springer-Verlag 2004.
17. T. Kerins, E. M. Popovici, W. P. Marnane. An FPGA Implementation of a Flexible Secure Elliptic Curve Crpytography Processor. In *Applied Reconfigurable Computing - ARC 2005*, pages 22-30, IADIS press 2005.
18. T. Kerins, W. P. Marnane and E. M. Popovici. Hardware Architectures for Arithemtic in $GF(p^m)$ for use in Public Key Cryptography. Preprint. 2004.
19. T. Kerins, W. P. Marnane, E. M. Popovici and P. S. L. M. Barreto. A Hardware Accelerator for Pairing Based Cryptosystems. Preprint. 2005.
20. S. Kwon. Efficient Tate pairing computation for supersingular elliptic curves over binary fields. Cryptology ePrint Archive, Report 2004/303, 2004. `http://eprint.iacr.org/2004/303`.
21. V. S. Miller. Short Programs for functions on curves. Unpublished manuscript. 1986. `http://crypto.stanford.edu/miller/miller.pdf`.
22. C. Paar and P. Soria-Rodriguez. Fast Arithemtic Architecturs for Public Key Algorithms over Galois Fields $GF((2^n)^m)$. In *Advances in Cryptology - Eurocrypt 1997* volume 1233 of *Lecture Notes in Computer Science* pages 363-378. Springer-Verlag 1997.
23. D. Page and N. P. Smart. Hardware implementation of Finite Fields of Characteristic Three. In *Cryptographic Hardware and Embedded Systems - CHES 2002* volume 2523 of *Lecture Notes in Computer Science* pages 529-539. Springer-Verlag 2002.
24. B. Schneier. *Appplied Cryptography* second edition. John Wiley & Sons, 1996.
25. M. Scott and P. S. L. M. Barreto. Compressed Pairings. In *Advances in Cryptology CRYPTO'2004* volume 3152 of *Lecture Notes in Computer Science*, pages 140-156. Springer-Verlag, 2004. Updated version: Cryptology ePrint Archive, Report 2004/032. `http://eprint.iacr.org/2004/303`
26. J. H. Silverman. *The Arithemtic of Elliptic Curves*. Number 106 in Graduate Studies in Mathematics. Springer-Verlag, Berlin, Germany, 1986.
27. E. R. Verheul. Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. In *Advances in Cryptology - Eurocrypt 2001* volume 2045 of *Lecture Notes in Computer Science*, pages 195-210. Springer-Verlag 2001.
28. Xilinx Inc. Virtex-2 Platform FPGAs : Complete Data Sheet. Ds031. 2004 `http://www.xilinx.com/bvdocs/publications/ds031.pdf`

Appendix A

Element $\check{c} = c_0 + c_1\rho + c_2\rho^2 = \check{a}^{-1}$ the inverse of $\check{a} = a_0 + a_1\rho + a_2\rho^2 \in GF(3^{3m})$ is calculated efficiently by the observation that $\check{c}\check{a} = 1 \in GF(3^{3m})$. The $GF(3^m)$ coefficients of $\check{c} \in GF(3^{3m})$ defined by $h_{\pm}(y)$ from (15) are calculated as illustrated in (20) for $h_+(y)$:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \delta_+^{-1} \begin{bmatrix} a_0^2 + a_2^2 - a_0a_2 - a_1(a_1 + a_2) \\ -a_0a_1 + a_2^2 \\ a_1^2 - a_0a_2 - a_2^2 \end{bmatrix} \tag{20}$$

where $\delta_+ = (a_0 - a_2)a_0^2 + (-a_0 + a_1)a_1^2 + (a_0 - a_1 + a_2)a_2^2$ and by (21) for polynomial $h_-(y)$:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \delta_-^{-1} \begin{bmatrix} a_0^2 - a_1^2 + (a_1 - a_0)a_2 + a_2^2 \\ -a_0 a_1 - a_2^2 \\ a_1^2 - a_0 a_2 - a_2^2 \end{bmatrix} \tag{21}$$

where $\delta_- = (a_0 - a_2)a_0^2 + (-a_0 - a_1)a_1^2 + (a_0 + a_1 + a_2)a_2^2 \in GF(3^m)$

Calculation of δ_+ and δ_- from (20) and (21) involves six multiplication operations in $GF(3^m)$ then these are inverted in $2m$ clock cycles using the $GF(3^m)$ inversion architecture discussed in [16,18]. The calculation of $\check{c}$ in (20) and (21) then involves a further six $GF(3^m)$ multiplication operations. In hardware this operation can be partly parallelized by performing three multiplication operations in parallel. This implies that inversion in $GF(3^m)$ can be performed in $4(\lceil m/D \rceil + n_m) + 2m$ clock cycles.

AES on FPGA from the Fastest to the Smallest

Tim Good and Mohammed Benaissa

Department of Electronic & Electrical Engineering,
University of Sheffield, Mappin Street, Sheffield, S1 3JD, UK
{t.good, m.benaissa}@sheffield.ac.uk

Abstract. Two new FPGA designs for the Advanced Encryption Standard (AES) are presented. The first is believed to be the fastest, achieving 25 Gbps throughput using a Xilinx Spartan-III (XC3S2000) device. The second is believed to be the smallest and fits into a Xilinx Spartan-II (XC2S15) device, only requiring two block memories and 124 slices to achieve a throughput of 2.2 Mbps. These designs show the extremes of what is possible and have radically different applications from high performance e-commerce IPsec servers to low power mobile and home applications. The high speed design presented here includes support for continued throughput during key changes for both encryption and decryption which previous pipelined designs have omitted.

Keywords: Advanced Encryption Standard (AES), Field Programmable Gate Array (FPGA), finite field, design exploration, high throughput, pipelined, low area, Application Specific Instruction Processor (ASIP).

1 Introduction

The research objective is to explore the design space associated with the Advanced Encryption Standard (AES) algorithm and in particular its Field Programmable Gate Array (FPGA) hardware implementation in terms of speed and area.

The Rijndael cipher algorithm developed by Vincent Rijmen and Joan Daemen won the competition run by the US government (NIST) in 2000 to select a new commercial cryptographic algorithm and was accorded the accolade the Advanced Encryption Standard (AES). This algorithm is documented in the freely available US government publication, FIPS-197 [1].

Subsequently, the AES has been the topic of much research to find suitable architectures for its hardware implementation. Architectural choices for a given application are driven by the system requirements in terms of speed and the resources consumed. This can simply be viewed as throughput and area, however, latency may also be important as may the cipher's mode of operation. The FIPS-197 specification details a number of modes of operation for the cipher, for example, the simplest is the Electronic Code Book (ECB). Additional resilience to attack can be gained by using one of the feedback modes, for example, Output Feed Back (OFB) mode unfortunately such modes also limit the effectiveness of pipelining.

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 427 – 440, 2005.

The use of FPGA has been expanding from its traditional role in prototyping to mainstream production. This change is being driven by commercial pressures to reduce design cost, risk and achieve a faster time to market. Advances in technology have resulted in mask programmed mass produced versions of FPGA fabrics being offered by the leading manufacturers which, for some applications, remove the necessity to move prototype designs from FPGA to ASIC whilst still achieving a low unit cost.

Previous attempts [2,3] at high speed pipelined design have been to use what is an effectively ASIC number-of-gates-in-critical-path design flow to place the pipeline cuts. This is fine where the target device is an ASIC, however, does not result in optimal pipeline cut placement for a given FPGA fabric. This paper presents an alternative flow specific to FPGA which results in optimal pipeline placement thus increased performance. The new high speed design reported here achieves a throughput of 25 Gbps on a Xilinx Spartan-III FPGA and has applications in the area of hardware accelerators for IPsec servers.

An additional novelty of the new high speed design presented in this paper is that the key may be periodically changed without loss of throughput and the operating mode may be changed between encryption and decryption at will. This enables the design to support a mode of operation where a batch of blocks may be encrypted or decrypted for each of a number of differently keyed concurrent channels without loss in throughput.

Reported low area architectures [4,5] have been based around a 32-bit datapath. As the AES operations MixColumns and KeyExpansion are fundamentally 32-bit it was previously believed that this was optimal. An ASIC design by Feldhofer et al [6] used an 8-bit datapath married to a 32-bit MixColumns operator. However, even MixColumns may be rewritten in an 8-bit form accepting a higher control overhead and reduced throughput. To the authors' knowledge, no 8-bit Application Specific Instruction Processor (ASIP) for AES has been reported in the literature. The results from the design of such a processor, which is believed to be the smallest, are documented in this paper. This design only occupies 60% of the smallest available Xilinx Spartan-II device (XC2S15) and achieves a throughput of 2.2 Mbps which is suitable for numerous applications in the mobile and home communications areas. This 8-bit design was compared to the two latest reported low area FPGA designs [4,5] which were based on a 32-bit architecture. Brief details of these designs are included together with their area cost results. A rival 'PicoBlaze' implementation is also presented as a benchmark to demonstrate the performance of a soft core microcontroller based design.

This paper concludes with a discussion on the relative merits of each architecture.

2 The Design

The intention here is to contrast a number of different architectures from the highest speed to the lowest area. An FPGA design flow is used throughout and performance results are presented together with comparison with the previously known best designs. The designs presented all support a 128-bit key. Xilinx ISE version 6.3 was used for the design flow and the results quoted are from post place and route figures

including all input and output delays. The new designs were coded in VHDL and validated using ModelSim.

2.1 Fully Parallel Loop Unrolled Architecture

FPGAs are particularly fast in terms of throughput when designs are implemented using deep pipeline cuts [2, 3, 7, 8, 9, 10, 11, 12]. The attainable speed is set by the longest path in the design so it is important to place the pipeline cuts such that the path lengths between the pipeline latches are balanced.

First, the algorithm must be converted into a suitable form for deep pipelining [2,7]. This is achieved by removing all the loops to form a loop-unrolled design where the data is moved through the stationary execution resources. On each clock cycle, a new item of data is input and progresses over numerous cycles through the pipeline resulting in the output of data each cycle, however, with some unavoidable latency.

One of the key optimisations was to express the SubBytes operation in computational form rather than as a look-up table. Earlier implementations used the look-up table approach (the "S" box) but this resulted in an unbreakable delay due to the time required pass through the FPGA block memories. The FIPS-197 specification provided the mathematical derivation for SubBytes in terms of Galois Field (2^8) arithmetic. This was efficiently exploited by hardware implementations using composite field arithmetic [2,7] which permitted a deeper level of pipelining thus improved throughput.

The method of placement of pipeline latches (or cuts) was to consider the synthesis estimates for various units within the design. In particular, for Xilinx FPGAs, the number of cascaded 4-input LUTs in the critical path together with routing delays dominate the minimum cycle period. The first stage of optimisation is to consider the routing delay as constant and only consider change in the number of cascaded LUTs. In further optimisation design cycle iterations, the secondary effects of excessive routing delays and fan out load were considered.

A simple function, such as the reduction-OR of a bit vector, can be used to generate LUT-levels versus cycle time results for the internal fabric of a specific

Table 1. Virtex-E performance versus logic levels

Logic Levels	Path Delay, ns	Max Clock Freq, MHz
1	2.176	459.6
2	3.321	301.1
3	4.466	223.9
4	5.611	178.2
5	6.756	148.0
6	7.901	126.6
7	9.046	110.5
8	10.191	98.1

technology. Table 1 shows such results for Xilinx Virtex-E. As can be seen from the table having pipeline registers between each LUT would yield the fastest design, however, there is a compromise in terms of the amount of fan-in required by the logic expressions in the design, the acceptable latency and realistic routing.

The Virtex FPGA slice consists of two LUTs and one D-type flip-flop (FD) so a single level of logic between FDs would under utilise cells resulting in an approximate factor of two increase in the design area thus an impact on speed due to the larger distances. Similarly, two levels of logic between FDs would not provide sufficient flexibility (number of input terms in an expression) for the AES algorithm thus is likely to result in a significant increase of area. Further, with only two LUT levels routing, propagation time, fan-out and congestion from a lack of suitable routing resources are very likely to dominate the cycle time. This leaves three logic levels as the aiming point for pipeline register placement.

There is a further complication in that the slice architecture includes a number of multiplexer (MUX) resources in addition to the LUTs these can be used to implement 2-input XOR and 2-input MUX functions without recourse to an extra level of LUTs. This factor must also be considered when placing the pipeline cuts.

For a given set of pipeline cuts the synthesis results may be examined to verify that the critical path contains the correct number of cascaded LUTs. This design process yielded the following optimal cut set; Figure 1 shows the composite field implementation of SubBytes [2] followed by ShiftRows (SR) and MixColumns (M) operations. The number of LUT-levels is shown adjacent to each design unit and the total in a given pipeline step (represented by the dashed lines) at the bottom of the diagram. From an initial implementation it was found that additional time had to be allowed for the excessively long routing associated with the ShiftRows operation. Thus both the ShiftRows and its inverse require extra time compared to the remainder of the design. The excess time is approximately equivalent to a time associated with using two LUTs. The circuit shown can perform both encryption and decryption operations.

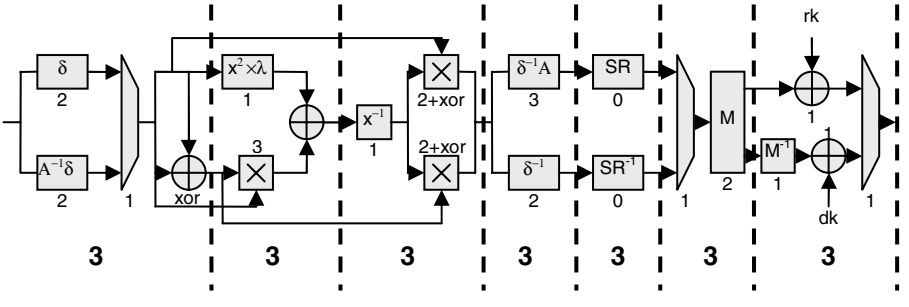


Fig. 1. Block diagram for each middle round

The same treatment was given to the placement of pipeline cuts in the final round (Figure 2) which conveniently turned out to require one less cut than the middle rounds. This was used to accommodate the single cut required for the first round to yield a regular timing pattern.

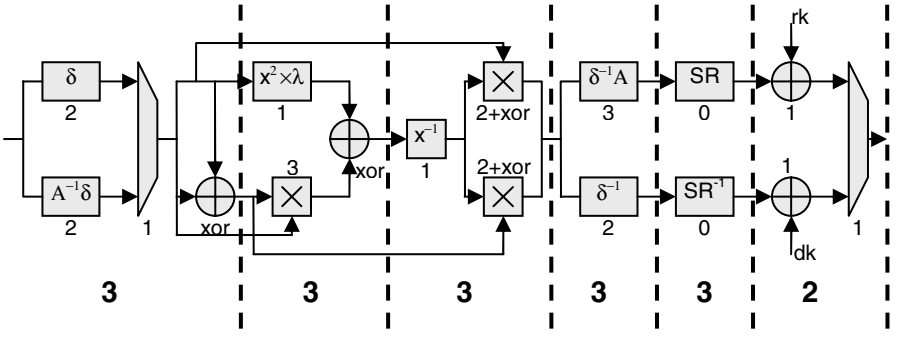


Fig. 2. Block diagram for final round

The key expansion also required implementing and in some previous designs had been overlooked. One key design decision was how frequently the key must be changed and whether continued throughput is required. In this design, it was decided that throughput should be maintained during key changes and that it was desirable to change between encryption and decryption on each cycle with key changes made on similar order to the latency.

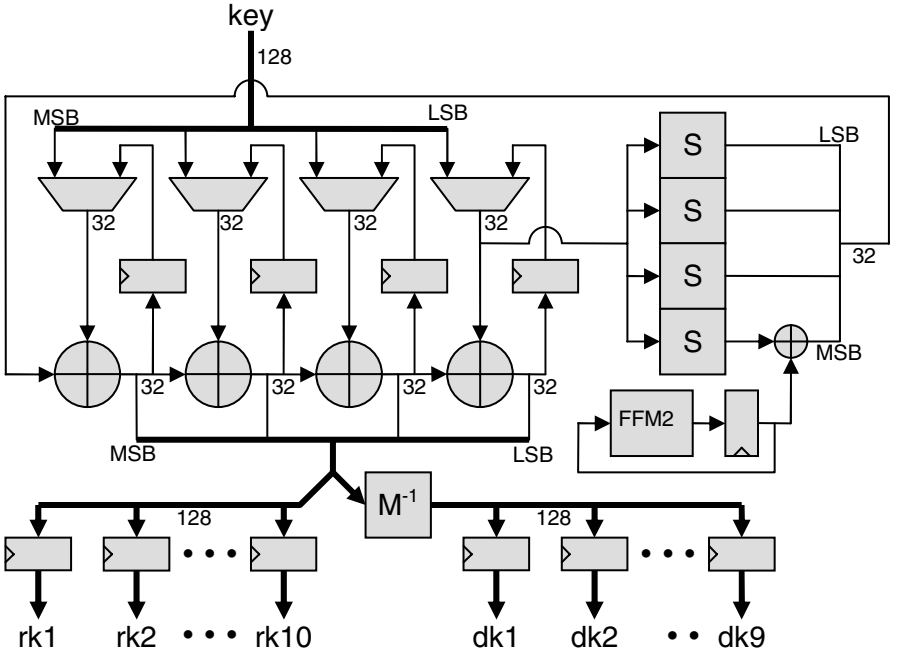


Fig. 3. Block Diagram of KeyExpander

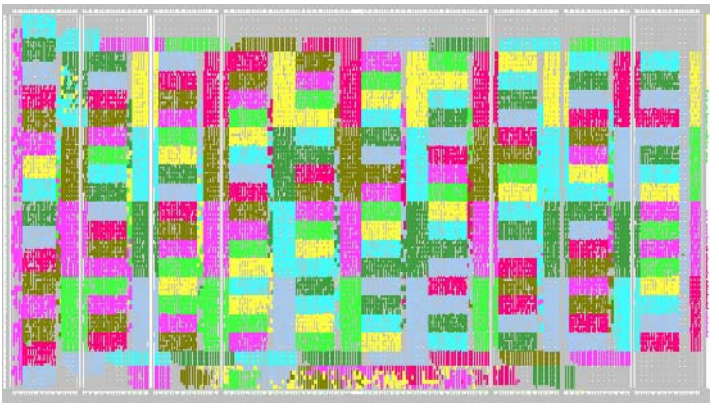


Fig. 4. Placement of design on Virtex-E (XCV2000E)

Table 2. Performance comparison of this work with previous designs

Design	FPGA Part	Freq. MHz	Thro'put Mbps	Latency ns	Area slices	Mbps / slice	Data path
Jarvinen et al [9]	Virtex-E XCV1000e-8	129.2	16,500		11,719	1.408	Enc
Saggesse et al [10]	Virtex-E XCV2000e-8	158	20,300		5810 + $100B_{RAM}$	1.091	Enc
Standaert et al [11]	Virtex-E XCV3200e-8	145	18,560		15,112	1.228	Enc
Hodjat et al [3] Excl. key expand	Virtex-II Pro XC2VP20	169.1	21,640	420	9,446 Excl. KE	2.290	Enc
Zambreno et al [8]	Virtex-II XC2V4000	184.1	23,570	163	16,938	1.391	Enc
Zhang (r=7), [2] Excl. key expand	Virtex-E XCV1000E-8	168.4	21,556	416	11,022 Excl. KE	1.956	Enc/ Dec
This work, 3LUT cut, key change support	Virtex-E XCV2000E-8	184.8	23,654	379	16,693	1.417	Enc/ Dec
This work, 3LUT cut, key change support	Spartan-III XC3S2000-5	196.1	25,107	357	17,425	1.441	Enc/ Dec

The pipeline cuts chosen meant that a given data item will pass completely through the AES cipher in 70 cycles. One issue with the AES KeyExpansion is that the decryption process starts with the final RoundKey and the only method of obtaining the final RoundKey is to progress through the entire key expansion. This issue was resolved by having separate encryption and decryption RoundKey registers and the new key being supplied suitably in advance to its data (140 cycles). Although the additional registers occupy a sizable amount of area it does permit maintaining throughput during key changes. The KeyExpander takes 10 cycles to fully generate the required set of RoundKeys. In order to match the latency through the main datapath, the KeyExpander was placed in a separate clock domain running at 1/7 of

the datapath clock. This allowed for many more levels of logic in the KeyExpander without it forming part of the critical path.

The architecture for the KeyExpander is shown in Figure 3. The InvMixColumns unit (M^{-1}) is included to maintain the order of the operations the same for both encryption and decryption. This is referred to in the FIPS-197 specification as "Equivalent Decryption". There is no pipelining in the KeyExpander and it evaluates one RoundKey every clock cycle (in its clock domain). The "RCON" values, defined in the FIPS-197 specification, are computed using repeated finite field doubling (FFM2 unit). Four non-pipelined, forward transform only, versions of the SubBytes operation were implemented using composite field arithmetic (S units). The output RoundKeys are registered to permit correct operation given key changes and selection between encryption and decryption (rk1 to rk10 for encryption and dk1 to dk9 for decryption). The first RoundKey (rk0) is obtained by directly registering the key input.

The placement of the design on a Virtex-E is shown in Figure 4 and the comparative results in Table 2. When comparing the quoted performance figures it is important to recognise the differences caused by changes in FPGA technology or more importantly with the level of support for key agility, encryption and decryption. Some designs did not include the key expansion in the results and other only supported the encryption datapath. This design shows an improvement in throughput over the previously known best design [2] of approximately 10% using the same FPGA technology. However, further savings can be made by moving from the Virtex-E to the lower cost Spartan-III devices with an increase in performance due to the more modern technology. The design achieves 25 Gbps throughput on the Spartan-III XC3S2000-5 device.

Further improvement in throughput, of say 20%-30%, is possible by adopting a 2-LUT cut, however factors such as fanout and congestion are likely to be a significant obstacle to obtaining an improved throughput-area figure.

Traditionally, such pipelined designs [3, 8, 9, 10, 11] only demonstrated any key agility in encryption only modes such as Counter mode (CTR). However, this design supports key agility for both encryption and decryption thus can support Electronic Code Book (ECB) mode. In a multi channel environment the key can be changed once ever 70 cycles thus support batch processing for a number of differently keyed concurrent channels without loss in throughput.

Further, it is a relatively simple task to extend the design by pipelining the key expansion, repeating its instantiation for all ten rounds and include registers (equivalent to approximately 15232 flip-flops) to support key changes each cycle. This would, in a multi channel environment, support any of the feedback modes, including Cipher Block Chaining (CBC) thus gaining improved security.

2.2 Round Based Architecture Using 32-Bit Datapath

There already exist a number of good 32-bit based designs [4, 5, 13]. In these designs the AES is implemented by breaking up a round into a number of smaller 32-bit wide operations. Thus a number of cycles is required to complete each round.

Such designs are based around a store for the "state" data (16 bytes for 128-bit) and look-up tables to perform the required AES operations of SubBytes and

MixColumns. One of the optimisations documented in the FIPS-197 specification is to combine the look-up table for the MixColumns and SubBytes operation into a single one. This is often referred to as the "T-box".

One of the key optimisations used by Rouvroy [5] was to exploit the larger (18kbit) block memories afforded by the Spartan-III and Virtex-II series FPGAs. This allowed for 4 off 32 bit x 256 word look up tables (ROMs) to be implemented per (dual port) block memory. So the required number of 8 off 32 bit x 256 word lookup tables can be implemented in two block memories, providing the four address buses needed (8-bits data in + 3-bits mode). The contents of the look up tables were chosen to provide convenient access to SubBytes and InvSubBytes required by the key expander and the finite field multiplications of the SubBytes table required for the combined SubBytes – MixColumns operation. The operation is completed by computing the exclusive-or of the four partial "T-box" values. The values stored are given by the following expression for the 8-bit value, a, using the SubBytes transformation and finite field multiplication by the given constant:

$$T(a) = \left[\begin{array}{c|c} 2 \bullet SB(a) & 14 \bullet ISB(a) \\ SB(a) & 9 \bullet ISB(a) \\ ISB(a) & 13 \bullet ISB(a) \\ 3 \bullet SB(a) & 11 \bullet ISB(a) \end{array}\right] \tag{1}$$

A further block memory was used to store the RoundKeys together with several multiplexers to route the data and key.

The following results (Table 3) were quoted together with a comparison with the previous design by Chadoweic and Gaj [4]. However, the figures quoted for

Table 3. Performance of existing 32-bit FPGA designs

	Chodowiec & Gaj [4]	Rouvroy et al [5]	Pramstaller et al [13]	Rouvroy et al [5]
Device	XC2S30-6	XC3S50-4	XCV1000E	XC2V40-6
Slices	222	163	1125	146
Throughput (Mbps)	166	208	215	358
RAM blocks	3	3	0	3
Throughput / Area(kbps / slice) ignoring block ram	750	1260	191	2450
Bits of block ram used	9600	34176	0	34176
Equiv slices for block ram	300	1068	0	1068
Total equiv. slices (area)	522	1231	1125	1214
Throughput / Area (kbps / slice)	318	169	191	295

throughput versus area failed to take into account the size of the block memories. This is of particular importance as the block memories on the Spartan-II are 4 kbit whereas those found on the Spartan-III and Virtex-II are 18 kbit. If these costs are taken into account then the result is substantially changed.

The cost of using a block memory in terms of an equivalent number of slices is still a matter of some debate. One option would be to make the comparison based on the physical area occupied by a slice and a block memory but quotable figures are not forthcoming from the manufacturers. An alternative is to consider the number of slices required to implement the equivalent distributed memory, however, this varies depending on the functionality required (for example single or dual port). Such estimates vary between 8 and 32 bits/slice. For this analysis a worst case figure of 32 bits/slice was used. The relative merits of the various designs and thus conclusions remain unchanged when the analysis was repeated for the lower estimate.

2.3 Application Specific Processor Architecture Using 8-Bit Datapath

The objective was to develop a small AES implementation. One option was to use the freely available Xilinx PicoBlaze soft core processor [14] for which the first version only requires 76 slices. However, for a practical design a small memory was needed thus the larger 96 slice KCPSM3 was selected. Additionally, the size of the ROM required to implement 365 instructions for the AES had to be considered together with an implementation for SubBytes. This results in a final design using the PicoBlaze of 119 slices plus the block memories which are accounted for here by an equivalent number of slices (once again 32 bits per slice was used). The resulting design had an equivalent slice count of 452 and with a 90.2 MHz maximum clock. Key expansion followed by encipher took 13546 cycles and key expansion followed by decipher 18885 cycles. The average encipher-decipher throughput was 0.71 Mbps.

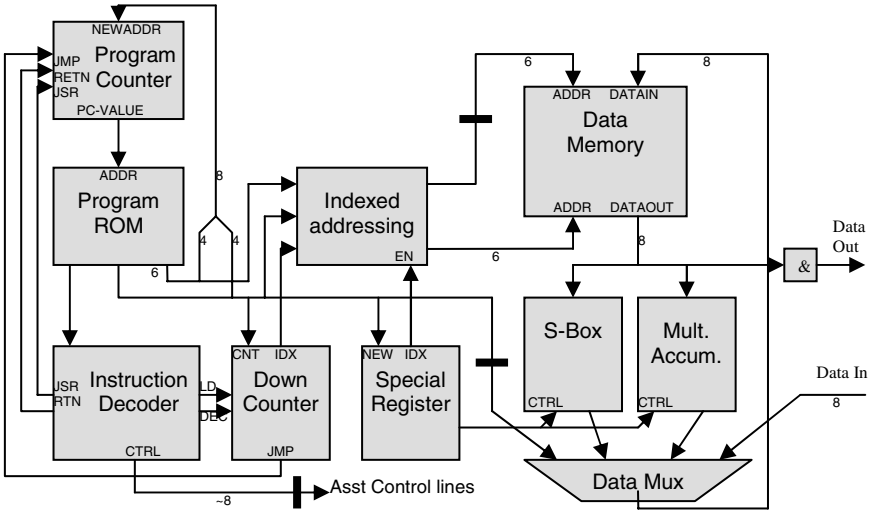


Fig. 5. ASIP Architecture

An application specific instruction processor (ASIP) was developed based around an 8-bit datapath and minimal program ROM size. Minimisation of the ROM size resulted in a requirement to support subroutines and looping. This added area to the control portion of the design but the saving was made in terms of the size of the ROM. The total design, including an equivalent number of slices for the block memories only occupies 259 slices to give a throughput of 2.2Mbps.

The datapath consisted of two processing units, the first to perform the SubBytes operation using resource shared composite field arithmetic and the second to perform multiply accumulate operations in Galois Field 2^8. A minimal set of instructions was developed (15 in total) to perform the operations required for the AES. The processor (Figure 5) used a pipelined design permitting execution of a new instruction every cycle.

Figure 6 is a pie chart depicting the balance of area between the various design units. Of the processor hardware approx 60% of the area is required for the datapath and 40% for the control.

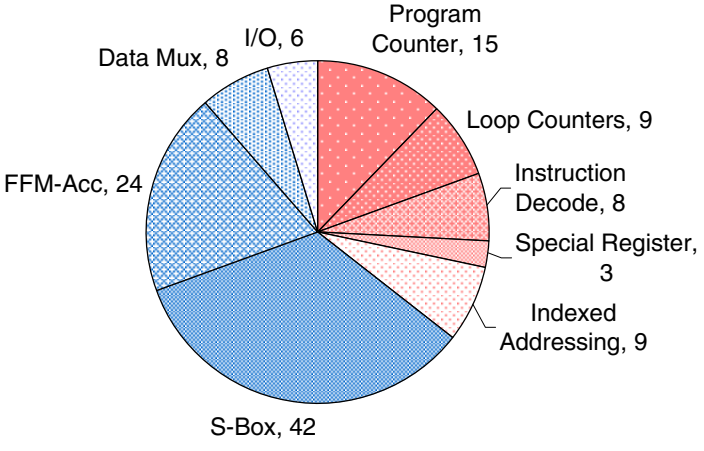


Fig. 6. Slice utilization versus design unit

As a good FPGA based 8-bit datapath for comparison could not be found, Table 4 shows comparison of this design with the state-of-the-art 32-bit designs using the relatively low cost Xilinx Spartan FPGAs. The two reference designs both quote throughput figures for a mode of operation where the key remains constant thus the time taken for key expansion was not included. A throughput figure was calculated for each design inclusive of the time taken for key expansion. The average for encipher and decipher was then calculated and is reported in Table 4 as the average throughput.

Figure 7 shows the placement of this design on a Spartan-II (XC2S15) part. The design requires 124 slices and two block memories. One memory formed the program ROM and the second was used as the ASIP's main memory (RAM). The AES application only required 360 bits of RAM thus the block memory was only partially utilized and could have be implemented as distributed memory with a cost of

42 additional slices and would then free up one of the block memories. There are also some concerns over the particular vulnerability of the block memories to power attacks so avoiding their use for key and data storage may be desirable. However, even avoiding use of the block memories does not negate such risks.

Table 4. Comparison with other designs using low cost FPGAs

	This design	Picoblaze based	Chodowiec & Gaj [4]	Rouvroy et al [5]
FPGA	Spartan-II XC2S15-6	Spartan-II XC2S15-6	Spartan-II XC2S30-6	Spartan-III XC3S50-4
Clock Frequency (MHz)	67	90	60	71
Datapath Bits	8	8	32	32
Slices	124	119	222	163
No. of Block RAMs used	2	2	3	3
Block RAM Size (kbits)	4	4	4	18
Bits of block RAM used	4480	10666	9600	34176
Est. equiv. slices for memory	140	333	300	1068
Total Equiv. Slices (area)	264	452	522	1231
Max Throughput (Mbps)	-	-	166	208
Ave. Throughput (Mbps)	2.2	0.71	69	87
Throughput/slice (kbps/slice)	8.3	1.9	132	70
Summary	Smallest	Software	Best speed/area	Fastest

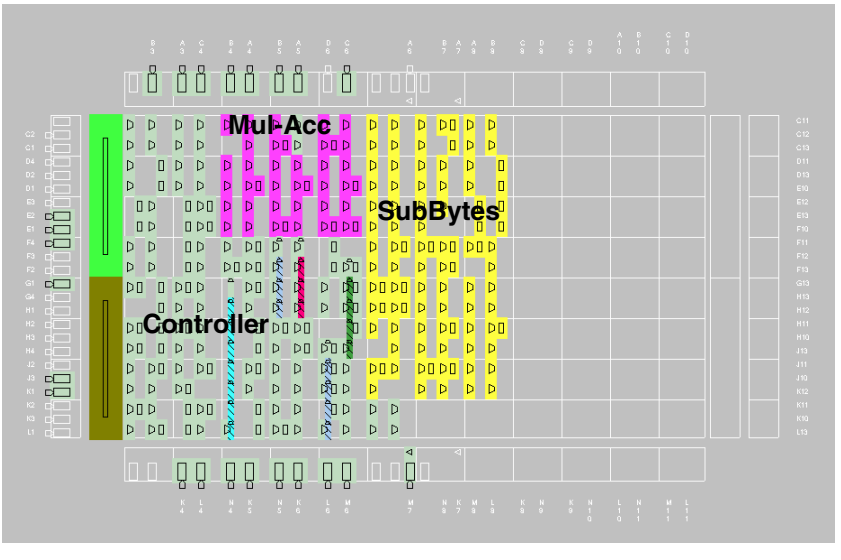


Fig. 7. Placement of low area design on Spartan-II (XC2S15)

3 Conclusions

This paper has presented a number of FPGA implementations from the fastest to the smallest. In terms of speed to area ratio the unrolled designs perform the best as there is no controller overhead. However, such designs are very large and need a 1 – 2 million gate device but achieve throughputs up to 25 Gbps. These designs have applications in fixed infrastructure such as IPsec for e-commerce servers.

The low area design described here achieves 2.2 Mbps which is sufficient for most wireless and home applications. The area required is small thus fundamentally low power so has utility in the future mobile area. The design requires just over half the available resources of the smallest available Spartan-II FPGA.

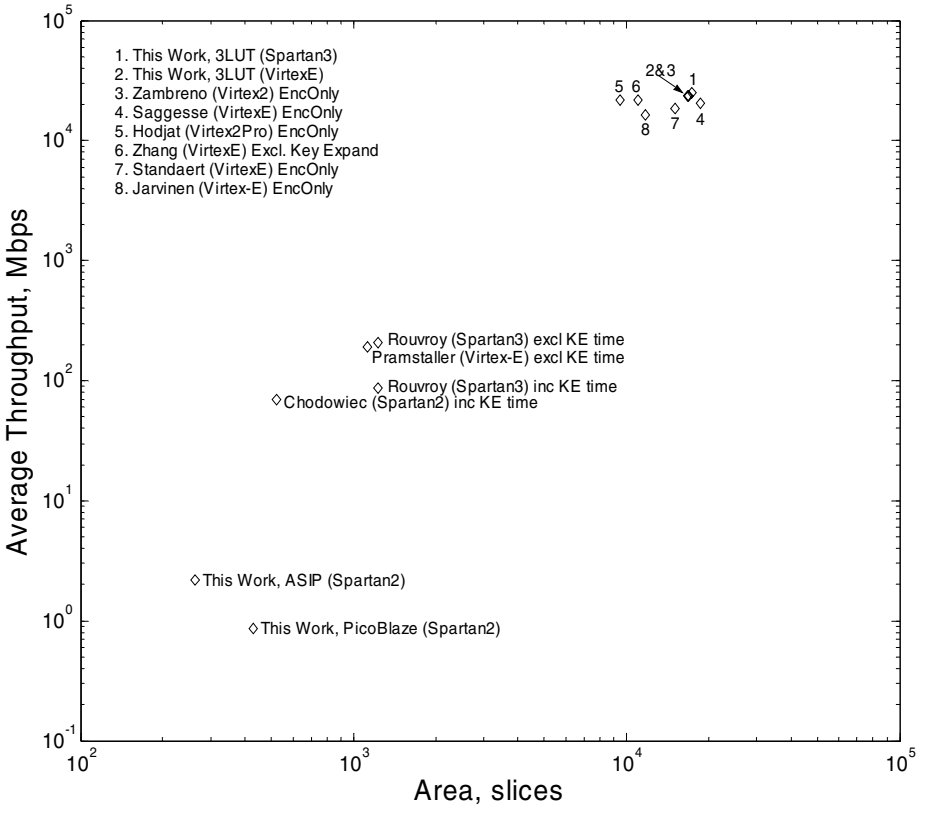


Fig. 8. Throughput versus area for the different FPGA designs

The advantage of the 8-bit ASIP over the more traditional 8-bit microcontroller architecture (PicoBlaze) is shown by the approximate factor of three improvement in throughput and 40% reduction in area (including estimated area for memories).

The 32-bit datapath designs occupy the middle ground between the two extremes and have utility where moderate throughput in the 100 – 200 Mbps is required.

The advantage of an FPGA specific optimisation over an ASIC number-of-gates approach has been demonstrated by the speed improvement made in the loop unrolled design.

The best architectural decision is to select the design of the lowest possible area meeting the throughput and operating mode requirement for the system being developed. Figure 8 shows the different designs in terms of their throughput and area.

Acknowledgement

This work was funded by the UK Engineering and Physical Sciences Research Council (EPSRC).

References

[1] National Institute of Standards and Technology (NIST), Information Technology Laboratory (ITL), *Advanced Encryption Standard (AES)*, Federal Information Processing Standards (FIPS) Publication 197, November 2001

[2] X. Zhang, K. K. Parhi, *High-speed VLSI architectures for the AES algorithm*, IEEE Trans. VLSI Systems, Vol. 12, Iss. 9, pp. 957 - 967, Sept. 2004

[3] A. Hodjat, I. Verbauwhede, *A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA*, 12th Annual IEEE Sypmosium on Field-Programmable Custom Computing Machines (FCCM'04), pp. 308-309, April 2004

[4] P. Chodowiec, K. Gaj, *Very Compact FPGA Implementation of the AES Algorithm*, Cryptographic Hardware and Embedded Systems (CHES 2003), LNCS Vol. 2779, pp. 319 – 333, Springer-Verlag, October 2003

[5] G. Rouvroy, F. X. Standaert, J. J. Quisquater, J. D. Legat, *Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications*, Procedings of the international conference on Information Technology: Coding and Computing 2004 (ITCC 2004), pp. 583 – 587, Vol. 2, April 2004

[6] M. Feldhofer, S. Dominikus and J. Wolkerstorfer, *Strong Authentication for RFID Systems Using the AES Algorithm*, CHES 2004, LNCS 3156, pp. 357-370, Springer-Verlag, 2004.

[7] A. Satoh, S. Morioka, K. Takano, S. Munetoh, *A Compact Rijndael Hardware Architecture with S-Box Optimization*, Proceedings of ASIACRYPT 2001, LNCS Vol. 2248, pp. 239 - 254, Springer-Verlag, December 2001

[8] J. Zambreno, D. Nguyen, A. Choudhary, *Exploring Area/Delay Trade-offs in an AES FPGA Implementation*, Proc. FPL 2004, 2004

[9] K. U. Jarvinen, M. T. Tommiska, and J. O. Skytta, *A fully pipelined memoryless 17.8 Gbps AES-128 encryptor*, Proc. Int. Symp. Field-Programmable Gate Arrays (FPGA 2003), Monterey, CA, pp. 207–215, Feb. 2003

[10] G. P. Saggese, A. Mazzeo, N. Mazocca, and A. G. M. Strollo, *An FPGA based performance analysis of the unrolling, tiling and pipelining of the AES algorithm*, Proc. FPL 2003, Portugal, Sept. 2003.

[11] F. Standaert, G. Rouvroy, J. Quisquater, and J. Legat, *Efficient implementation of Rijndael encryption in reconfigurable hardware: Improvements & design tradeoffs*, Proc. CHES 2003, Cologne, Germany, Sept. 2003.

[12] M. McLoone and J.V. McCanny, *High Performance Single-Chip FPGA Rijndael Algorithm Implementations*, CHES 2001, Paris, France, 2001

[13] N. Pramstaller and J. Wolkerstorfer, *A Universal and efficient AES co-processor for Field Programmable Logic Arrays*, FPL 2004, LNCS Vol. 3203, pp. 565-574, Springer-Verlag, 2004.

[14] K. Chapman, *PicoBlaze 8-bit Microcontroller*, Xilinx, 2002 http://www.xilinx.com/products/design_resources/proc_central/grouping/picoblaze.htm

A Very Compact S-Box for AES

D. Canright

Naval Postgraduate School, Monterey CA 93943, USA
dcanright@nps.edu

Abstract. A key step in the Advanced Encryption Standard (AES) algorithm is the "S-box." Many implementations of AES have been proposed, for various goals, that effect the S-box in various ways. In particular, the most compact implementations to date of Satoh et al.[14] and Mentens et al.[6] perform the 8-bit Galois field inversion of the S-box using subfields of 4 bits and of 2 bits. Our work refines this approach to achieve a more compact S-box. We examined many choices of basis for each subfield, not only polynomial bases as in previous work, but also normal bases, giving 432 cases. The isomorphism bit matrices are fully optimized, improving on the "greedy algorithm." Introducing some NOR gates gives further savings. The best case improves on [14] by 20%. This decreased size could help for area-limited hardware implementations, e.g., smart cards, and to allow more copies of the S-box for parallelism and/or pipelining of AES.

1 Introduction

The Advanced Encryption Standard (AES) was specified in 2001 by the National Institute of Standards and Technology [10]. The purpose is to provide a standard algorithm for encryption, strong enough to keep U.S. government documents secure for at least the next 20 years. The earlier Data Encryption Standard (DES) had been rendered insecure by advances in computing power, and was effectively replaced by triple-DES. Now AES will largely replace triple-DES for government use, and will likely become widely adopted for a variety of encryption needs, such as secure transactions via the Internet.

A wide variety of approaches to implementing AES have appeared, to satisfy the varying criteria of different applications. Some approaches seek to maximize throughput, e.g., [7], [16] and [3]; others minimize power consumption, e.g., [8]; and yet others minimize circuitry, e.g., [13], [14], [17], and [2]. For the latter goal, Rijmen[12] suggested using subfield arithmetic in the crucial step of computing an inverse in the Galois Field of 256 elements—reducing an 8-bit calculation to several 4-bit ones. Satoh et al.[14] further extended this idea, using the "tower field" approach of Paar[11], breaking up the 4-bit calculations into 2-bit ones, which resulted in the smallest AES circuit to date.

Mentens et al.[6] recently examined whether the choice of representation (basis in each subfield) used by [14] was optimal. They compared 64 different choices (including that in [14]), based on the number of '1' entries in the two

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 441–455, 2005.

transformation matrices used in encryption and on the number of binary XOR operations used in one of the 4-bit operations in the subfield. Based on these criteria, they determined that a different choice is better than that in [14], and estimated the improvement at 5%.

The current work improves on the compact implementation of [14] and extends the work of [6] in the following ways. Many choices of representation (432 different isomorphisms) were compared, including all those in [6]. The cases in [6] use a *polynomial basis* in each subfield (as in [14]), while we also consider a *normal basis* for each subfield. It turns out the best case uses all normal bases. And while [14] used the popular "greedy algorithm" to reduce the number of gates in the bit matrices required in changing representations, we fully optimized each matrix by an exhaustive tree-search algorithm, resulting in the minimum number of gates. (Based on our fully optimized matrices, comparisons of matrices using the simple "number of '1' entries" criterion of [6] gives incorrect comparisons in 37% of the cases, and even the greedy algorithm gives incorrect comparisons in 20% of the cases.) We included logic optimizations both at the hierarchical level of the Galois arithmetic and at the low level of individual logic gates. We were thus able to replicate the very compact *merged* S-box reported in [14], which includes both the S-box function and its inverse, including a Galois inverter and all four transformation matrices as well as multiplexors for selecting which input and output transformations are used[15]. Hence our comparisons of the different cases are based on complete, optimized implementations of the merged S-box (rather than the two criteria of [6]), and it turns out the best case for the merged architecture is also the best for the architecture with a separate S-box and inverse S-box. Also, although the bit operations of Galois arithmetic correspond directly to XOR and AND (or NAND) gates, here certain combinations of operations are implemented more compactly using XOR and OR (or NOR) gates. These refinements combine to give a merged S-box circuit that is 20% smaller than in [14], a significant improvement.

1.1 The Advanced Encryption Standard Algorithm

The AES algorithm, also called the Rijndael algorithm, is a symmetric block cipher, where the data is encrypted/decrypted in blocks of 128 bits. Each data block is modified by several rounds of processing, where each round involves four steps. Three different key sizes are allowed: 128 bits, 192 bits, or 256 bits, and the corresponding number of rounds for each is 10 rounds, 12 rounds, or 14 rounds, respectively. From the original key, a different "round key" is computed for each of these rounds. For simplicity, the discussion below will use a key length of 128 bits and hence 10 rounds.

There are several different modes in which AES can be used [9]. Some of these, such as Cipher Block Chaining (CBC), use the result of encrypting one block for encrypting the next. These feedback modes effectively preclude pipelining (simultaneous processing of several blocks in the "pipeline"). Other modes, such as the "Electronic Code Book" mode or "Counter" modes, do not require feedback, and may be pipelined for greater throughput.

The four steps in each round of encryption, in order, are called *SubBytes* (byte substitution), *ShiftRows*, *MixColumns*, and *AddRoundKey*. Before the first round, the input block is processed by AddRoundKey. Also, the last round skips the MixColumns step. Otherwise, all rounds are the same, except each uses a different round key, and the output of one round becomes the input for the next. For decryption, the mathematical inverse of each step is used, in reverse order; certain manipulations allow this to appear like the same steps as encryption with certain constants changed. Each round key calculation also requires the *SubBytes* operation. (More complete descriptions of AES are available from several sources, e.g., [10].)

Of these four steps, three of them (*ShiftRows*, *MixColumns*, and *AddRoundKey*) are *linear*, in the sense that the output 128-bit block for such steps is just the linear combination (bitwise, modulo 2) of the outputs for each separate input bit. These three steps are all easy to implement by direct calculation in software or hardware.

The single *nonlinear* step is the *SubBytes* step, where each byte of the input is replaced by the result of applying the "S-box" function to that byte. This nonlinear function involves finding the inverse of the 8-bit number, considered as an element of the Galois field $GF(2^8)$. The Galois inverse is not a simple calculation, and so many current implementations use a table of the S-box function output. This table look-up method is fast and easy to implement.

But for hardware implementations of AES, there is one drawback of the table look-up approach to the S-box function: each copy of the table requires 256 bytes of storage, along with the circuitry to address the table and fetch the results. Each of the 16 bytes in a block can go through the S-box function independently, and so could be processed in parallel for the byte substitution step. This effectively requires 16 copies of the S-box table for one round. To fully pipeline the encryption would entail "unrolling" the loop of 10 rounds into 10 sequential copies of the round calculation. This would require 160 copies of the S-box table (200 if round keys are computed "on the fly"), a significant allocation of hardware resources.

In contrast, this work describes a direct calculation of the S-box function using sub-field arithmetic, similar to [14]. While the calculation is complicated to describe, the advantage is that the circuitry required to implement this in hardware is relatively simple, in terms of the number of logic gates required. This type of S-box implementation is significantly smaller (less area) than the table it replaces, especially with the optimizations in this work. Furthermore, when chip area is limited, this compact implementation may allow parallelism in each round and/or unrolling of the round loop, for a significant gain in speed.

The rest of the paper describes our specific algorithm in detail. (See [1] for a thorough, detailed presentation of the 432 different versions considered in finding the best one.) Section 2 explains the basic idea of the algorithm and the resulting structure of the Galois inverter. Section 3 discusses ways to optimize the calculation, Section 4 describes the changes of representation, and Section 5 describes the results. Finally, Section 6 summarizes the work.

2 The S-Box Algorithm Using Subfield Arithmetic

The S-box function of an input byte (8-bit vector) $\boldsymbol{a}$ is defined by two substeps:

1. *Inverse*: Let $\boldsymbol{c} = \boldsymbol{a}^{-1}$, the multiplicative inverse in $GF(2^8)$ (except if $\boldsymbol{a} = \boldsymbol{0}$ then $\boldsymbol{c} = \boldsymbol{0}$).
2. *Affine Transformation*: Then the output is $\boldsymbol{s} = M\,\boldsymbol{c} \oplus \boldsymbol{b}$, with the constant bit matrix M and byte $\boldsymbol{b}$ shown below:

$$\begin{pmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{pmatrix} = \begin{pmatrix} 1\,1\,1\,1\,1\,0\,0\,0 \\ 0\,1\,1\,1\,1\,1\,0\,0 \\ 0\,0\,1\,1\,1\,1\,1\,0 \\ 0\,0\,0\,1\,1\,1\,1\,1 \\ 1\,0\,0\,0\,1\,1\,1\,1 \\ 1\,1\,0\,0\,0\,1\,1\,1 \\ 1\,1\,1\,0\,0\,0\,1\,1 \\ 1\,1\,1\,1\,0\,0\,0\,1 \end{pmatrix} \begin{pmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

where bit #7 is the most significant, with all bit operations modulo 2.

The second, affine substep is easy to implement; the algorithm for the first substep, finding the inverse, is described below. (Some familiarity with Galois arithmetic is assumed. A succinct introduction to Galois fields is given in [5]; for more depth and rigor, see [4]. Also, [1] conveys just enough theory to understand this algorithm.)

The AES algorithm uses the particular Galois field of 8-bit bytes where the bits are coefficients of a polynomial (this representation is called a *polynomial basis*) and multiplication is modulo the irreducible polynomial $q(x) = x^8 + x^4 + x^3 + x + 1$, with addition of coefficients modulo 2. Let A be one root of $q(x)$; then the standard polynomial basis is $[A^7, A^6, A^5, A^4, A^3, A^2, A, 1]$. (Note: we will usually use uppercase Roman letters for specific elements of $GF(2^8)$, lowercase Greek letters for elements of the subfield $GF(2^4)$, uppercase Greek letters for the sub-subfield $GF(2^2)$, and lowercase Roman letters for bits in $GF(2)$.)

Direct calculation of the inverse (modulo an eighth-degree polynomial) of a seventh-degree polynomial is not easy. But calculation of the inverse (modulo a second-degree polynomial) of a first-degree polynomial is relatively easy, as pointed out by Rijmen [12]. This suggests the following changes of representation.

First, we represent a general element G of $GF(2^8)$ as a linear polynomial (in y) over $GF(2^4)$, as $G = \gamma_1 y + \gamma_0$, with multiplication modulo an irreducible polynomial $r(y) = y^2 + \tau y + \nu$. All the coefficients are in the 4-bit subfield $GF(2^4)$. So the pair $[\gamma_1, \gamma_0]$ represents G in terms of a polynomial basis $[Y, 1]$ where Y is one root of $r(y)$.

Alternatively, we could use the *normal basis* $[Y^{16}, Y]$ using both roots of $r(y)$. Note that

$$r(y) = y^2 + \tau y + \nu = (y + Y)(y + Y^{16}) \ , \tag{1}$$

so $\tau = Y + Y^{16}$ is the *trace* and $\nu = (Y)(Y^{16})$ is the *norm* of Y.

Second, we can similarly represent $GF(2^4)$ as linear polynomials (in z) over $GF(2^2)$, as $\gamma = \Gamma_1 z + \Gamma_0$, with multiplication modulo an irreducible polynomial $s(z) = z^2 + Tz + N$, with all the coefficients in $GF(2^2)$. Again, this uses a polynomial basis $[Z, 1]$, where Z is one root of $s(z)$; or we could use the normal basis $[Z^4, Z]$. As above, T is the trace and N is the norm of Z.

Third we represent $GF(2^2)$ as linear polynomials (in w) over $GF(2)$, as $\Gamma = g_1 w + g_0$, with multiplication modulo $t(w) = w^2 + w + 1$, where g_1 and g_0 are single bits. This uses a polynomial basis $[W, 1]$, with W one root of $t(w)$; or a normal basis would be $[W^2, W]$. (Note that the trace and norm of W are 1.)

This allows operations in $GF(2^8)$ to be expressed in terms of simpler operations in $GF(2^4)$, which in turn are expressed in the simple operations of $GF(2^2)$. In each of these fields, addition (the same operation as subtraction) is just bitwise XOR, for any basis.

In $GF(2^8)$ with a *polynomial* basis, multiplication mod $y^2 + \tau y + \nu$ is given by

$$(\gamma_1 y + \gamma_0)(\delta_1 y + \delta_0) = (\gamma_1\delta_0 + \gamma_0\delta_1 + \gamma_1\delta_1\tau)y + (\gamma_0\delta_0 + \gamma_1\delta_1\nu) \ . \tag{2}$$

From this it is easy to verify that the inverse is given by

$$\begin{aligned}(\gamma_1 y + \gamma_0)^{-1} &= [\theta^{-1}\ \gamma_1]\, y + [\theta^{-1}\ (\gamma_0 + \gamma_1\tau)] \\ \text{where } \theta &= \gamma_1^2\nu + \gamma_1\gamma_0\tau + \gamma_0^2 \ .\end{aligned} \tag{3}$$

So finding an inverse in $GF(2^8)$ reduces to an inverse and several multiplications in $GF(2^4)$. Analogous formulas for multiplication and inversion apply in $GF(2^4)$. Simpler versions apply in $GF(2^2)$, where the inverse is the same as the square (for $\Gamma \in GF(2^2)$, $\Gamma^4 = \Gamma$); note then that a zero input gives a zero output, so that special case is handled automatically.

The details of these calculations change if we use a *normal* basis at each level. In $GF(2^8)$, recall that both Y and Y^{16} satisfy $y^2 + \tau y + \nu = 0$ where $\tau = Y^{16} + Y$ and $\nu = (Y^{16})Y$, so $1 = \tau^{-1}(Y^{16} + Y)$. Then multiplication becomes

$$\begin{aligned}(\gamma_1 Y^{16} + \gamma_0 Y)(\delta_1 Y^{16} + \delta_0 Y) &= [\gamma_1\delta_1\tau + \theta]\, Y^{16} + [\gamma_0\delta_0\tau + \theta]\, Y \\ \text{where } \theta &= (\gamma_1 + \gamma_0)(\delta_1 + \delta_0)\nu\tau^{-1} \ ,\end{aligned} \tag{4}$$

and the inverse is

$$\begin{aligned}(\gamma_1 Y^{16} + \gamma_0 Y)^{-1} &= [\theta^{-1}\ \gamma_0]\, Y^{16} + [\theta^{-1}\ \gamma_1]\, Y \\ \text{where } \theta &= \gamma_1\gamma_0\tau^2 + (\gamma_1^2 + \gamma_0^2)\nu \ .\end{aligned} \tag{5}$$

Again, finding an inverse in $GF(2^8)$ involves an inverse and several multiplications in $GF(2^4)$, and analogous formulas apply in the subfields.

These formulas can be simplified with specific choices for the coefficients in the minimal polynomials $r(y)$ and $s(z)$. The most efficient choice is to let the trace be unity, so from here on we let $\tau = 1$ and $T = 1$. (This is better than choosing the norm to be unity—we can't have both, and neither can be zero.)

The above shows that both polynomial bases and normal bases give comparable amounts of operations, at this level; both types remain roughly comparable

at lower levels of optimization. (Of course, one could choose other types of basis at each level, but both polynomial and normal bases have structure that leads to efficient calculation, which is lacking in other bases.) We considered all of the subfield polynomial and normal bases that had a trace of unity. There are eight choices for the norm ν that make $r(y) = y^2 + y + \nu$ irreducible over $GF(2^4)$, and two choices for N that make the polynomial $s(z) = z^2 + z + N$ irreducible over $GF(2^2)$. Each of these polynomials $r(y)$, $s(z)$, and $t(w)$ has two distinct roots, and for a polynomial basis we may choose either, or for a normal basis we use both. So altogether there are $(8 \times 3) \times (2 \times 3) \times (1 \times 3) = 432$ possible cases (including the all-polynomial case used in [14]).

We compared all of these cases, in terms of complete implementations of the merged S-box architecture of [14], including all low-level optimizations appropriate to each case. The most compact was judged to be the one giving the least number of gates (using a 0.13-μm CMOS standard cell library[15]) for the merged S-box, where the encryptor and decryptor share a $GF(2^8)$ inverter. As it happens, this is also the best case for an architecture using a separate encryptor and decryptor (each with an inverter).

The most compact case uses normal bases for all subfields. Here we will give the relevant Galois elements as hexadecimal numbers, for bit vectors in terms of the standard polynomial basis for $GF(2^8)$ (powers of A). For $GF(2^8)$, the norm ν = 0xEC, and Y = 0xFF, so the basis is [0xFE,0xFF] (recall that for each of the normal bases, the sum of the two elements is the trace, which is unity). For $GF(2^4)$, N = 0xBC and Z = 0x5C, so the basis is [0x5D,0x5C]. (These two levels are related by $\nu = N^2 Z$.) And for $GF(2^2)$, W = 0xBD, and the basis is [0xBC,0xBD]. (Those two levels are related by $N = W^2$ and $W = N^2$.)

2.1 Hierarchical Structure

Here we show the structure of this best-case inverter. To clarify the subfield operations needed, we will use $\oplus$ and $\otimes$ for addition and multiplication in the subfield. In $GF(2^8)$ the only operation required is the inverse; the normal basis inverter is shown in Figure 1 and the polynomial basis inverter in Figure 2, for comparison. The operations required in the subfield $GF(2^4)$ include an inverter (same form as in $GF(2^8)$), three multipliers, two adders (bitwise XOR), and the combined operation of squaring then scaling (multiplying) by the norm ν. Note: in $GF(2^2)$ inversion is the same as squaring, which is free with a normal basis:

$$(g_1 W^2 + g_0 W)^{-1} = (g_1 W^2 + g_0 W)^2 = g_0 W^2 + g_1 W \quad . \tag{6}$$

The $GF(2^4)$ multiplier is shown in Figure 3 for a normal basis; the polynomial basis version has the same operations in a slightly different arrangement. The operations required in the subfield $GF(2^2)$ include three multipliers, four adders, and scaling by the norm N. The $GF(2^2)$ multiplier has the same structure, except lacks scaling by the norm (since the norm of W is 1), and in $GF(2)$, $\otimes$ means AND.

The other operation needed in $GF(2^4)$ is the combined operation of squaring then scaling by ν (the "square-scale operation"). The form of this operation

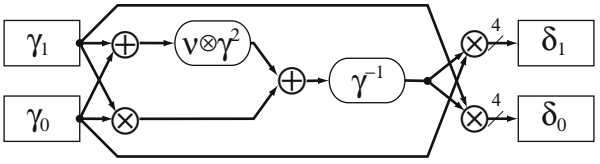


Fig. 1. Normal $GF(2^8)$ inverter: $(\gamma_1 Y^{16} + \gamma_0 Y)^{-1} = (\delta_1 Y^{16} + \delta_0 Y)$. The datapaths all have the same bit width, shown at the output (4 bits here); addition is bitwise exclusive-OR; and sub-field multipliers appear below. The $GF(2^4)$ inverter has the same structure. In $GF(2^2)$ inverting is free: a bit swap.

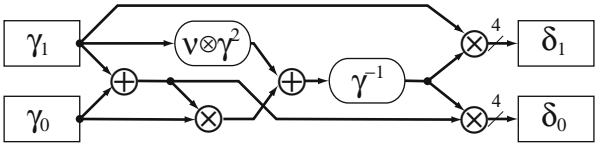


Fig. 2. Polynomial $GF(2^8)$ inverter: $(\gamma_1 y + \gamma_0)^{-1} = (\delta_1 y + \delta_0)$. The $GF(2^4)$ inverter has the same structure. In $GF(2^2)$ inverting (same as squaring) requires only one XOR.

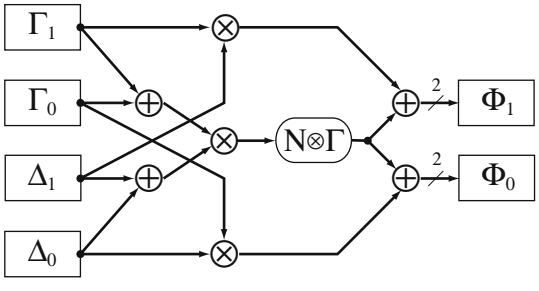


Fig. 3. Normal $GF(2^4)$ multiplier: $(\Gamma_1 Z^4 + \Gamma_0 Z) \otimes (\Delta_1 Z^4 + \Delta_0 Z) = (\Phi_1 Z^4 + \Phi_0 Z)$. The $GF(2^2)$ multiplier has the same structure except lacks the scaling by N, since the norm in the subfield is 1.

varies, depending not only on the type of basis in $GF(2^4)$, but also on the representation ν in that basis; there are a dozen different versions. Here scaling the square of $\gamma = \Gamma_1 Z^4 + \Gamma_0 Z$ by $\nu = N^2 Z$ gives

$$\nu \otimes (\Gamma_1 Z^4 + \Gamma_0 Z)^2 = [(\Gamma_1 \oplus \Gamma_0)^2] Z^4 + [(N \otimes \Gamma_0)^2] Z \ . \tag{7}$$

The only "new" operation required in the subfield $GF(2^2)$ is squaring, but this is the same as inversion, and for a normal basis is free.

The remaining operation needed in the subfield $GF(2^2)$ is scaling by $N = W^2$ (since squaring is free, this also give the square-scale operation in the $GF(2^4)$ inverter):

$$N \otimes (g_1 W^2 + g_0 W) = [g_0] W^2 + [g_0 \oplus g_1] W \quad , \tag{8}$$

requiring a single XOR.

Also, combining the multiplication in $GF(2^2)$ with scaling by N gives a small improvement; this combination appears in the $GF(2^4)$ multiplier:

$$\begin{aligned} & N \otimes (g_1 W^2 + g_0 W) \otimes (d_1 W^2 + d_0 W) \\ & \quad = [f \oplus ((g_1 \oplus g_0) \otimes (d_1 \oplus d_0))] W^2 + [f \oplus (g_1 \otimes d_1)] W \\ & \text{where } f = g_0 \otimes d_0 \ . \end{aligned} \tag{9}$$

3 Inverter Optimizations

Here we will show the optimizations in the $GF(2^8)$ inverter for this best case. There are similar optimizations for other cases, described in [1]. All these optimizations were carefully calculated by hand, and so should be at least as good as versions given by automatic optimization tools.

3.1 Common Subexpressions

Eliminating redundancy where low-level subexpressions appear more than once in the above hierarchical structure reduces the size of the Galois inverter.

As [14] mentions, one place this occurs is when the same factor is input to two different multipliers. Each multiplier computes the sum of the high and low halves of each factor (see Figure 3), so when a factor is shared then this addition in the subfield can be removed from one of the multipliers. For example, a 2-bit factor shared by two $GF(2^2)$ multipliers saves one XOR (addition in the 1-bit subfield). Moreover, since each $GF(2^4)$ multiplier includes three $GF(2^2)$ multipliers, then a shared 4-bit factor implies three corresponding shared 2-bit factors in these subfield multipliers. So each shared 4-bit factor saves five XORs (one 2-bit addition and three 1-bit additions).

The normal-basis inverters for $GF(2^8)$ and $GF(2^4)$ share all three factors among the three multipliers; however, the corresponding polynomial-basis inverters each have only two shared factors (see Figures 1 and 2). This gives an advantage of five XORs to using a normal basis in $GF(2^8)$, from the additional shared factor.

A more subtle saving occurs in the $GF(2^4)$ inverter. There the bit sums computed for common factors can be used in the following square-scale operation, which saves one XOR. A similar optimization occurs in the $GF(2^8)$ inverter; combining the bit sums for shared input factors with parts of the square-scale operation saves three XORs.

3.2 Logic Gate Optimizations

Mathematically, computing the Galois inverse in $GF(2^8)$ breaks down into operations in $GF(2)$, i.e., the bitwise operations XOR and AND. However, it can be advantageous to consider other logical operations that give equivalent results.

For example, for the 0.13-μm CMOS standard cell library considered[15], a NAND gate is smaller than an AND gate. Since the AND output bits in the $GF(2^2)$ multiplier are always combined by pairs in a following XOR, then the AND gates can be replaced by NAND gates. That is, $[\,(a \otimes b) \oplus (c \otimes d)\,]$ is equivalent to [(a NAND b) XOR (c NAND d)]. This gives a slight size saving.

Also, in this library an XNOR gate is the same size as an XOR gate. This is useful in the affine transformation of the S-box, where the addition of the constant $\boldsymbol{b} = 0\text{x}63$ means applying a NOT to some output bits. In most cases, this can be done by replacing an XOR by an XNOR in the bit-matrix multiply, so is "free."

While the above logic optimizations are not original, here is one we have not seen elsewhere. Note that the combination $[\,a \oplus b \oplus (a \otimes b)\,]$ is equivalent to [a OR b]. In the few places in the inverter where this combination occurs, we can replace 2 XORs and an AND by a single OR, a worthwhile substitution. (Actually, 2 XORs and a NAND are replaced by a NOR, smaller than an OR.) In fact, the NOR gate is smaller than an XOR gate, so even when some rearrangement is required to get that combination, it is worthwhile even if the NOR ends up replacing only a single XOR. Our implementation uses 6 NORs in the $GF(2^8)$ inverter (including two in the $GF(2^4)$ inverter).

4 Changes of Representation

This algorithm involves two different representations, or isomorphisms, of the Galois Field $GF(2^8)$. The standard AES form uses a vector of 8 bits (in $GF(2)$) as the coefficients of the 8 powers of A, the root of the defining polynomial $q(x) = x^8 + x^4 + x^3 + x + 1$. The subfield form for $GF(2^8)$ uses a pair of 4-bit coefficients (in $GF(2^4)$) of Y^{16} and Y (for a normal basis), the roots of $r(y) = y^2 + y + \nu$. Then each element of $GF(2^4)$ is a pair of two-bit coefficients (in $GF(2^2)$) of Z^4 and Z, the roots of $s(z) = z^2 + z + N$. And in $GF(2^2)$, each element pair of one-bit coefficients (in $GF(2)$) of W^2 and W, the roots of $t(w) = w^2 + w + 1$. So the subfield representation uses pairs of pairs of pairs of bits.

One approach to using these two forms, as suggested by [13], is to convert each byte of the input block once, and do all of the AES algorithm in the new form, only converting back at the end of all the rounds. Since all the arithmetic in the AES algorithm is Galois arithmetic, this would work fine, provided the key was appropriately converted as well. However, the *MixColumns* step involves multiplying by constants that are simple in the standard basis (2 and 3, or A and $A + 1$), but this simplicity is lost in the subfield basis (in our best basis, 2 and 3 become 0xA9 and 0x56). For example, scaling by 2 in the standard basis takes only 3 XORs; the most efficient normal-basis version of this scaling requires 18 XORs. Similar concerns arise in the inverse of *MixColumns*, used in decryption. This extra complication more than offsets the savings from delaying the basis change back to standard. Then, as in [14], the affine transformation can be combined with the basis change (see below). For these reasons, it is most

efficient to change into the subfield basis on entering the S-box and to change back again on leaving it.

Each change of basis means multiplication by an 8×8 bit matrix. Letting X refer to the matrix that converts from the subfield basis to the standard basis, then to compute the S-box function of a given byte, first we do a bit-matrix multiply by X^{-1} to change into the subfield basis, then calculate the Galois inverse by subfield arithmetic, then change basis back again with another bit-matrix multiply, by X. This is followed directly by the affine transformation (substep 2), which includes another bit-matrix multiply by the constant matrix M. (This can be regarded another change of basis, since M is invertible.) So we can combine the matrices into the product MX to save one bit-matrix multiply, as pointed out by [14]. Then adding the constant $\boldsymbol{b}$ completes the S-box function.

The inverse S-box function is similar, except the XOR with constant $\boldsymbol{b}$ comes first, followed by multiplication by the bit matrix $(MX)^{-1}$. Then after finding the inverse, we convert back to the standard basis through multiplication by the matrix X.

For each such constant-matrix multiply, the gate count can be reduced by "factoring out" combinations of input bits that are shared between different output bits (rows). One way to do this is known as the "greedy algorithm," where at each stage one picks the combination of two input bits that is shared by the most output bits; that combination is then pre-computed in a single (XOR) gate, which output effectively becomes a new input to the remaining matrix multiply. The greedy algorithm is straightforward to implement, and generally gives good results.

But the greedy algorithm may not find the best result. We used a brute-force "tree search" approach to finding the optimal factoring. At each stage, each possible choice for factoring out a bit combination was tried, and the next stage examined recursively. (Some "pruning" of the tree is possible, when the bit-pair choice in the current stage is independent of that in the calling stage and had been checked previously. The C program is given in [1].) This method is guaranteed to find the minimal number of gates; the big drawback is that one cannot predict how long it will take, due to the combinatorial complexity of the algorithm.

The "merged" S-box and inverse S-box of [14] complicates this picture, but reduces the hardware overall when both encryption and decryption are needed. There, a block containing a single $GF(2^8)$ inverter can be used to compute either the S-box function or its inverse, depending on a selector signal. Given an input byte $\boldsymbol{a}$, both $X^{-1}\,\boldsymbol{a}$ and $(MX)^{-1}\,(\boldsymbol{a}+\boldsymbol{b})$ are computed, with the first selected for encryption, the second for decryption. That selection is input into the inverter, and from the output byte $\boldsymbol{c}$, both $(MX)\,\boldsymbol{c}+\boldsymbol{b}$ and $X\,\boldsymbol{c}$ are computed; again the first is selected for encryption, the second for decryption.

With this merged approach, these basis-change matrix pairs can be optimized together, considering X^{-1} and $(MX)^{-1}$ together as a 16×8 matrix, and similarly (MX) and X, each pair taking one byte as input and giving two bytes as output. (Then $(MX)^{-1}\,(\boldsymbol{a}+\boldsymbol{b})$ must be computed as $(MX)^{-1}\,\boldsymbol{a}+[(MX)^{-1}\,\boldsymbol{b}]$.)

Combining in this way allows more commonality among rows (16 instead of 8) and so yields a more compact "factored" form. Of course, this also means the "tree search" optimizer has a much bigger task and longer run time. (Using an Intel Xeon processor under Linux, optimization times for a 16×8 matrix varied from a few minutes to many weeks.)

The additive constant $\boldsymbol{b}$ of the affine transformation requires negating specific bits of the output of the basis change. (Actually, for the merged S-box, the multiplexors we use are themselves negating, so it is the bits other than those in $\boldsymbol{b}$ that need negating first.) As mentioned in Section 3.2, this usually involves replacing an XOR by an XNOR in the basis change (both are the same size in the CMOS library we consider), but sometimes this is not possible and a NOT gate is required.

The change of basis matrix X for our best case is given below :

$$X = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \tag{10}$$

The other three matrices are easily computed from X. The combined 16×8 bit matrices for the merged architecture, fully optimized by our tree search algorithm, are given in [1]

At this time, not all of the matrices for all of the cases considered below have been fully optimized, but the data so far indicate how full optimization can improve on the greedy algorithm. For the architecture with separate encryptor and decryptor, all cases have been fully optimized: of 1728 matrices (8×8) optimized, 762 (44%) were improved by at least one XOR, and of those, 138 (18% of improved ones) were improved by two XORs, and 11 (1.4% of improved ones) were improved by three XORs. For the merged architecture, the top 27 cases have been optimized (we gave up on one matrix in case 28 after estimating optimization would take 5 years). Of 55 matrices (16×8) optimized, 24 (44%) were improved by one XOR, 10 (18%) were improved by two XORs, and 6 (11%) were improved by three XORs, so altogether 73% were improved.

With so many optimized matrices, we could evaluate how well matrix comparisons based on the greedy algorithm or on the number of '1' entries correctly predicted the comparisons between the corresponding fully optimized matrices. We called a prediction incorrect when it predicted that one matrix was better than another, but the fully optimized version turned out worse or the same (or predicted same when one was better). For the 1492128 comparisons among the 1728 optimized 8×8 matrices, the greedy algorithm gave incorrect predictions for 19.9% of comparisons while the number of '1's incorrectly predicted 37.5%. The results for the 1485 comparisons among the 55 optimized 16×8 matrices were more dramatic: the greedy prediction was incorrect for 30.7% and the '1's prediction incorrect for 43.7%.

Since we have not yet fully optimized the (16×8) matrices for all of the 432 possible cases, it is remotely possible that some other case could turn out to be better than the case we call "best." We *have* optimized all cases whose estimated size, based on the greedy algorithm, was within 9 XORs of the actual size of our best case (except in one case, where only 1 of the 2 matrices was optimized; it improved by 2 XORs). So far, the best improvement in a single 16×8 matrix is 3 XORs, and the best improvement in the pair of matrices for a single case is 5 XORs. For some other case to be best, full optimization must improve a matrix pair, beyond what the greedy algorithm found, by at least 10 XORs. We consider this highly unlikely, and so are confident that we have indeed found the best of all 432 cases.

5 Implementation Results

The size of our best S-box is shown in Table 1, for three architectures: merged S-box and inverse S-box (one inverter, all four tranformation matrices, and two 8-bit selectors), only S-box (for just encrypting), and only inverse S-box (for just decrypting). Results are shown by number and type of logic operations, and also by total "gates," where the number refers to the equivalent number of NAND gates, using our standard cell library. We use the equivalencies 1 XOR/XNOR = $\frac{7}{4}$ NAND gates, 1 NOR = 1 NAND gate, 1 NOT = $\frac{3}{4}$ NAND gate, and 1 MUX21I = $\frac{7}{4}$ NAND gates [15]. Our merged S-Box, equivalent in size to 234 NANDs, is an improvement of 20% over that of Satoh et al. at 294 NANDs[14]. While Mentens et al.[6] use a different cell library, if we just compare equivalent NANDs our merged S-box is 14% smaller than their S-box at 272 NANDs.

Table 2 shows the effects of different levels of optimization of the inverter. Note in particular that the NOR substitution discussed in 3.2 further reduces the inverter by 9%. Table 3 show how different choices of basis affect the results. For fair comparisons, since we have not calculated fully optimized matrices and the NOR substitution improvements for all four bases shown, we show *our implementations* using greedy-algorithm matrices and exclude the NOR substitution. Our best basis is the only one of the four that uses normal bases.

Table 1. Best Case Results. Here are our best results for a complete implementation of a merged S-box & inverse, S-box alone, and inverse S-box alone. All use our best case basis with all optimizations.

best	XOR	NAND	NOR	NOT	MUX	total gates
merged	94	34	6	2	16	234
S-box	80	34	6	0	0	180
$(\text{S-box})^{-1}$	81	34	6	0	0	182

Table 2. Levels of Optimization. Here the first line shows the inverter based on the hierarchical structure (of 2.1); the next shows the improvement due to the removal of common subexpressions (of 3.1); the last shows the additional improvement from the NOR substitution (of 3.2). All use our best basis.

inverter	XOR	NAND	NOR	total gates
hierarchical	88	36	0	190
w/ low-level opt.	66	36	0	152
w/ NOR subst.	56	34	6	138

Table 3. Choice of Basis. Here we compare four different choices of basis: our best case, the best case of Mentens[6], the basis used by Satoh et al.[14], and our worst case. Each shows *our* complete implementation of a merged S-box & inverse, S-box alone, and inverse S-box alone. For comparison, all use the same level of optimization (using greedy-algorithm matrices and excluding the NOR substitution).

basis	type	XOR	NAND	NOT	MUX	total gates
ours	merged	107	36	2	16	253
	S-box	91	36	0	0	195
	$(\text{S-box})^{-1}$	91	36	0	0	195
Mentens	merged	118	36	0	16	271
	S-box	96	36	0	0	204
	$(\text{S-box})^{-1}$	97	36	0	0	206
Satoh	merged	119	36	3	16	275
	S-box	100	36	0	0	211
	$(\text{S-box})^{-1}$	99	36	0	0	209
worst	merged	131	36	0	16	293
	S-box	107	36	0	0	223
	$(\text{S-box})^{-1}$	106	36	0	0	222

The merged S-box and inverse was implemented as a Verilog module, shown in [1], including all our optimizations. While this compact implementation is intended for ASICs, we tested this implementation using an FPGA. Specifically, we used an SRC-6E Reconfigurable Computer, which includes two Intel processors and two Virtex II FPGAs. As implemented on one FPGA, the function evaluation takes just one tick of the 100 MHz clock, the same amount of time needed for the table look-up approach.

We also implemented a complete AES encryptor/decryptor on this same system, using our S-box. Certain constraints (block RAM access) of this particular system prevent using table lookup for a fully unrolled pipelined version; 160 copies of the table (16 bytes/round $\times$ 10 rounds) would not fit (we precompute the round keys). So for this system, our compact S-box allowed us to implement a fully pipelined encryptor/decryptor, where in the FPGA, effectively one block is processed for each clock tick. (In fact, we could even fit all 14 rounds needed for 256-bit keys.)

6 Conclusion

The goal of this work is an algorithm to compute the S-box function of AES, that can be implemented in hardware with a minimal amount of circuitry. This should save a significant amount of chip area in ASIC hardware versions of AES. Moreover, this area savings could allow many copies of the S-box circuit to fit on a chip for parallelism within each round, and perhaps enough to "unroll" the loop of 10 rounds for full pipelining (for non-feedback modes of encryption), on smaller chips.

This algorithm employs the multi-level representation of arithmetic in $GF(2^8)$, similar to the previous compact implementation of Satoh et al[14]. Our work shows how this approach leads to a whole family of 432 implementations, depending on the particular isomorphism (basis) chosen, from which we found the best one. (A detailed exposition of this nested-subfield approach, including specification of all constants for each choice of representation, is given in [1].) Another improvement involves replacing some XORs and NANDs with NORs. And in factoring the transformation (basis change) matrices for compactness, rather than rely on the greedy algorithm as in prior work, we fully optimized the matrices, using our tree search algorithm with pruning of redundant cases. This gave an improvement over the greedy algorithm in 73% of the 16×8 matrices and 44% of the 8×8 matrices that we optimized.

Our best compact implementation gives a merged S-box that is 20% smaller than the previously most compact version of [14]. We have shown that none of the other 431 versions possible with this subfield approach is as small. (We did not examine issues of timing, latency and delay, but these should be comparable with [14].) This compact S-box could be useful for many future hardware implementations of AES, for a variety of security applications.

Acknowledgements

Many thanks to Akashi Satoh for his patient and very helpful discussions.

References

1. D. Canright. A very compact Rijndael S-box. Technical Report NPS-MA-04-001, Naval Postgraduate School, September http://library.nps.navy.mil/uhtbin/hyperion-image/NPS-MA-05-001.pdf, 2004.
2. Pawel Chodowiec and Kris Gaj. Very compact FPGA implementation of the AES algorithm. In C.D. Walter et al., editor, *CHES2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 319–333. Springer, 2003.
3. Kimmo U. Jarvinen, Matti T. Tommiska, and Jorma O. Skytta. A fully pipelined memoryless 17.8 gbps AES128 encryptor. In *FPGA03*. ACM, 2003.
4. Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications*. Cambridge, New York, 1986.
5. F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, New York, 1977.

6. Nele Mentens, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. A systematic evaluation of compact hardware implementations for the Rijndael S-box. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, page 323333. Springer, 2005.
7. Sumio Morioka and Akashi Satoh. A 10 Gbps full-AES crypto design with a twisted-BDD S-box architecture. In *IEEE International Conference on Computer Design*. IEEE, 2002.
8. Sumio Morioka and Akashi Satoh. An optimized S-box circuit arthitecture for low power AES design. In *CHES2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 2003.
9. NIST. Recommendation for block cipher modes of operation. Technical Report SP 800-38A, National Institute of Standards and Technology (NIST), December 2001.
10. NIST. Specification for the ADVANCED ENCRYPTION STANDARD (AES). Technical Report FIPS PUB 197, National Institute of Standards and Technology (NIST), November 2001.
11. C. Paar. *Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields*. PhD thesis, Institute for Experimental Mathematics, University of Essen, Germany, 1994.
12. Vincent Rijmen. Efficient implementation of the Rijndael S-box. available at `http://www.esat.kuleuven.ac.be/~rijmen/rijndael/sbox.pdf`, 2001.
13. Atri Rudra, Pradeep K. Dubey, Charanjit S. Jutla, Vijay Kumar, Josyula R. Rao, and Pankaj Rohatgi. Efficient Rijndael encryption implementation with composite field arithmetic. In *CHES2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 171–184. Springer, 2001.
14. A. Satoh, S. Morioka, K. Takano, and Seiji Munetoh. A compact Rijndael hardware architecture with S-box optimization. In *Advances in Cryptology - ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 239–254. Springer, 2001.
15. Akashi Satoh. personal communication, July 2004.
16. Nicholas Weaver and John Wawrzynek. High performance, compact AES implementations in Xilinx FPGAs. available at `http://www.cs.berkeley.edu/~nweaver/papers/AES_in_FPGAs.pdf`, September 2002.
17. Johannes Wolkerstorfer, Elisabeth Oswald, and Mario Lamberger. An ASIC implementation of the AES Sboxes. In *CT-RSA*, volume 2271 of *Lecture Notes in Computer Science*, pages 67–78. Springer, 2002.

Author Index