Distinguishing Multiplications from Squaring Operations

Frederic Amiel[1], Benoit Feix[2], Michael Tunstall[3], Claire Whelan[4], and William P. Marnane[5]

[1] AMESYS,
1030, Avenue Guillibert de la Lauzière,
13794 Aix-en-Provence, Cedex 3, France
f.amiel@amesys.fr
[2] Inside Contactless
41 Parc Club du Golf, 13856 Aix-en-Provence, Cedex 3, France
bfeix@insidefr.com
[3] Department of Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol BS8 1UB, United Kingdom
tunstall@cs.bris.ac.uk
[4] TDS (Time Data Security) Ltd.,
2060 Castle Drive, Citywest Business Campus,
Naas Road, Dublin 24, Ireland
claire.whelan@tds.ie
[5] Department of Electrical and Electronic Engineering,
University College Cork, Cork, Ireland
liam@eleceng.ucc.ie

Abstract. In this paper we present a new approach to attacking a modular exponentiation and scalar multiplication based by distinguishing multiplications from squaring operations using the instantaneous power consumption. Previous approaches have been able to distinguish these operations based on information of the specific implementation of the embedded algorithm or the relationship between specific plaintexts. The proposed attack exploits the expected Hamming weight of the result of the computed operations. We extrapolate our observations and assess the consequences for elliptic curve cryptosystems when unified formulæ for point addition are used.

Keywords: Side channel attacks, differential power analysis, modular multiplication and exponentiation, RSA, square and multiply algorithm.

1 Introduction

Side channel attacks on RSA [23] target the algorithm for modular exponentiation, the computation of which is dependent on the private key. It has been shown in the literature that an attacker can derive a private key by observing the power consumption during the computation of a naïvely implemented modular exponentiation [17]. This attack targeted implementations of the square and

R. Avanzi, L. Keliher, and F. Sica (Eds.): SAC 2008, LNCS 5381, pp. 346–360, 2009.

multiply algorithm, which has been shown to be vulnerable to this technique, referred to as Simple Power Analysis (SPA). This vulnerability was present because the power consumption during the computation of a squaring operation was different to that of a multiplication, and could, therefore, be distinguished by simply monitoring the power consumption trace of the target device. This attack can allow an attacker to simply read the private key from a power consumption trace.

One of the first countermeasures proposed was a square and multiply *always* algorithm [11], which consists of a squaring operation followed by a (possibly fake) multiplication. While this algorithm achieves the effect of ensuring regular behaviour regardless of the value of the bits of the exponent, it has a large impact on efficiency. A more efficient approach, known as side channel atomicity, was proposed in [10]. While this approach does make the operations computed behave identically in terms of the instantaneous power consumption, other information being processed, such as the operand value being operated on, may leak information and provide an attacker with the necessary insight to recover the private key.

In this paper, we describe an attack that can be applied to algorithms implemented using side channel atomicity [8] without knowledge of the plaintext used. This is possible because the statistically expected Hamming weight of the result of a multiplication and a squaring operation has an exploitable difference, which is visible in the instantaneous power consumption. This highlights the importance of randomising the exponent used to calculate a modular exponentiation. A similar attack was previously proposed in [1] but requires that the architecture of a hardware implementation is known. The attack is also somewhat similar to the attack described in [27]. However, our attack is based on the distribution of the Hamming weights of the values being manipulated by a device, rather than a thorough analysis of the structure of hardware implementations of multipliers [27,29].

In some previously proposed attacks, similar power consumption traces during squaring (or doubling) operations in two separate acquisitions have been exploited by choosing or knowing the plaintexts being manipulated [14,19,30]. However, these attacks can be prevented by blinding the plaintext, and these attacks are not possible when classical padding schemes are used. The advantage of the attack described in this paper is that an attacker does not need any plaintext information. Indeed, we assume that an attacker does not have access to this information.

The implications of the proposed attack are explored further, and we analyse how attacks based on the statistically expected difference in Hamming weight of a multiplication and a squaring operation can be applied to implementations of the elliptic curve point scalar multiplication algorithm central to many elliptic curve schemes.

This paper is organised as follows. Section 2 describes why the Hamming weight is of interest in side channel analysis. Section 3 details the difference in expected Hamming weight between the results of a multiplication and squaring operation. Section 4 gives practical results using different long integer modular

multiplications on a classical ARM7 microprocessor to validate the theoretical analysis given. New attacks based on this difference analysis are presented on public key algorithms in Section 5. In Section 6 we analyse the countermeasures which can be used in implementations of the algorithms discussed. We conclude our research in Section 7.

Notation: The base of a value is determined by a trailing subscript, which is applied to the whole word preceding the subscript. For example, $\texttt{FE}_{16}$ is 254 expressed in base 16, $d = (d_{\ell-1}, d_{\ell-2}, \ldots, d_0)_2$ gives a binary expression for d.

2 The Hamming Weight

It has been demonstrated that in microprocessors the instantaneous power consumption is typically proportional to the Hamming weight of data being manipulated at a given point in time [8]. This difference in Hamming weight was first exploited in [17] to attack block ciphers. In this attack, an attacker acquires M power consumption traces (w_i for $i \in \{1, 2, \ldots, M\}$) during the computation of a block cipher, and chooses one bit b of an intermediate state generated during the computation of a block cipher. For a given hypothesis for a secret key value (or portion of the key) K this bit is predicted and used to determine whether a corresponding power consumption trace is a member of one of two possible sets. The first set S_0 will contain all the traces where b is equal to zero, and the second set S_1 will contain all the remaining traces, i.e. where the output bit b is equal to one.

A differential trace Δ is calculated by finding the average of each set and then subtracting the resulting values from each other, where all operations on waveforms are conducted in a pointwise fashion, i.e. this calculation is conducted on the first point of each acquisition to produce the first point of the differential trace, the second point of each acquisition to produce the second point of the differential trace, etc.

$$\Delta = \frac{\sum_{w_i \in S_0} w_i}{|S_0|} - \frac{\sum_{w_i \in S_1} w_i}{|S_1|}$$

A differential trace is produced for each value that K can take. In DES the first subkey will be treated in groups of six bits, so 64 (i.e. 2^6) differential traces will be generated to test all the combinations of six bits. The differential trace with the highest peak will validate a hypothesis for K.

In this paper we propose a novel attack based on a similar difference in Hamming weight. However, in the proposed attack it is not necessary to predict the value of a bit b, as the difference in Hamming weight is produced by the statistically expected Hamming weight of the result of the computed operations. A similar attack was previously proposed in [1] but requires that the architecture of a hardware implementation is known.

Another commonly used model to describe the power consumption is the Hamming distance model [8], where the power consumption is proportional to

the Hamming weight of data being manipulated at a given point in time XORed with some previous state. An analysis of how one would perform the proposed attack in this case is beyond the scope of this paper.

In smart card implementations of RSA it has traditionally been necessary to use a cryptographic coprocessor, which would typically be modelled using the Hamming distance model [8]. However, it has been practically demonstrated in [2] that the Hamming weight model applies to many public key implementations using arithmetic coprocessors. Some modern smart card chips are using 32-bit architectures [3,20], which allow for efficient implementations of RSA without requiring a cryptographic coprocessor. In these cases the Hamming weight model is likely to apply.

3 Defining the Difference in Hamming Weight

In this section we will describe the difference in Hamming weight of a multiplication and squaring operation for random inputs, to describe why the expected difference in Hamming weight between a multiplication and squaring operation occurs.

If we consider the classical binary method of long integer multiplication, the least significant bit will be set to one, if and only if both least significant bits in the multiplicands are equal to one. The probability of the least significant bit of the output being one is, therefore, equal to $1/4$. In the case of a squaring operation the least significant bit will be equal to one if the least significant bit of the input is equal to one. For a random input this will occur with probability $1/2$.

The next least significant bit has a higher chance of being equal to one if we consider a multiplication with random inputs. However, if we conduct a squaring operation this bit will always be equal to zero. This is because there are only two bits that could affect this bit in the output. The two values that could affect this bit are 10_2 and 11_2. In the case of 10_2 only the least significant bit in the output is set to one and nothing affects the second bit, in the case of 11_2 both bits will affect the second most significant bit. The bits will therefore cancel and produce a carry. I.e. the output of every squaring operation will be equal to 0 or 1 mod 4.

This reasoning can be continued with increased complexity for more significant bits and will be valid for all bit lengths, until more bits than half the total number of bits being considered are included. After this point the least significant bit ceases to directly affect each bit, and will only have an effect via the carry.

Defining the exact extent of this difference for n-bit operands is a non-trivial problem. A method for defining the probability density function of the product of uniformly distributed random variables is defined in [15]. This method defines a means of computing the probability density function of the result of the product of two random values that are distributed over a continuous uniform distribution. Where, for two random values uniformly distributed in the interval $[0, \ell]$, the product can take every real value in $[0, \ell^2]$. Random values generated

in a microprocessor will, by necessity, be distributed on a discrete uniform distribution. If we consider two discrete random values uniformly distributed in the interval $[0, \ell]$, the product cannot take every integer value in $[0, \ell^2]$. This is because no integer value in $(\ell, \ell^2]$ that is coprime with respect to the integer values in $[0, \ell]$ can be made from the product of two discrete random values distributed between $[0, \ell]$. The most efficient method of defining the probability distribution, and computing the expected Hamming weight of the result, is to simply count all the possible outcomes.

We will consider the multiplication and squaring of random values of bit length n, with no modular reduction. This is because we are interested in the distribution of the single-precision operations required to compute multi-precision operations. We will therefore assume that the values multiplied together will have an equal bit length. If we consider that the values multiplied together have a bit length of n, then the input values are, therefore, uniformly distributed over the integer values in the interval $[0, 2^n - 1]$.

The difference in the distributions can be demonstrated by evaluating the expected output of a multiplication and a squaring operation by calculating the mean Hamming weight of all the possible results, i.e. the expected Hamming weight of the result of squaring an n-bit value, X, is calculated as

$$\mathrm{E}(X^2) = \sum_{i=0}^{2^n-1} H(i^2) \cdot \Pr\left[X = i\right] = \frac{1}{2^n} \sum_{i=0}^{2^n-1} H(i^2)\,,$$

and the Hamming weight of the result of multiplying two n-bit values, X and Y, is calculated as

$$\mathrm{E}(X \cdot Y) = \sum_{i=0}^{2^n-1} \sum_{j=0}^{2^n-1} H(i \cdot j) \cdot \Pr\left[X = i \wedge Y = j\right] = \frac{1}{2^{2n}} \sum_{i=0}^{2^n-1} \sum_{j=0}^{2^n-1} H(i \cdot j)\,,$$

where H is a function that computes the Hamming weight in both cases.

This can be readily computed for bit lengths of less than, or equal to, 16. For bit lengths greater than 16 it starts to become time consuming to compute the expected Hamming weight of the output of a multiplication. Figure 1 shows the expected difference in Hamming weight for bit lengths between one and 16, and the difference appears to tend to slightly less than one as the bit length increases.

If we consider multiplication and squaring operations with 16-bit inputs, the reason for the difference in the expected result can be demonstrated if we consider the probability of each bit being equal to one. For random, uniformly distributed, 16-bit inputs the probability of each of the 32 bits in the output being equal one for a multiplication and squaring operation can be derived if all the possible inputs are considered. A plot of the probabilities for each bit for the multiplication and squaring operation is given in Figure 2. Further details on this expected difference for 32-bit variables are given in the Appendix A.

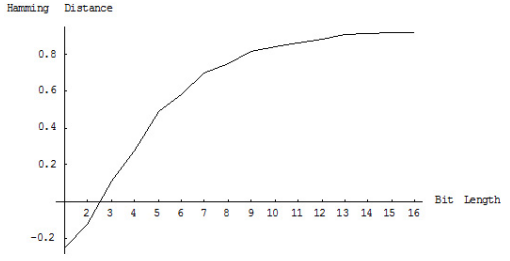

Fig. 1. The expected difference in Hamming weight between the output of a multiplication and a squaring operation, for bit lengths 1 to 16

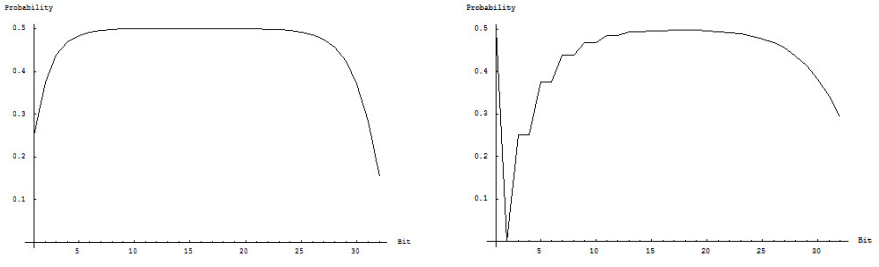

Fig. 2. The probability that each bit of the result of a multiplication (left) and a squaring operation (right) is equal to one with random 16-bit inputs

4 Demonstrating the Difference in Practice

Certain multiplication algorithms were implemented on a standard 32-bit microprocessor. The results of manipulating the power traces acquired while these multiplication algorithms were being computed are described in this section.

Long Integer Multiplication. A 128-bit multiplication using the long integer multiplication algorithm was implemented on a microprocessor and 3000 acquisitions[1] were taken for multiplications and squaring operations with random, uniformly distributed inputs. The implementation was based on the description given in [18], and is given in Algorithm 1.

The difference between the two average traces is shown in Figure 3. There are four peaks in the trace that correspond to the four squaring operations conducted by the chip to compute the square of the input, i.e. for $X = (x_3, x_2, x_1, x_0)_b$, where b is 2^{32}, there will be four occurrences in the 16 multiplications where $i = j$ when $x_i \cdot x_j$ is computed. If averaged traces corresponding to the same operation are subtracted from each other no significant peaks are produced.

[1] Similar results are possible with 500 traces. However, the results are not as clear.

Algorithm 1. Long Integer Multiplication

Input: $X = (x_{z-1}, \ldots, x_1, x_0)_b$, $Y = (y_{z-1}, \ldots, y_1, y_0)_b$
Output: $W = (w_{2z-1}, \ldots, w_1, w_0)_b = X \cdot Y$

$W \leftarrow 0$
for $i = 0$ **to** $z - 1$ **do**
  $c \leftarrow 0$
  for $j = 0$ **to** $z - 1$ **do**
    $(uv)_b \leftarrow w_{i+j} + x_j \cdot y_i + c$
    $w_{i+j} \leftarrow v$; $c \leftarrow u$
  end
  $w_{2z-1} \leftarrow u$
end
return W

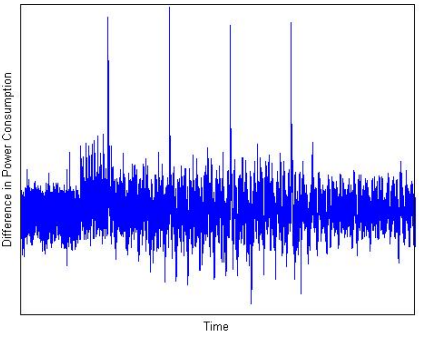

Fig. 3. The difference between two averaged power consumptions for long integer multiplication

Montgomery Multiplication. One of the most common methods of calculating modular multiplication is using Montgomery multiplication [21]. This is because of its efficiency, especially as it can be parallelised in hardware and does not require any time-consuming word-by-word divisions.

Montgomery multiplication [21] does not return the simple product of X and Y modulo M. The algorithm actually returns XYR^{-1} mod M, where R^{-1} mod M is introduced by the algorithm ($R = b^z$), which imposes certain restrictions on its use. The conditional subtraction has been shown to be unnecessary, and undesirable in a secure implementation, and was not included in our implementation [25,26].

A description of Montgomery multiplication is given in Algorithm 2. Here b is the size of the basic data unit, usually a machine word, and z is the number of words in the representation of M, X and Y.

As previously, a 128-bit multiplication algorithm was implemented and 3000 acquisitions were taken for multiplications and squaring operations with random,

Algorithm 2. Montgomery Multiplication

Input: $X = (x_{z-1}, \ldots, x_1, x_0)_b$, $Y = (y_{z-1}, \ldots, y_1, y_0)_b$,
$M = (m_{z-1}, \ldots, m_1, m_0)_b$, $R = b^z$ with $gcd(M, b) = 1$, and $M' = -M^{-1} \mod b$

Output: $A = (a_{z-1}, \ldots, a_1, a_0)_b = X \cdot Y \cdot R^{-1} \mod M$

$A \leftarrow 0$
for $i = 0$ **to** $z - 1$ **do**
  $u_i \leftarrow (a_0 + x_i \cdot y_0)M' \mod b$
  $A \leftarrow (A + x_i \cdot Y + u_i \cdot M)/b$
end
if $A \geq M$ **then** $A \leftarrow A - M$
return A

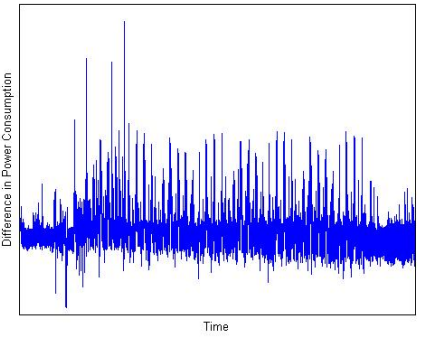

Fig. 4. The difference between two averaged power consumptions for Montgomery multiplication

uniformly distributed inputs. The difference in the average trace produced by each set of acquisitions is shown in Figure 4.

The first peak will be produced by the calculation of $a_0 + x_0 \cdot y_0 \mod b$, as a_0 is set to zero so the difference in the distribution will be visible even where this can be calculated with one instruction (e.g. on ARM microprocessors).

In the implementation analysed the processor computed $A \leftarrow A + u_0 \cdot M$ followed by $A \leftarrow A + x_i \cdot Y$. The group of peaks following the first peak are caused by the repeated manipulation of u_0 when it is multiplied by M. The peaks are dependent on the value of M, and therefore M', that is being used and will vary from one analysis to another.

This is followed by a large peak that corresponds to the computation of $x_0 \cdot y_0$, which is subsequently combined with $u_0 \cdot M$ by adding the result to A. The next group of three peaks are created by the manipulation of A when it is combined with $u_1 \cdot M$.

This is followed by three groups of small peaks that correspond to the multiplication of three squaring operations conducted by the chip to compute the square of the input, i.e. three instances where $i = j$ when $x_i \cdot y_j$ is computed.

5 Exploiting the Difference in Tamper Resistant Cryptographic Primitives

Once an exponentiation algorithm has been chosen, for instance Barrett or Montgomery exponentiation [5,21], a common countermeasure to protect embedded implementations from Simple Power Analysis, consists in using side channel atomicity. This was introduced in [10], where they deem an algorithm to be secure if it can be broken down into indistinguishable blocks. In this section we describe how these schemes can be attacked by observing the difference between a multiplication and a squaring operation.

5.1 Recovering the Exponent in Atomic Exponentiations

The simplest exponentiation algorithm is the square and multiply algorithm, that functions by scanning the bits of an exponent from left to right. An accumulator is initially set to one and for each bit of the exponent scanned the accumulator is operated upon. For each bit the accumulator is squared, and when a bit is equal to one the accumulator is multiplied by the value being raised to the power of the exponent. The square and multiply atomic exponentiation algorithm simply means that squaring operations are computed using the same algorithm as multiplications and the side channel becomes identical [10].

If a series of power consumption traces are taken, the points corresponding to each operation (multiplication or squaring) can be identified using a method similar to that described in [27] for identifying multiplications with a constant value. The average power consumption trace of each operation can be compared to the operation preceding, or following, it by performing a pointwise subtraction. If this corresponds to subtracting the power consumption trace of a squaring from that of a multiplication peaks will be visible (as shown in Figure 3), in the case where the opposite occurs the same peaks will occur but will be negative. It is interesting to note that an attacker does not need to have any knowledge of the values being manipulated.

An attacker would therefore be able to determine a k-bit exponent by making $\frac{3}{2}k - 1$ comparisons, i.e. comparing each operation with one neighbouring operation. This can be decreased by a factor of two, where an attacker can be sure that each comparison gives noise free information, by only including each operation in a comparison once.

If we consider the (M, M^3) algorithm, as described in [10], analysing the power consumption traces is sufficient to decrease the security of the algorithm. However, we cannot recover the entire private exponent d. The (M, M^3) algorithm functions in a similar manner to the square and multiply algorithm, but there are three possible cases when parsing the bits of the private exponent from left to right. When $d_i = 0$ a squaring operation is performed. When $d_i d_{i-1} = 10_2$ the device computes a squaring operation, a multiplication with M and then a squaring operation. The third case occurs when $d_i d_{i-1} = 11_2$, where the device computes two squaring operations followed by a multiplication with M^3.

In the remainder of this section we will denote a multiplication by M and a squaring operation by S. Any sequence of operations MSM is particular because it indicates that the last two operations correspond to the secret bits $d_i d_{i-1} = 10_2$ and that $d_{i+2} d_{i+1} = 11_2$. Indeed, this sequence can only be part of a longer sequence $SSMSM$. We can also identify any bits of an exponent set to zero when there are more than two consecutive squaring operations.

Through simulations of this attack we were able to determine that an attacker can retrieve, on average, 37% of the bits of a private exponent by exploiting the sequence MSM, and a further 17% of the bits by identifying repeated squaring operations. Thus, an attacker would be able to retrieve 54% of the bits of a private exponent using the attack method proposed above.

In case where the public exponent is small (for instance 3 or $2^{16}+1$), half of the most significant bits of d are intrinsically leaked as showed in [7]. Thus, combined with the side channel leakage, up to 3/4 of the bits of a private exponent could be considered to be recoverable by an attacker. However, there are currently no factorisation techniques in the literature that can benefit from such partial information, although an interesting approach has been published in [13], where the authors assume that the exponent is modified by a small random value. How the proposed attack can be applied to an implementation where this occurs is discussed in Section 6.2.

5.2 Recovering the Scalar in ECC Using Unified Addition Formulæ

In the context of Elliptic Curve Cryptosystems (ECC), the ability to distinguish a multiplication operation from a squaring operation can also facilitate the extraction of secret information. The calculation of the point scalar multiplication of $r\boldsymbol{P}$, where r is a secret scalar value, $\boldsymbol{P}$ is a point on the prescribed elliptic curve, and the operation of $r\boldsymbol{P}$ is known as point scalar multiplication is central to a number of ECC schemes, such as EC-DH [6]. One of the most side-channel naïve methods to calculate $r\boldsymbol{P}$ is the double and add method, which involves accumulatively doubling and adding the point $\boldsymbol{P}$, the sequence of which is determined by the binary representation of r [4]. This method is inherently vulnerable to Simple Power Analysis and other side-channel attacks.

A countermeasure, known as unified addition formulæ, to make the double operation indistinguishable from the addition operation was proposed in [6,9]. This method defined formulæ for the calculation for point addition and point doubling, which is equivalent for both operations. Specifically, the slope for each operation is equivalent. For example, the slope calculated during the addition of the points $\boldsymbol{P} = (x_1, y_1)$, $\boldsymbol{Q} = (x_2, y_2)$ is

$$\lambda = \frac{x_1^2 + x_1 x_2 + x_2^2 + a_2 x_1 + a_2 x_2 + a_4 - a_1 y_1}{y_1 + y_2 + a_1 x_2 + a_3},$$

regardless of whether $\boldsymbol{P}$ is equal, or not equal, to $\boldsymbol{Q}$. Hence, no discernible difference between the addition and doubling of a point is present in the formula. In light of the work described in this paper, a difference between these operations can be identified. The calculation of $x_1 \cdot x_2$ in the calculation of λ will allow

an attacker to determine whether an addition or a doubling operation is being performed, since when a double is performed $x_1 \cdot x_2 = x_1^2$, and will be vulnerable to the attack process described in Section 5.1.

Similarly, this potential exploit can be witnessed when the elliptic curve points are represented and operated on as projective coordinates, which will be the case in most practical implementations. Unified formula for point addition and multiplication using projective coordinates was also given by [9] and further examined by [24]. In this case the addition of the points $\boldsymbol{P} = (X_1, Y_1, Z_1)$, $\boldsymbol{Q} = (X_2, Y_2, Z_2)$, with $x_i = X_i/Z_i$ and $y_i = Y_i/Z_i$ is

$$X_3 = 2FW \qquad Y_3 = R(G - 2W) \qquad Z_3 = 2F^3$$

where $U_1 = X_1Z_2$, $U_2 = X_2Z_1$, $S_1 = Y_1Z_2$, $S_2 = Y_2Z_1$, $Z = Z_1Z_2$, $T = U_1 + U_2$, $M = S_1 + S_2$, $F = ZM$, $L = MF$, $G = TL$, $R = T^2 - U_1U_2 + AZ^2$ and $W = R^2 - G$. Notice that, when a point doubling operation is being performed, the computation of Z and U_1U_2 in R will be squaring operations and, hence, our attacks can also be applied to such implementations.

6 Countermeasures

As presented in the previous sections, both side channel atomicity and unified point addition formulæ are potentially vulnerable to attack according to the expected difference highlighted in Section 3. Some of the countermeasures that could be used to prevent this attack are discussed in this section.

6.1 Blinding

The most common countermeasure used to protect RSA against DPA consists in modifying plaintext with a random value, either using an additive method $m_b = m + r_1 n \mod r_2 n$, where r_1 and r_2 are random values, or in a multiplicative way $m_b = r_1^e \cdot m$ where e is the public exponent. With such a countermeasure, classical DPA [17], and related attacks (such as the attacks presented in [19] or [2]), can no longer be applied.

However, plaintext blinding is not sufficient to protect against the attack described in this paper. It is, therefore, necessary to change the order of the multiplication and squaring operations between different exponentiations. The most common solution consists of computing $d_b = d + r_1\phi(N)$, where ϕ is Euler's Totient function and r_1 is a small random value [16]. An equivalent solution can be used to protect the double and add algorithm [11].

6.2 The Big Mac Attack

In [27], Walter presented the Big Mac attack, which demonstrates a powerful attack on devices with a high level of side channel leakage, i.e. devices where only a few power consumption traces are required to successfully conduct a power analysis. Furthermore, in [28] it is explained that using longer keys in asymmetric

cryptosystems improves the probability of a Big Mac attack succeeding. This idea can be extended here to obtain a kind of Big Mac power attack which, would enable the attack described in Section 5.1 to be conducted on one power consumption trace. In such a case, the blinding of d would not provide adequate protection to defend against the attack described in this paper. The attack could also be applied to other schemes, such as the Diffie-Hellman key exchange [12] and the DSA [22].

An attack on a single power consumption trace would consist of identifying the points in a power consumption trace that correspond to the computation of $x_j \cdot y_i$ when $i = j$ (e.g. in Algorithm 1), and extract these points where each point its then treated as a separate trace. These small traces can then be used in place of a trace representing the entire operation in exactly the same manner described in Section 5.1. The points to be used can be identified by analysing an unprotected algorithm, as described in [2].

The success of this attack will depend on the length of the key and the word size of the processor, i.e. long keys and small word size will provide an accurate average and raise the probability of achieving a successful attack [28]. This demonstrates that the blinding of the private key d may not be adequate to prevent the attack presented in this paper.

7 Conclusion

This paper shows that the statistically expected difference in operations computed by a microprocessor can be used to distinguish between a multiplication and a squaring operation. All that is required is that the plaintexts used contain enough variation that the computations adhere to the distributions defined in Section 3. This is an improvement over previously published results [14,19,27,29,30], as the described attack requires no knowledge of the plaintext being manipulated or of the architecture of the multiplier. Moreover, the proposed attacks will work when classical padding schemes are used. Further work that is being conducted by the authors consists of analysing the algorithms that are potentially vulnerable to this attack, and the development of inexpensive countermeasures.

Acknowledgements

The authors would like to thanks James Curran of University College Cork for helpful discussions in the early stages of this work. The work described in this paper has been supported in part by the European Commission IST Programme under Contract IST-2002-507932 ECRYPT and EPSRC grant EP/F039638/1 “Investigation of Power Analysis Attacks”. Also the support of the Informatics Commercialisation initiative of Enterprise Ireland is gratefully acknowledged.

References

1. Akishita, T., Takagi, T.: Power analysis to ECC using differential power between multiplication and squaring. In: Domingo-Ferrer, J., Posegga, J., Schreckling, D. (eds.) CARDIS 2006. LNCS, vol. 3928, pp. 151–164. Springer, Heidelberg (2006)
2. Amiel, F., Feix, B., Villegas, K.: Power analysis for secret recovering and reverse engineering of public key algorithms. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 110–125. Springer, Heidelberg (2007)
3. ARM. SecurCore family, `http://www.arm.com/products/CPUs/families/SecurCoreFamily.html`
4. Avanzi, R.-M., Cohen, H., Doche, C., Frey, G., Lange, T., Nguyen, K., Verkauteren, F.: Handbook of Elliptic and Hyperelliptic Curve Cryptography. Taylor & Francis Ltd, Abington (2008)
5. Barrett, P.: Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 311–323. Springer, Heidelberg (1987)
6. Blake, I., Seroussi, G., Smart, N.: Advances in Elliptic Curve Cryptography. Lecture Note Series, vol. 317. Cambridge University Press, London Mathematical Society (2005)
7. Boneh, D., Durfee, G., Frankel, Y.: An attack on RSA given a small fraction of the private key bits. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 25–34. Springer, Heidelberg (1998)
8. Brier, É., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
9. Brier, É., Joye, M.: Weierstraß elliptic curves and side-channel attacks. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 335–345. Springer, Heidelberg (2002)
10. Chevallier-Mames, B., Ciet, M., Joye, M.: Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. IEEE Transactions on Computers 53(6), 760–768 (2004)
11. Coron, J.-S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999)
12. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory 22(6), 644–654 (1976)
13. Fouque, P.-A., Kunz-Jacques, S., Martinet, G., Muller, F., Valette, F.: Power attack on small RSA public exponent. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 339–353. Springer, Heidelberg (2006)
14. Fouque, P.-A., Valette, F.: The doubling attack – why upwards is better than downwards. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 269–280. Springer, Heidelberg (2003)
15. Glen, A.G., Leemis, L.M., Drew, J.H.: Computing the distribution of the product of two continuous random variables. Computaional Satatistics and Data Analysis 44(3), 451–464 (2004)
16. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
17. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)

18. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1997)
19. Messerges, T.S., Dabbish, E.A., Sloan, R.H.: Power analysis attacks of modular exponentiation in smartcards. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 144–157. Springer, Heidelberg (1999)
20. MIPS-Technologies. SmartMIPS ASE, `http://www.mips.com/content/Products/`
21. Montgomery, P.: Modular multiplication without trial division. Mathematics of Computation 44, 519–521 (1985)
22. National Institute of Standards and Technology. Digital signature standard (DSS), FIPS–186-2 (2000)
23. Rivest, R., Shamir, A., Adleman, L.M.: Method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
24. Stebila, D., Thériault, N.: Unified point addition formulæ and side-channel attacks. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 354–368. Springer, Heidelberg (2006)
25. Walter, C.D.: Montgomery exponentiation needs no final subtractions. Electronic Letters 35(21), 1831–1832 (1999)
26. Walter, C.D.: Montgomery's multiplication technique: How to make it smaller and faster. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 80–93. Springer, Heidelberg (1999)
27. Walter, C.D.: Sliding windows succumbs to big mac attack. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 286–299. Springer, Heidelberg (2001)
28. Walter, C.D.: Longer keys may facilitate side channel attacks. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 42–57. Springer, Heidelberg (2004)
29. Walter, C.D., Samyde, D.: Data dependent power use in multipliers. In: Montuschi, P., Shwarz, E. (eds.) 17th Symposium on Computer Arithmetic (ARITH), pp. 4–12. IEEE, Los Alamitos (2005)
30. Yen, S.-M., Lien, W.-C., Moon, S.-J., Ha, J.C.: Power analysis by exploiting chosen message and internal collisions – vulnerability of checking mechanism for RSA-decryption. In: Dawson, E., Vaudenay, S. (eds.) Mycrypt 2005. LNCS, vol. 3715, pp. 183–195. Springer, Heidelberg (2005)

A Appendix

In Section 3 we discussed the expected Hamming weight of multiplication and squaring operations for random, uniformly distributed, 16-bit inputs. Given the complexity of evaluating all of the possible inputs to a multiplication, it is not possible to evaluate the expected Hamming weight and the corresponding distribution of the individual bits for larger bit length. Given that the implementations described in Section 4 are on a 32-bit chip, it would be helpful to attempt to describe the corresponding distribution.

To characterise the distribution of the individual bits in the result of a 32-bit squaring operation all the possible input values were evaluated and the result is plotted on the right hand side of Figure 5. It is not possible to evaluate a 32-bit multiplication in the same way as there are 2^{64} possible inputs to a

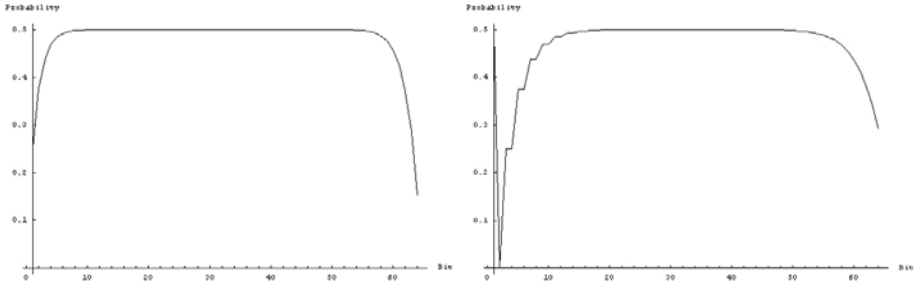

Fig. 5. The distribution of the individual bits of the result of a multiplication (left) and a squaring operation (right) with random 32-bit inputs

32-bit multiplication. An approximation to the distribution was generated by evaluating the product of 2^{32} pairs uniformly distributed 32-bit random values.

The form of the difference is similar to that shown of 16-bit operations in Section 3, but with a larger region where the distribution of the bits are identical.

Given the very regular nature of multiplication algorithms, it would seem reasonable to assume that the same difference will occur for all bit lengths. However, it is not possible to demonstrate this because of the complexity of evaluating all the possible inputs.