Int. J. Inf. Secur. (2011) 10:255–266
DOI 10.1007/s10207-011-0135-4

REGULAR CONTRIBUTION

Using templates to distinguish multiplications from squaring operations

Neil Hanley · Michael Tunstall · William P. Marnane

Published online: 27 May 2011

Abstract Since side channel analysis was introduced as a method to recover secret information from an otherwise secure cryptosystem, many countermeasures have been proposed to prevent leakage from secure devices. Among these countermeasures is side channel atomicity that makes operations indistinguishable using side channel analysis. In this paper, we present practical results of an attack on RSA signature generation, protected in this manner, based on the expected difference in Hamming weight between the result of a multiplication and a squaring operation. This work presents the first attack that we are aware of where template analysis can be used without requiring an open device to characterize an implementation of a given cryptographic algorithm. Moreover, an attacker does not need to know the plaintexts being operated on and, therefore, blinding and padding countermeasures applied to the plaintext do not hinder the attack in anyway.

Keywords Side channel analysis · Hamming weight · Template attack · RSA

N. Hanley · W. P. Marnane
Claude Shannon Institute for Discrete Mathematics, Coding and Cryptography, Department of Electrical and Electronic Engineering, University College Cork, Cork, Ireland
e-mail: neilh@eleceng.ucc.ie

W. P. Marnane
e-mail: l.marnane@ucc.ie

M. Tunstall (✉)
Department of Computer Science, University of Bristol, Merchant Venturers Building, Woodland Road, Bristol BS8 1UB, UK
e-mail: tunstall@cs.bris.ac.uk

1 Introduction

Side channel analysis of cryptographic devices can potentially allow the recovery of secret information by an attacker, even when the underlying algorithms are mathematically secure. An attacker would seek to identify a suitable side channel, such as timing [21], power [22], and electromagnetic [14] leakage. Many attacks using these side channels have been described in the literature such as differential [7,22] and template [9] attacks, as well as countermeasures [10,12,20] intended to thwart them.

In [2], a method of distinguishing between multiplication and squaring operations using acquired traces of power consumption or electromagnetic emanations was demonstrated. In this paper, we extend that work by using template attacks to distinguish between these operations to propose a practical attack. The attack functions with no knowledge of the input to the algorithm, other than that it is random and uniformly distributed. Therefore, the attack is unaffected by padding and/or blinding applied to the input message.

Typically, generating templates to attack a device requires an open device where cryptographic keys are known or can be controlled. The device is then characterized and templates are generated. These templates are then used to derive a cryptographic key on an identical device with very few traces. In this paper, we present an attack where a device with a known key can be used to generate templates. Where a verification function is available that uses the same subfunctions as a signature generation function, this function could be used to generate templates under the assumption that the public key is known.

Previous work applying templates to public key primitives, such as that described in [18,24], has targeted the precise Hamming weight of intermediate values. The work described in this paper targets the distribution of the

Hamming weight of intermediate values but differs in that the value itself is unknown even when generating the templates. The templates are generated so as to be able to distinguish whether an operation is a squaring operation or multiplication, rather than directly recovering an intermediate value.

The paper is organized as follows: in Sect. 2, the side channel atomic RSA algorithm is described as the target of this work. Further implementation details are given with a description of the Montgomery multiplication algorithm. An overview of template attacks is also given in this section. In Sect. 2.4, the previous work dealing with the the Hamming weight difference between multiplications and squaring operations is revisited. Both simulated and practical results of our attack using templates are provided in Sect. 3. In Sect. 4, we discuss how the attack would change given a device that leaks information corresponding to the Hamming distance model and demonstrate that the power consumption of the ARM7 used in Sect. 3.2 shows an example of both models. The potential application of our attack to implementations secured with various other countermeasures is described in Sect. 5. Finally, we draw our conclusions in Sect. 6.

2 Preliminaries

2.1 RSA

The principal operation of the RSA [32] signature scheme is a modular exponentiation in $(\mathbb{Z}/N\mathbb{Z})^*$. That is, a signature s is generated from a message m by computing $s = \mu(m)^d \bmod N$, where d is the private key, N is the product of two large primes, and μ is an appropriate padding function. This signature can be verified by checking whether $\mu(m)$ is equal to $s^v \bmod n$. We define $d \equiv v^{-1} \pmod{\phi(N)}$ where ϕ is Euler's Totient function.

One of the most widely known algorithms for implementing an exponentiation is the square-and-multiply algorithm, where an exponent e is read from left-to-right bit-by-bit. Starting with an accumulator set to one, a squaring operation is performed if a bit is equal to zero, and a squaring operation followed by a multiplication (with the value being raised to e) if a bit is equal to one.

It has been shown that bit values of an exponent can be distinguished by observing a suitable side channel, such as the execution time [21], power consumption [22], or electromagnetic emanations [14,30]. This is because an attacker can potentially derive the difference between a multiplication and a squaring operation if, for example, a different number of registers in a hardware implementation is used (which would lead to the operations having different power signatures), or they require a different amount of time to compute.

The simplest countermeasure is to always perform the multiplication and squaring operations [12]; however, this incurs a significant performance penalty. Another option is to remove the difference between a squaring operation and a multiplication. This means that, to square a value x, the calculation of $x^2 \bmod N$ is replaced with $x \cdot x \bmod N$. This idea is put forward in [10], where two instructions are defined as side channel equivalent if they are indistinguishable through side channel analysis, and algorithms are termed side channel atomic if the algorithm can be broken down into indistinguishable blocks. This principle, applied to the square-and-multiply algorithm, is demonstrated in Algorithm 1.

Algorithm 1: Side channel atomic square and multiply algorithm

Input: $m, x < m, e \geq 1, \ell$ the binary length of e (i.e., $2^{\ell-1} \leq e < 2^{\ell}$)
Output: $A_0 = x^e \bmod m$

```
1 A0 ← 1;  A1 ← x;  i ← ℓ − 2;  k ← 0
2 while i ≥ 0 do
3     A0 ← A0 · Ak mod m
4     k ← k ⊕ bit(e, i)
5     i ← i − ¬k
6 end
7 return A0
```

This algorithm would prevent an attacker from being able to distinguish a multiplication from a squaring operation by simply observing the difference in a side channel, since an optimized squaring operation is not used.

2.2 Montgomery multiplication

When implementing a modular exponentiation, a commonly used multiplication algorithm is Montgomery multiplication [26], since the modular reduction is interleaved with the multiplication. The result of a modular multiplication using this algorithm is not the product of x and y modulo m. The algorithm actually returns $x\,y\,R^{-1} \bmod m$, where $R^{-1} \bmod m$ is introduced by the algorithm ($R = b^n$, where the modulus consists of n words of size b), shown in Algorithm 2. In order to use Montgomery multiplication, x and y need to be converted to their Montgomery representation, i.e., $\widetilde{x} \leftarrow x\,R \bmod m$ and $\widetilde{y} \leftarrow y\,R \bmod m$. Then, when $\widetilde{x}$ and $\widetilde{y}$ are multiplied together using Montgomery multiplication, the result is $x\,y\,R \bmod m$. This algorithm requires $2n\,(n+1)$ single-precision multiplications [25].

Algorithm 2 has been demonstrated to be vulnerable to side channel because of the final conditional subtraction that takes place if $A \geq M$. This conditional subtraction affects the entire execution time of an exponentiation leading to an attack based on total time taken to compute an exponentiation [37]. It has also been shown that individual subtractions will be visible in the power consumption, or electromagnetic

Algorithm 2: Montgomery multiplication

Input: $m = (m_{n-1}, \ldots, m_1, m_0)_b, x = (x_{n-1}, \ldots, x_1, x_0)_b$, $y = (y_{n-1}, \ldots, y_1, y_0)_b$ with $0 \leq x, y < m$, $R = b^n$, $\gcd(m, b) = 1$ and $m' = -m^{-1} \bmod b$.
Output: $A \leftarrow x\, y\, R^{-1} \bmod m$.

```
A ← 0
for i = 0 to n − 1 do
    u_i ← (a_0 + x_i y_0)m' mod b
    A ← (A + x_i y + u_i m)/b
end
if A ≥ m then A ← A − m
return A
```

emanations, leading to more efficient attacks [35,36]. The simplest countermeasure would be to always conduct the subtraction and take the result as required. However, this approach is problematic since the bit that is used to make this choice may be visible in a side channel. Moreover, this could potentially be attacked by a fault attack, where a fault in a subtraction that does not affect the result of an exponentiation identifies a dummy subtraction [39]. More effective countermeasures involve increasing the number of iterations of the main loop so that the final subtraction becomes unnecessary [17,34]. However, these attacks and countermeasures do not have any impact on the attack described below and can be considered beyond the scope of this paper.

2.3 Template attacks

Template attacks, introduced by Chari et al. [9], are a powerful form of side channel attack that allow secret information to be extracted from a device with very few power traces. They work on the premise that an attacker has an identical device to that which is being attacked under his control, such that he can choose, or knows, all the inputs to the device (this includes any cryptographic keys). This allows an attacker to characterize the power consumption of the device prior to an attack. A template attack is, therefore, a two stage attack, as outlined in [13]. The first stage is template generation, where a device under the attacker's control is used to characterize the power consumption. This is followed by template classification where, using the templates generated previously, information is extracted from the device under attack.

In the attack presented in this paper, we can consider a more relaxed model than that proposed by Chari et al. since we do not require a device where cryptographic keys need be modified to generate templates. An attacker merely needs to be able to execute a multiplication or squaring operation where the input is known to be random. Assuming that an attacker wishes to recover a RSA private key, this can be achieved by using an identical device where the private key is known. Another approach would be to use the verification function in the same device assuming that the public key is known.

2.3.1 Template generation

Templates consist of estimates for the mean vector $\mathbf{m}_i$ and noise covariance matrix $\mathbf{C}_i$, as defined in Eqs. 1 and 2 where $i \in \{1, \ldots, n\}$ and n signifies the number of different possible values or operations that an attacker wishes to analyze. These values are each constructed from a large number of traces, $\mathbf{t}_j$ where $j \in \{1, \ldots, k\}$, and k is typically in the region of 1,000. However, the actual value of k will vary from one device to another and is also dependent on the size of the bus in the device under attack, as well as the operation being performed.

$$\mathbf{m}_i = \frac{1}{k} \sum_{j=1}^{k} \mathbf{t}_{i,j} \tag{1}$$

Here $\mathbf{t}_{i,j}$ represents the jth acquisition of the ith possible value.

$$\mathbf{C}_i = \frac{1}{k-1} \sum_{j=1}^{k} \left(\mathbf{t}_{i,j} - \mathbf{m}_i\right) \left(\mathbf{t}_{i,j} - \mathbf{m}_i\right)^T \tag{2}$$

When recording power traces, a high sampling rate is often used, to capture small fluctuations in power consumption. This leads to the length of a trace being very large (e.g., for the 1,024-bit modular multiplication traces on the ARM7 used in Sect. 3.2, a sampling rate of 100 MS/s led to a trace length of over a million points). It is computationally unfeasible to construct templates including all of these points. Methods are available to reduce redundant information in a trace, such as integration within a clock cycle or extracting the maximum value per clock cycle [23]. These methods significantly reduce processing time by removing redundancy within the trace without a significant loss in the information that leaks from the trace. However, for the construction of templates, further reduction is required to extract the features that the templates will be based on. One option is to sum the absolute differences of the mean traces and select the required number of highest points, as described in [31]. It is typically only necessary to retain one point per clock cycle since the inclusion of other points does not add any further information. Note that certain trace reduction methods already have this effect.

2.3.2 Template classification

To extract key information, an attack trace $\mathbf{t}$ is required, which is a power trace of the operation that is being targeted. The trace must first be reduced in size and features selected, i.e.,

extracting the same information that was used when generating templates. For each of the n templates, the probability of the trace corresponding to a given template can be calculated using Eq. 3:

$$\Pr(\mathbf{t} \mid \mathbf{m}_i, \mathbf{C}_i) = \frac{1}{\sqrt{(2\pi)^n |\mathbf{C}_i|}} e^{-\frac{1}{2}(\mathbf{t}-\mathbf{m}_i)\mathbf{C}_i^{-1}(\mathbf{t}-\mathbf{m}_i)^T} \quad (3)$$

The probability of each of the n possible templates can then be calculated using Bayes' theorem, which is given in Eq. 4:

$$\Pr(k_j \mid \mathbf{t}) = \frac{\Pr(\mathbf{t} \mid k_j)\ \Pr(k_j)}{\sum_{l=1}^{K} \Pr(\mathbf{t} \mid k_l)\ \Pr(k_l)} \quad (4)$$

The success of the attack is increased if a set of D power traces, $\mathbf{T}$, for a constant secret key is available. In this scenario, Bayes' theorem applied iteratively can be used, thereby increasing the power of the attack, as shown in [29]. Numerical errors during the classification due to the exponentiation in Eq. 3 can be avoided by taking the logarithm of the probability template as described in [23].

2.4 The difference in hamming weight

It was observed in [2] that the expected Hamming weight of the result of a multiplication is different than the result of a squaring operation. The power consumption, and electromagnetic emanations, of a microprocessor is often proportional to the Hamming weight of the values being manipulated [7,14]. The difference in the expected Hamming weight of the result of a multiplication and a squaring operation can be observed by measuring these side channels [2]. We can note that this observation applies to a single-word operation and that this difference will be present in any word-by-word multiplication algorithm.

This observation assumes that the power consumption, or electromagnetic emanations, of the device being attacked is proportional to the Hamming weight of the values being manipulated, referred to as the Hamming weight model. However, many devices also conform to the Hamming distance model [7], i.e., the power consumption, or electromagnetic emanations, of the device being attacked is proportional to the number of bits that change at a given point in time. This is of particular relevance to CMOS logic since a gate will consume little power unless it changes state. However, other features in a microprocessor can correspond to the Hamming weight model. In this paper, we assume that the target device corresponds to the Hamming weight model (we discuss this topic in more depth in Sect. 4).

If we assume that a multiplication takes place between two random uniformly distributed κ-bit values, the probability of a given bit being equal to zero, or one, can be computed by observing the distribution of each bit over all the possible multiplicands. If $\kappa = 16$, for example, this can readily be computed, as is shown in Fig. 1. A significant difference in the probability of the bits of the least significant byte being set to one can be observed. This difference will be present in each of the single-precision multiplications that are required to compute a Montgomery multiplication.

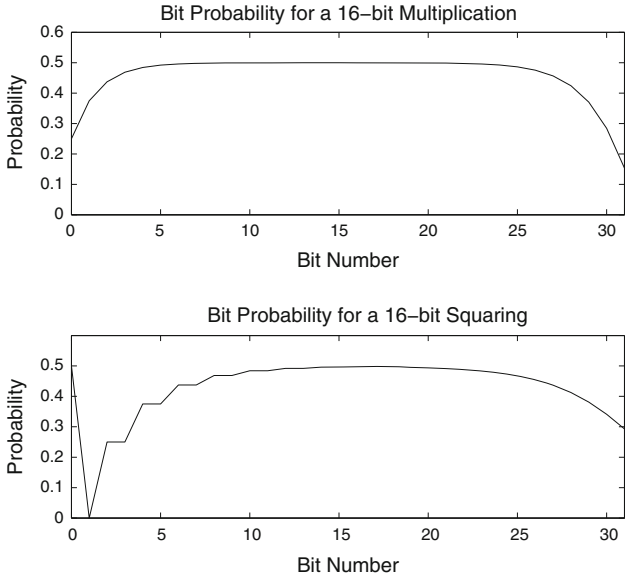

Fig. 1 The probability that each bit of the result of a multiplication and a squaring operation is equal to one with random 16-bit multiplicands

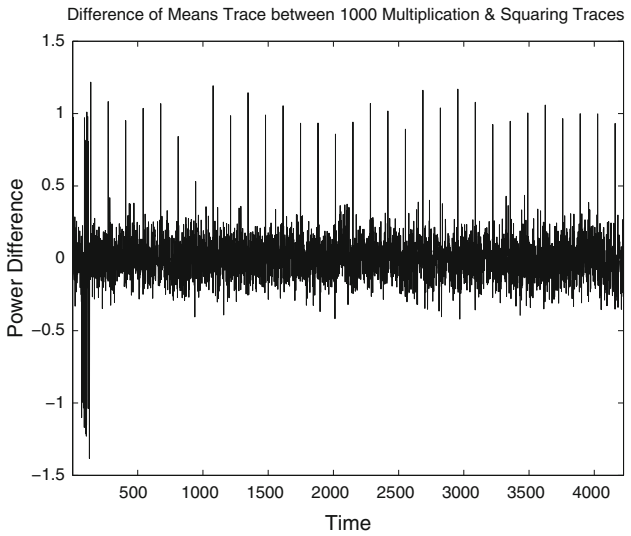

Fig. 2 Difference between mean traces for multiplication and squaring operations

In order to demonstrate this difference, simulated traces of a 1,024-bit Montgomery multiplication were generated. Algorithm 2 was implemented with a radix value, b, of 32-bits, leading the number of words, n, required also being 32. Therefore, $2n(n+1) = 2{,}112$ single-precision multiplications were present for each trace. The Hamming weight of each intermediate multiplication value was then recorded to generate the simulated power values. Figure 2 shows the

difference in the mean power consumption for 1,000 multiplication and squaring operations. There are 33 distinct peaks clearly visible in this trace. Comparing with Algorithm 2, these peaks correspond to the single-precision multiplications $x_i\, y_j$, where $i = j$. There are 33 distinct peaks since there are two such multiplications when $i = 0$. It can also be seen where mean power traces for the same operation are subtracted and no peaks are present. There are also a number of smaller peaks present at the beginning when computing $u_0 m$ since $u_0 = \left(x_0 y_0 m'\right) \bmod b$.

This property leads to the attack outlined in [2], where if many traces for a constant exponent are available, bits can be recovered by subtracting the mean of adjacent operations. However, attacks based on this observation require a large number of acquisitions in order to be able to observe this difference to extract key bits and are not practicable.

3 Application of templates

In this section, we demonstrate that the expected Hamming weight can be used to create templates to distinguish between a multiplication and a squaring operation with a high probability using a single trace. We further discuss how this could be applied to a side channel atomic exponentiation algorithm to determine an RSA private key. This is demonstrated using simulated traces, and then its practical relevance is shown using traces acquired by measuring the instantaneous power consumption of an ARM7 microprocessor.

As stated previously, we assume that the power consumption is proportional to the proportional to the Hamming weight of the values being manipulated at a given point in time.

3.1 Attack using simulated traces

To illustrate our attack, we generated simulated traces for 20 thousand 1,024-bit Montgomery multiplication operations in the same manner as used for Fig. 2. Two sets of 10 thousand traces were acquired, representing multiplications and squaring operations using uniformly distributed random inputs. The first 5 thousand traces of each set were designated the template generation traces, and the last 5 thousand were designated attack traces. Using the peaks present in the difference of means trace, as illustrated in Fig. 2, templates were built for the two operations. The peaks are chosen as the points of interest to build the templates from as they indicate the points where the expected Hamming weight of a multiplication and squaring operation differ and affect the power consumption.

To determine whether an operation is a multiplication or squaring, each trace was classified using both multiplication and squaring templates. Since we are considering a binary classification system, a threshold was determined where any probability greater than the threshold indicates the template for the correct operation has been used. Conversely, this means that anything less than the threshold is the other operation. Classification for both templates must still be computed to allow the application of Bayes theorem as outlined in Sect. 2.3.2. To calculate the threshold, each trace used to generate the templates was classified individually and the midpoint between the mean of the resultant probabilities selected.

Figure 3 shows the classification success rate of 1,000 attack traces as a function of the number of traces used to generate the templates, i.e., the percentage of individual traces that were successfully classified. It can be seen that as the number of traces used to generate the templates increases, the attack traces are more accurately classified, giving an expected success rate of 82% for a given trace when 5,000 traces are used to generate the templates. This is due to a more accurate estimation for the mean and covariance matrix of the templates.

If an exponent is constant, as would be the case with a naïve implementation of RSA, multiple traces with different inputs can be used in an amplified template attack, as described in [29]. By multiplying the probabilities for consecutive traces, the correct operation can be identified with a high probability after very few traces, as illustrated in the example in Fig. 4. In this example, using templates built from 5,000 traces, 1,000 sets of five traces are classified, and the success rate for each set is plotted taking consecutive traces into account (as opposed to the success rate for each individual trace in the last example). The success rate increases as more traces are taken into account, correctly classifying the operation 94% of the time if five traces from the same exponent are available.

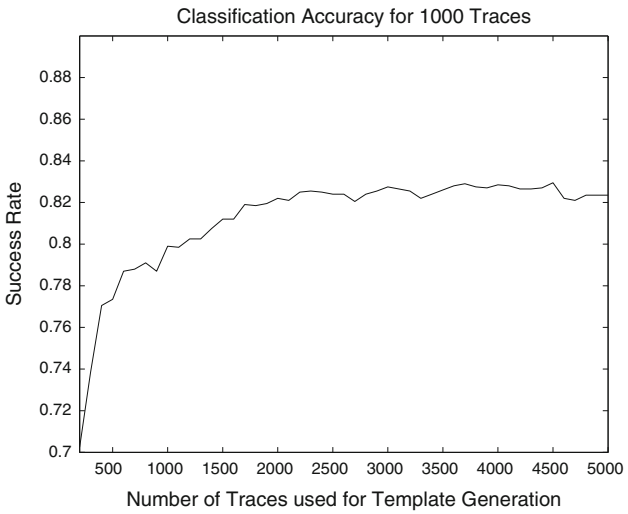

Fig. 3 Success rate of template attack—1,024-bit simulated traces

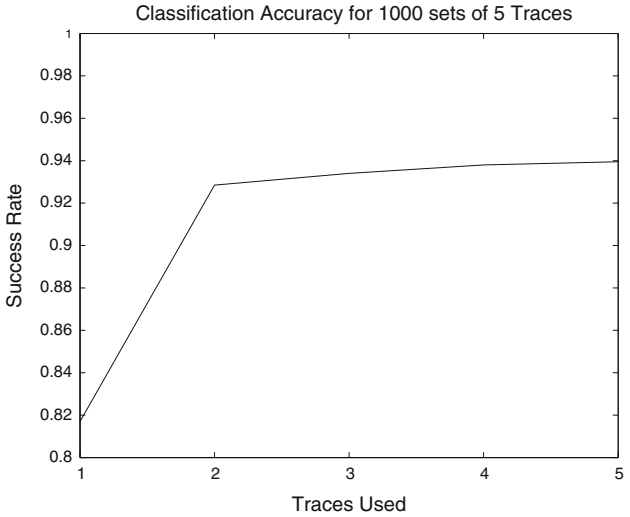

Fig. 4 Success rate of amplified template attack—1,024-bit simulated traces

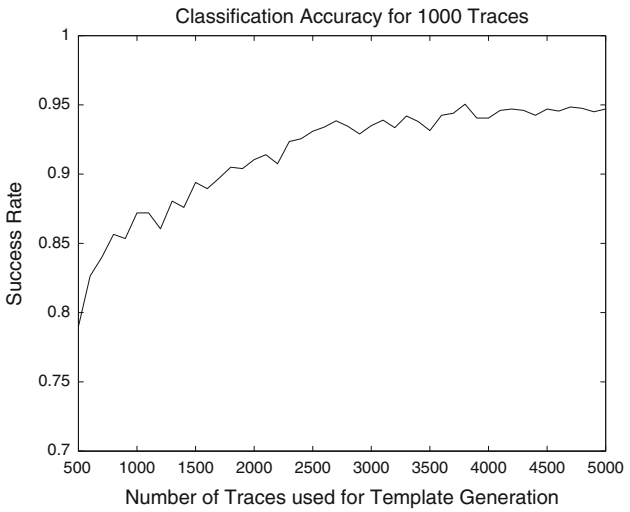

Fig. 5 Success rate of template attack—4,096-bit simulated traces

We can note that there will be more information to exploit as the bit length of the targeted operations increases, since there will be more single-precision multiplications required to compute each operation. This observation was also utilized in the attack presented in [35]. In the case of a 4,096-bit multiplication operation, there are four times as many information bearing single-precision multiplications to compute than in a 1,024-bit operation. This allows us to more accurately classify an operation using a single trace. To verify this, we generated simulated traces as before, except for with 4,096-bit inputs. Figure 5 shows us that the success rate has increased from 82 to 95%. Therefore, for example, in the case of a blinded implementation of the RSA signature scheme with a key length equal to (or greater than) 4,096-bits, one can deduce the operations from a single trace using templates with a high probability of success.

3.2 Attack using acquired traces

To verify the practical implications of our attack, we also implemented a 1,024-bit Montgomery Mulitplication on an ARM7 microprocessor [3], which has a 32-bit architecture leading to the same number of single-precision multiplications that leak useful information as the simulated case. For simplicity, the Montgomery multiplication algorithm was implemented with a redundant subtraction since this operation will have no effect on the analysis described in this section. The traces were acquired with a sampling rate of 100 Ms/s, while the ARM7 microprocessor was clocked with a 7.37 MHz clock. To reduce the effect of noise, the traces were filtered using a 10 MHz lowpass filter. To reduce the computational complexity of the resultant attack, the maximum point within each clock cycle was extracted to shorten the trace length, as described in [23]. This also has the effect of synchronizing the traces where the offset is less than the width of a clock cycle. Using the same inputs as before, 10,000 traces for each operation were acquired and split evenly into generation and attack traces.

Figure 6 shows the difference of means trace between the operations, the crosses indicating the points retained to generate the templates. Here, two points per multiplication are retained per single-precision multiplication. The lower plot shows the detail of a single peak from the upper plot. We can see that there are two peaks where the power difference between a multiplication and squaring operation can be observed. Therefore more points are available to build our templates with than in the simulated case.

The effect of the extra peaks available to generate the templates can be seen in Fig. 7, where the success rate is

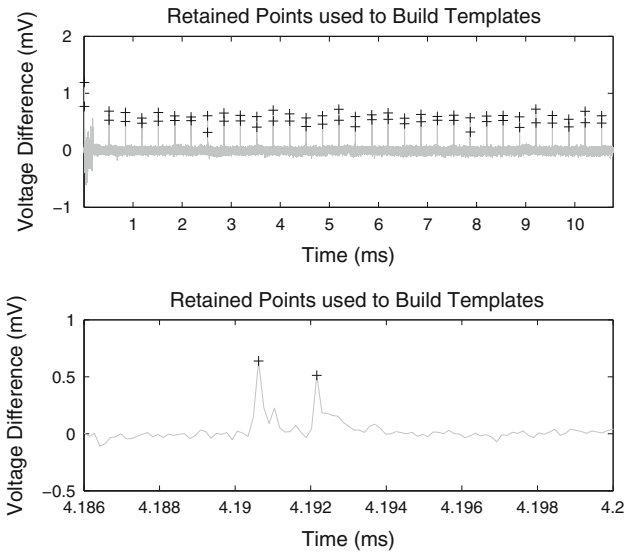

Fig. 6 Difference of means trace between multiplication and squaring operations

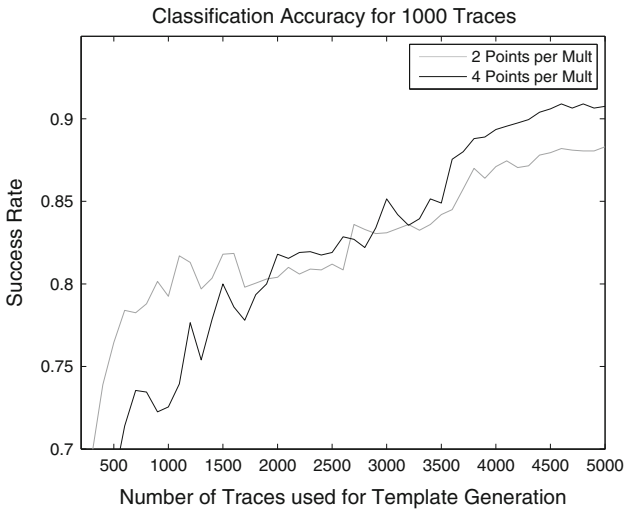

Fig. 7 Classification success rate of 1,000 traces

higher with actual power traces than in the simulated case. The attacker is also not limited to retaining a single point for each peak. In this scenario, the highest success rate of over 90% comes when four points per single-precision multiplication are used to build templates. Retaining further points leads to the templates modeling the generation set of traces more accurately, but not the general output of the multiplication operation as a whole, which is what is required to successfully classify the attack traces. The success rate of an amplified template attack also shows a corresponding increase with the ARM7 traces, giving a success rate of 97% when templates are generated with 5,000 traces for each operation, and five traces are available with a constant exponent.

4 Power consumption model

In Sect. 3, we assume that the power consumption is proportional to the Hamming weight of the data being manipulated at a given point in time, referred to as the Hamming weight model. In this section, we show that this is the case for an ARM7 microprocessor and demonstrate that some instructions leak information consistent with the Hamming weight model and other instructions leak information consistent with the Hamming distance model. That is, the power consumption is proportional to the number of bits that change state from clock cycle to another. We then describe how the attack described above would be affected if all of the information available was consistent with the Hamming distance model.

4.1 Side channel leakage on an ARM7 microprocessor

The multiplier on an ARM7 microprocessor is typically implemented so that it computes the multiplication of two 32-bit words in 2–5 clock cycles. The number of clock cycles required depends on the bit length of one of the multiplicands, that is the multiplication will terminate early if a number of the most significant bytes of one multiplicand are set to zero. The precise algorithm is defined in Algorithm 3.

Algorithm 3: A functional description of ARM7TDMI multiplication

Input: The 32-bit integers x and y.
Output: The 64-bit result $t = x \cdot y$.

$t_0 \leftarrow 0$
for $i = 0$ **up to** 3 **step** 1 **do**
  $t_1 \leftarrow x \cdot y_{7\ldots0}$
  $t_0 \leftarrow t_0 + (t_1 \ll 8i)$
  $y \leftarrow y \gg 8$
  if $y = 0$ **then return** t_0
end

Several attacks are described in [16] that demonstrate that naïvely using this multiplier to implement cryptographic algorithms will provide an attacker with an exploitable vulnerability. That is, the number of clock cycles taken by each multiplication can be seen in the power consumption and used to deduce information on the values being operated on.

This means that the multiplier on an ARM7 microprocessor has to be used such that it will perform a multiplication in a constant amount of time irrespective of the bit length of the multiplicands. An example of such an algorithm is shown in Algorithm 4 [16]. For the implementation of our attack, described in Sect. 3.2, this algorithm was used to provide an implementation with a minimum level of security. That is, an implementation that is not vulnerable to attack by simply inspecting a small number of power traces.

Algorithm 4: A constant-time algorithm to replace ARM7 multiplication

Input: The 32-bit integers x and y.
Output: The 64-bit result $r = x \cdot y$.

$\gamma \leftarrow (y \wedge \mathtt{00FFFFFF}_{(16)}) + \mathtt{01000000}_{(16)}$
$\tau \leftarrow (y \wedge \mathtt{FF000000}_{(16)}) \gg 24$
$r \leftarrow x \cdot \gamma$
$r \leftarrow r + ((x \cdot \tau) \ll 24)$
$r \leftarrow r - (x \ll 24)$
return r

The side channel leakage that enables the attack described in Sect. 3 can be characterized by observing the correlation between a series of traces of the Hamming weight of the result of a multiplication and the power consumption. This was proposed as an attack but also will validate a hypothesized model [7,22]. This is illustrated in Fig. 8 that shows a

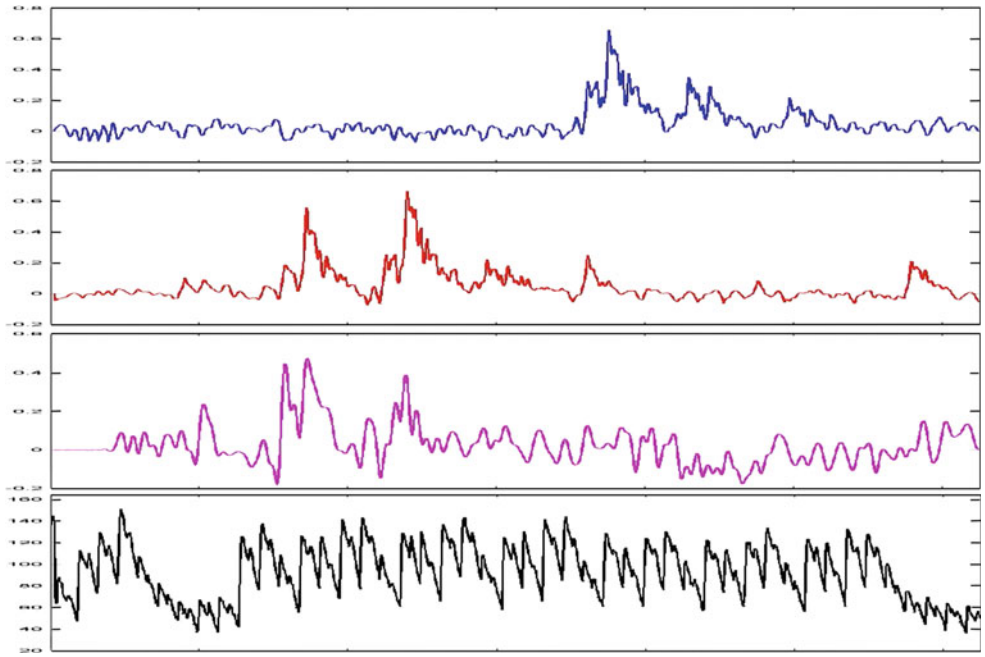

Fig. 8 The *lower trace* shows the power consumption over time in millivolts. A multiplication is visible as a significant dip in the power consumption towards the beginning of the trace. The *upper middle* (resp. top) *trace* shows the correlation with the Hamming weight of the least (resp. most) significant word of the output of a multiplication at each point in time. The *lower middle trace* shows the difference in mean power consumption caused by computing a multiplication or a squaring operation

series of traces to demonstrate that the expected difference in the power consumption, observed in Sect. 3.2, is caused by the power consumption being consistent with the Hamming weight model.

In Fig. 8, the bottom trace shows the power consumption during and after the computation of a multiplication. In this case, the multiplication is the computation of $r \leftarrow x \cdot \gamma$ in line 3 of Algorithm 4 and is visible as a significant dip in the power consumption toward the beginning of the trace. The upper middle (respectively, top) trace shows the correlation with the Hamming weight of the least (respectively, most) significant word of the output of a multiplication at each point in time. The lower middle trace shows the difference of two mean traces, as shown in Fig. 6, one of which is the power consumption trace when the multiplication computed is a squaring operation and the other is a multiplication.

Two peaks are visible in the difference of means that correspond to the correlation with the least significant word of the result of the multiplication. We can note that no difference is visible during the computation of the multiplication or the subsequent manipulation of the most significant word of the result of the multiplication. The observed difference is therefore present in the manipulation of the least significant word after the multiplication has taken place, rather than during the computation of the multiplication.

A similar attack, also based on the difference between mean power consumption traces for multiplication and squaring operations but caused by a different effect, is described by Akishita and Takagi [1], who analyzed the switching in a hardware multiplier unit implemented in CMOS circuitry. These units are typically used as a hardware coprocessor for performing group exponentiation algorithms in resource constrained devices such as smart cards. As noted in Sect. 2.4, data dependent power consumption in CMOS circuits occurs when logic gates switch state. Looking at a hardware multiplier, the overall transition probabilities for the gates are different depending on whether the input values are equal or not. As in the attack described in [2], this is caused by the difference in the probability distribution of the output of the multiplication. In this paper, we target the difference in power consumption caused by the distribution of the output directly. The attack in [1] targets the difference in power consumption while calculating this output. Hardware co-processors in CMOS logic often follow the Hamming distance model so the power consumption at the output register might not be directly related to the expected distribution. We discuss this further below.

4.2 Defining an attack using the hamming distance model

As described above, the ARM7 microprocessor used in the implementation of our attack has side channel leakage that is consistent with the Hamming weight model. However, hardware implementations and some microprocessor functions

are consistent with the Hamming distance model. That is, the number of transitions that occur at a given point in time from some previous state. When considering the Hamming distance model, there are two different scenarios that we can consider, these are:

4.2.1 Constant previous state

If the previous state is a constant value, the probability of a bit changing state because of the result of a multiplication will be modified. The probabilities shown in Fig. 1 can be considered to represent the probabilities for the Hamming distance model where the previous state is equal to zero. If some bits of a previous state are set to one, then the probabilities corresponding to that bit are inverted. A difference in Hamming weight will remain that should be exploitable by the attack described in this paper. However, the size of the difference will depend on the value of the previous state and will have an effect on the success rate of the attack.

4.2.2 Variable previous state

If the previous state is unpredictable, the attack becomes more problematic. An attacker will have to determine the distribution of the Hamming weight of the previous state. Then an attacker would need to determine the difference between this distribution and the distribution of the Hamming weight of the result of a multiplication and squaring operation.

If the previous state has a definite distribution, an attack may be possible if the expected difference is large enough. However, an evaluation of this topic is beyond the scope of this paper and is left as open problem.

5 Application to secure implementations

When implementing a cryptographic algorithm on a device that is potentially vulnerable to side channel analysis, one would typically include specific countermeasures, in conjunction with side channel atomicity, to prevent an attacker being able to derive information on cryptographic keys. When implementing RSA, this would typically mean implementing the blinding countermeasures described by Coron [12].

These countermeasures modify the exponentiation algorithm such that the values being operated on at a given point in time cannot be predicted by an attacker. Given that both the side channel leakage models discussed above are based on the number of bits being manipulated at a given point in time, these countermeasures randomize the bitwise representation of all the variables being manipulated. For instance, to implement a function to compute a RSA signature $s = \mu(m)^d \bmod N$, as defined in Sect. 2.1, each variable would be modified with a random value as shown in Algorithm 5. This is referred to as blinding since an attacker is unable to determine the values being manipulated at a given point in time.

Algorithm 5: Blinded exponentiation

Input: m, d, N, μ a padding function, ϕ Euler's Totient function and non-zero random values r_i, for $i \in \{1, 2, 3\}$.
Output: $s = \mu(m)^d \bmod N$.

1 $m' \leftarrow \mu(m) + r_1 N$
2 $N' \leftarrow r_2 N$
3 $d' \leftarrow r_3 \phi(N) + d$
4 $s' \leftarrow \mu(m)^{d'} \bmod N'$
5 **return** $s' \bmod N$

The addition of $r_1 N$ to $\mu(m)$ provides a redundant representation of $\mu(m)$ modulo N and is referred to a message blinding. Given that $\mu(m) + r_1 N \equiv \mu(m) \pmod N$ and this will remain true for all the intermediate states during the execution of the algorithm, the computation should not take place in $(\mathbb{Z}/N\mathbb{Z})^*$ since the blinding would be removed. The bit length of the modulus is therefore increased by multiplying it by a random value so that the computation takes place in $(\mathbb{Z}/r_2 N\mathbb{Z})^*$. The values held in memory at a given point during the computation of the exponentiation cannot be predicted without knowing these random values. However, the values held in memory at a given point in time would represent the same value in $(\mathbb{Z}/N\mathbb{Z})^*$ for a fixed exponent.

The bitwise representation of the exponent is also modified by replacing the exponent d with $r_3 \phi(N) + d$, where ϕ is Euler's Totient function and is referred to as exponent blinding. Any value raised to a multiple of $\phi(N)$ will be equal to itself in $(\mathbb{Z}/N\mathbb{Z})^*$, i.e., it is an identity function. Adding a multiple of $\phi(N)$ to the exponent changes the bitwise representation of the exponent without changing the result of its use in $(\mathbb{Z}/N\mathbb{Z})^*$.

One would typically choose the values of r_i, for $i \in \{1, 2, 3\}$, to be at least 32 bits. As noted in [33], one would also want the bit length of r_2 to be longer than the longest run of zeros in the bitwise representation of $\phi(N)$ so that all the bits of d are blinded. Furthermore, some side channel attacks could potentially derive an exponent if the bit length of these random values is too small [11]. The discussion of this topic is beyond the scope of this paper, and we will assume that each random value has the same bit length as one word of the processor computing the algorithm.

5.1 Effect on the attack

When implementing the attack described in Sect. 3, the use of exponent blinding changes the required strategy, while the use of message blinding aids an attacker.

The use of message blinding randomizes the bitwise representation of the values being operated on by the exponent. This aids an attacker since the values being operated on will be random and uniformly distributed and any values set to a constant will be randomized, e.g., the X509 and PKCS padding schemes set large parts of their output bytes to fixed values. The increase in the bit length of the modulus will also mean that it will require an extra word in memory. This also means that the Montgomery multiplication (see Algorithm 2) will require one more iteration of the main loop, providing one more single-precision multiplication that could be analyzed by an attacker (i.e., one more single-precision multiplication that will either be a multiplication or a squaring operation). This will provide another point in the traces that can be included in the template attack. As illustrated in Sect. 3, the more single-precision multiplications available to build templates with, the higher the success rate of the attack.

The use of exponent blinding randomizes the bit wise representation of the exponent and therefore randomizes the sequence of multiplication and squaring operations that are computed. An attacker would be unable to take several acquisitions to improve the quality of classification since the operation being computed at a given point in time during the execution of the algorithm will vary from instantiation to another. This means that an attacker has to determine whether an operation is a multiplication or a squaring operation from one acquisition. Moreover, an attacker also needs to acquire a single trace that includes the power consumption during an entire exponentiation to attempt to derive the exponent used, and therefore a value equivalent to the exponent d in $(\mathbb{Z}/N\mathbb{Z})^*$. Deriving partial information from two traces will provide partial information on two values that are equivalent to the exponent d in $(\mathbb{Z}/N\mathbb{Z})^*$, which will not be sufficient to derive d.

The results from Sect. 3 are directly applicable to a secure implementation, but only if we add the constraint that an attacker can use one target trace. In Sect. 3.2 an attacker could expect to correctly identify 90% of the bits of an exponent used in an ARM implementation of Algorithm 5.

The incorrect classification of operations can be corrected to a certain extent, since the Hamming weight of the entire exponent can be computed by observing the total number of operations, and that a multiplication will always be preceded, and followed by, a squaring operation (see Algorithm 1). One approach to conducting this analysis was proposed by Green et al. [15] using hidden Markov models.

6 Conclusion

In this paper, we demonstrate that the principle of side channel atomicity is not valid simply because the same code is executed for a given function for all possible inputs. This was done by characterizing the difference in expected Hamming weight of the result of a multiplication and squaring operation given random uniformly distributed inputs to generate templates, based on previous descriptions of this difference [2]. These templates are generated by considering single-precision multiplications rather than multiplications between multi-precision values.

These templates can be used to characterize multiplications and squaring operations to determine a private exponent when an RSA signature is generated using Algorithm 1. Previous work in this area has concentrated on building templates on intermediate values (e.g., such as observing where the input value is reused in a left-to-right exponentiation algorithm) rather than the expected distribution of the result of a given function [18,24].

The advantage of this work over previous work based on the model originally proposed by Chari et al. [9] is that an attacker does not need an open identical device to conduct the attack. That is, an attacker does not need a device where all the input and cryptographic keys can be changed to arbitrary values. An attacker can use an identical device with a known key, or a verification functions that uses the same operations. Furthermore, the values being operated on do not need to be known, an attacker just needs to know that the values are random.

The proposed attack can be used to derive the private key used in RSA, where one needs to obtain the least significant quarter of an exponent to be able to derive the entire exponent [6]. This naturally extends to the computation of a RSA signature using the Chinese remainder theorem (CRT). Again, one needs to obtain a quarter of the least significant bits of the private key modulo one of the two prime factors of the modulus N (i.e., one of the two exponents used in this algorithm) [5]. The proposed method can also be used to derive a blinded exponent with a certain probability; however, there are currently no results detailing how many bits one would need to identify (i.e., how efficient an attack would be).

The work presented in this paper also applies to elliptic curve scalar multiplication over large prime fields. If strongly unified formulae (e.g., these have been defined for Edward's curves [4] and Weierstraß curves [8]) are used then an algorithm equivalent to Algorithm 1 can be defined. An attacker could potentially distinguish an addition from a doubling operation using the attack described above. While the bit length of the prime field using in elliptic curve cryptography is significantly smaller than the bit length of an RSA modulus, there are numerous operations in the prime field for each operation, i.e., the addition and doubling operation, that could be exploited.

As noted in [24], this type of attack naturally extends to attacking implementations of the digital signature algorithm (DSA), and its elliptic curve equivalent (ECDSA) [38].

In each case, a known value is raised to the power of a random value as part of the signature generation. If some bits of the random value can be determined in one, or several, instantiations of the signature scheme the private key can be derived [27,28]. The work described in this paper should allow an attacker to derive sufficient bits of this random value to break a naïve implementation of these algorithms with a reasonable probability.

The template attack described in this paper will, counter-intuitively, become more effective with longer key lengths due to the extra single-precision multiplications required, as previously noted in [35] for similar reasons. This gives a strong argument for using regular exponentiation algorithms (see [19] for a summary of this topic) rather than side channel atomic algorithms.

Acknowledgments The work described in this paper has been supported in part by the Informatics Commercialization Initiative of Enterprise Ireland, Science Foundation Ireland under Grant No. [SFI/08/RFP/ENE1643], the European Commission IST Program under Contract ICT-2007-216676 ECRYPT II and EPSRC grant EP/F0 39638/1 "Investigation of Power Analysis Attacks".

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, vol. 3928 of LNCS, pp. 151–164. Springer, Berlin (2006)
2. Amiel, F., Feix, B., Tunstall, M., Whelan, C., Marnane, W.P.: Distinguishing multiplications from squaring operations. In: Youm, H.Y., Yung, M. (eds.) SAC 2008, vol. 5932 of LNCS, pp. 148–162. Springer, Berlin (2009)
3. ARM Limited: ARM7TDMI technical reference manual (revision r4p1), http://infocenter.arm.com/ (2004)
4. Bernstein, D.J., Lange, T.: Faster addition and doubling on elliptic curves. In: Kurosawa, K. (ed.) ASIACRYPT 2007, vol. 4833 of LNCS, pp. 29–50. Springer, Berlin (2007)
5. Blömer, J., May, A.: New partial key exposure attacks on RSA. In: Boneh, D. (ed.) CRYPTO 2003, vol. 2729 of LNCS, pp. 27–43. Springer, Berlin (2003)
6. 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 98, vol. 1514 of LNCS, pp. 25–34. Springer, Berlin (1998)
7. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004, vol. 3156 of LNCS, pp. 16–29. Springer, Berlin (2004)
8. Brier, E., Joye, M.: Weierstraß elliptic curve and side-channel attacks. In: Naccache, D., Paillier, P. (eds.) PKC 2002, vol. 2274 of LNCS, pp. 335–345. Springer, Berlin (2002)
9. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002, vol. 2523 of LNCS, pp. 13–28. Springer, Berlin (2003)
10. Chevallier-Mames, B., Ciet, M., Joye, M.: Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. IEEE Trans. Comput. **53**(6), 760–768 (2004)
11. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., Verneuil, V.: Horizontal correlation analysis on exponentiation. Cryptology ePrint Archive, Report 2010/394, http://eprint.iacr.org/ (2010)
12. Coron, J.-S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Koç, C.K., Paar, C. (eds.) CHES 99, vol. 1717 of LNCS, pp. 292–302. Springer, Berlin (1999)
13. Fahn, P.N., Pearson, P.K.: IPA: a new class of power attacks. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999, vol. 1717 of LNCS, pp. 173–186. Springer, Berlin (1999)
14. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: concrete results. In: Koç, C.K., Naccache, D., Paar, C. (eds.) CHES 2001, vol. 2162 of LNCS, pp. 251–261. Springer, Berlin (2001)
15. Green, P.J., Noad, R., Smart, N.P.: Further hidden Markov model cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005, vol. 3659 of LNCS, pp. 61–74. Springer, Berlin (2005)
16. Großschädl, J., Oswald, E., Page, D., Tunstall, M.: Side channel analysis of cryptographic software via early-terminating multiplications. In: Lee, D., Hong, S. (eds.) ICISC 2009, vol. 5984 of LNCS, pp. 176–192. Springer, Berlin (2010)
17. Hachez, G., Quisquater, J.-J.: Montgomery exponentiation with no final subtractions: Improved results. In: Koç, C.K., Paar, C. (eds.) CHES 2000, vol. 1965 of LNCS, pp. 293–301. Springer, Berlin (2000)
18. Herbst, C., Medwed, M.: Using templates to attack masked Montgomery ladder implementations of modular exponentiation. In: Chung, K.-I., Sohn, K., Yung, M. (eds.) WISA 2008, vol. 5379 of LNCS, pp. 1–13. Springer, Berlin (2009)
19. Joye, M., Tunstall, M.: Exponent recoding and regular exponentiation algorithms. In: Preneel, B. (ed.) AFRICACRYPT 2009, vol. 5580 of LNCS, pp. 334–349. Springer, Berlin (2009)
20. Joye, M., Tymen, C.: Protections against differential analysis for elliptic curve cryptography: An algebraic approach. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001, vol. 2162 of LNCS, pp. 377–390. Springer, Berlin (2001)
21. Kocher, P.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO '96, vol. 1109 of LNCS, pp. 104–113. Springer, Berlin (1996)
22. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO '99, vol. 1666 of LNCS, pp. 388–397. Springer, Berlin (1999)
23. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer, Berlin (2007)
24. Medwed, M., Oswald, E.: Template attacks on ECDSA. In: Chung, K.-I., Sohn, K., Yung, M. (eds.) WISA 2008, vol. 5379 of LNCS, pp. 14–27. Springer, Berlin (2009)
25. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, USA (1997)
26. Montgomery, P.: Modular multiplication without trial division. Math. Comput. **44**, 519–521 (1985)
27. Nguyen, P.Q., Shparlinski, I.E.: The insecurity of the digital signature algorithm with partially known nonces. J. Cryptol. **15**(3), 151–176 (2002)
28. Nguyen, P.Q., Shparlinski, I.E.: The insecurity of the elliptic curve digital signature algorithm with partially known nonces. Des. Codes Cryptogr. **30**, 201–217 (2003)
29. Oswald, E., Mangard, S.: Template attacks on masking—resistance is futile. In: Abe, M. (ed.) CT-RSA 2007, vol. 4377 of LNCS, pp. 243–256. Springer, Berlin (2007)
30. Quisquater, J.-J., Samyde, D.: Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In: Attali, I., Jensen, T.P. (eds.) Smart card programming and security, International conference on research in smart cards—E-smart 2001, vol. 2140 of LNCS, pp. 200–210. Springer, Berlin (2001)
31. Rechberger, C., Oswald, E.: Practical template attacks. In: Lim, C.H., Yung, M. (eds.) WISA 2004, vol. 3325 of LNCS, pp. 440–456. Springer, Berlin (2004)
32. Rivest, R., Shamir, A., Adleman, L.M.: Method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)

33. Smart, N., Oswald, E., Page, D.: Randomised representations. IET Proc. Inf. Sec. **2**(2), 19–27 (2008)
34. Walter, C.D.: Montgomery exponentiation needs no final subtractions. Electron. Lett. **35**(21), 1831–1832 (1999)
35. Walter, C.D.: Longer keys may facilitate side channel attacks. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2004, vol. 3006 of LNCS, pp. 42–57. Springer, Berlin (2004)
36. Walter, C.D.: Simple power analysis of unified code for ECC double and add. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004, vol. 3156 of LNCS, pp. 191–204. Springer, Berlin (2004)
37. Walter, C.D., Thompson, S.: Distinguishing exponent digits by observing modular subtractions. In: Naccache, D. (ed.) CT-RSA 2001, vol. 2020 of LNCS, pp. 192–207. Springer, Berlin (2001)
38. ANSI X9.62.: Public key cryptography for the financial services industry, the elliptic curve digital signature algorithm (ECDSA) (1999)
39. Yen, S.-M., Joye, M.: Checking before output may not be enough against fault based cryptanalysis. IEEE Trans. Comput. **49**(9), 967–970 (2000)