Address-bit Differential Power Analysis on Boolean Split Exponent Counter-measure

Christophe Negre[1,2]
[1]DALI, Université de Perpignan, France
[2]LIRMM, Université de Montpellier et CNRS, France

Keywords: Side Channel Analysis, Address Bit Differential Power Analysis, Exponentiation, Randomization.

Abstract: Current public key cryptographic algorithms (RSA, DSA, ECDSA) can be threaten by side channel analyses. The main approach to counter-act such attacks consists in randomizing sensitive data and address bits used in loads and stores of an exponentiation algorithm. In this paper we study a recent counter-measure ”Boolean split exponent” (Tunstall et al. 2018) preventing differential power analysis on address bits. We show that one of their proposed protections has a flaw. We derive an attack exploiting this flaw and we successfully apply it on a simulated power consumption of an RSA modular exponentiation.

1 INTRODUCTION

Side channel analysis is a serious threat for devices performing cryptographic computation. Specifically, it can threaten exponentiation x^{κ} involved in currently used cryptosystems (RSA, DSA and ECDSA). In 1996, in a seminal work (Kocher, 1996) Kocher showed that a statistical analysis of the computation time can leak out the secret key used in RSA cryptosystem. Afterwards, it was shown that the power consumption can be used to extract secret information: with simple power analysis (SPA) (Kocher et al., 1999) one can read on the power trace the sequence of operations (squaring/multiplication) performed during an exponentiation and then deduce the secret exponent. In (Kocher et al., 1999) the authors introduce the differential power analysis (DPA) which computes differences on the power traces to determine the bits of the key.

To counter-act these side channel analyses, the basic approach is to randomly mask sensitive data: we can blind the point x by multiplying it by a random element (Coron, 1999) or use randomized representation (Clavier et al., 2010), and we can use additive or multiplicative mask of the exponent (Coron, 1999; Tunstall and Joye, 2010). But this is not always sufficient, since it was shown (Itoh et al., 2003) that the loads and stores performed during the exponentiation induce a power consumption correlated to the corresponding address bits. This can be exploited to mount a differential power analysis on the address bits (ADPA). Recently some strategies (Izumi et al., 2010; Tunstall et al., 2018) were proposed to counter-act this attack by randomizing the address bits used in the loads and stores of the exponentiation algorithm.

Contribution. In this paper we analyse a potential flaw in the signed version of the boolean exponent splitting approach of (Tunstall et al., 2018). We show that the sequence of operations (squarings and multiplications) are not fully regular: we show that if we can distinguish a squaring from a multiplication we can deduce the bit used to randomize the address bits. Then a classical differential power analysis on the address bits can be performed to determine the secret exponent. We validate this attack by simulating power traces based on the Hamming weight model. We provide a method for distinguishing a squaring from a multiplication which has a high level of confidence. Then we show that an ADPA attack can be performed and successfully deduce the secret exponent with a few thousand power traces.

Organization of the Paper. In Section 2 we review power analyses on exponentiation algorithm and related counter-measures. In Section 3 we show that there is a flaw in the counter-measures “boolean splitting exponent” of (Tunstall et al., 2018). In Section 4 we present our approach for power consumption simulation of modular exponentiation, and we apply an attack on the “boolean splitting exponent” exploiting the flaw shown in Section 3. We ends the paper in Section 5, with a few concluding remarks.

Negre, C.
Address-bit Differential Power Analysis on Boolean Split Exponent Counter-measure.
DOI: 10.5220/0009891306320637
In *Proceedings of the 17th International Joint Conference on e-Business and Telecommunications (ICETE 2020) - SECRYPT*, pages 632-637
ISBN: 978-989-758-446-6

2 REVIEW OF POWER ANALYSES

In currently used public key cryptosystems (e.g. RSA, DSA, ECDSA), the sensitive operation is an exponentiation x^κ for $x \in \mathbb{G}$ a finite group or ring and κ a secret exponent. With the square-and-multiply algorithm one can compute x^κ as a sequence of squarings $R \leftarrow R^2$ followed by a multiplication $g \leftarrow g \times x$ when the i-th bit $\kappa_i = 1$. This basic approach is not secure when considering side channel analysis. For example, if we monitor the power consumption of an exponentiation, then, a simple power analysis (Kocher et al., 1999) can identify on the trace the operations performed (square or multiplication) on the device and then deduce the secret exponent κ. To counteract this attack modified versions of the square-and-multiply exponentiation are recommended for which the sequence of computed operations is not related to bits of the exponent κ. The most popular method is the Montgomery ladder (Joye and Yen, 2002) which involves two variables R_0 and R_1, for which, during the exponentiation, we always have $R_1 = xR_0$ and the loop iteration is always a multiplication followed by a squaring (cf. Algorithm 1).

Algorithm 1: Montgomery-ladder.

```
Require: $x \in \mathbb{G}$, one $n$-bit integers $\kappa = \sum_{i=0}^{n-1} \kappa_i 2^i$
Ensure: $x^\kappa$
1: $R_0 \leftarrow 1_\mathbb{G}$; $R_1 \leftarrow x$;
2: for $i = n-1$ down to 0 do
3:     $R_{1-\kappa_i} \leftarrow R_{\kappa_i} \cdot R_{\neg\kappa_i}$;
4:     $R_{\kappa_i} \leftarrow R_{\kappa_i}^2$;
5: return $R_0$
```

The Montgomery ladder is a good protection against SPA, but it is not robust against attack like differential power analysis (DPA) (Kocher et al., 1999), collision attack (Fouque and Valette, 2003; Yen et al., 2006). Indeed these attacks guess one or several bits of the key and they predict the power consumption at some iteration of the exponentation. If this prediction is correct this means that the guessed bits are also correct otherwise one can try another guess. To counteract such attacks at the algorithmic level, the best approaches randomize data in order to render power consumption unpredictable:

- *Point blinding.* The strategy here is to hide the data x and R_0, R_1 by either multiplying x by a random element (Coron, 1999), adding a random mask or by randomizing the representation of x (Clavier et al., 2010).
- *Exponent masking.* In this case we randomly modify the exponent. (Coron, 1999) proposed to add to κ a random mutiple $r \times N$ where N is the order of $\mathbb{G}$. One can also (Tunstall and Joye, 2010) randomize the exponent as $\kappa' = \beta^{-1}\kappa \mod N$ and compute x^κ as $(x^\beta)^{\kappa'}$.

Point blinding and *Exponent masking* counter-measures induce an overhead which, for the later, is important since the level of randomization have to be larger than the longest run of 0 or 1 in κ (Smart et al., 2008).

Point blinding approaches used alone are not sufficient to counter-act DPA attack on the address bits (ADPA (Itoh et al., 2003)). Indeed guessing a few key bits leads to a prediction of the address bits involved in the loads and stores performed during the exponentiation. Predicting these address bits, leads to predicting the behavior of the power consumption during the loads or stores and performing a DPA on several power traces validates the guessed key bit or not. Since point blinding counter-measures do not modify the sequence of operations done during the exponentiation, they do not protect the implementation from an ADPA.

Consequently, to counter-act such DPA on address bits, the authors in (Izumi et al., 2010) proposed a first version of the Montgomery-ladder which randomizes the loads and stores. This work was subsequently improved in (Tunstall et al., 2018) which provides two kinds of randomized Montgomery-ladder. In their first approach they use one random bit a_i to split the key bit $\kappa_i = a_i \oplus b_i$ and which decides the order of loads R_0 and R_1 for the multiplication $R_0 \times R_1$ in Step 3 of Algorithm 1. This bit a_i is also used to randomly store the results of the squaring in Step 4 and the multiplication in Step 3 of Algorithm 1 done in one iteration of the Montgomery ladder. This requires a bit b' which memorizes how the value are stored in the two registers R_0 and R_1 at the end of a loop iteration. This approach is shown in Algorithm 2.

Algorithm 2: Montgomery-Ladder with XOR split exponent I (Tunstall et al., 2018).

```
Require: $x \in \mathbb{G}$, two $n$-bit integers $A = \sum_{i=0}^{n-1} a_i 2^i$ and
    $B = \sum_{i=0}^{n-1} a_i 2^i$
Ensure: $x^\kappa$ where $\kappa = A \oplus B$
1: $R_0 \leftarrow 1_\mathbb{G}$; $R_1 \leftarrow 1_\mathbb{G}$; $R_2 \leftarrow 1_\mathbb{G}$; $b' \xleftarrow{R} \{0,1\}$;
   $R_{\neg b'} \leftarrow x$;
2: for $i = n-1$ down to 0 do
3:     $R_2 \leftarrow R_{a_i} \cdot R_{\neg a_i}$;
4:     $R_{a_i} \leftarrow R^2_{(b_i \oplus b') \oplus a_i}$;
5:     $R_{\neg a_i} \leftarrow R_2$;
6:     $b' \leftarrow b_i$;
7: return $R_{b'}$
```

The authors in (Tunstall et al., 2018) propose a second approach (Algorithm 3) which randomizes the signed version of the Montgomery ladder. This algorithm perform the exponentiation as a sequence of two multiplications one between R_0 and R_1 and between R_0 and either $U_0 = x$ or $U_1 = x^{-1}$. In Algorithm 3 the addresses used in the store instructions are fixed and then they are not correlated to the bits of the exponent. The random bit a_i (and thus $b_i = \kappa_i \oplus a_i$) is used to randomize the load instructions in Step 4 and Step 5. The bit b' is used to memorize how the data are placed in the registers R_0 and R_1 at the end of a loop iteration.

Algorithm 3: Montgomery-Ladder with XOR split exponent II (Tunstall et al., 2018).

```
Require: $x \in \mathbb{G}$, two $n$-bit integers $A = \sum_{i=0}^{n-1} a_i 2^i$ and
    $B = \sum_{i=0}^{n-1} a_i 2^i$
Ensure: $x^\kappa$ where $\kappa = A \oplus B$
 1: $R_0 \leftarrow 1_\mathbb{G}$; $R_1 \leftarrow 1_\mathbb{G}$; $R_2 \leftarrow 1_\mathbb{G}$; $b' \xleftarrow{R} \{0,1\}$;
    $R_{\neg b'} \leftarrow x$;
 2: for $i = n-1$ down to 0 do
 3:     $R_0 \leftarrow R_{b_i \oplus b'} \cdot R_{(b_i \oplus b') \oplus a_i}$;
 4:     $R_1 \leftarrow R_0 \cdot U_{b_i}$;
 5:     $b' \leftarrow b_i$;
 6: return $R_{b'}$
```

3 WEAKNESSES OF BOOLEAN SPLIT RANDOMIZATION

In (Tunstall et al., 2018) the authors claim that their algorithm combined with a point blinding technique has the same security level but with a lower cost as the method based on randomizing the exponent (Coron, 1999; Tunstall and Joye, 2010). For Algorithm 2 and 3, this is not entirely true:

- Randomizing the exponent alone prevents from the three attacks : DPA, collision and ADPA. This approach ensure that the sequence of points $R_{0,i}$ and $R_{1,i}$ in $\mathbb{G}$ for $i = n-1, \ldots, 0$ computed during the exponentiation are always different. This renders impossible to predict the power consumption of operation involving $R_{0,i}$ and $R_{1,i}$ and then prevents DPA and collision attack. Randomizing the exponent also randomly changes the bits of the exponent, which implies that the stores and loads are also randomized and thus unpredictable.
- Randomizing the loads and stores like it is done in Algorithm 2 and 3 does not modify the sequence of computed element $R_{0,i}$ and $R_{1,i}$ of $\mathbb{G}$. So a correct guess of the bits of the exponent would lead to a correct guess of R_0 and R_1 at the considered iteration. So if we want to hide these values we have to inject a high level of randomization otherwise a small amount of leakage could be exploited and would lead to a successful DPA. The only approach which produces a randomized and unpredictable sequence of computed points $R_{0,i}$ and $R_{1,i}$ in the exponentiation is the point blinding of (Coron, 1999) which multiplies x by a random value.

The table below summarizes the strength of the counter measures reviewed in Section 2 when they are used alone. This table shows that exponent randomizations are the most robust methods.

Table 1.

Counter-measure	DPA and CA	ADPA
Point blinding	yes	no
Exponent randomization	yes	yes
Address randomization	no	yes

Algorithm 3 has a more important flaw. The main problem of Algorithm 3 is that in Step 3, a multiplication of two different data R_0 and R_1 is done when $a_i = 1$ but when $a_i = 0$ this multiplication is either $R_0 \times R_0$ or $R_1 \times R_1$, which are squarings done using a multiplication routine. But distinguishig such a squaring done with multiplication routine from a genuine multiplication can be detected by power analysis. In (Hanley et al., 2011) the authors showed that the power consumption of these operations can be distinguished using template methodology.

If we successfully distinguish the power trace of $R_0 \times R_1$ from either $R_1 \times R_1$ or $R_0 \times R_0$, for unknown R_0 and R_1, we can deduce the value of a_i for $i = n-1, \ldots, 0$. Then the randomization of loads and stores is broken and DPA can be conducted on the address bits of Algorithm 3.

4 EXPERIMENTATION

In this section we present experimental results of the proposed attack on address randomization of Algorithm 3. The attack is performed on a simulated power traces using the hamming weight leakage model.

4.1 Simulation of Power Consumption

We target an RSA modular exponentiation with an RSA modulus N of bit length 1024. We consider an implementation of the modular exponentiation RSA based on a word-level Montgomery modular multiplication algorithm (Bosselaers et al., 1993). Then in order to simulate the power consumption of a full

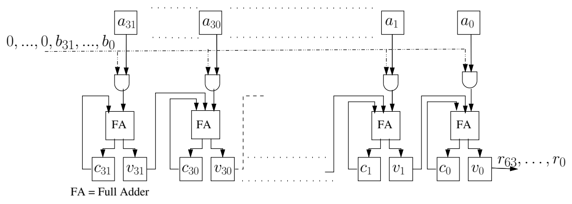

Figure 1: 32-Bit sequential multiplier.

modular multiplication we only need to generate the power consumption of w-bit additions and multiplications.

32-bit multiplier and adder. We defined a circuit performing multiplication of 32-bit integers and a circuit performing addition of 32-bit integers. The multiplier shown in Fig. 1 is a bit sequential multiplier. It computes $r = a \times b$ where $a = (a_{31}, \ldots, a_0)_2$ and $b = (b_0, \ldots, b_1)_2$ as shown in the following pseudocode:

```
for $i = 0$ to $32$ do
    $r \leftarrow r + 2^i(b_i \times a)$
```

The i-th bit r_i is output after the i-th clock-cycle. The additions in Fig. 1 are done through carry save adder in order to reduce the critical path delay. This means that the carries are not propagated and but are saved in the flip-flops c_i for $i = 0, \ldots, 31$. After the first 32 iterations the first 32 bits $r_0, \ldots, r_{31}$ of r are generated and output. But it remains to perform 32 more iterations, with input 0 in place of b_i ,to propate the carries and generate $r_{32}, \ldots, r_{63}$.

We do not provide the circuit of the 32-bit adder, since it is a classical 32 bit adder, which can be easily found in the literature.

To get the power consumption of one clock-cycle of the multiplier we compute the hamming weight of signal flowing in the wires. We split the clock cycle into four parts P_1, P_2, P_3 and P_4. We assume that at beginning of a cycle there are only 0 on all wires. Then we propagate data from the flip-flop and deduce the simulated consumption:

- Each wire containing a signal 1 and connecting a flip-flop and a first gate contributes to 1 on the consumption of P_1, P_2, P_3 and P_4.
- Each wire containing a 1 and connecting a first gate and a second gate contributes to to the consumption of P_2, P_3 and P_4.
- Each wire containing a 1 and connecting a second gate and a third gate contributes to 1 to the consumption of P_3 and P_4.
- Each wire containing a 1 and connecting a third gate and the flip-flop contributes to 1 to the consumption of P_4.

The above simulation of the consumption is highly simplified but we believe that it is sufficient to validate the potential threat of the proposed attack on Algorithm 3. We show in Fig. 2 an example of the simulated power consumption of one w-bit multiplication.

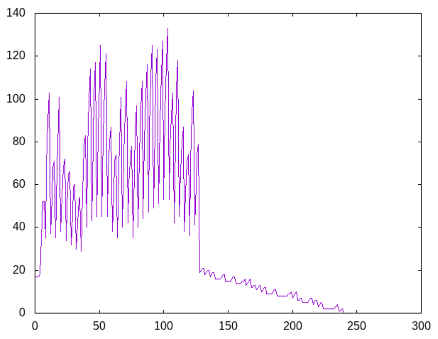

Figure 2: Power trace of a 32-bit multiplication.

Word level Montgomery modular multiplication. For a modular multiplication we use the word level version (Bosselaers et al., 1993) of the Montgomery modular multiplication with word size w. Given two integers A and B consisting of s words the modular multiplication are performed as follows

```
1: $R \leftarrow (0, \ldots, 0)_{2^w}$
2: for $i = 0$ to $s - 1$ do
3:     $q \leftarrow (R + A[i] \times B) \times N' \mod 2^w$
4:     $R \leftarrow (R + q \times N + A[i] \times B)/2^w$
```

Step 3 consists of two w-bit multiplications and one w-bit addition. Step 4 involves two products of a ws-bit integer by a w-bit integer and two additions of two ws-bit integers. Each of these operations are computed through a sequence of w-bit multiplications and/or additions.

4.2 Distinguishing a Square from a Multiplication

We would like to distinguish a multiplication $(A \times B) \mod N$ from a squaring $(A \times A) \mod N$. The main idea to get such distinguisher is that during the multiplication of $(A \times A) \mod N$ for each $i \neq j$ the same product is done twice as $A[i] \times A[j]$ and in reverse operand $A[j] \times A[i]$ in Step 3 the word level Montgomery multiplication. But since these multiplications involve the same data their power trace must be correlated.

Then we tried to compute the covariance of the power traces of the 32-bit multiplication in order to determine if they are correlated or not. But this strategy was not successful. Probably this is due to the non symmetric form of the considered 32-bit multiplier. We tried another strategy where we evaluate the

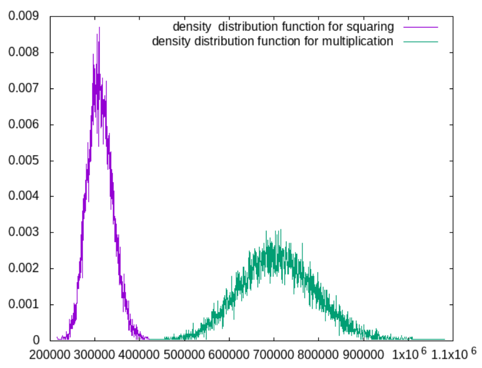

Figure 3: Square and non square repartition of power consumption.

difference of the mean of the power traces:

$$\sigma_{i,j}(A,B) = |\int Tr(Mul(A[i],B[j])| - |\int Tr(Mul(B[i],A[j])|$$

This computed value might be low if $A = B$, since, in this case, many computed bits appearing in the multiplication $Mul(A[i],A[j])$ also appear in $Mul(A[j],A[i])$ and they cancel out in the difference of $\sigma_{i,j}(A,A)$. If $A \neq B$ these bits do not cancel out resulting in a higher value for $\sigma_{i,j}(A,B)$. This effect is amplified if we add up $\sigma_{i,j}(A,B)$ over all $i < j$:

$$\sigma(A,B) = \sum_{0 \leq i < j \leq s} \sigma_{i,j}(A,B).$$

In order to check this fact we performed the following experiment: for a sufficiently large number of time ($\cong 10000$) we chose randomly A and B and computed $\sigma(A,B)$ and $\sigma(A,A)$. We obtained the resulting density distribution function shown in Fig. 3.

Fig. 3 clearly shows that we can easily distinguish a squaring from a genuine multiplication. We estimated the mean value m and the standard deviation d for the two cases: squaring (S) and genuine multiplication (M):

$$\begin{aligned} m_S &= 310000, d_S &= 28000 \\ m_M &= 710038, d_M &= 82000 \end{aligned}$$

Given a power trace for a multiplication $A \times B \bmod N$, we compute $\sigma(A,B)$ and we deduce that $A = B$ if $\sigma(A,B)$ is close to m_S and $A \neq B$ if it is close to m_M. To a get an estimation of the probability of success of this approach, we use the following inequality of Bienaymé-Tchebychev for a random variable X with mean m and standard deviation d:

$$\mathbb{P}(|X - m| > kd) < 1/k^2.$$

We chose $k = 3$ leading to a level of confidence in our distinguisher of $1 - 1/9$.This is validated by an experiment for 5000 samples for which we get 99.5% of success.

4.3 Final Step: ADPA

Now, since we have a reliable distinguisher, we can proceed to the next step: finding the bits of the exponent with a DPA attack on the address bits. Given a power trace of an exponentiation $x^\kappa \bmod N$ we can find the random bit a_i used in Algorithm 3 for all $i = 0, \ldots, n-1$. Our goal now is to find difference of the power trace related to the address bits involved in loads or stores. Let us analyze the loads and stores in order to select interisting power traces and compute a difference producing a peak if the guess is incorrect and a flat trace if the guess is correct.

The table below shows the relation between the $(i+1)$-th iteration and i-th iteration for the operation done in Step 5 in Algorithm 3 depending on the value of b', k_i and a_i.

Table 2.

$(i+1)$-th iter.		i-th iter.	
bit	Op. (Step 5)	bits	Op. (Step 4)
$\mathbf{b' = 0}$	$\mathbf{R_1 \leftarrow R_0U_0}$	$\mathbf{k_i = 0, a_i = 0}$	$\mathbf{R_1 \leftarrow R_0U_0}$
$b' = 0$	$R_1 \leftarrow R_0U_0$	$k_i = 0, a_i = 1$	$R_1 \leftarrow R_0U_1$
$\mathbf{b' = 1}$	$\mathbf{R_1 \leftarrow R_0U_1}$	$\mathbf{k_i = 0, a_i = 0}$	$\mathbf{R_1 \leftarrow R_0U_0}$
$b' = 1$	$R_1 \leftarrow R_0U_1$	$k_i = 0, a_i = 1$	$R_1 \leftarrow R_0U_1$
$\mathbf{b' = 0}$	$\mathbf{R_1 \leftarrow R_0U_0}$	$\mathbf{k_i = 1, a_i = 0}$	$\mathbf{R_1 \leftarrow R_0U_1}$
$b' = 0$	$R_1 \leftarrow R_0U_0$	$k_i = 1, a_i = 1$	$R_1 \leftarrow R_0U_0$
$\mathbf{b' = 1}$	$\mathbf{R_1 \leftarrow R_0U_1}$	$\mathbf{k_i = 1, a_i = 0}$	$\mathbf{R_1 \leftarrow R_0U_1}$
$b' = 1$	$R_1 \leftarrow R_0U_1$	$k_i = 1, a_i = 1$	$R_1 \leftarrow R_0U_0$

If we focus on the cases $a_i = 0$ (the row in bold in the table), we can notice that:

- Case 1: $b' \oplus k_i = 0$. The operations in Step 5 of the two iterations are the same. In particular the address for U_i is the same.
- Case 2. $b' \oplus k_i = 1$. The operations in Step 5 of the two iterations are different. In particular the address bits for U_i are different.

Consequently, we proceed by guessing the value of $g = k_i \oplus k_{i+1}$. Then we select the power traces such that $a_i = 0$ and such that $a_{i+1} \oplus g = 0$. If our guess is correct we will have:

$$\begin{aligned} 0 &= a_{i+1} \oplus g = a_{i+1} \oplus k_{i+1} \oplus k_i \\ &= b_{i+1} \oplus k_i = b' \oplus k_i. \end{aligned}$$

This means that if the guess g is correct, then Case 1 applies, which means that computing the difference of the power traces of loop i and $i+1$ would lead to a zero difference for the address of U_i, and the difference will be flat. If the guess is not correct, we are in Case 2, and the difference would be equal to the consumption of the address of U_0 minus the one for U_1. Adding a sufficient number of such differences would lead to a flat difference if the guess is correct and a peak if it is not correct.

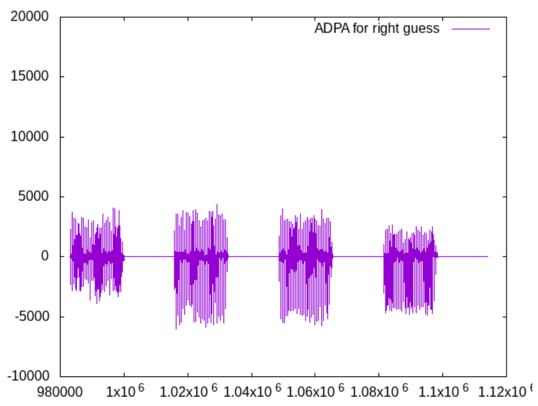

Figure 4: Difference for a correct guess on $k_{i+1} \oplus k_i$.

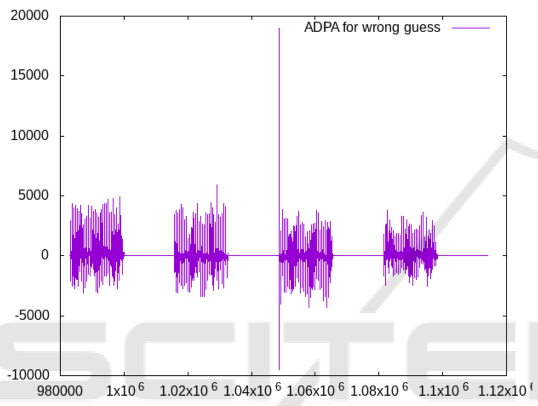

Figure 5: Difference for a wrong guess on $k_{i+1} \oplus k_i$.

In Fig. 4 and Fig. 5 we provide the ADPA obtained for an RSA of size 1024 bits and using 4000 traces. We can see the peak in Fig. 5 showing that the guess g is not correct. There is no peak in Fig. 4 which means that the guess is correct.

This experimentation shows that the proposed approach is effective to extract the whole key. This means that Algorithm 3 does not provide the claimed protection from ADPA. This attack works even if the elements are blinded at the beginning of the exponentiation, by either a randomized representation or a multiplication with a random element.

5 CONCLUSION

In this paper we considered two exponentiation algorithms (Algorithm 2 and 3) proposed in (Tunstall et al., 2018) with randomized store and load in order to counter-act address bit differential power analysis. We analyzed the security of these approaches, and we showed that Algorithm 3 has a an important flaw. Indeed, the operation done in Step 3 of Algorithm 3 is a square or multiplication depending on the bit used for load and store randomization. With a simulated power consumption we showed that we can distinguish a square from a multiplication. This means that the randomization of loads and stores in Algorithm 3 is not effective anymore and an ADPA can be conducted to recover the whole secret key with a few thousand power traces.

REFERENCES

Bosselaers, A., Govaerts, R., and Vandewalle, J. (1993). Comparison of Three Modular Reduction Functions. In *CRYPTO'93*, volume 773 of *LNCS*, pages 175–186.

Clavier, C., Feix, B., Roussellet, M., and Verneuil, V. (2010). Horizontal Correlation Analysis on Exponentiation. In *ICICS 2010*, volume 6476 of *LNCS*, pages 46–61.

Coron, J.-S. (1999). Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In *CHES 1999*, pages 292–302.

Fouque, P. and Valette, F. (2003). The Doubling Attack – Why Upwards Is Better than Downwards. In *CHES 2003*, pages 269–280.

Hanley, N., Tunstall, M., and Marnane, W. (2011). Using templates to distinguish multiplications from squaring operations. *Int. J. Inf. Sec.*, 10(4):255–266.

Itoh, K., Izu, T., and Takenaka, M. (2003). Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA. In *CHES 2002*, LNCS, pages 129–143.

Izumi, M., Ikegami, J., Sakiyama, K., and Ohta, K. (2010). Improved countermeasure against address-bit DPA for ECC scalar multiplication. In *DATE 2010*, pages 981–984.

Joye, M. and Yen, S. (2002). The Montgomery Powering Ladder. In *CHES 2002*, volume 2523 of *LNCS*, pages 291–302.

Kocher, P. (1996). Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO '96*, volume 1109 of *LNCS*, pages 104–113.

Kocher, P. C., Jaffe, J., and Jun, B. (1999). Differential Power Analysis. In *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397.

Smart, N. P., Oswald, E., and Page, D. (2008). Randomised representations. *IET Inform. Security*, 2(2):19–27.

Tunstall, M. and Joye, M. (2010). Coordinate Blinding over Large Prime Fields. In *CHES 2010*, pages 443–455.

Tunstall, M., Papachristodoulou, L., and Papagiannopoulos, K. (2018). Boolean Exponent Splitting. Technical Report 2018/1226, IACR Cryptology ePrint Archive.

Yen, S., Ko, L., Moon, S., and Ha, J. (2006). Relative Doubling Attack Against Montgomery Ladder. In *ICISC 2005*, LNCS, pages 117–128.