# Power Analysis for Secret Recovering and Reverse Engineering of Public Key Algorithms

Frederic Amiel[1,⋆], Benoit Feix[2,⋆⋆], and Karine Villegas[1]

[1] GEMALTO, Security Labs,
La Vigie, Avenue du Jujubier, ZI Athélia IV,
F-13705 La Ciotat Cedex, France
firstname.familyname@gemalto.com
[2] INSIDE CONTACTLESS
41 Parc Club du Golf
13856 Aix-en-Provence, Cedex 3, France
bfeix@insidefr.com

**Abstract.** Power Analysis has been deeply studied since 1998 in order to improve the security of tamper resistant products such as Trusted Platform Module (TPM). The study has evolved from initial basic techniques like simple and differential power analysis to more complex models such as correlation. However, works on correlation techniques have essentially been focused on symmetric cryptography. We analyze here the interests of this technique when applied to different smartcard coprocessors dedicated to asymmetric cryptography implementations. This study leads us to discover and realize new attacks on RSA and ECC type algorithms with fewer curves than classical attacks. We also present how correlation analysis is a powerful tool to reverse engineer asymmetric implementations.

**Keywords:** Public key cryptography, arithmetic coprocessors, exponentiation, side-channel analysis, reverse engineering.

## 1 Introduction

Public key cryptography has been widely used since its introduction by Diffie and Hellman [DH76] in 1976. Nowadays most famous applications are RSA [RSA78], invented in 1978 by Rivest, Shamir, and Adleman, and elliptic curves cryptosystems independly introduced by Koblitz [Kob87] and Miller [Mil86].Both kinds of asymmetric schemes require arithmetic operations in finite fields. For instance the use of modular arithmetic is necessary for exponentiation primitive in RSA or DSA [NIS00], as well as for point multiplication in elliptic curves. Therefore to obtain efficient computations, dedicated arithmetic coprocessors have been introduced in embedded devices.

For years tamper resistant devices have been considered as secure until 1996 when Kocher introduced the first side-channel attack (SCA) based on execution

⋆ This author has recently left Gemalto.

⋆⋆ Part of this work has been done when this author was with Gemalto.

C. Adams, A. Miri, and M. Wiener (Eds.): SAC 2007, LNCS 4876, pp. 110–125, 2007.

time measurements [Koc96]. A few years later power analysis was introduced by Kocher, Jaffe and Jun [KJJ99]. Their techniques, named simple power analysis (SPA) and differential power analysis (DPA), threaten any naive cryptographic algorithm implementation. Because an electronic device is composed of thousands of logical gates that switch differently depending on the executed operations, the power consumption depends on the executed instructions and the manipulated data. Thus by analyzing the power consumption of the device on an oscilloscope it is possible to observe its behavior and then to deduce from this power curve some secret data. Later, in 1999, Messerges, Dabbish and Sloan [MDS99] applied DPA to modular exponentiation which is the heart of several public key algorithms. In 2004, Brier, Clavier and Olivier [BCO04] introduced correlation power analysis (CPA) with a leakage model. This method has proven its efficiency on symmetric key algorithms, and needs very few curves to recover a secret key compared to classical DPA.

In this paper, we focus on this technique for which application to asymmetric algorithms has not been yet publicly reported. We introduce new attacks, illustrated by concrete experiments, to apply CPA on these algorithms. Indeed any arithmetic operation can be threatened by correlation analysis. For instance, we show how to reveal on a single correlation curve the whole private exponent during RSA exponentiation, and even during RSA CRT exponentiations. In addition we introduce a new case for CPA: the ability to realize precise reverse engineering. Once secret implementation and component hardware design have been recovered more powerful attacks can be envisaged.

The paper is organized as follows. Section 2 gives an overview of asymmetric algorithms embedded implementations. Section 3 describes well-known SCA techniques related to this article. New applications of correlation analysis on public key algorithms are discussed in Section 4. Practical results on different smartcard coprocessors are presented, they validate our attacks and their efficiency compared to classical differential power analysis. In Section 5 we present another application domain of correlation analysis by describing how it can be used to realize reverse engineering. We conclude our research in Section 6.

## 2 Public Key Embedded Implementations

We introduce here principles used later in this paper: modular multiplication and exponentiation, especially the ones designed by Montgomery that are particularly suitable for embedded implementations, and the RSA public key cryptosystem.

### 2.1 Modular Multiplication

Chip manufacturers usually embed arithmetic coprocessors to compute modular multiplications $x \times y \mod n$ for long integers $x$, $y$ and $n$.

Montgomery introduced in [Mon85] an efficient algorithm named Montgomery Modular Multiplication. Other techniques exist: interleaved multiplication-reduction with Knuth, Barrett, Sedlack or Quisquater methods [Dhe98].

**Montgomery modular multiplication**
Given a modulus $n$ and two integers $x$ and $y$, of size $v$ in base $b$, with $\gcd(n, b) = 1$ and $r = b^{\lceil log_b(n) \rceil}$, MontMul algorithm computes:

$$\mathsf{MontMul}(x,\, y,\, n) = x \times y \times r^{-1} \mod n$$

**Algorithm 2.1.** MontMul: Montgomery modular multiplication algorithm
INPUT: $n$, $0 \leq x = (x_{v-1}x_{v-2} \dots x_1x_0)_b, y = (y_{v-1}y_{v-2} \dots y_1y_0)_b \leq n-1$ , $n' = -n^{-1} \mod b$
OUTPUT: $x \times y \times r^{-1} \mod n$

**Step 1.** $a = (a_{v-1}a_{v-2} \dots a_1a_0) \leftarrow 0$
**Step 2.** **for** $i$ from 0 to $v-1$ **do**
  $u_i \leftarrow (a_0 + x_i \times y_0) \times n' \mod b$
  $a \leftarrow (a + x_i \times y + u_i \times n)/b$
**Step 3.** **if** $a \geq n$ **then** $a \leftarrow a - n$
**Step 4.** **Return**($a$)

Refer to papers [Mon85] and [KAK96] for details of MontMul implementation.

## 2.2 RSA

RSA signature of a message $m$ consists in computing the value $s = m^d \mod n$. Signature $s$ is then verified by computing $m = s^e \mod n$. Integers $e$ and $d$ are named the *public exponent* and the *private exponent*, $n$ is called the *modulus*.

Some of the attacks introduced in this paper aim at recovering this private exponent $d$ during decryption. Many implementations of the RSA algorithm rely on the Chinese Remainder Theorem (CRT) as it greatly improves performance in terms of execution speed, theoretically up to four times faster, cf. Alg. 2.2..

**Algorithm 2.2.** RSA CRT
INPUT: $p$, $q$, $d_p$, $d_q$, $i_q = q^{-1} \mod p$: the private elements, $m$: the message
OUTPUT: $s$: the signature

**Step 1.** Compute $m_p = m \mod p$ and $m_q = m \mod q$
**Step 2.** Compute $s_p = m_p^{d_p} \mod p$ and $s_q = m_q^{d_q} \mod q$
**Step 3.** Compute $s = s_q + ((s_p - s_q) \times i_q \mod p) \times q$
**Step 4.** **Return**($s$)

In this case SCA is applied either to exponentiations to recover $d_p$ and $d_q$, or to the recombination step to find $q$ or to the initial reductions to recover $p$ and $q$.

Modular exponentiation is the most time-consuming operation of RSA primitives. It is then essential to use an efficient method for exponentiation. Alg. 2.3.

below, based on MontMul, gives the Montgomery exponentiation algorithm and is particularly suited for embedded RSA implementations.

For a given modulus $n = (n_{v-1}n_{v-2}\dots n_1 n_0)_b$, we define $r = b^{\lceil log_b(n)\rceil}$ and the following function $f_n$:

$$\begin{aligned} f_n : [0, n-1] &\longrightarrow [0, n-1] \\ x &\longrightarrow x \times r \mod n \end{aligned}$$

Let $x$ and $y$ be integers such that $0 \leq x, y < n$, we denote $\overline{x} = f_n(x)$ and $\overline{y} = f_n(y)$. We have the following property: MontMul($\overline{x}$, $\overline{y}$, $n$)$=x\times y\times r \mod n = f_n(x\times y)$ which is very useful to define the Montgomery modular exponentiation, cf. Alg. 2.3..

**Algorithm 2.3.** MontExp: Montgomery Square and Multiply from left to right

INPUT: integers $m$ and $n$ such that $m < n$, $k$-bit exponent $d = (d_{k-1}d_{k-2}\dots d_1 d_0)_2$
OUTPUT: MontExp($m$,$d$,$n$)$= m^d \mod n$

**Step 1.** $a = r$
**Step 2.** $\overline{m} = f_n(m)$
**Step 3. for** $i$ from $k-1$ to $0$ **do**
  $a =$ MontMul($a$,$a$,$n$)
  **if** $d_i = 1$ **then** $a =$ MontMul($a$,$\overline{m}$,$n$)
**Step 4.** $a = a \times r^{-1} \mod n =$ MontMul($a$,1,$n$)
**Step 5. Return**($a$)

## 3 Power Analysis

Among the different side-channel analysis techniques, we present in this section DPA applied to modular exponentiation and recall the principles of CPA based on a Hamming distance linear model.

### 3.1 Differential Power Analysis on Exponentiation

We want to recover the secret exponent $d$ during Alg. 2.3.. We explain here the Zero-Exponent Multiple-Data (ZEMD) attack from Messerges, Dabbish and Sloan [MDS99]. Suppose we know the $u$ most significant bits of $d$; i.e. $d_{k-1}\dots d_{k-u}$, and we want to recover the $(u+1)^{st}$ bit of $d$. We make the guess $d_{k-u-1} = g$ with $g = 0$ or 1, and we want to confirm this guess. We execute on the device to attack $t$ executions of the algorithm with input messages $m_1 \dots m_t$ and collect the curves $C_1 \dots C_t$ corresponding to the power consumption of these executions.

Let $S_\epsilon$ be the integer $S_\epsilon = \sum_{j=0}^{\epsilon-1} d_{k-1-j}.2^{\epsilon-j-1}$. A selection function $D(m_j, d_{k-u-1})$ is defined and used to split the set of curves into two subgroups such as: $G_{0,u+1} = \{C_j \text{ such that } D(m_j, d_{k-u-1}) = 0\}$ and $G_{1,u+1} = \{C_j \text{ such that } D(m_j, d_{k-u-1}) = 1\}$. For instance $D(m_j, d_{k-u-1})$ could be equal to the least

significant bit of $f_n(m_j^{S_{u+1}}) = f_n(m_j^{2S_u+g})$ (if the guess of $g$ is correct). Then compute the differential trace $T_{u+1}$:

$$T_{u+1} = \frac{\sum_{C_j \in G_{1,u+1}} C_j}{|G_{1,u+1}|} - \frac{\sum_{C_j \in G_{0,u+1}} C_j}{|G_{0,u+1}|}$$

Finally if the guess of $d_{k-u-1}$ is correct, the trace $T_{u+1}$ will have DPA peaks in the part of the curve corresponding to the manipulation of data associated to $D(m_j, d_{k-u-1})$, for instance in the next square. If the guess of $d_{k-u-1}$ is wrong, no peak should be visible on trace $T_{u+1}$. Once $d_{k-u-1}$ is recovered, the same analysis can be applied successively to the following secret bits of exponent $d$ with $T_{u+2}$, $T_{u+3}$ ... This attack can be improved by multi-bit selection. In that case, the function $D(m_j, d_{k-u-1})$ takes into consideration several bits of the value $f_n(m_j^{2S_u+d_{k-u-1}})$ [Mes00].

We refer the reader to Appendix C where differential trace results are presented.

### 3.2 Correlation Power Analysis

As published by Brier, Clavier and Olivier [BCO04], it is known that CPA can be applied to obtain successful attacks on symmetric algorithms, for instance DES and AES, with fewer messages than classical DPA. The power consumption of the device is supposed to be linear in $H(D \oplus R)$, the Hamming distance of the data manipulated $D$, with respect to a *reference state* $R$. The linear correlation factor is used to correlate the power curves with this value $H(D \oplus R)$. The maximum correlation factor is obtained for the right guess of secret key bits.

Let $W$ be the power consumption of the chip, its consumption model is:

$$W = \mu H(D \oplus R) + \nu.$$

The correlation factor $\rho_{C,H}$ between the set of power curves $C$ and values $H(D \oplus R)$ is defined as: $\rho_{C,H} = \frac{cov(C,H)}{\sigma_C \sigma_H}$.

The principle of the attack is then the following:

- Perform $t$ executions on the chip with input data $m_1 \dots m_t$ and collect the corresponding power curves $C_1 \dots C_t$.
- Predict some intermediate data $D_i$ as a function of $m_i$ and key hypothesis $g$.
- Produce the set of the $t$ predicted Hamming distances: $\{H_{i,R} = H(D_i \oplus R), i = 1 \dots t\}$ .
- Calculate the estimated correlation factor:

$$\hat{\rho}_{C,H} = \frac{cov(C,H)}{\sigma_C \sigma_H} = \frac{t \sum (C_i H_{i,R}) - \sum C_i \sum H_{i,R}}{\sqrt{t \sum C_i^2 - (\sum C_i)^2} \sqrt{t \sum H_{i,R}^2 - (\sum H_{i,R})^2}}, i = 1 \dots t$$

When the attacker makes the right guesses for values of the reference state $R$ and secret leading to data $D$, the correlation factor $\rho$ is maximum. It can also

be seen graphically by tracing the correlation curve $C_{\rho,g}$. Of course, peak(s) of correlation is (are) visible on $C_{\rho,g}$ when the guess is correct. The attacker has recovered a part of the secret value and a reference state during the execution. $R$ can be an *opcode* value or a *look-up-table* address for instance.

# 4 Correlation Power Analysis of Asymmetric Implementations

Previous CPA publications were mainly focused on symmetric algorithms such as DES and AES. The use of CPA against public key implementations has never been publicly investigated, except for Joye who theoretically introduced its application to a second order attack in ECC [Joy04]. This is the subject of this section where we present new attacks based on CPA.

## 4.1 Correlation on Intermediate Value in Modular Exponentiation

When computing an RSA exponentiation, if a guess $g$ (0 or 1) is made for a bit $d_{k-u-1}$ of the secret exponent, for a message $m_j$ you can aim to correlate the power curve with the full data $R \oplus m_j^{2S_u+g} \mod n$. A more realistic choice is to select, depending on the size $b$ of the chip multiplier, only a part of the intermediate data :

$$(R \oplus (m_j^{2S_u+g} \mod n)) \wedge \omega_{b,s}$$

where $\omega_{b,s} = b^s(b-1)$ and $s \in [0, v-1]$, for instance choose $s = 0$. Thus from ZEMD DPA we derive a ZEMD CPA (Alg. 4.4. with MontExp).

**Algorithm 4.4.** ZEMD CPA on Montgomery exponentiation
INPUT: $n$ the modulus, $m_1, \ldots, m_t$ $t$ messages
OUTPUT: the secret exponent $d = (d_{k-1}d_{k-2}\ldots d_1 d_0)_2$

**Step 1.** Choose $s \in \{0, .., v\text{-}1\}$
**Step 2. for** $u$ from 0 to $k-1$ **do**
Guess $d_{k-1-u} = 1$
$$A_1 = \left\{ H((R \oplus f_n(m_i^{2S_u + d_{k-u-1}})) \wedge \omega_{b,s}), \quad i = 1, \ldots, t \right\}$$
$\rho_1 = \hat{\rho}_{C,A_1}$
Conclude $d_{k-1-u} = 1$ **if** $C_{\rho_1}$ has correlation peaks **else** $d_{k-1-u} = 0$
**Step 3. Return**($d$)

This attack can be optimized by simultaneously searching for many bits ($\alpha$) of $d$, kind of $\alpha$-ary CPA. In that case you have to compare $2^\alpha - 1$ correlation values, the maximum correlation value corresponding to the right guess of the $\alpha$ bits of $d$.

**Reference state value:** The difficulty in correlation analysis is the knowledge of the reference state value $R$, which must be known or at least guessed by the attacker. The natural choice is to take $R = 0$. In that case the correlation

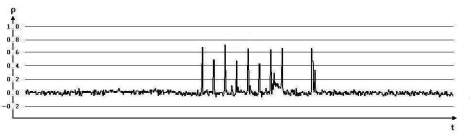

**Fig. 1.** ZEMD CPA on MontExp: correct guess

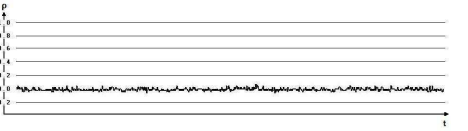

**Fig. 2.** ZEMD CPA on MontExp: wrong guess

model is reduced from the Hamming distance model to the Hamming weight one. Indeed we can expect to have either a hardware erase operation on initial register(s) of the multiplication algorithm, or the combinatorial part of the hardware modular multiplier to be in a stall state. If not, suppose $b = 2^8$ and $s = 0$: if $R$ is a constant value then try the whole 256 different possible values for $R$. The correlation analysis will be successful only for right guess of $d_{k-1-u}$ and $R$. A more complex case can be envisaged to evaluate $H(R \oplus D)$: it consists in choosing for $R$ previous intermediate data, for instance $R = u_i$, and for the newly obtained data $D$ the value $D = u_{i+1}$ at step 2 in Alg. 2.1. (Practical results are shown in Fig. 1 and Fig. 2.)

The correlation peaks can appear either during the data handling of the guessed intermediate value leading to the output result of the current operation, or caused by setting this value as input operand of the next operation (for instance, in the next square). Therefore, there are two different possible sources of correlation.

### 4.2 Correlation on Multiplicand Data

The drawback of ZEMD CPA is that the attack must be iterated for each guessed bit of $d$ (or even $l$-bit per $l$-bit), we need then to compute $k$ (or $k/l$) correlation curves. A more efficient attack can be considered when the correlation peaks are caused by the handling of the input operand. Indeed, during an exponentiation, for each multiplication (as opposed to squarings), one of the multiplicands is constant and equal to $m$ (or $f_n(m)$ for Montgomery). Therefore, by computing correlation on this multiplicand value we can expect to obtain CPA peaks each

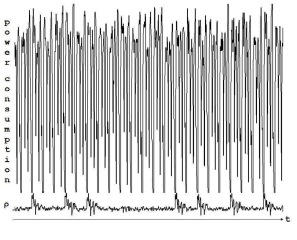

**Fig. 3.** CPA on multiplicand partial size during exponentiation, $R = 0$

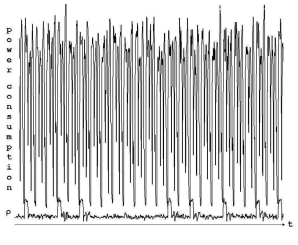

**Fig. 4.** CPA on multiplicand full size during exponentiation, $R = 0$

time it is manipulated. Thus all the multiplications by $m$ (or $f_n(m)$) could be seen on this single correlation curve. The full secret $d$ is thus recovered with a single correlation computation on all the curves. This attack has been realized with success on different chips. Practical results for a partial and a full correlation on the multiplicand value during an exponentiation are shown in Fig. 3 and Fig. 4. Indeed in Fig. 4 each multiplication can be seen on the correlation curve and as result $d$ can be easily deduced as it would be done in SPA.

**Algorithm 4.5.** CPA on Multiplicand Data in Montgomery exponentiation
INPUT: $n$ the modulus, $m_1, \ldots, m_t$ $t$ messages
OUTPUT: secret exponent $d = (d_{k-1}d_{k-2}\ldots d_1 d_0)_2$

**Step 1.** Choose $s \in \{0, .., v-1\}$
**Step 2.** Compute $A = \{H((R \oplus f_n(m_i)) \wedge \omega_{b,s}), \quad i = 1, \ldots, t\}$
**Step 3.** Compute $\rho = \hat{\rho}_{C,A}$ and its related correlation curve $C_\rho$
**Step 4.** Detect on $C_\rho$ the peaks to identify all the multiplications and deduce $d$
**Step 5. Return**($d$)

### 4.3 Correlation on RSA CRT

This section introduces new CPA attacks to recover the private key elements for each step of the Algorithm 2.2..

**Correlation during CRT modular exponentiations**
During modular exponentiations of message $m_j$; $s_p = m_j^{d_p} \mod p$ and $s_q = m_j^{d_q} \mod q$, it is not possible to apply ZEMD because $p$ and $q$ are unknown to the attacker.

However it is possible to do correlation on the multiplicand's value to recover $d_p$ when $m_j < p$ and $d_q$ when $m_j < q$. Choose $m_j < \min(p, q)$ to recover simultaneously $d_p$ and $d_q$. For instance if $p$ and $q$ are both $k$-bit primes, select messages $m_j$ in $[2, 2^{k-1}]$.

Note that this attack is not applicable to RSA CRT using Montgomery exponentiation as $f_p(m_j)$ and $f_q(m_j)$ are unpredictable. However it can be done on other exponentiations, using different modular arithmetic such as Barrett.

**Correlation during the CRT recombination**
The recombination (Step 3 of Alg. 2.2.) gives the ability to guess bits of the value of $q$ by CPA as $s$ is known. Indeed, one can observe that for the upper half bits of $s$ we have:

$$\left\lfloor \frac{s}{q} \right\rfloor = ((s_p - s_q) \times i_q \mod p) + \left\lfloor \frac{s_q}{q} \right\rfloor = (s_p - s_q) \times i_q \mod p$$

As $(s_p - s_q) \times i_q \mod p$ is an operand of the recombination step, it is then obvious that for the right guess of $q$ we should be able to obtain the best correlation factor by estimating this operand value. In practice, the attack realization needs to guess the value of $q$ by groups of $b$ bits starting from the most significant

words. For instance take $b = 2^8$ and the right guess of $b$ bits of $q$ corresponds to the highest correlation value obtained among the 256 guesses. For more details please refer to Alg. 4.6..

**Algorithm 4.6.** CPA on RSA CRT recombination

INPUT: $s_1, \ldots, s_t$ $t$ signatures
OUTPUT: the secret element $q = (q_{v-1}q_{v-2}\ldots q_1 q_0)_b$

**Step 1.** $q = 0$
**Step 2.**
**for** $i$ from $v-1$ to $0$ **do**
$g_{max} = 0$, $\rho_{max} = 0$
**for** $g$ from 0 to $b-1$ **do**
$\hat{q} = q + (g + 0.5) \times b^i$
$A = \left\{ H((R \oplus \left\lfloor \frac{s_j}{\hat{q}} \right\rfloor) \wedge \omega_{b,i}), \quad j = 1 \ldots t \right\}$
$\rho_g = \hat{\rho}_{C,A}$
**if** $|\rho_g| > |\rho_{max}|$ **then** $g_{max} = g$, $\rho_{max} = \rho$
$q = q + g_{max} \times b^i$
**Step 3. Return**($q$)

Choosing $\hat{q} = q + (g + 0.5) \times b^i$ instead of $\hat{q} = q + g \times b^i$ as estimator gives the correct decision when the correct value belongs to $\left[g \times b^i, (g+1) \times b^i\right] + q$.

On our implementation based on a Montgomery multiplier, one should take into account that the estimation of $A$ will depend on $f_n$ (see Paragraph 2.2).

**Correlation during the initial reductions**
Attacks on the initial reductions $m \mod p$ and $m \mod q$ would aim at recovering $p$ and $q$. Previous studies have been presented by Boer, Lemke and Wicke [BLW02] and Akkar [Akk04]. Contrary to those previous attacks, CPA works with any messages and fewer curves on any arithmetic operation such as addition or subtraction. Indeed, for message reduction and no matter what the algorithm is, the first steps always require a subtraction and/or an addition between a part of the message and the secret modulus $p$ or $q$. Note that even if the implementation is supposed to be protected against SPA, like the secure shift and reduce division algorithm [JV02], it is possible to perform CPA. For example, if $p$ has $k$ bits and the messages used for CPA, $m_1, \ldots m_t$, have $2k$ bits, the guesses will be performed on operands $\left\lfloor \frac{m_i - m_i \mod 2^k}{2^k} \right\rfloor - p$, for $i = 1, \ldots, t$. All the bytes of $p$ or $q$ from the least significant bytes to the most significant ones can be retrivied using this technique.

### 4.4 Application to Elliptic Curves Cryptosystems

ECDSA and El Gamal are two of the most widely known elliptic curves schemes. For details we suggest the reader refer to Appendix A and [ACD+06].

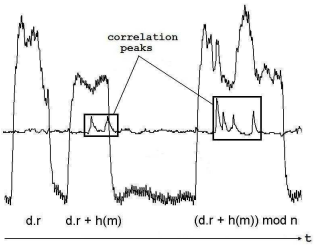


**Fig. 5.** CPA on ECDSA, correct guess

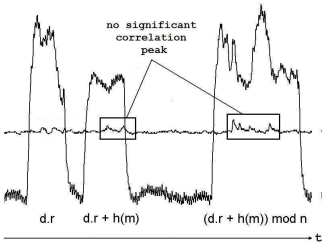


**Fig. 6.** CPA on ECDSA, wrong guess

**CPA on El Gamal Decryption:** This primitive can be vulnerable to CPA during the scalar multiplication $P_k = [d]Q_k$ where $d$ is the private key. $Q_k$ is a point of the curve returned in the ciphertext. CPA targets $d$. If the scalar multiplication is performed with the Double and Add algorithm (cf. Appendix A), different processes are done whether a given bit of $d$ is equal to 0 or 1. An attacker could identify the addition part by correlating consumption curves with $g(x_Q)$ a function of coordinate(s) of the point added according to the targeted implementation. For instance $g(x_Q)$ can be $x_Q \wedge \omega_{b,0}$.

**CPA on ECDSA signature:** here, $d \times r \mod n$ operation, with $d$ the private key and $r$ known as part of the signature, could be sensitive to CPA. On a modular multiplier based on Montgomery implementation (see Alg. 2.1 for further details) such leakage could occur during $x_i \times y_0$ or $x_i \times y$ operations. Fig. 5 illustrates a successful attack, the most significant byte of $d$ is recovered among 256 guesses. The attack algorithm is described in Appendix B.

### 4.5 Practical Results and Remarks

All the attacks presented have been tested with success on several smartcard coprocessors using different modular arithmetic implementations, cf. Fig. 7. In

| Chip | Algorithm | Attack | Curves for DPA | Curves for CPA |
|---|---|---|---|---|
| Coprocessor 1 | RSA Exponentiation | Intermediate value | 1500 | 150 |
| | RSA Exponentiation | Multiplicand value | 2500 | 300 |
| Coprocessor 2 | RSA Exponentiation | Intermediate value | 500 | 100 |
| | RSA Exponentiation | Multiplicand value | 1500 | 250 |
| | RSA CRT | Recombination step | No success | 150 |
| Coprocessor 3 | RSA CRT | Recombination step | No success | 4000 |
| | ECDSA | $d \times r \mod n$ | No success | 4000 |
| | RSA Exponentiation | Intermediate value | No success | 1000 |
| | RSA Exponentiation | Multiplicand value | No success | 3000 |

**Fig. 7.** Practical results

our experiments, the best results are obtained for $b$ parameter in $\omega_{b,s}$ equal to the radix of the multi-precision multiplier. On the other hand, another important parameter is the reference state value $R$. Successful results have been obtained with $R = 0$, which confirms the hypothesis that either a hardware erase operation is done on initial register(s) of the multiplication algorithm, or the combinatorial part of the hardware modular multiplier is in a stall state.

## 5 Reverse Engineering

Side channel analysis has already been used by Novak [Nov03] and Clavier [Cla04] to reverse engineer secret GSM A3/A8 algorithms. Later Daudigny, Ledig, Muller and Valette processed similarly on a DES implementation [DLMV].

Previously we showed that each data manipulation by the coprocessor could be observed by CPA. It has been used to recover secret keys and data in the previous analysis. We now show how CPA can also be used to recover the design of the coprocessor embedded in the chip.

In most software implementations, encryption and signature verification primitives do not implement any countermeasure as they do not manipulate secret data. Such primitives could then be used by an attacker to increase knowledge about the hardware multiplier. It could be useful as in software implementations, unsecure and secure primitives share the same hardware resources.

Fig. 8 gives the details of the leakage of a hardware multiplier during a square operation in RSA exponentiation.

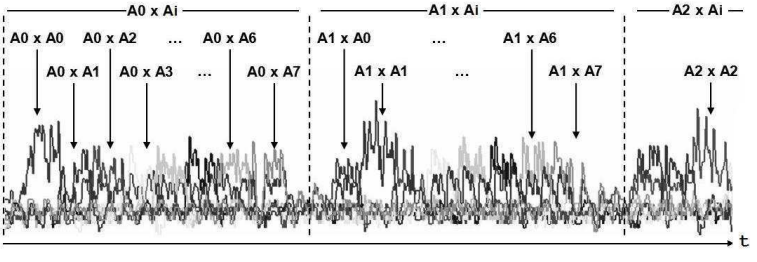


**Fig. 8.** Reverse engineering by CPA on Montgomery multiplication

By CPA, we are able to determine the precise implementation of the $b$-bit words multiplications in the modular multiplication algorithm: the size $b$ of the hardware multiplier and the order in which the operands are used can be determined. We can precisely localize the instant when each data $x_i$, $y_i$, $x_i \times y$, $u_i \times n$, etc. in Alg. 2.1. is manipulated. The order of use of such intermediate values depends on the algorithm supposed to be implemented (for instance FIOS, CIOS

in [KAK96]). Thus, for each guessed implementation, we can observe if the expected intermediate values appear and their exact position on the correlation curves. If not, the guess of the implementation is incorrect. Else, by combining many correlation curves for the different consecutives intermediate values, the exact implementation of the multiplication can be deduced. Fig. 8 illustrates part of the analysis we made on a chip using a Montgomery multiplier. We successfully recovered the size $b$ of the hardware multiplier, the sequence and the kind of operations processed and thus the algorithm implemented for MontMul.

With such a precise knowledge, more complex attacks can be considered and achieved to defeat the classical DPA countermeasures. Indeed higher order power analysis attacks can also be envisaged with more precision on the cycles to combine together. In [Wal01] Walter introduced the *Big Mac Attack* to recover a secret exponent $d$ only with the single power curve of the executed exponentiation. Such a reverse engineering by CPA gives the necessary knowledge required to realize this attack in the best conditions.
Such information can also be used to recover secret variables in asymmetric algorithms based on private specifications when the basic structure of the algorithm is known. Of course this implies that the attacker also needs to learn about the implementation done: the kind of coordinates that are used for elliptic curves (projective, Jacobian), kind of modular arithmetic, etc.

Furthermore in a fault attack scenario, the benefit of such information cannot be neglected as the effect of each fault injection during hardware multiplier execution could become predictable and with a really precise effect.

To avoid risks of reverse engineering we advise randomizing public elements during the computations using techniques such as described in [Cor99], [Koc96].

## 6 Conclusion

Several new attacks based on CPA have been presented in this paper. These attack schemes threaten most of public key algorithms such as RSA, RSA CRT or Elliptic Curves ones (ECES, ECDSA) if efficient countermeasures, such as blinding technics, are not implemented. Through experiments, we have demonstrated that CPA can detect and characterize leakages of arithmetic coprocessors. Therefore it gave us the ability to successfully implement the attacks we present by using the power consumption model based on the Hamming distance between the handled data and a constant reference state. As results of these experiments, we have proven also that the use of CPA is an improvement compared to classical attacks such as DPA. The efficiency of CPA has led us to proceed to reverse engineering of the design of coprocessors and then it has given the knowledge of the type of algorithm implemented (Montgomery, Barrett, ...), the size of hardware multiplier, the way to interleave operations and words in the multiplication algorithm etc. Furthermore, we have stressed that such precise knowledge of an arithmetic coprocessors can make realistic high-order side channel attacks

or even fault attack scenarios and improves their efficiency. Therefore, in order to avoid any potential attacker gaining experience on a secure device, we suggest using randomization techniques even for encryption or signature verification primitives usually considered as not sensitive.

## Acknowledgements

The authors would like to thank Benoit Chevallier-Mames and Christophe Clavier for their fruitful comments and advices on this paper.

## References

[ACD+06] Avanzi, R.-M., Cohen, H., Doche, C., Frey, G., Lange, T., Nguyen, K., Verkauteren, F.: Handbook of elliptic and hyperelliptic curve cryptography (2006)

[Akk04] Akkar, M.-L.: Attaques et méthodes de protections de protections de systèmes de cryptographie embarquée. PhD thesis, Université de Versailles (2004)

[BCO04] Brier, E., 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)

[BLW02] den Boer, B., Lemke, K., Wicke, G.: A DPA attack against the modular reduction within a CRT implementation of RSA. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 228–243. Springer, Heidelberg (2003)

[Cla04] Clavier, C.: Side channel analysis for reverse engineering (SCARE), an improved attack against a secret A3/A8 GSM algorithm. IACR Cryptology eprint archive (049) (2004)

[Cor99] 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)

[DH76] Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory 22(6), 644–654 (1976)

[Dhe98] Dhem, J.-F.: Design of an efficient public-key cryptographic library for RISC-based smart cards. PhD thesis, Université catholique de Louvain, Louvain (1998)

[DLMV] Daudigny, R., Ledig, H., Muller, F., Valette, F.: Scare of the DES. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 393–406. Springer, Heidelberg (2005)

[Joy04] Joye, M.: Smart-card implementation of elliptic curve cryptography and DPA-type attacks. In: Quisquater, J.-J., Paradinas, P., Deswarte, Y., El Kalam, A.A. (eds.) CARDIS, pp. 115–126. Kluwer, Dordrecht (2004)

[JV02] Joye, M., Villegas, K.: A protected division algorithm. In: CARDIS 2002. Proceedings of the Fifth Smart Card Research and Advanced Application Conference (2002)

[KAK96] Koç, Ç.K., Acar, T., Kaliski, B.-S.: Analysing and comparing Montgomery multiplication algorithms. IEEE Micro 16(3), 26–33 (1996)

[KJJ99] Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)

[Kob87] Koblitz, N.: Elliptic curve cryptosystems. Math. of Comp. 48(177), 203–209 (1987)

[Koc96] 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)

[MDS99] 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)

[Mes00] Messerges, T.S.: Power analysis attacks and countermeasures for cryptographic algorithms. PhD thesis, University of Illinois at Chicago (2000)

[Mil86] Miller, V.S.: Use of elliptic curves in cryptography. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 489–502. Springer, Heidelberg (1987)

[Mon85] Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation 44(170), 519–521 (1985)

[NIS00] NIST. Digital signature standard (DSS). Federal Information Processing Standards Publication 186-2 (January 2000)

[Nov03] Novak, R.: Side-channel attack on substitution blocks. In: Zhou, J., Yung, M., Han, Y. (eds.) ACNS 2003. LNCS, vol. 2846, pp. 307–318. Springer, Heidelberg (2003)

[RSA78] Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21, 120–126 (1978)

[Wal01] 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)

# A Elliptic Curves

## A.1 Double and Add Algorithm

**Algorithm A.7.** Double and Add

INPUT: $E$ the curve, $Q = (x_Q, y_Q)$ a point of $E$, $d = (d_{k-1}d_{k-2}\dots d_1 d_0)$ a scalar of $k$ bits
OUTPUT: value $P = [d]Q$

**Step 1.** $P = Q$
**Step 2. for** $i$ from $k-2$ to $0$ **do**
  $P = 2P$
  **if** $d_i = 1$ **then** $P = P + Q$
**Step 3. Return**$(P)$

### A.2 ECDSA Signature

Let the knowledge of the domain parameters be $D = (p, a, b, G, n, h)$, the key pair be $(d, Q)$ with $Q = [d]G$ and the required hash function be $h(.)$.

**Algorithm A.8.** ECDSA signature
INPUT: $D$, private key $d = (d_{k-1}d_{k-2}\dots d_1d_0)_2$, message $m$, $h(.)$.
OUTPUT: $m$ signature$= (s, r)$.

**Step 1.** Select a random $u$, $1 \leq u \leq n-1$
**Step 2.** Compute $R = [u]G = (x_R, y_R)$
**Step 3.** Compute $r = x_R \mod n$, **if** $r = 0$ go to Step1
**Step 4.** Compute $s = u^{-1}\left(h(m) + d \times r\right) \mod n$, **if** $s = 0$ go to Step1
**Step 5. Return**$(r, s)$

## B Algorithm for CPA on ECDSA

**Algorithm B.9.** CPA on ECDSA signature
INPUT: $(s_1, r_1), \dots, (s_t, r_t)$ $t$ signatures
OUTPUT: the secret element $d = (d_{k-1}d_{k-2}\dots d_1d_0)_b$

**Step 1.** $d = 0$
**Step 2.**
**for** $i$ from $k-1$ to $0$ **do**
$g_{max} = 0$, $\rho_{max} = 0$
**for** $g$ from $0$ to $b-1$ **do**
$\hat{d} = d + (g + 0.5) \times d^i$
$A = \left\{ H((R \oplus f(\hat{d} \times r_j)) \wedge \omega_{b,i}), \quad j = 1 \dots t \right\}$
$\rho_g = \hat{\rho}_{C,A}$
**if** $|\rho_g| > |\rho_{max}|$ **then** $g_{max} = g$, $\rho_{max} = \rho$
$d = d + g_{max} \times b^i$
**Step 3. Return**$(d)$

## C ZEMD DPA Practical Result

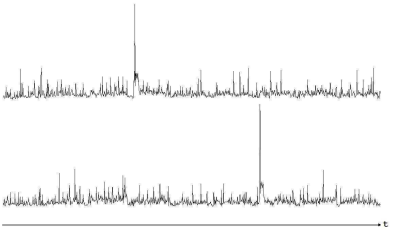


**Fig. 9.** ZEMD DPA: right guesses for $d_{n-2}$ and $d_{n-4}$, single bit selection

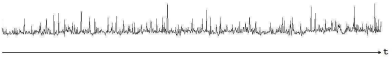


**Fig. 10.** ZEMD DPA: wrong guess for $d_{n-2}$, single bit selection