A New Type of Timing Attack: Application to GPS

Julien Cathalo, François Koeune, and Jean-Jacques Quisquater

Université catholique de Louvain
Place du Levant 3
1348 Louvain-la-Neuve, Belgium
{cathalo,fkoeune,jjq}@dice.ucl.ac.be

Abstract. We investigate side-channel attacks where the attacker only needs the Hamming weights of several secret exponents to guess a long-term secret. Such weights can often be recovered by SPA, EMA, or simply timing attack. We apply this principle to propose a timing attack on the GPS identification scheme. We consider implementations of GPS where the running time of the exponentiation (commitment phase) leaks the exponent's Hamming weight, which is typical of a square and multiply algorithm for example. We show that only 800 time measures allow the attacker to find the private key in a few seconds on a PC with a success probability of 80%. Besides its efficiency, two other interesting points in our attack are its resistance to some classical countermeasures against timing attacks, and the fact that it works whether the Chinese Remainder Technique is used or not.

Keywords: Side-Channel Attacks, Timing Attacks, GPS, Identification Schemes

1 Introduction

Timing attacks are certainly less powerful than power or electromagnetic analysis. On the other hand, the very limited equipment they require and the simplicity of the measurements make them much easier to deploy, even for a non-skillful adversary with very limited resources. Moreover, there are situations in which power consumption or electromagnetic radiations cannot be measured while the running time may be obtained, for example by measuring the delay between question and answer [3].

GPS is an identification scheme initially proposed by Girault [7]. It was recently selected in Nessie's portfolio of cryptographic primitives [4]. This protocol was designed for smart cards, allowing fast identification even on low-cost processors. The security of GPS was proven in [11]. The scheme is complete, sound (under the hypothesis that computing discrete logarithms with small exponents modulo $n = pq$ is hard), and statistically zero-knowledge. Careless implementations might nevertheless be subject to side-channel attacks.

Timing attacks against exponentiation schemes have been known for several years, and various countermeasures have been proposed against them. However, since some of these countermeasures are precisely based on randomizing the exponent, one may feel tempted to believe that an algorithm where the exponent is random by nature, and where a different exponent is used for each execution may not be subject to a timing attack. We show in this paper that this is not true.

We propose a timing attack on GPS allowing recovering the prover's private key provided the exponentiation's running time is dependent (in our example, linear) in the exponent's Hamming weight. In our scenario, the attacker impersonates the verifier, and is able to measure precisely the computation time for the commitment step. Apart from this, the attacker has no knowledge of the implementation, such as multiplication algorithm, time needed for an individual multiplication, ...

To our knowledge, none of the previously known timing attacks needed only the leakage of the exponent's Hamming weight ([5] and [12], for example, were based on modular reductions occurring in Montgomery multiplications). As a consequence, several of the countermeasures proposed against timing attacks turn out to be useless in our context.

Lastly, previous timing attacks required big amounts of data to work. Our attack only needs 600 timings to obtain the key with a probability of success of 72% after a few hours of treatment on a single PC, while 1000 timings allow 89% success after a few seconds of treatment. Collecting such a number of timings is easy (we recall that [8] states that the total time of the commitment followed by the answer takes less than 100 milliseconds with a crypto-processor card).

Notations. For any integer b, we denote by $|b|$ its size in bits, and by b_j the j-th bit of b, starting from the least significant bit, i.e. $b = b_{|b|-1} \dots b_0$.
For an interval I, $b \in_u I$ means that b is chosen at random in I.
$l \oplus m$ denotes the addition $l + m$ modulo 2.

2 The GPS Identification Scheme

This section briefly reminds the principle of GPS (see [2] for more details) then focuses on the commitment step.

2.1 Short Description of GPS

An authority generates two strong primes p and q and computes $n = pq$. It also chooses an integer g. GPS authors [8] recommend the value $g = 2$, as it fits the security requirements while allowing good performance. Note that this choice has no effect on our attack.

Three parameters A, B and S are needed. We recall their minimum sizes as recommended in [8].

– S is the size of the (long-term) private key, $|S| \geq 160$
– B is the size of the challenge, $|B| = 16, 32$ or 64

– A is the size of the ephemeral keys, $|A| = |B| + |S| + 64$ or $|A| = |B| + |S| + 80$

We set $E = A + (B - 1)(S - 1)$.
Two cases may occur:

– Each user has his own modulus. In this case, $|n|$ should be at least 1024, and the factors of n, p and q, can be revealed to the prover, allowing her to use the Chinese Remainder Technique to speed-up the commitment phase.
– Several users share the same modulus n. In that case, $|n|$ should be at least 2048.

The prover's private key is a random element $x \in [0, S[$. Her public key is $X = g^{-x} \bmod n$. An identification with GPS proceeds as follows:

1. *Commitment:* The prover picks a random $y \in [0, A[$. She computes $Y = g^y \bmod n$ and sends Y to the verifier.
2. *Challenge:* The verifier sends a random integer $c \in [0, B[$ to the prover.
3. *Response:* The prover checks that $c \in [0, B[$ and computes $z = y + cx$. She sends z to the prover.
4. *Verification:* The verifier checks that $z \in [0, E[$ and $g^z X^c \equiv Y \pmod n$.

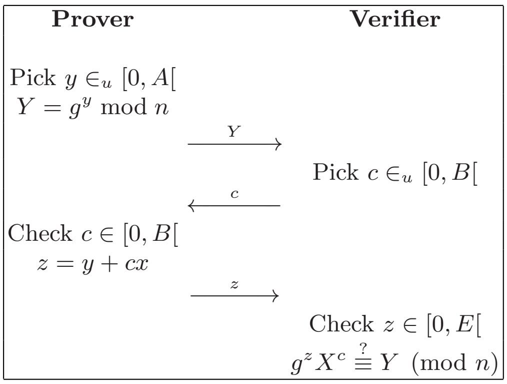

Fig. 1. A round of GPS

2.2 About the Commitment Step

Two methods are possible to manage the commitment issue. The first one consists in pre-computing commitments outside the card and then storing them in the card, using several tricks to save memory space (see [2] for details). This solution is interesting since it can be implemented on a card without crypto-processor, but it becomes a problem for applications that require a lot of identifications to be performed (such as pay-TV, for example): the card has to be reloaded with fresh commitments in a secure way periodically, or can only perform a limited number of identifications (an optimized version with 6000 precomputed commitments needs about 36 kb of memory space).

The second method consists in making the card compute the commitments itself. This solution allows it to perform enough identifications for any application. Two variants of this method are possible: the computation can be done online, i.e. during the identification, or offline. The online variant requires a crypto-processor. In that case, the commitment can be computed in less that 100 milliseconds [8]. In the offline variant, the card computes the commitment before the execution of the identifcation itself. Speed is not an issue anymore, thus no crypto-processor is needed; however, the card needs to be supplied with current – and therefore, for classical smart cards, to be connected to a reader – during this computation.

3 Context and Attack Overview

In this section, we introduce the context of our attack, then draw its main principles.

3.1 Scenario

We assume that the attacker is able to accurately measure the computation time of the commitment. We furthermore assume that this computation time is roughly linear in the Hamming weight of the exponent. This section briefly discusses the realism of these assumptions.

First of all, this will of course only be possible if the commitments are computed by the card, either online or offline. In the offline case, running times may be difficult to obtain by direct measurements, but indirect methods (e.g. based on power consumption) are still possible.

We also assume that the attacker is able to impersonate a honest verifier, which is a weak assumption: as resistance to dishonest verifier attacks is a natural requirement for an identification scheme, there is usually no need to bother about the verifier's identity. In its core version, GPS does not perform any verification step on the verifier before entering the commitment phase.

As far as accuracy of the measurement is concerned, we believe our assumption to be realistic, for example in the context of a smart card (a typical target for side-channel attacks) since smart cards are usually not equipped with an internal clock, but have their clock signal provided by the reader they are put in; in this context, accurate running time is easy to obtain with a rogue reader. To resist side-channel attacks, some modern smart cards get equipped with an internal clock. However, our attack seems to be robust against small imprecision in running time, and could therefore be carried out simply by measuring the elapsed time between startup and commitment.

Finally, the running time will clearly be a linear function of the Hamming weight of the exponent in the case of a classical square and multiply algorithm.

Other methods, such as sliding windows or Walter's division chains [15], are discussed in section 6.

Remark: For the sake of simplicity, we will assume in this description that A is a power of 2 (in fact, it is likely that many implementations choose this value, because it makes generation of random elements in $[0, A[$ easier). Taking this value allows us to use a very simple formula linking the Hamming weight of the ephemeral key y and the probability for one of its bits to be 0. However, a different A makes computations more fastidious, but has no effect on the attack's efficiency.

3.2 Test Platform

To validate our attack, we performed timing measurements using a smart card development kit [10,1]. Although not strictly practical, this scenario should be very close to the reality. Since the emulator is designed to allow implementors to optimize their code before "burning" actual smart cards, its predictions almost perfectly match the smart card's behaviour. We therefore believe physical attack of an actual smart card should not induce much more measurement errors than the ones we encountered here.

3.3 Attack Overview

The core principle of the attack goes as follows: suppose the attacker has obtained $w(y)$, the Hamming weight of some y generated by the prover. Impersonating the verifier and sending the challenge $c = 1$, he has also obtained the prover's response, $z = y + x$.

He starts by attacking the least significant bit of the secret, x_0. If the Hamming weight of y is smaller than half of y's bit length, he may assume that the least significant bit of y is more likely to be equal to 0 than to 1. More precisely, he assumes that the probability $P_0 = P(y_0 = 0)$ is equal to $1 - \frac{w(y)}{|A|}$.

Therefrom, and knowing the value of z, he can easily deduce a probability for the least significant bit of x to be zero.

Of course, this single guess has a non-negligible chance to be wrong. However, if the attacker is able to repeat this experiment over several identifications and compute the average of these probabilities, then this error risk shrinks.

Once he has obtained x_0, the attacker can deduce the actual value of the least significant bit $y_0^{(i)}$ of every message of his sample, which will allow him to attack the second bit x_1. Note that it is not necessary to collect new sets of samples.

The next section takes a deeper look at the attack.

4 How to Recover the Key: Full Details

The attack proceeds in three phases: time measurements, computation of a candidate for the key, key recovering.

4.1 Phase I: Time Measurements

In phase I, the attacker impersonates the honest verifier k times, always sending the same challenge[1] $c = 1$, and collects a list of couples $(t^{(i)}, z^{(i)}), i = 1..k$, where $t^{(i)}$ is the computation time for the commitment of the i-th interaction, and $z^{(i)}$ is the corresponding answer ($z^{(i)} = y^{(i)} + x$).

Assuming the exponentiation's running time is roughly a linear function of the Hamming weight, i.e.

$$t^{(i)} = \alpha \times w(y^{(i)}) + \beta + \epsilon^{(i)},$$

with unknown parameters α and β ($\epsilon^{(i)}$ represents the error), the attacker estimates the Hamming weights of the set of ephemeral keys.

Figure 2 gives a clear idea of how parameters α and β can be estimated. The left graph shows the sorted running times of 80 exponentiations performed on our test platform (with the following parameters: $g = 2$, $|n|$=1024, $A = 2^{240}$ and without CRT) and the right graph shows the corresponding Hamming weights. Linear regression techniques on the left graph immediately provide good estimates for α and β.

These figures might deserve a bit more attention. As we can see, running times are grouped by "steps", and the attacker can safely assume that these "steps" correspond to several exponents having the same Hamming weight, whereas the average height between two "steps" is the time for a modular multiplication (i.e. α). The right graph shows that this estimate is pretty accurate: actual Hamming weight is rarely further than 2 from its estimate.

Remarks:

- We purposely took a small number of samples in order not to overload the figures. In practice, using a much larger set (say, 1000 samples) will of course reduce the error.
- We emphasize that this estimation is possible *whether the Chinese Remainder Technique is used or not.* Without CRT, the exponentiation time leaks information on $w(y)$. But with CRT, the card first computes $g^y \bmod p$ then $g^y \bmod q$ (since $y \ll p, q$, we have $y = y \bmod (p-1) = y \bmod (q-1)$), and the timing leaks information on $2 \times w(y)$.
- Smart card implementations of GPS probably require a constant time between the insertion of the card in the reader and the beginning of the computation of the commitment, meaning that the attacker gets a list $(t^{(i)} + R)$, where R is constant and unknown to the attacker, instead of a list $(t^{(i)})$. Thanks to the linear regression, this has no effect on the attack's efficiency.

[1] Always sending $c = 1$ improves both the precision and the simplicity of the attack, but this is not a necessary condition. For example, the principle of the attack also works when the verifier always sends powers of two as challenges: the pair $(w(y), z = y + 2^v x)$ will leak information on bits $x_{|S|-1} \dots x_v$ of x.

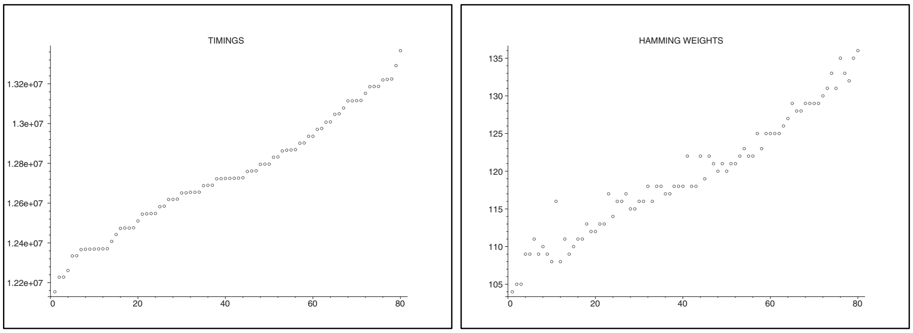

Fig. 2. Sorted timings for 80 samples and corresponding Hamming weights

4.2 Phase II: Computing a Candidate for the Key

In phase II, the attacker uses the collected data $(w^{(i)}, z^{(i)})$ to recover a candidate $\overline{x}$ for the private key.

Remark: Our procedure obviously works independently from the way the attacker obtained the Hamming weights. Although we put ourselves in the context of a timing attack, other methods, such as power or electromagnetic analysis, would work as well.

Estimation of $P(y_j^{(i)} = 0)$. At step j of the procedure, we must estimate $P(y_j^{(i)} = 0)$ for each $i = 1, \dots, k$. A basic estimation would be $P(y_j^{(i)} = 0) = 1 - (w^{(i)}/|A|)$, but we use two tricks to refine it:

- since $y^{(i)}$ is at least 64 bits longer than x, the first bits of $y^{(i)}$ are very likely to be the same as the first bits of $z^{(i)}$. We have $z^{(i)}_{|A|-1} \dots z^{(i)}_{|S|+10} = y^{(i)}_{|A|-1} \dots y^{(i)}_{|S|+10}$ with probability $1 - \frac{1}{2^{10}}$. For simplicity, we will assume in the following that this relationship always holds.
- if we already know $x_{j-1} \dots x_0$, then we can use $z^{(i)}$ to compute $y^{(i)}_{j-1} \dots y^{(i)}_0$.

This leads to the following estimation of $P(y_j^{(i)} = 0)$:

$$P(y_j^{(i)} = 0) = 1 - \frac{w(y^{(i)}_{|S|+9} \cdots y^{(i)}_j)}{|S| + 10 - j}$$

with

$$w(y^{(i)}_{|S|+9} \cdots y^{(i)}_j) = w^{(i)} - w(z^{(i)}_{|A|-1} \cdots z^{(i)}_{|S|+10}) - w(y^{(i)}_{j-1} \cdots y^{(i)}_0)$$

Our procedure will only store k values $w^{(i)}$ corresponding to the weight of the part of $y^{(i)}$ that has not been guessed yet.

Guessing x_0. For each $i = 1, \ldots, k$, we have $z^{(i)} = y^{(i)} + x$. Thus $z_0^{(i)} = y_0^{(i)} \oplus x_0$. Therefore we have $P(x_0 = 0) = P(z_0^{(i)} = y_0^{(i)})$. For each i, we estimate $P(y_0^{(i)} = 0)$, then $P(z_0^{(i)} = y_0^{(i)})$:

$$P(z_0^{(i)} = y_0^{(i)}) = \begin{cases} P(y_0^{(i)} = 0) \text{ if } z_0^{(i)} = 0 \\ 1 - P(y^{(i)} = 0) \text{ if } z_0^{(i)} = 1 \end{cases}$$

Using all the couples $(w^{(i)}, z^{(i)})$, we can use the following estimation:

$$P(x_0 = 0) = \frac{1}{k} \sum_{i=1}^{k} P(z_0^{(i)} = y_0^{(i)})$$

If we get $P(x_0 = 0) > 1/2$, we set $\overline{x}_0 = 0$; else we set $\overline{x}_0 = 1$.

Guessing x_j for $j > 0$. To try to guess x_j for $j > 0$, the situation is a little different, because we now have to deal with a carry: $z_j^{(i)} = y_j^{(i)} \oplus x_j \oplus carry_j^{(i)}$. We use our previous guesses $(\overline{x}_{j-1}, \ldots, \overline{x}_0)$ to guess the carry. As we already explained, we also use $(\overline{x}_{j-1}, \ldots, \overline{x}_0)$ to refine our estimation of $P(y_j^{(i)} = 0)$.

Each couple $(w^{(i)}, z^{(i)})$ leads to an estimation of $P(x_j = 0)$:

$$P(x_j = 0) \simeq P(y_j^{(i)} = z_j^{(i)} \oplus carry_j^{(i)})$$

We take the mean of these estimations for $i = 1, \ldots, k$.

$$P(x_j = 0) = \frac{1}{k} \sum_{i=1}^{k} P(y_j^{(i)} = z_j^{(i)} \oplus carry_j^{(i)})$$

Again, if we get $P(x_j = 0) > 1/2$, we set $\overline{x}_j = 0$; else we set $\overline{x}_j = 1$.

Our procedure *FindCandidate* stores the list $carry^{(i)}, i = 1, \ldots, k$, where $carry^{(i)}$ equals $carry_{j-1}^{(i)}$ at the beginning of procedure *PartialEstimate*(i, j) and $carry_j^{(i)}$ at the end of this procedure.

Remark: If the estimation is wrong at step j, meaning that $\overline{x}_j \neq x_j$, the main consequence is an error on the next carry [2]. It is easy to see that this error will essentially propagate like a carry error in a classical addition, meaning that only a small block of the output bits will be erroneous. Our experiments confirmed this fact.

[2] Actually, a wrong estimation on x_j also induces an error on the updated estimation $w^{(i)}$. This error is small and does not affect the success of our attack.

Algorithm 1 $PartialEstimate(i,j)$: computes the i-th estimation of $P(x_j = 0)$ and the carry $carry_j^{(i)}$

```
$\overline{y}_{j-1}^{(i)} := z_{j-1}^{(i)} \oplus carry_{j-1}^{(i)} \oplus \overline{x}_{j-1}^{(i)}$
$w^{(i)} := w^{(i)} - \overline{y}_{j-1}^{(i)}$
if $\overline{x}_{j-1} + \overline{y}_{j-1}^{(i)} + carry_{j-1}^{(i)} \geq 2$ then
    $carry_j^{(i)} := 1$
else
    $carry_j^{(i)} := 0$
end if
Clear $carry_{j-1}^{(i)}$
Store $carry_j^{(i)}$
$P_j^{(i)} := 1 - (w^{(i)}/(|S| + 10 - j))$
if $z_j^{(i)} \oplus carry_j^{(i)} = 0$ then
    Return $P_j^{(i)}$
else
    Return $1 - P_j^{(i)}$
end if
```

Algorithm 2 $FindCandidate$: computes $\overline{x}$

```
for i=1 to $k$ do
    $w^{(i)} := w^{(i)} - w(z_{|A|-1}^{(i)} \dots z_{|S|+10}^{(i)})$
end for
for $j = 0$ to $|S| - 1$ do
    $P := 0$
    for $i = 1$ to $k$ do
        $P := P + PartialEstimate(i,j)$
    end for
    $P := P/k$
    if $P > 0.5$ then
        $\overline{x}_j := 0$
    else
        $\overline{x}_j := 1$
    end if
    Store $\overline{x}_j$
end for
Return $\overline{x}_{|S|-1} \dots \overline{x}_0$
```

4.3 Phase III: Using the Candidate to Find the Key

At the end of Phase II, the attacker finds a candidate $\overline{x}$. If $g^{-\overline{x}} \equiv X \pmod{n}$, the attack is a success. Even if $\overline{x} \neq x$, the attacker can make an exhaustive search on values x' such that $d(x', \overline{x})$, the Hamming distance between x' and $\overline{x}$, is small, testing whether $g^{-x'} \equiv X \pmod{n}$. The maximum distance such that the attack can succeed obviously depends on the capacity of the attacker; practical values are given in the next section.

5 Practical Results

We chose the following values to perform our tests: $g = 2$, $|n| = 1024$, $A = 2^{240}$, then used the test platform described in section 3.2 to obtain running times. We did not use the Chinese Remainder Technique.

For Phase III, we assumed that the attacker can perform 400 tests per second (which roughly corresponds to computations on a 2 Ghz PC with the GMP libray). This leads to the following classification for the candidates:

1. If $\overline{x} = x$, $\overline{x}$ is already the correct key
2. $d(\overline{x}, x) \leq 2$, the attacker will need less than 16 seconds on average to find the correct key; we say that $\overline{x}$ is a "seconds" candidate
3. $d(\overline{x}, x) \leq 4$, the attacker will need less than 9 hours on average to find the correct key; we say that $\overline{x}$ is a "hours" candidate
4. $d(\overline{x}, x) \leq 5$, the attacker will need less than 12 days on average to find the correct key; we say that $\overline{x}$ is a "days" candidate

Our attack's results are summarized in Table 1.

Table 1. Simulation Results

k (number of samples)	200	400	600	800	1000
Immediate keys	0%	2%	21%	52%	72%
"seconds" candidates	0%	3%	54%	80%	89%
"hours" candidates	0%	6%	72%	94%	97%
"days" candidates	0%	10%	77%	96%	98%
avg. distance $d(\overline{x}, x)$	45.97	16.11	3.88	1.43	0.7

Phase II itself completes in a few seconds. When the number of samples is very small, it may be worth noticing that even if the attack does not produce directly exploitable results, it does anyway reveal substantial information about the key.

6 Countermeasures

Several countermeasures against timing attacks are known today. However, some care must be taken, as they will not all be efficient against our attack.

First of all, most timing attacks known so far exploited the modular reduction occurring in a Montgomery (resp. Barrett, ...) multiplication. Therefore, several countermeasures [6,13,14] typically consisted in removing the time variation in this multiplication. Since our attack is not based on the same property, these countermeasures are pointless.

Another frequently suggested countermeasure consists in randomizing the exponent by adding a random multiple of $\varphi(n)$ to it, a modification that does not affect the final result. However, GPS, that was designed for efficiency, uses a short (typically, between 240 and 304 bits) exponent. Clearly, adding a multiple of $\varphi(n)$ (typically, 1024 or 2048 bits) will imply a very serious performance drawback.

Simple sliding window techniques are probably too basic to make the attack impossible (in short, there is still a correlation between running time and Hamming weight).

Nevertheless, some other countermeasures are possible.

Pre-computed Commitments A natural countermeasure is to use pre-computed commitments. This obviously makes the attack impossible; the advantages and the drawbacks of such a method have already been discussed in section 2.2.

Square and Multiply Always Using dummy multiplications during the exponentiation allows to hide the hamming weight of the ephemeral keys. This countermeasure increases the computation time by about 30%.

MIST algorithm: A much more efficient countermeasure could be the use of Walter's MIST exponentiation algorithm [15]. MIST is not strictly constant-time (the greater the exponent, the longer the division chain). However, the information it could leak is probably limited to the strong bits of the exponent [16], which are already known to any eavesdropper (as they correspond to the strong bits of the answer). Thus MIST, which was designed to resist power analysis for RSA-like systems, i.e. when a same exponent is used many times, seems to fit out needs with GPS, where a new exponent is used for each commitment.

Remark: We were also suggested to modify the pseudo-random number generator, as a possible countermeasure. The most straightforward way to do this is to take $A = 2^a$ and ensure that the outputs $y \in [0, 2^a[$ of the generator are such that $w(y) = a/2$. The corresponding information on individual bits of y is null. However, we believe this countermeasure must be considered with great caution: the security of identification schemes such as GPS is strongly related to the randomness of the commitments, and tampering with "randomness modifications" is always very risky.

7 Conclusion

We proposed a strategy for a side-channel attack on GPS, and showed that it is realistic and efficient in a timing attack context. We believe that several refinements of the attack are still possible to further improve its efficiency. As usual with side-channel attacks – that do not target cryptographic primitives themselves, but rather specific implementations – this paper does not question the security of the GPS primitive. Instead, we showed how straightforward implementations can easily be broken, and pointed out the question of efficient countermeasures.

Acknowledgements. The authors wish to thank Colin Walter, Gaël Hachez and Mathieu Ciet for fruitful discussions and Sylvie Baudine for her permanent English support.

References

1. Cascade (Chip Architecture for Smart CArds and portable intelligent DEvices). Project funded by the European Community, see `http://www.dice.ucl.ac.be/crypto/cascade`.
2. O. Baudron, F. Boudot, P. Bourel, E. Bresson, J. Corbel, L. Frisch, H. Gilbert, M. Girault, L. Goubin, J.-F. Misarsky, P. Nguyen, J. Patarin, D. Pointcheval, J. Stern, and J. Traoré. GPS, an asymetric identification scheme for *on the fly* authentification of low cost smart cards. Submitted to the European NESSIE Project, 2000. Available at `https://www.cryptonessie.org/`.
3. D. Boneh and D. Brumley. Remote timing attacks are practical. Submitted to Usenix Security. Available at `http://crypto.stanford.edu/~dabo/papers/ssl-timing.pdf`, 2003.
4. Nessie consortium. Portfolio of recommended cryptographic primitives. Available at `https://www.cryptonessie.org/deliverables/decision-final.pdf`, 2003.
5. J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A practical implementation of the timing attack. In J.-J. Quisquater and B. Schneier, editors, *Proc. CARDIS 1998, Smart Card Research and Advanced Applications*, LNCS. Springer, 1998.
6. J.F. Dhem. *Design of an efficient public-key cryptographic library for RISC-based smart cards.* PhD thesis, Université catholique de Louvain – UCL Crypto Group – Laboratoire de microélectronique (DICE), May 1998.
7. M. Girault. Self-Certified Public Keys. In D.W. Davies, editor, *Advances in Cryptology - Proceedings of EUROCRYPT 1990*, volume 0547 of *Lecture Notes in Computer Science*, pages 490–497. Springer, 1991.
8. Marc Girault, Guillaume Poupard, and Jacques Stern. Some modes of use of the GPS identification scheme. In *Proceedings of the third Nessie workshop*, November 6–7 2002.
9. P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO '96, Santa Barbara, California*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.
10. Advanced RISC Machines Ltd. *ARM Software Developpment Toolkit version 2.11: User guide.* Advanced RISC Machines Ltd, 1997. Document number: ARM DUI 0040C.
11. G. Poupard and J. Stern. Security Analysis of a Practical *on the fly* Authentification and Signature Generation. In K. Nyberg, editor, *Advances in Cryptology – Proceedings of EUROCRYPT 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 422–436. Springer, 1998.
12. W. Schindler. A timing attack against RSA with the Chinese remainder theorem. In Ç. Koç and C. Paar, editors, *Proc. of Cryptographic Hardware and Embedded Systems (CHES 2000)*, volume 1965 of *LNCS*, pages 109–124. Springer, 2000.

13. Colin D. Walter. Montgomery Exponentiation Needs no Final Subtractions. *Electronics Letters*, 35(21):1831–1832, October 1999.
14. Colin D. Walter. Montgomery's Multiplication Technique: How to Make It Smaller and Faster. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES '99*, volume 1717 of *Lectures Notes in Computer Science (LNCS)*, pages 80–93. Springer-Verlag, August 1999.
15. Colin D. Walter. MIST: An efficient, randomized exponentiation algorithm for resisting power analysis. In *Topics in Cryptology – CT-RSA 2002*, Lecture Notes in Computer Science. Springer, April 2002.
16. Colin D. Walter. Seeing through MIST given a small fraction of an RSA private key. In M. Joye, editor, *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *Lectures Notes in Computer Science (LNCS)*. Springer, 2003.