Microcontroller Implementation of Simultaneous Protections Against Observation and Perturbation Attacks for ECC

Audrey Lucas[1] and Arnaud Tisserand[2]
[1]*CNRS, IRISA UMR 6074, INRIA Centre Rennes - Bretagne Atlantique and Univ Rennes, Lannion, France*
[2]*CNRS, Lab-STICC UMR 6285 and University South Britany, Lorient, France*

Keywords: Elliptic Curve Cryptography, Side Channel Attack, Fault Injection Attack, Protection, Countermeasure.

Abstract: Scalar multiplication is the main operation in elliptic curve cryptography. In embedded systems, it is vulnerable to both observation and perturbation attacks. Most of protections only target one of these two types of attacks. Unfortunately, many protections against one type of attack may reduce the protection against the other one. In this paper, we simultaneously deal with protections against both types of attacks. Two countermeasures are presented for scalar multiplication and implemented on a Cortex-M0 microcontroller. The first one protects finite field operations over point coordinates. The second one protects the scalar (or key) bits.

1 INTRODUCTION

Elliptic curve cryptography (ECC) is promoted for providing public-key cryptography (PKC) support in embedded systems due to its smaller cost, *e.g.* silicon area and energy, and better performances than RSA (Cohen and Frey, 2005; Hankerson et al., 2004).

Embedded systems are widespread in our society, thus their protection against various types of attacks is essential. Due to their proximity with other users, potentially malicious ones, embedded circuits are vulnerable to physical attacks. In this paper, we focus on side channel attacks (SCAs) and fault attacks (FAs). The first ones, use observations of physical parameters, such as computation timings or power consumption, which are analyzed using statistical tools to deduce links between physical measurements and internal secret values. The second ones, use perturbations of the circuit such as variations of the power supply or electromagnetic radiations to inject fault(s) during algorithms execution. These faults are exploited to deduce internal secret values.

Numerous countermeasures exist against SCAs and FAs at various levels: mathematics, algorithm, architecture, circuit. Most of these protections only target one type of attack. For example, uniformization schemes are efficient against SCAs but not for FAs. Some error correcting codes can be used against FAs but not for SCAs. Unfortunately, many protections, against one type of attack, leave or may make the implementation vulnerable to the other type of attacks.

In this work, we simultaneously deal with protections against both types of attacks. We propose two countermeasures, developed onto specific curves, for scalar multiplication (SM) in ECC. They are probably adaptable onto other curves. The first one protects finite field operations over point coordinates. The second one protects the scalar itself during SM.

Our paper is organized as follows. Sections 2, 3 and 4 respectively recall background elements on ECC, SCAs/FAs attacks and ECC attacks and protections. Our two propositions are presented in Section 5. Section 6 reports implementation results on Cortex-M0 microcontrollers and the μNaCl library (Düll et al.,).

2 BACKGROUND ON ECC

ECC (Hankerson et al., 2004; Cohen and Frey, 2005) is a PKC based on elliptic curves (ECs).

In the case of prime fields $\mathbb{F}_p$, short Weierstrass curves form E_{WS} and Montgomery (Montgomery, 1987) curves E_M are defined, with $a,b \in \mathbb{F}_p$ and specific conditions on a,b (see books (Hankerson et al., 2004; Cohen and Frey, 2005)), respectively by:

$$\begin{aligned} E_{WS} : y^2 &= x^3 + ax + b, \\ E_M : by^2 &= x^3 + ax^2 + x. \end{aligned} \tag{1}$$

In this paper, we only consider these curves onto prime fields $\mathbb{F}_p$.

Lucas, A. and Tisserand, A.
Microcontroller Implementation of Simultaneous Protections Against Observation and Perturbation Attacks for ECC.
DOI: 10.5220/0006884604040411
In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications (ICETE 2018) - Volume 2: SECRYPT*, pages 404-411
ISBN: 978-989-758-319-3

The most critical operation in ECC is the scalar multiplication (SM) $[k]P$ between a curve point P and a scalar k (either the public or private key). When k is private, it must be protected. SM can be performed by various algorithms based on point addition (`ADD`) and point doubling (`DBL`) operations at curve level. When `ADD` and `DBL` have different behaviors, their differences can be a leakage source in observation attacks. The easiest way to perform SM is the `double and add` (DA) algorithm 1. In case of E_M, the `Montgomery ladder` (ML) algorithm 2. is commonly used.

Algorithm 1: SM - `double and add`.

```
   Input: $P$ and $k = (k_{m-1}, \dots, k_0)_2$
   Result: $[k] \cdot P$
1  $T \leftarrow O$
2  for $i = m-1$ to 0 do
3      $T \leftarrow 2 \cdot T$                  DBL
4      if $k_i = 1$ then
5          $T \leftarrow T + P$              ADD
6  return $T$
```

Algorithm 2: SM - `Montgomery ladder`.

```
   Input: $P$ and $k = (k_{m-1}, \dots, k_0)_2$
   Result: $[k] \cdot P$
1  $T_1 \leftarrow O$,    $T_2 \leftarrow P$
2  for $i = m-1$ to 0 do
3      if $k_i = 1$ then
4          $T_1 \leftarrow T_1 + T_2$          ADD
5          $T_2 \leftarrow 2 \cdot T_2$          DBL
6      else
7          $T_2 \leftarrow T_1 + T_2$          ADD
8          $T_1 \leftarrow 2 \cdot T_1$          DBL
9  return $T_1$
```

In order to perform $T_1 + T_2$, the x coordinate of $T_1 - T_2$ can be known. During ML $[k]P$ internal iterations, $T_1 - T_2$ is always equal to the base point P.

Several `ADD` and `DBL` formulas for different curves are available on the EFD website (Bernstein and Lange,).

3 BACKGROUND ON PHYSICAL ATTACKS

Embedded systems have to face attacks at both logical and physical levels. Logical attacks target mathematical properties of cryptosystems, networking protocols, weak software implementations, *etc*. For instance, very efficient factorization algorithms and parallel implementations have been used against RSA 768 bits a few years ago. In this work, we do not consider these attacks. Physical attacks are totally different from logical ones and require specific protections. Embedded systems have to be protected against them since circuits in charge of security tasks can be very close to the attackers. Typical physical attacks include: reverse engineering, observation (or SCAs) and perturbation (or FAs). In this paper, we only consider SCAs and FAs. It is possible to combine them such as (Roche et al., 2011).

3.1 SCAs and Countermeasures

SCAs observe physical parameters such as timings (Kocher, 1996), power consummation (Mangard et al., 2007) or electromagnetic radiations (EM) (Agrawal et al., 2002) at run time. They exploit potential correlations between measurements of physical parameter(s) and some secret data manipulated during execution.

SCAs are often decomposed into two types. On one hand, simple power analysis (SPA) uses a single trace of power measurements. For instance, algorithm 1 is vulnerable to SPA. On the other hand, various attacks use multiple traces and statistical tools. For instance, differential power analysis (DPA) (Kocher et al., 2011) uses difference of averages and correlation power analysis (CPA) (Brier et al., 2004) uses Pearson correlation. Both simple and differential-like attacks exist for other physical parameters (*e.g.* EM).

For SCA protection, one must avoid, or strongly reduce, dependencies between secret values and observable variations of the physical parameter(s). A first type of protection is denoted uniformization: operations sequences must be indistinguishable whatever the actual secret bits manipulated in the circuit. Useless operations can be added to uniformize some algorithms. A second type of SCA protection is denoted randomization: a random activity generates a scramble in the measurements. Statistic tools consider this random activity as data and their results are disturbed. For instance, random useless operations or random masks can be added. Many variations and combinations of uniformization and randomization protections have been proposed.

3.2 FAs and Countermeasures

Lasers, electromagnetic radiations, variations in supply voltage or circuit temperature, glitches in clock signals are used to disturb the circuit by injecting fault(s) during algorithm execution (Bar-El et al., 2006; Verbauwhede et al., 2011). These faults can be temporary or permanent and equivalent at logical level to a bit flip, bit set, bit reset or bit stuck-at (on single or multiple bits).

FAs exploit some unspecified circuit behavior, directly or not, in order to deduce the secret. For instance, they can use differences between faulty and correct outputs thanks to differential fault analysis (DFA) (Biham and Shamir, 1997).

Safe-error analysis (SEA) (Yen and Joye, 2000) checks if the injected fault has an impact on the final result. By determining whether a corrupted data was effectively used or not, SEA is very efficient against SCA protections based on useless/dummy operations.

Attackers can produce fault(s) on data, control or external memory. In this paper, we only consider faults on data since we target software implementations with on-chip memory.

Two types of protections exist against FAs: detection and correction schemes. Detection schemes allow various policy solutions when an attack occurs: execution stop and re-run, algorithm change, erasing/destroying secret values, *etc*. Detection can be achieved at various levels: in hardware using intrusion sensors, at algorithm using redundant computations (spatial and/or temporal) or data integrity checks for instance. Correction schemes use methods performing the expected operations even in presence of faults (e.g. use of majority voters). In this paper, we only consider detection schemes.

4 ATTACKS AND PROTECTIONS ON ECC

In this section, several SCAs, FAs and related protections for ECC are recalled. Attacks objective is to recover the secret scalar/key k from execution(s) of the scalar multiplication $Q = [k]P$.

4.1 SCAs on ECC and Protections

During SM, each sequence of curve-level operations depends on the actual scalar bits. If `ADD` and `DBL` operations can be distinguished (through physical measurements) and DA algorithm is used, then SM is vulnerable to SPA. Indeed, a 1 key bit generates a `DBL` followed by an `ADD`, while a 0 key bit only generates a `DBL`. If partial traces for `ADD` and `DBL` are different (even with a few differences), an attacker is able to distinguish what operation is made and then recover the key bits from the trace as illustrated in Figure 1.

Several other SCAs on ECC exist including timings, DPA, zero-value point attacks (Akishita and Takagi, 2003) or doubling attacks (Fouque and

1		0	0	0	1	
DBL	ADD	DBL	DBL	DBL	DBL	ADD

Figure 1: Basic DA algorithm.

Valette, 2003). In practice, some randomization schemes can be applied against DPA-like attacks in many protocols. Then SPA-like ones are considered as a major threat in ECC. In this paper, we only deal with SPA-like attacks.

Among SCA protections uniformization and randomization have been widely used in ECC.

Among uniformization countermeasures, `double and add always` (DAA) (Coron, 1999) and ML are typical SPA protections. The DAA algorithm is similar to DA where a useless `ADD` is added when the key bit is zero. This is good for SPA protection but very bad for SEA ones (injecting a fault during the useless `ADD` has not impact on the output, then the attacker knows that the operation was a dummy one and the corresponding key bit was 0).

ML is widely used in practice since it is SPA and SEA resistant. The same operations sequence is made regardless of the key bits. Attackers cannot distinguish `ADD` and `DBL` patterns. Furthermore, all intermediate computations impact the final result.

Among randomization countermeasures, scalar randomization and point blinding protections have been proposed against DPA (Coron, 1999). Scalar randomization consists in performing $[k]P = [k + r \cdot \lambda]P$ where r is the order of E and λ is a random number. Point blinding performs $[k]P = [k](P+R) - [k]R$ instead of $[k]P$, where R is a random point. Other randomization countermeasures use projective coordinates. Before each SM execution, P coordinates are randomized thanks to the multiplication by a random number λ, so $P = (\lambda x_P, \lambda y_P, \lambda z_P)$. This new P is employed during SM.

4.2 FAs on ECC and Protections

Attackers can inject faults on several types of data during SM: curves parameters, scalar, field representation (Ciet and Joye, 2005), base point (Biehl et al., 2000) and current point (Blömer et al., 2006). The attacker aims DFA or transferring the ECDLP (discrete logarithm problem) onto a weaker curve. Commonly, the transfer is possible since b parameter in curve equation 1 is unused during SM. Below, two attack examples with different targets are recalled.

In (Biehl et al., 2000), the base point $\widetilde{P}$ belongs to $\widetilde{E}_{SW}$ instead of E_{SW}. Curve $\widetilde{E}_{SW}$ has a smaller order than E_{SW} and it is defined by:

$$\widetilde{b} = y^2 - x^3 - ax. \qquad (2)$$

As $[k]P = [k]\widetilde{P}$, the DLP is transferred onto subgroup of smaller order and an attacker recovers k mod $ord(\widetilde{P})$. By reiterating with several other $\widetilde{P}$, the attacker recovers the key value thanks to the Chinese remainder theorem (CRT).

In (Bao et al., 1997), the attack proposed on RSA can also apply to ECC. The fault is supposed one bit flip located on the random key bit at index j. Let $\widetilde{Q}$ the faulty SM result, then Q and $\widetilde{Q}$ can be written as:

$$\begin{aligned} \widetilde{Q} &= [\widetilde{k}]P = \sum_{i=0}^{j-1} k_i 2^i P + \widetilde{k}_j 2^j P + \sum_{i=j+1}^{m-1} k_i 2^i P \\ Q &= [k]P = \sum_{i=0}^{m-1} k_i 2^i P \end{aligned} \tag{3}$$

Knowing both Q and $\widetilde{Q}$, one can compute $Q - \widetilde{Q}$ which helps to deduce the key bit k_j. Indeed, if $Q - \widetilde{Q} = -2^j P$ then $k_j = 0$ and if $Q - \widetilde{Q} = 2^j P$ then $k_j = 1$. Finally, the attacker recovers the full key iteratively for the other j ranks.

Most FAs on ECC are defeated using point verification (PV) (Biehl et al., 2000) at various steps of SM. For instance, PV ensures that final or current point belongs to the curve by injecting their coordinates into the curve equation. Thus, PV protects against FA which targets the curve parameters and the point P of SM $[k]P$.

In case of Montgomery curve, the y coordinate is unused during SM, and PV is equivalent to verify if $C = \frac{x^3+ax^2+x}{b}$ is a square with Legendre symbol (LS). As a reminder LS is computed by $C^{\frac{p-1}{2}}$, and if $C \equiv 0 \bmod p$ (p is prime) then, LS equals to 0. In other case, LS equals to 1 if C is square modulo p and -1 else. However, PV is ineffective against FA targeting scalar bits k_js.

5 PROPOSED PROTECTIONS

As protections against one type of attack may weaken the implementation against the other one, simultaneously dealing with both SCAs and FAs is important but it can be tricky. For instance, basic uniformization schemes using dummy operations for SCA protection may be weak against SEA. Or in case of FA protection, adding redundancy checks must not reduce the robustness against SCAs (for instance by breaking the uniform behavior). In most of the literature, FAs and SCAs are considered independently.

We propose two combined countermeasures for protecting SM simultaneously against both SPA-like attacks and major FAs. Standard protections against DPA-like attacks can be used on top of our countermeasures. Our first countermeasure is an extension of PV proposed for protection against FAs but we added uniformization for SPA protection. Our second countermeasure, called iteration counter, protects the scalar bits against FAs with a uniform behavior for SPA protection. In this section, we describe these countermeasures for Weierstrass curves (with both jacobian and projective coordinates) and Montgomery curves (with XZ coordinates).

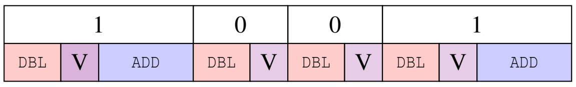

Figure 2: Point verification in DA.

Their cost will be first evaluated in terms of the number of $\mathbb{F}_p$ operations: multiplication M, square S and addition/subtraction A. In Section 6, detailed comparisons will be reported for microcontroller implementation.

5.1 Point Verification (PV)

PV injects point coordinates into the curve equation to verify that the checked point effectively belongs to the curve. PV can be integrated in SM at different periods leading to various trade-offs between security and performance. For low cost protection, PV can be performed at beginning and end of SM. Then, a FA on an intermediate point is detected very late. For earlier detection, but with a higher run time, PV can be performed every d iterations or randomly during SM. The strongest protection is obtained for $d = 1$ where PV is performed at each SM iteration. In this paper, we denote ℓ the number of executed PVs during one SM ($1 \leq \ell \leq m+1$).

Care must be taken when applying PV to not weakening the implementation against SCAs. For instance, using PV after DBL operations is safer (they do not depend on k bits).

For our first countermeasure, we explore and modify PV to ensure a uniform behavior against SPA-like attacks. The sequence of $\mathbb{F}_p$ operations added for PV, denoted V, can be used to ”fill” the differences between ADD and DBL as illustrated in Figure 2. We modified the complete computations to ensure the exact same behavior, *i.e.* same sequence of $\mathbb{F}_p$ operations, for both ADD and DBL+V to make our SM uniform (against SPA).

5.1.1 Uniform PV on Weierstrass Curves

We first present uniformization of SM with PV for projective and jacobian coordinates where ADD is more complex than DBL. We include PV in DBL, denoted DBL+V, to ensure a uniform behavior of SM.

Weierstrass curves onto $\mathbb{F}_p$ in projective coordinates are defined below with $a = -3$:

$$E_{WP} : y^2 z = x^3 + axz^2 + bz^3. \quad (4)$$

Multiplication by b, denoted $\mathtt{M}_b$, can be implemented with a generic multiplication or additions depending on b value (*e.g.* sparse decomposition).

Table 1 reports the cost of curve operations and verification `V`. Obviously, `V` cost is too small to directly uniformize SM. We add operations in `V` by multiplying by y such that:

$$\mathtt{V} : y^3 z = x^3 y + axyz^2 + byz^3. \quad (5)$$

After factorization of some operations and add of operation $\mathtt{M}_b$ to `ADD` (this is not a dummy operation, see the source code), the new cost for `DBL+V` is equal to the `ADD` cost $(11\mathtt{M} + 6\mathtt{S} + 18\mathtt{A} + 1\mathtt{M}_b)$. The final overhead for our uniform SM is $6\ell\mathtt{M} + 4\ell\mathtt{A} + 2\ell\mathtt{M}_b$.

Table 1: Operations costs for Weierstrass curves.

	Projective	Jacobian
`ADD`	11M+6S+18A	11M+ 5S+13A
`DBL`	5M+6S+14A	3M+ 6S+ 15A
Basic `V`	4M+3S+5A+1M$_b$	1M+6A+1M$_b$
`V`	6M+4A+2M$_b$	7M+ 7A+2M$_b$

For jacobian coordinates, we first transform `DBL` (remove 1 `S` and add 1 `M`):

$$\begin{array}{rcl} xx & = & x_1^2 \\ yy & = & y_1^2 \\ t_0 & = & x_1 \cdot yy \\ t_1 & = & t_0 + t_0 \\ s & = & t_1 + t_1 \\ t_2 & = & xx + xx \\ t_3 & = & t_2 + t_2 \end{array} \Rightarrow \begin{array}{rcl} yy & = & y_1^2 \\ t_0 & = & x_1 + x_1 \\ t_1 & = & t_0 + x_1 \\ t_2 & = & t_0 + t_0 \\ s & = & t_2 \cdot yy \\ t_3 & = & t_1 \cdot x_1 \end{array} \quad (6)$$

Then, the verification equation is transformed thanks to a multiplication by the z coordinate:

$$\mathtt{V} : y^2 z = x^3 z + axz^5 + bz^7 \quad (7)$$

After integration into `DBL` and some factorizations, our uniform SM overhead is $7\ell\mathtt{M} + 7\ell\mathtt{A} + 2\mathtt{M}_b$.

Table 2: Overheads.

	Projective	Jacobian
PV	$6\ell\mathtt{M} + 4\ell\mathtt{A} + 2\ell\mathtt{M}_b$	$7\ell\mathtt{M} + 7\ell\mathtt{A} + 2\ell\mathtt{M}_b$
`DAA`	$\simeq 6.5\ell\mathtt{M} + 3\ell\mathtt{S} + 9\ell\mathtt{A}$	$\simeq 6.5\ell\mathtt{M} + 2.5\ell\mathtt{S} + 7.5\ell\mathtt{A}$

Despite `ADD` and `DBL` have the same cost, their behaviors are distinguishable. Then, we reschedule the operations sequences of `ADD` and `DBL+V` to ensure the exact same behavior.

We compare our uniform PV with DAA in Table 2. Our uniform SM with PV protects against both SPA and FA (only SPA for DAA) and has a smaller cost than DAA.

Obviously, one can use a smaller security level with less frequent PVs (with $d > 1$) leading to smaller overheads (while $d = 1$ for DAA).

5.1.2 Uniform PV on Montgomery Curves

Direct PV is too expensive for Montgomery curves using XZ coordinates. The unused y coordinate forces to check current point using LS. Furthermore, Montgomery curves with XZ coordinates and ML are more efficient than Weierstrass curves.

Our proposition takes advantage of a constant inside ML Algorithm 2. Indeed, $T_2 - T_1$ is always equal to P at each iteration. If point T_1 is faulted, it becomes $\widetilde{T_1}$ and $T_2 - \widetilde{T_1} \neq P$. The computation $T_2 - T_1$ with addition formulas is possible at each iteration since $T_1 + T_2$ is also computed.

$T_1 + T_2 = (x_3, z_3)$ is performed using the x coordinate of P denoted x_P. After, $T_2 - T_1$ is made with x_3. A first verification consist in comparing between result of $T_2 - T_1$ and P. This verification does not detect attacks since the x_3 must be normalized by z_3. Then, a final test is equivalent to $x_3(1 - z_P) = 0$ which is always true since $z_P = 1$.

To solve this problem, SM is modified. Instead to deal with one bit, the new `ladderStep`, denote `ML_V`, deals with simultaneously two bits (the iterations number is halved). Indeed, the curve operations are performed as in the algorithm 3. The variable T_6 is the new T_2 after `ML_V`. If $k_i \neq k_{i+1}$ then, T_5 replaces T_1. Else T_1 is replaced by T_7. In order to perform PV, we note that $T_6 - P = T_7$ and $T_6 + P = T_5$. As T_5 and T_7 are calculated earlier, $T_8 = T_6 \pm P$ can be performed with the x coordinate of the new T_1 (T_5 or T_7).

Algorithm 3: `ML_V`.

```
   Input: $T_1, T_2, xor \leftarrow k_i \oplus k_{i+1}$
1  $T_3 \leftarrow 2T_1$
2  $T_4 \leftarrow T_1 + T_2$
3  $T_5 \leftarrow 2T_4$          new $T_1$, if $xor = 1$
4  $T_6 \leftarrow T_3 + T_4$     new $T_2$
5  $T_7 \leftarrow 2T_3$          new $T_1$, if $xor = 0$
6  $T_8 \leftarrow T6 + P$        use x of new $T_1$
7  if $xor = 1$ then
8      $T_8 = T_7$
9  else
10     $T_8 = T_5$
```

If a fault is performed during SM then, the equality between T_8 and T_7 ($xor = 1$) and between T_8 and T_5 ($xor = 0$) is wrong.

The cost of the original `LadderStep` is $5\mathtt{M} + 4\mathtt{S} + 8\mathtt{A} + 1\mathtt{M}_a$. As `ML_V` deals with two key bits instead of one, its overhead equals to $8\mathtt{M}+4\mathtt{S}+7\mathtt{A}+\mathtt{M}_a$. Thus, for one SM the overhead is $4\ell\mathtt{M}+2\ell\mathtt{S}+4.5\ell\mathtt{A}+\frac{\ell}{2}\mathtt{M}_a$.

The verification in `ML_V` is faster than verifying if x^3+ax^2+x is a square using LS. Nevertheless, this PV does not ensure that P belongs to curve. Thus, the first attack from Section 4 is possible. To avoid this, the y coordinate is kept and a basic PV is performed at the beginning of SM. In latter steps of SM, the y coordinate can be removed without security reduction. The cost of the beginning curve equation computation is $1M+2S+3A+\mathtt{M}_a$. Finally, our uniform PV simultaneously protects intermediate point and curve parameters against major FAs and SPA.

As ML is uniform, if late detection is acceptable, one can only use PV at beginning and end of SM for a very low cost.

5.2 Iteration Counter (`IC`)

During SM, key bits manipulations are very short and have a different behavior compared to field operations. But injecting faults during them is also possible (see Sec. 4). PV only protects curve parameters and verified points against FAs but not key bits (PV is verified with faulted key bits).

In order to protect all scalar bits against FAs, we propose the iteration counter countermeasure. It ensures that the executed SM iterations effectively correspond to the actual key k even in presence of FAs with a uniform behavior.

A naive solution is a check sum which counts the Hamming weight of k. Nevertheless, this idea is not sufficient when attackers can flip two key bits at different indexes ($i \neq i'$).

Another solution is to count `ADD`s using a weight depending on the iteration index i. When $k_i=1$, index i is added to a register *reg* (remember that i is small). Attackers have to forge multiple bit flips according to interesting values of i, which is very unlikely. Then the overwhelming majority of faults in k are detected. Unfortunately, when $k_i=0$, *reg* is not modified leading to a small but measurable activity drop. This second solution is good against FAs but not sufficient against SPA.

Our final solution consists in splitting *reg* into 4 registers $r_1,\dots,r_4$ as illustrated in Figure 3. Thanks to these registers, the `cswap` function (used during ML) can switch both the `IC` registers and current point coordinates according to key bits at each ML iteration.

When $k_i \neq k_{i-1}$, `cswap` switches (r_1, r_2) with (r_3, r_4). Regardless of key bits, random values are added to r_3 and r_4. If $i\%2=0$, i is added to first part of r_1. Else, i is added to first part of r_2.

At the end of the SM, r_1 and r_2 are shifted and added. This result is compared to a reference value. This new version, denoted `ICC`, costs 1 swap, 5 small integer additions, 2 shifts and $2m$ small random number generations.

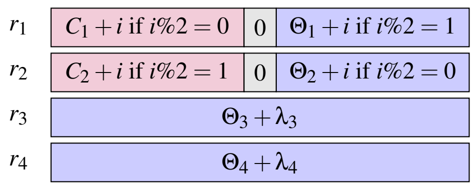

Figure 3: `ICC` split on registers during SM iterations.

`ICC` detects bit flip attacks and is resistant to SPA. But it does not detect bit set or bit reset faults (the bits are forced to 1 and 0 respectively) used in some SEAs. Indeed, if the i-th bit is set to 1 and actually $k_i=1$, this "non-modification" is not detected. But SEAs can be avoided using masking schemes such as (Coron, 1999).

5.3 Fault Detection Policy

Protection against SCAs is based on good properties of the algorithms and implementations without detection at run time. But fault detection can be an active process. Several detection policies can be applied at run time: stopping the execution, erasing the secret data, re-computing with the same or another algorithm, continuing computations with a random key. The policy choice depends on the application and related threats.

In this work, we implemented the continuation using a random scalar. A random scalar k_r is generated before SM. If an attack is detected, k_r is used instead of k as soon as possible. The cost of this additional protection is the random generation of m bits (*i.e.* k length) and $k \leftrightarrow k_r$ swap when a detection occurs.

6 IMPLEMENTATION RESULTS

We implemented our countermeasures on a 32-bit Cortex-M0 microcontroller (STM32F0 Discovery board) and the μNaCl library (Düll et al.,) at 128-bit security level. This is a variant of the NaCl (Daniel J. Bernstein and Schwabe,) library where the Bernstein curve (Bernstein, 2006) (E_M with $a=48662$, $b=1$ and $p=2^{255}-19$) is implemented thanks to ML and XZ coordinates.

The main SM variable is `state` composed of: coordinates of points T_1 and T_2, x_P coordinate, scalar, previous bit and downwards counter.

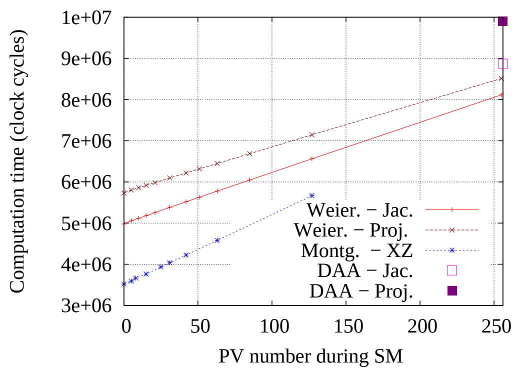

Figure 4: Clock cycles depending on PV numbers.

The SM loop on key bits is handled by the `cswap` which swaps T_1 and T_2 when the current key bit is different from the last one. After `cswap`, a `LadderStep` is performed on `state`.

We use the structure and parameters defined in μNaCl for SM with ML onto Montgomery curves. In case of Weierstrass curves, we use the same base field with the NIST parameters and y-coordinate is added to `state`.

In order to implement `ICC`, the variable `Reg` is created to hold r_1, r_2, r_3 and r_4. Moreover, the small random numbers are generated with random number generator of μNaCl. Before performing `ICC` computations, the `swap` function is used on `Reg` at each iteration.

Figure 5 illustrates the overhead of PV during the SM with a 256-bit scalar. The practical overhead is larger than theoretical overhead. The original library was optimized to fill the processors registers efficiently. When PV is added more memory pressure leads to slower execution.

Cost increases linearly with the number of PVs, this leads to trade-offs between the detection level and the performance. The PV overhead for Montgomery curves is 62% in worst case against only 2.3% when PV is only used the beginning and the end of SM. Similar observations can be made for Weierstrass curves. Nevertheless, overheads of uniform algorithm (the worst case) are smaller than DAA. In addition to the number of clock cycles, the overhead of size code and intermediate RAM are reported in Table 3.

7 CONCLUSION

We proposed two ECC countermeasures combined simultaneous protection against SCAs and FAs. They protect field operations on point coordinates and the scalar bits against both major fault and SPA-like attacks with various security *vs.* performance trade-offs. They have been implemented for short Weierstrass and Montgomery curves on a 32-bit microcontroller.

A uniform PV in Weierstrass curves was proposed. It leads to faster SM than DAA with early detection of FAs (at each SM iteration if needed) and protection against SPA-like attacks. For Montgomery curves, a PV was proposed against FAs (with a uniform behavior). For protecting scalar bits against FAs, a specific countermeasure called iteration counter was proposed. It is low cost and robust to SPA-like attacks.

Our code will be distributed as open source software. Future works will focus on new randomization schemes and other types of ECs and point coordinates.

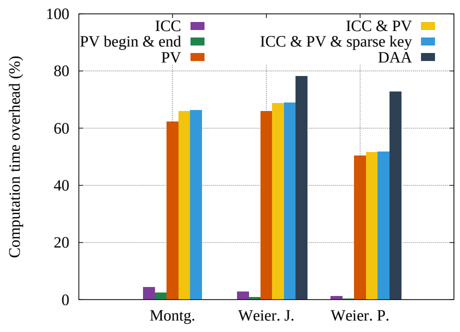

Figure 5: Overhead depending on type of protection.

Table 3: Experimental results overhaed.

	protection type	code size	RAM size
Montg. XZ	`ICC`	2.7%	2.8%
	PV end	10.6%	11%
	PV	12.6%	13.2%
	PV+`ICC`+answer	17.3%	16.9%
Weier. Jac.	`ICC`	2.5%	2.6%
	PV	2.9%	2.6%
	PV begin+end	3.9%	4%
	PV+`ICC`+answer	5.6%	5.8%
	DAA	0.4%	0.4%
Weier. Proj.	`ICC`	2.3%	2.4%
	PV	2.1%	2.2%
	PV begin+end	2.1%	2.2%
	PV+`ICC`+answer	4.8%	5%
	DAA	0.4%	0.2%

REFERENCES

Agrawal, D., Archambeault, B., Rao, J. R., and Rohatgi, P. (2002). The EM Side-Channel(s). In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 29–45.

Akishita, T. and Takagi, T. (2003). Zero-Value Point Attacks on Elliptic Curve Cryptosystem. In *Proc. Information Security - ISC*, pages 218–233.

Bao, F., Deng, R. H., Han, Y., and more authors (1997). Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults. In *Proc. Security Protocols*, pages 115–124.

Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., and Whelan, C. (2006). The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382.

Bernstein, D. J. (2006). Curve25519: New Diffie-Hellman Speed Records. In *Proc. Public Key Cryptography - PKC*, pages 207–228.

Bernstein, D. J. and Lange, T. Explicit-formulas database. `http://hyperelliptic.org/EFD/`.

Biehl, I., Meyer, B., and Müller, V. (2000). Differential Fault Attacks on Elliptic Curve Cryptosystems. In *Proc. Advances in Cryptology - CRYPTO*, pages 131–146.

Biham, E. and Shamir, A. (1997). Differential Fault Analysis of Secret Key Cryptosystems. In *Proc. Advances in Cryptology*, pages 513–525.

Blömer, J., Otto, M., and Seifert, J. (2006). Sign Change Fault Attacks on Elliptic Curve Cryptosystems. In *Proc. Fault Diagnosis and Tolerance in Cryptography - FDTC*, pages 36–52.

Brier, E., Clavier, C., and Olivier, F. (2004). Correlation Power Analysis with a Leakage Model. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 16–29.

Ciet, M. and Joye, M. (2005). Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. *Des. Codes Cryptography*, 36(1):33–43.

Cohen, H. and Frey, G., editors (2005). *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Discrete Maths and Applications. Chapman & Hall/CRC.

Coron, J. (1999). Resistance Against Differential Power Analysis For Elliptic Curve Cryptosystems. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 292–302.

Daniel J. Bernstein, T. L. and Schwabe, P. μNaCl library. `https://nacl.cr.yp.to/`.

Düll, M., Haase, B., and Sánchez, A. H. μNaCl library. `http://munacl.cryptojedi.org/index.shtml`.

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

Hankerson, D., Menezes, A., and Vanstone, S. (2004). *Guide to Elliptic Curve Cryptography*. Springer.

Kocher, P. (1996). Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proc. Advances in Cryptology - CRYPTO*, pages 104–113.

Kocher, P., Jaffe, J., Jun, B., and Rohatgi, P. (2011). Introduction to differential power analysis. *J. Cryptographic Engineering*, 1(1):5–27.

Mangard, S., Oswald, E., and Popp, T. (2007). *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer.

Montgomery, P. L. (1987). Speeding the pollar and elliptic curves methods of factorisation. *Mathematics of Computation*, 48(177):243–264.

Roche, T., Lomné, V., and Khalfallah, K. (2011). Combined Fault and Side-Channel Attack on Protected Implementations of AES. In *Proc. Smart Card Research and Advanced Applications -CARDIS*, pages 65–83.

Verbauwhede, I., Karaklajic, D., and Schmidt, J. (2011). The Fault Attack Jungle - A Classification Model to Guide You. In *Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 3–8.

Yen, S. and Joye, M. (2000). Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Trans. Computers*, 49(9):967–970.