Error Detection for Borrow-Save Adders Dedicated to ECC Unit

Julien Francq, Jean-Baptiste Rigaud, Pascal Manet, Assia Tria, Arnaud Tisserand

2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography

Error Detection for Borrow-Save Adders Dedicated to ECC Unit

Julien Francq[1,3], Jean-Baptiste Rigaud[1],
Pascal Manet[2], Assia Tria[2]
[1]École des Mines de Saint-Étienne, [2]CEA-LETI
[1,2]Centre Microélectronique de Provence G. Charpak
Laboratoire SESAM
880 Avenue de Mimet, 13120 Gardanne, France
[1]{name@emse.fr}, [2]{surname.name@cea.fr}

Arnaud Tisserand[3]
[3]LIRMM, CNRS–Univ. Montpellier 2
161 rue Ada,
34392 Montpellier Cedex 05, France
[3]{surname.name@lirmm.fr}

Abstract

Differential Fault Analysis (DFA) is a real threat for elliptic curve cryptosystems. This paper describes an elliptic curve cryptoprocessor unit resistant against fault injection. This resistance is provided by the use of parity preserving logic gates in the operating structure of the ECC unit, which is based on borrow-save adders. The proposed countermeasure provides a high coverage fault detection and induces an acceptable area overhead (+ 38 %).

1. Introduction

Since it offers the highest strength per bit of any public-key cryptography system known today, Elliptic Curve Cryptography (ECC) is a good alternative to RSA cryptosystems for ensuring secret exchange [22]. Usually, ECC brings many benefits: faster computations, less power consumption, limited storage and smaller keys and certificates. As a consequence, ECC is a good candidate for smart cards and embedded systems.

ECC is considered mathematically secure since it is based on the Elliptic Curve Discrete Logarithm Problem (ECDLP). Nevertheless, the secret key processed by an elliptic curve cryptosystem can be retrieved if this latter is implemented without caution using physical attacks. Among physical attacks, Simple Power Analysis (SPA [15]) can be an efficient method to extract the key. In such attacks, information about secret key is deduced directly thanks to the study of the power trace from a single secret key computation. Implementations of elliptic curve point multiplication algorithms are particularly vulnerable because the usual formulas used for the two main elliptic curve point multiplication operations called addition and doubling are quite different. Consequently, they can have power traces which can be distinguished. In order to protect elliptic curve cryptosystems against SPA, some efficient countermeasures have already been proposed in [7], [4] or [5]. A second type of attack, called “fault attacks”, consists in forcing the device to perform erroneous computations by changing some bits of a parameter or an intermediate result [2], [6], [3]. Faults can be induced thanks to various means as temperature variations, electromagnetic perturbations, X-rays and ion beams injection, glitches on the supply voltage or the external clock, or light illumination [1]. In order to circumvent this powerful kind of attack, some standard hardware countermeasures can be implemented such as detectors (temperature, supply voltage, frequency, light) for example. If an embedded error detection scheme detects an error, an alarm can be raised and/or a random result can be sent to the output of the cryptosystem. Another obvious countermeasure can be to check the output using an additional computation step. Unfortunately for curve-based cryptography, and contrary to RSA, checking the result implies to perform the whole computation twice or requires a space redundancy duplication, which is very costly. As a consequence, the use of partial checking methods working in parallel with the main computation is preferred.

This paper presents the incorporation of a fault detection method based on parity-preserving logic gates in some parts of an elliptic curve unit. In [18], the feasibility of this approach had already been theoretically demonstrated, but no synthesis result of parity-preserving circuits had been reported. The main contribution of this paper is to demonstrate that this method is acceptable in practice by giving implementation results. A specific part of the unit, the borrow-save adders, becomes a high-level fault-tolerant structure.

This paper is organized as follows. Section 2 recalls some notations and mathematical background. Section 3 presents known fault attacks on ECC and previous counter-

978-0-7695-3314-8/08 $25.00 © 2008 IEEE
DOI 10.1109/FDTC.2008.17

measures. Then, our parity fault-tolerant unit using parity preserving logic gates is detailed in Section 4. The performance of this unit, the impact of the implemented countermeasure and the fault coverage are presented in Section 5.

2. Mathematical Preliminaries

2.1 Elliptic Curves over $\mathbb{F}_p$

Elliptic curve cryptography was proposed independently by Koblitz [14] and Miller [16] in 1985. See [11] for a complete introduction to ECC. Many protocols use ECC for digital signature (ECDSA), encryption (ECIES), keys exchange (ECDH) or key generation (ECQMV).

An elliptic curve over a finite field $\mathbb{F}_p$ (p is a large prime number) is the set of points (x, y) which satisfies the Weierstrass equation $E : y^2 = x^3 + ax + b$, where x, y, a and b are in $\mathbb{F}_p$ and $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$. The point at infinity ∞ is added to E. The set E forms an additive group with the properties (P_1 and P_2 are in E):

1. ∞ is the neutral element: $P_1 \oplus \infty = \infty \oplus P_1 = P_1$,
2. point opposite (or negation) is inexpensive on elliptic curves: $-P_1 = (x_1, -y_1)$,
3. in affine coordinates, the result of the addition of the points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ gives the final point (x_3, y_3), such that:

$$x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda(x_1 - x_3) - y_1$$

$$\text{where } \lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{if } P_1 \neq \pm P_2 \text{ (addition)} \\ \dfrac{3x_1^2 + a}{2y_1}, & \text{if } P_1 = P_2 \text{ (doubling)} \end{cases}$$

The main ECC primitive is the point scalar multiplication. This one-way function is defined as follows:

$$E \times \mathbb{Z} \to E, (P, k) \mapsto Q = [k]P = \underbrace{P \oplus P \oplus \cdots \oplus P}_{k \text{ times}}$$

One basic method to compute the point scalar multiplication is the double-and-add algorithm (Algorithm 1).

2.2 Borrow-Save Addition

Borrow-Save (BS) is a radix-2 signed-digit redundant representation [10]. The integer X is represented by $(x_{l-1} \cdots x_1 x_0)_{\mathrm{BS}}$ where the l digits x_i are in $\{-1, 0, 1\}$ and coded using 2 bits x_i^+ and x_i^- such that $x_i = x_i^+ - x_i^-$ and

$$X = \sum_{i=0}^{l-1} x_i 2^i = \sum_{i=0}^{l-1} (x_i^+ - x_i^-) 2^i$$

Algorithm 1 Double-and-add

```
Input: $P$,  $k = (k_{l-1}...k_1k_0)_2$
Output: $Q = [k]P$
$Q \leftarrow \infty$
for $i = l - 1$ downto $0$ do
    $Q \leftarrow [2]Q$
    if $k_i = 1$ then
        $Q \leftarrow Q \oplus P$
    end if
end for
return $Q$
```

Borrow-Save Addition (BSA) can be performed using the constant time algorithm presented in Algorithm 2 and illustrated on Figure 1. Due to the signed-digit representation, $-X$ is obtained by swapping x_i^+ and x_i^- for all ranks i.

Algorithm 2 Borrow-Save Addition

```
Input: $X = (x_{l-1}...x_1x_0)_{\mathrm{BS}}$ and $Y = (y_{l-1}...y_1y_0)_{\mathrm{BS}}$
Output: $S = (s_l...s_1s_0)_{\mathrm{BS}} = X + Y$
$c_0^+ \leftarrow 0$,  $s_0^- \leftarrow 0$
for $i = 0$ to $l - 1$ do                                  ▷ parallel loop
    $2c_{i+1}^+ - c_i^- \xleftarrow{PPM} x_i^+ + y_i^+ - x_i^-$
end for
for $i = 0$ to $l - 1$ do                                  ▷ parallel loop
    $2s_{i+1}^- - s_i^+ \xleftarrow{PPM} y_i^- + c_i^- - c_i^+$
end for
$s_l^+ \leftarrow c_l^+$
return $S$
```

The BSA operator depicted on Figure 1 uses 2 rows of PPM cells [10]. A PPM is very close to a full-adder (just 1 extra inverter) and computes $2c^\pm - s^\mp = x^\pm + y^\pm - x^\mp$. A logical implementation can be $s = x^\pm \oplus y^\pm \oplus x^\mp$ and $c = x^\pm y^\pm + x^\pm \overline{x^\mp} + y^\pm \overline{x^\mp}$. Figure 1 clearly shows that the BSA computation time does not depend on the operand size l and is $\mathrm{T}(\mathrm{BSA}(l)) = 2 \cdot \mathrm{T}(\mathrm{PPM})$.

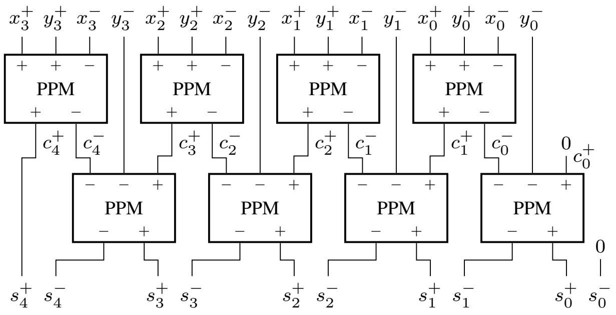
x_3^+ y_3^+ x_3^- y_3^- x_2^+ y_2^+ x_2^- y_2^- x_1^+ y_1^+ x_1^- y_1^- x_0^+ y_0^+ x_0^- y_0^-
PPM
+ + −
+ −
c_4^+ c_4^- c_3^+ c_2^- c_2^+ c_1^- c_1^+ c_0^-
0
c_0^+
− − +
− +
0
s_4^+ s_4^- s_3^+ s_3^- s_2^+ s_2^- s_1^+ s_1^- s_0^+ s_0^-

Figure 1. 4-Digit Borrow-Save Adder

Known attacks on the ECDLP	**Security constraints on parameters**
Exhaustive search	n sufficiently large (e.g., $n \geq 2^{80}$)
Pohlig-Hellman and Pollard's rho attack	For maximum resistance: # $E(\mathbb{F}_p) = hn$ with $h \in \{1, 2, 3, 4\}$ and n prime ($n > 2^{160}$)
Attack on prime-field-anomalous curves	# $E(\mathbb{F}_p) \neq p$
Weil and Tate pairing attack	$n \nmid (p^k - 1)$ for all $1 \leq k \leq C$, where C is large enough [1]

Table 1. Consequences of known attacks on the choice of the curve parameters

3 Fault Attacks on ECC

3.1 Known Fault Analysis on ECC

The security of elliptic curve cryptography schemes lies on the ECDLP: given points P and Q, there exists no subexponential algorithm to find k, such that $Q = [k]P$ (in the general case). For security reasons, the ECDLP should be intractable. As a consequence, the elliptic curve parameters (particularly p, a, b, P, and $n = \text{ord}_E(P)$) should be carefully chosen in order to be resistant against all the known attacks on the ECDLP. If this is the case, this elliptic curve is considered to be a cryptographically "strong" curve. Table 1 lists some attacks on the ECDLP and the consequences induced on the choice of a strong curve parameters.

An attacker's way of applying DFA on ECC is to disturb the representation of P (P becomes $\hat{P}$), such that the cryptosystem applies its point multiplication algorithm to a value which is not a point on the given (or selected) curve but on another curve, expected to be cryptographically less secure (it is considered to be a "weak" curve). The result of this computation is the point $\hat{Q}$ on this new weak curve which can be exploited to compute the secret key k. This idea had been first described by Biehl *et al.* in [2].

Because parameter b is not used in point addition or doubling, an elliptic curve can be completely defined as:

$$E(a, b) = E(a, y^2 - x^3 - ax). \tag{1}$$

If a "point" $\hat{P} = (\hat{x}, \hat{y}) \in \mathbb{F}_p \times \mathbb{F}_p$ but $\hat{P} \notin E$ then the computation of $\hat{Q} = [k]\hat{P}$ will take place on the curve:

$$\hat{E}(a, \hat{b}) = \hat{E}(a, \hat{y}^2 - \hat{x}^3 - a\hat{x}) \tag{2}$$

	$x_1 \rightarrow \hat{x_1}$	$p \rightarrow \hat{p}$	$a \rightarrow \hat{a}$
$P \rightarrow$	$\hat{P} = (\hat{x_1}, y_1)$	$\hat{P} = (\hat{x_1}, \hat{y_1})$	unchanged
Device outputs	$\hat{Q} = [k]\hat{P}$	$\hat{Q} = [k]\hat{P}$	$\hat{Q} = [k]P$
$(\hat{x_Q}, \hat{y_Q})$	$= [k](\hat{x_1}, y_1)$	$= [k](\hat{x_1}, \hat{y_1})$	$= [k](x_1, y_1)$
$\hat{Q} \in$	$\hat{E}(a, \hat{b})$	$\hat{E}(a, \hat{b})$	$\hat{E}(\hat{a}, \hat{b})$
Unknown values	$\hat{x_1}$	$\hat{p}$	$\hat{a}$, $\hat{b}$
Useful relations	$\hat{x}_1^3 + a\hat{x_1} + \hat{b} - y_1^2 = 0 \pmod{p}$ [2]	$\hat{p} \mid (b_Q - b_1)$ [3]	$y_1^2 = x_1^3 + \hat{a}x_1 + \hat{b}$, $\hat{y}_Q^2 = \hat{x}_Q^3 + \hat{a}\hat{x_Q} + \hat{b}$

Table 2. Attacks proposed by Ciet *et al.* in [6]

Using this fact, an attacker can choose carefully a "point" $\hat{P}_i = (\hat{x}_i, \hat{y}_i) \in \hat{E}_i$ such that with $\hat{b_i} = \hat{y_i}^2 - \hat{x_i}^3 - a\hat{x_i}$ $\text{ord}_{\hat{E}_i}(\hat{P}_i) = n_i$ is small. Then, the cryptosystem will compute $\hat{Q}_i = [k]\hat{P}_i$. Because $\hat{P}_i$ and $\hat{Q}_i$ are known, and n_i is small, the attacker can compute the discrete logarithm to retrieve $k \pmod{n_i}$. The attacker can iterate this procedure with other input points and using Chinese Remainder Theorem, the correct value of k can be finally retrieved.
A simple countermeasure is to check whether the points P and Q are on the strong curve E. As a consequence, the attack described before may not be easy in practice. Biehl *et al.* also proposed two other attacks making the assumption that only few bit-errors (or exactly one) are inserted into the base point P. These attacks are based on a rather idealized fault model.

Ciet *et al.* in [6] refine the ideas of Biehl *et al.* by relaxing their fault model. It is shown how truly random errors (hence practical computation faults) injected in the coordinates of P, in the field representation, or in the curve parameters can allow to retrieve the key k. In a cryptographic device, the system parameters are stored in non-volatile memory (e.g. EEPROM) and are transferred into working memory (e.g. RAM) for the computations. In a first scenario, Ciet *et al.* assume a permanent fault in an unknown position in any system parameter defining the elliptic curve. In a second scenario, it is analyzed the consequence of faults during the transfer of the system parameters. Table 2 lists the attacks proposed in [6].

Blömer *et al.* showed in [3] how sign changes of points can be used to recover the value of k. While the previous attacks forced the device to output points that are not on the original elliptic curve, the following Sign Change Fault Attacks (SCFA) uses a faulty point on the original curve.
It is only shown in this paper the SCFA on the standard

[1] C must be large enough so that the DLP in $\mathbb{F}^*_{p^C}$ is considered intractable (if $n > 2^{160}$, $C = 20$ suffices).

[2] with $\hat{b} = \hat{y}_Q^2 - \hat{x}_Q^3 - a\hat{x_Q} \pmod{p}$

[3] with $b_Q = \hat{y}_Q^2 - \hat{x}_Q^3 - a\hat{x_Q} \pmod{p}$ and $b_1 = y_1^2 - x_1^3 - ax_1 \pmod{p}$

double-and-add algorithm (Algorithm 1). It must be noticed that this attack can be also mounted against the Non-Adjacent-Form-based binary method and the Montgomery ladder [4] where the y coordinate is used.
The goal of the adversary is to change the sign of the y coordinate of the variable Q in line 5 of Algorithm 1 during some unknown loop iteration $0 \leq i \leq \ell - 1$, such that $Q \oplus P$ becomes $-Q \oplus P$. It is needed to be able to mount $c = \ell/m$ $\log(2\ell)$ attacks on the same input (P, k, $E(a,b)$) to recover k with probability at least 1/2. Moreover, a correct result Q must be known. The secret key bits of k will be retrieved in pieces of r bits, such that $1 \leq r \leq m$. The faulty value $\hat{Q}$ that results from a SCFA like defined before can be written:

$$\hat{Q} = -Q + 2L_i(k), \text{with } L_i(k) = \sum_{j=0}^{i} k_j 2^j P \qquad (3)$$

$L_i(k)$ is the only part which is unknown in (3): it is a multiple of P. If only a small number of bits of k are unknown, these latter can be guessed and verified using equation (3). The complete attack is divided into three phases. In the first phase, it must be collected c outputs $\hat{Q}$ of Algorithm 1 by inducing a sign-change fault in Q for random values of i. In the second phase it is guessed parts of k (stored in a variable x) and computed a test candidate T_x. In the third phase, it is tested T_x with all faulty results $\hat{Q}$ obtained in the first phase.
Blömer *et al.* proved that their Algorithm succeeds to recover k with a $O(c\ell M3^m)$ complexity (M is the maximal cost of a full scalar multiplication or a scalar multiplication including the induction of a sign-change fault) with probability at least 1/2.

The fault analysis presented in [2] and [6] allows the key recovering thanks to the mathematical analysis of only faulty results, contrary to the attack depicted in [3] where both correct and faulty executions are needed.

Another fault attack, which is called Safe-Error Attack (SEA), only checks if the computation is correctly performed or not. This definition involves that the adversary does not need to know the faulty decryption value, but only if his attack is successful or not. There are two types of SEAs: the CSEA and the MSEA. The CSEA consists in inducing any temporary random computational fault inside the ECC unit [24]. It can be applied to attack the value of key bits in a double-and-add always scheme with a dummy addition: after injecting a fault during the ECC unit computation, if the final result is correct, the addition was dummy [12]. In order to perform a MSEA, it is needed to induce a temporary memory fault inside a register or memory location [23]. The MSEA implies stronger requirements than CSEA in terms of controllability of fault location and timing. Thus, this attack appears rather hypothetical.

3.2 Previous Countermeasures

In order to circumvent fault attacks depicted in subsection 3.1, some countermeasures have already been proposed.
Check if points are on the initial curve. Attacks proposed in [2] and the attack on the base point P in [6] can be counteracted when the device checks if P is on the original curve. This can be done thanks to the curve parameter b. This latter can become an integrity check:

$$\text{for } P = (x, y), b = y^2 - x^3 - ax$$

It must be noticed that the attack presented in [3] uses a faulty point on the original curve. As a consequence, the proposed countermeasure is inefficient for this attack.
CRC checksums. In order to prevent curve parameters (particularly a and p) from the attacks in [6], it can be computed CRC on them. Before each use of these parameters, a new CRC is computed and then compared to the old one.
Randomization. Scalar k can be randomized. It can be split thanks to a random value r. Different splitting methods can be implemented, from the simplest ($[d]P = [k - r]P + [r]P$) to the most complex ($[k]P = [k \bmod r]P + [\lfloor k/r \rfloor]([r]P)$).
Use of a combined curve. In [3], it is proposed a countermeasure which generalizes the Shamir's idea [19] for RSA to ECC. The modulus is extended in a first computation (p becomes $N = p_0 p$) and then reduced (modulo p_0) in a second one. Instead of computing directly $[k]P$, it can be performed

$$Q_N = [k]P_N \text{ and } Q_{p_0} = [k]P_{p_0},$$
$$\text{where } P_N = P \pmod{N} \text{ and } P_{p_0} = P \pmod{p_0}.$$

At the end, it must be checked if $Q_{p_0} = Q_N \pmod{p_0}$. If this is not the case, some errors have been induced.
Montgomery Ladder without using y coordinates. Since y coordinate is not used in Montgomery Ladder, it is not possible to successfully attack this latter thanks to a SCFA [3].

4 Proposed ECC Unit

We present in this section the chosen algorithms for computing modular operations and the global architecture of the proposed ECC unit.

4.1 Algorithms for Modular Operations

This ECC unit is built to be able to compute unified addition formulae described by Déchène *et al.* in [8]. As a consequence, this architecture is also SPA resistant.

Required Modular Operations. Three different types of modular operations are needed to compute the formulae of Déchène *et al.*: additions, multiplications and inversions (to express the final point in affine coordinates at the end of the point multiplication). All these arithmetic operations are done in borrow-save representation (see subsection 2.2). Modular additions are computed thanks to an algorithm given in [21], and Modular Montgomery Multiplication (MMM) is used to compute modular multiplications [17]. Fermat's theorem is chosen to implement modular inversions.

Chosen Modulo through Point Multiplication. The initial version of the Montgomery multiplication described in [17] takes as inputs $X < p$ and $Y < p$, and finally computes $X.Y.R^{-1}$ (mod p), with $R = 2^l$ (see Algorithm 3). At step 5 of algorithm 3, $0 < S < 2p$, so a final subtraction is needed to ensure $S < p$. Unfortunately, this final subtraction is a time- and area-consuming process. Moreover, an attack can be performed on this, especially when unified addition formulae are chosen to implement scalar multiplication [20]. As a consequence, a new MMM algorithm without final subtraction must be written (see Algorithm 4), and the result S is given modulo $2p$: this is the chosen modulo for all the calculations.

Algorithm 3 MMM with final subtraction

```
Input: $p$ ($l$ bits), $X < p$ ($l$ bits), $Y < p$ ($l$ bits),
    $gcd(p, 2) = 1$, $R = 2^l$
Output: $S = X.Y.R^{-1}$ (mod $p$)
1. $S \leftarrow 0$
2. for $i = 0$ to $l - 1$ do
3.     $m_i \leftarrow s_0 \oplus x_i.y_0$
4.     $S \leftarrow (S + x_i.Y + m_i.p)/2$
5. end for
6. if $S \geqslant p$ then
7.     $S \leftarrow S - p$
8. end if
9. return $S$
```

Algorithm 4 MMM without final subtraction

```
Input: $p$ ($l$ bits), $X < 2p$ ($l + 1$ bits), $Y < 2p$ ($l + 1$ bits),
    $gcd(p, 2) = 1$
Output: $S = \text{MMM}(X, Y, 2p) = X.Y.2^{-(l+2)}$ (mod $2p$)
$S \leftarrow 0$
for $i = 0$ to $l + 1$ do
    $m_i \leftarrow s_0 \oplus x_i.y_0$
    $S \leftarrow (S + x_i.Y + m_i.p)/2$
end for
return $S$
```

Algorithm for Modular Multiplication. The Algorithm 4 must be modified in order to make the computations in borrow-save representation (see Algorithm 5).

Algorithm 5 MMM without final subtraction in borrow-save representation

```
Input: $p$ ($l$ bits), $-2p < X < 2p$, $-2p < Y < 2p$,
    $gcd(p, 2) = 1$
Output: $(S^+, S^-) = X.Y.2^{-(l+2)}$ (mod $2p$)
$(S^+, S^-) \leftarrow (0, 0)$
for $i = 0$ to $l + 1$ do
    $m_i \leftarrow (s_0^+ \oplus s_0^-) \oplus (x_i^+ \oplus x_i^-).(y_0^+ \oplus y_0^-)$
    if $(x_i^+, x_i^-, m_i) = (0, 0, 0)$ or $(1, 1, 0)$ then
        $(S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (0, 0)]$
    else if $(x_i^+, x_i^-, m_i) = (1, 0, 0)$ then
        $(S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (Y^+, Y^-)]$
    else if $(x_i^+, x_i^-, m_i) = (0, 1, 0)$ then
        $(S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (Y^-, Y^+)]$
    else if $(x_i^+, x_i^-, m_i) = (0, 0, 1)$ or $(1, 1, 1)$ then
        $(S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (p, 0)]$
    else if $(x_i^+, x_i^-, m_i) = (1, 0, 1)$ then
        $(S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), ((Y + p)^+, (Y + p)^-)]$
    else if $(x_i^+, x_i^-, m_i) = (0, 1, 1)$ then
        $(S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), ((Y + p)^-, (Y + p)^+))]$
    end if
    $(S^+, S^-) \leftarrow (S^+/2, S^-/2)$
end for
return $(S^+, S^-)$
```

Algorithm for Modular Addition. The algorithm initially described in [21] allows to compute $X + Y$ (mod p). This algorithm must be modified to finally compute $X + Y$ (mod $2p$).

Algorithm 6 Modular Addition

```
Input: $2^l \leq 2p < 2^{l+1}$, $-2p < X < 2p$, $-2p < Y < 2p$
Output: $S = X + Y$ (mod $2p$), with $S = S^+ - S^-$
$(T^+, T^-) \leftarrow \text{BSA}[(X^+, X^-), (Y^+, Y^-)]$
if $tv = 4t_{l+2} + 2t_{l+1} + t_l < 0$ then
    $(S^+, S^-) \leftarrow \text{BSA}[(T^+, T^-), (2p, 0)]$
else if $tv > 0$ then
    $(S^+, S^-) \leftarrow \text{BSA}[(T^+, T^-), (-2p, 0)]$
else if $tv = 0$ then
    $(S^+, S^-) \leftarrow \text{BSA}[(T^+, T^-), (0, 0)]$
end if
return $S$
```

Algorithm for Modular Inversion. Using Fermat's theorem, the inverse of a value X modulo p can be computed thanks to a modular exponentiation of X by $p - 2$: $X^{-1} = X^{p-2}$ (mod p), if gcd(X,p) = 1. This algorithm can be used in this case because p is prime. A standard algorithm for computing modular exponentiation is the square-and-multiply algorithm.

4.2 Architecture

The figure of our ECC unit is depicted in figure 2.

Modular Addition. When modular addition is performed ($X + Y \pmod{2p}$, with $X \neq Y$ or $X = Y$), MUX1 and MUX2 respectively select the couple (X^+,X^-) and (Y^+,Y^-) and a first addition is computed thanks to BSA1. MUX4 selects (T^+,T^-) and MUX3 chooses the value which must be added to (T^+,T^-) thanks to the value of $4t_{l+2} + 2t_{l+1} + t_l$. BSA2 adds these two variables, and DEMUX sends the result to its output (DEMUX$^+$, DEMUX$^-$). SUBTRACTER affects the operands before BSA1 in order to make a subtraction:

- $[(X^+, X^-), (Y^+, Y^-)] \leftarrow [(X^+, X^-), (Y^-, Y^+)]$ for $X - Y \pmod{2p}$
- $[(X^+, X^-), (Y^+, Y^-)] \leftarrow [(X^-, X^+), (Y^-, Y^+)]$ for $-X - Y \pmod{2p}$

Modular Multiplication. SHIFT1 and SHIFT2 are only used to implement the division by 2 in MMM. At the beginning of a MMM, the value of $(Y + P)$ (denoted YP in Figure 2) is computed: MUX1 and MUX2 respectively select the couple (Y^+,Y^-) and (P^+, P^-), BSA1 makes the addition, MUX3 and MUX4 respectively select the couple (T^+,T^-) and (0,0), BSA2 adds (T^+ T^-) and (0,0) and DEMUX sends the result to its output named (YN_s$^+$,YN_s$^-$). If X is denoted $X = (x_{l+1}x_lx_{l-1} \cdots x_2x_1x_0)_2$, SELECTOR1 and SELECTOR2 respectively computes m_{i1} (resp. m_{i2}) by treating even (resp. odd) indexes of X. These values command MUX2 and MUX4 in order to choose the value which must be added to (S^+ S^-). Thus, SELECTOR1, BSA1 and SHIFT1 computes the result (T^+,T^-) which is treated by SELECTOR2, BSA2 and SHIFT2. In this mode, MUX1, MUX3 and DEMUX respectively selects the couple (S^+,S^-), (T^+,T^-) and (S_s^+,S_s^-). At the last step of the MMM, DEMUX sends the result to its output (DEMUX$^+$, DEMUX$^-$).

Modular Inversion. Modular inversion can be implemented as a series of modular multiplications.

5 Parity-Preserving Logic Gates

5.1 General Properties

Parhami introduced in [18] a class of logic gates for which the parity of the outputs matches that of the inputs. For example, a parity-preserving logic gate (PPLG) which has 2 input bits (a, b) and 2 output bits (p, q) must hold the property:

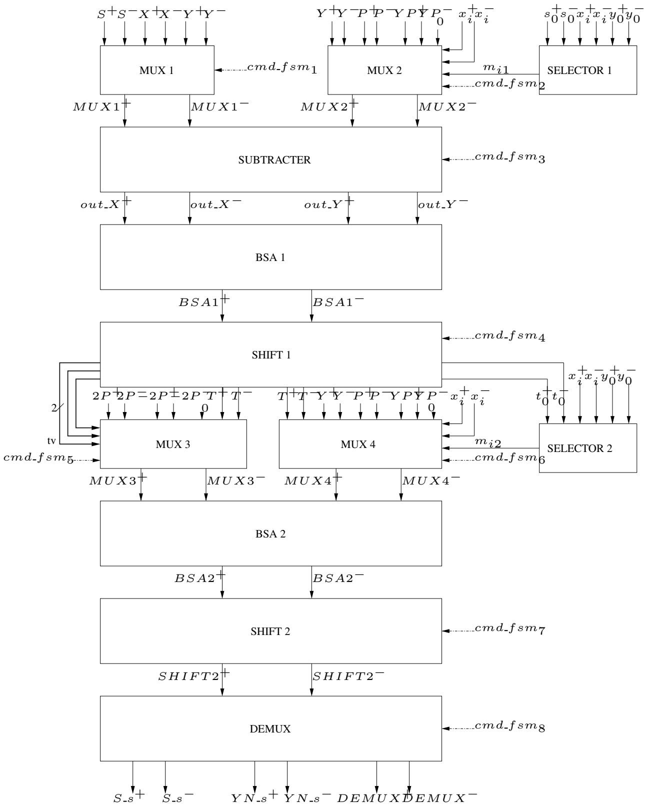
S^+ S^- X^+ X^- Y^+ Y^-
Y^+ Y^- P^+ P^- YP^+ YP^- 0 x_i^+ x_i^-
s_0^+ s_0^- x_i^+ x_i^- y_0^+ y_0^-
MUX 1
cmd_fsm_1
MUX 2
m_{i1}
cmd_fsm_2
SELECTOR 1
$MUX1^+$
$MUX1^-$
$MUX2^+$
$MUX2^-$
SUBTRACTER
cmd_fsm_3
out_X^+
out_X^-
out_Y^+
out_Y^-
BSA 1
$BSA1^+$
$BSA1^-$
SHIFT 1
cmd_fsm_4
$2P^+$ $2P^-$ $2P^+$ $2P^-$ 0 T^+ T^-
T^+ T^- Y^+ Y^- P^+ P^- YP^+ YP^- 0 x_i^+ x_i^-
t_0^+ t_0^- x_i^+ x_i^- y_0^+ y_0^-
2
tv
cmd_fsm_5
MUX 3
MUX 4
m_{i2}
cmd_fsm_6
SELECTOR 2
$MUX3^+$
$MUX3^-$
$MUX4^+$
$MUX4^-$
BSA 2
$BSA2^+$
$BSA2^-$
SHIFT 2
cmd_fsm_7
$SHIFT2^+$
$SHIFT2^-$
DEMUX
cmd_fsm_8
S_s^+ S_s^- YN_s^+ YN_s^- $DEMUX^+$ $DEMUX^-$

Figure 2. Original ECC unit

$$a \oplus b = p \oplus q$$

These PPLGs are also reversible in the sense that they allows the reproduction of the circuits inputs from observed outputs. In this paper, it is only used the parity-preserving property of these gates. In [18], the author proved that the only 2-input (a, b), 2-output (p, q) reversible gate which is also a PPLG that complements both inputs unconditionally. This is not a sufficiently interesting result for building complex circuits. Consequently, the author searched and found the only two 3-input (a, b, c), 3-output (p, q, r) reversible logic gates which are also PPLGs (with the condition $p = a$ [4]): the Fredkin gate (denoted FRG, [9]) and the Feynman double-gate (F2G).

If designers want to optimize the performance of the circuit using some fanouts or feedbacks, it must paid attention that the parity of the global circuit is maintained. Conse-

[4] It must be noted that all the already proposed reversible logic gates in the literature hold this relation.

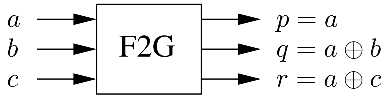

Figure 3. Feynman double-gate (F2G)

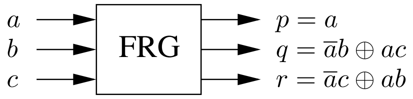

Figure 4. Fredkin gate (FRG)

quently, making a reversible circuit robust or fault tolerant is much more difficult than a conventional logic circuit.

5.2 Our approach

It is detailed in this subsection a procedure for implementing parity-preserving circuits in the general case. An illustration is given in this paper by protecting only the borrow-save adders (BSA1 and BSA2), which are the operating components of the initial ECC unit (see Figure 2). Future works will consist in protecting other components of the ECC unit, like control logic.

Choose the protected part of the circuit. Designers should investigate the tradeoff between the degree of fault tolerance of the resulting circuit (i.e. how many BSA output bits are protected and what is the protection level?) and its performance (area and speed). Moreover, besides controlling the validity of outputs, designers can add parity checks on intermediate variables: these additional checks should imply a reasonable penalty on the circuit performance.

This paper chooses to check the validity of two BSA output bits at a time (called s^-_{i+1} and s^+_i). Consequently, five BSA input bits are potentially concerned: a^+_i, b^+_i, a^-_i, b^-_i, and c^+_i.

Get the corresponding logic equations. The chosen protected part of the circuit should be sufficiently simple to get its resulting logic equations easily.

In our case, the intermediate result called c^-_i (respectively c^+_i) computed by the PPM cell which processes the input bits a^+_i, b^+_i, a^-_i (a^+_{i-1}, b^+_{i-1}, a^-_{i-1}) can be written:

$$\begin{cases} c^-_i = a^+_i \oplus b^+_i \oplus a^-_i \\ c^+_i = a^+_{i-1}.b^+_{i-1} + a^+_{i-1}.\overline{a^-_{i-1}} + b^+_{i-1}.\overline{a^-_{i-1}} \end{cases}$$

Finally, the results s^-_{i+1} and s^+_i computed by the PPM cell which processes the input bits c^-_i, b^-_i, c^+_i can be written:

$$\begin{cases} s^-_{i+1} = c^-_i.b^-_i + c^-_i.\overline{c^+_i} + b^-_i.\overline{c^+_i} \\ s^+_i = c^-_i \oplus b^-_i \oplus c^+_i \end{cases}$$

Transform the logic equations in Galois field. F2G and FRG can compute "not" logic function, and FRG can implement "and" logic function. On the other hand, among these gates, no one can implement "or" logic function. Thus, logic expressions must be expressed in Galois field thanks to the following property:

$$x + y = x \oplus y \oplus x.y$$

In our application, the variables c^+_i and s^-_{i+1} can be finally written: $\begin{cases} c^+_i = a^+_{i-1}.b^+_{i-1} \oplus a^+_{i-1}.\overline{a^-_{i-1}} \oplus b^+_{i-1}.\overline{a^-_{i-1}} \\ s^-_{i+1} = c^-_i.b^-_i \oplus c^-_i.\overline{c^+_i} \oplus b^-_i.\overline{c^+_i} \end{cases}$

Proof. (for s^-_{i+1})

$s^-_{i+1} = c^-_i.b^-_i + c^-_i.\overline{c^+_i} + b^-_i.\overline{c^+_i}$

$= (c^-_i.b^-_i \oplus c^-_i.\overline{c^+_i} \oplus c^-_i.b^-_i.\overline{c^+_i}) + b^-_i.\overline{c^+_i}$

$= c^-_i.b^-_i \oplus c^-_i.\overline{c^+_i} \oplus c^-_i.b^-_i.\overline{c^+_i} \oplus b^-_i.\overline{c^+_i} \oplus c^-_i.b^-_i.\overline{c^+_i} \oplus c^-_i.b^-_i.\overline{c^+_i} \oplus c^-_i.b^-_i.\overline{c^+_i}$

$= c^-_i.b^-_i \oplus c^-_i.\overline{c^+_i} \oplus b^-_i.\overline{c^+_i}$

This proof can be also applied to the variable c^+_i.

Implement these equations thanks to PPLGs. The protected circuit will only consist of the FRG (white boxes in Figure 5 and 6) and the F2G (grey boxes in Figure 5 and 6). It must be noticed that no fanout is advised. If a signal is needed by more than one cell, it is preferable to duplicate it thanks to a parity-preserving logic gate. For example, if a signal x is needed to be duplicated, a F2G cell can be used with the inputs ($a = x$, $b = 0$, $c = 0$): its outputs will be equal to ($p = x$, $q = x$, $r = x$).

If a PPLG output signal is not used, it must be sent to the output of the component to maintain its global parity. If two PPLG outputs have the same values and are not used by other PPLGs, they can be simplified or unconnected in the case that the reversibility property of these PPLGs is not used: these bits are called "garbage bits". For example, if a F2G have the outputs ($p = x$, $q = x$, $r = x$), and if q and r are not used by other PPLGs, they can be unconnected instead of sending them to the component output. PPM1 have eight garbage bits (see Figure 5), PPM2 six (see Figure 6).

The elementary cell of our fault-tolerant BSA (called FTBSA in the following), is depicted in Figure 7. It is made of two cells, called PPM1 (see Figure 5) and PPM2 (see Figure 6). PPM1 computes c^+_{i+1}, c^-_i, and p_1, which are the remaining output bits of PPM1. PPM2 finally outputs the results s^-_{i+1} and s^+_i; it also computes p_2, which are the remaining output bits of PPM1. The implemented parity-checker computes:

$$\begin{cases} par_1 = a^+_i \oplus b^+_i \oplus a^-_i \oplus c^-_i \\ par_2 = a^+_i \oplus b^+_i \oplus a^-_i \oplus p_1 \oplus c^+_{i+1} \\ par_3 = c^+_i \oplus b^-_i \oplus c^-_i \oplus s^-_{i+1} \oplus p_2 \oplus s^+_i \end{cases}$$

Signal p_o is then calculated:

$$p_o = par_1 + par_2 + par_3$$

Thus, if $p_o = 1$, an (or some) error(s) is (are) detected.

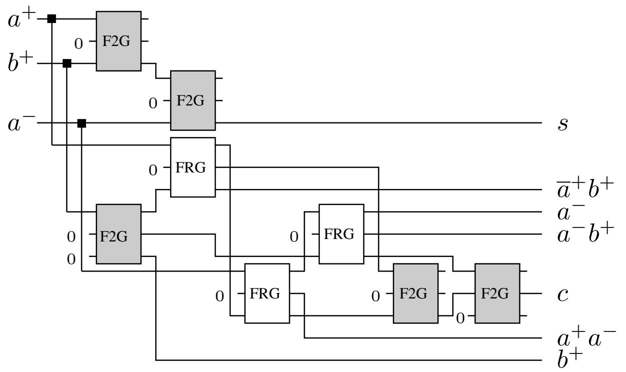

Figure 5. PPM1

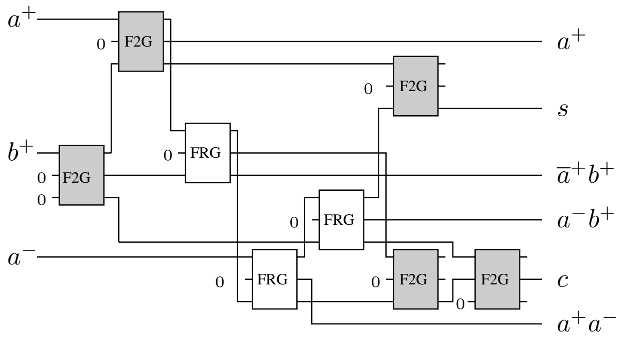

Figure 6. PPM2

6 Results

6.1 Cost of the proposed solution

All the architectures presented in this paper have been synthetized in C35 CORELIB technology using Design Vision tool. Table 3 contains the impact of our countermeasure only on the BSA. The overhead is great; the next step will be to implement the FTBSA thanks to optimized PPLGs: for example, instead of implementing F2Gs with inputs ($a = x, b = y, c = 0$), it will be implemented only $x \oplus y$.

Overhead. The impact of the countermeasure implemented in components BSA1 and BSA2 is given in Table 4. The area overhead is acceptable (+ 38 %), but the main drawback of this countermeasure is the induced latency. Like in BSA case, some optimizations are possible, particularly the implementation of optimized PPLGs or pipelining.

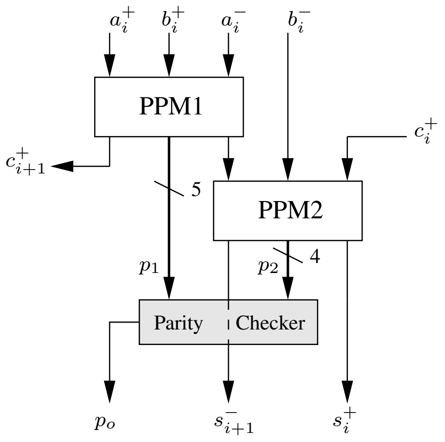

Figure 7. FTBSA

Architecture	**Area** (μm^2)	**Latency** (ns)
BSA-160 w/o EDC	134,440	1.39
BSA-160 with EDC	698,157	5.69
overhead	×5.19	×4.09

Table 3. Area and time evaluation of BSA-160, with and without error detection capabilities (EDC)

6.2 Fault Coverage

The error detection capabilities of the proposed parity check were studied using simulated fault injections into the BSA. In these simulations, two random 164-bit elements were generated, addition was started and a fault was injected during the calculation. The corrupted data was finally checked against the parity bits. The faults (one or two faulty bits at once) could appear anywhere in the adder (164×35 bits).

Some faults appear not to affect the result, they are filtered during the computation (it can appear through a FRG for example). Some are not detected even with only one faulty wire due to multiple use of some wires through the PPLGs if one of the outputs is filtered. The ratio of undetected faults goes from 5 to 12% passing from one to two faults. This ratio should decrease for three faults thanks to the parity properties. We will evaluate this case in future works.

Architecture	**Area** (μm^2)	**Latency** (ns)
ECC-160 w/o EDC	3,096,103	8.38
ECC-160 with EDC	4,270,313	19.96
overhead	×1.38	×2.38

Table 4. Area and time evaluation of ECC-160, with and without error detection capabilities (EDC)

Number of faulty bits	**1 bit**	**2 bits**
Size of the space search	5740	32941860
Detected faults	4592	28590818
Undetected faults	328	3668352
Non-faulty computation	820	682690

Table 5. Evaluation of the detection capabilities

6.3 Additional Remarks

At present, the proposed countermeasure only computes parity bits. The treatment of this information is critical since it must not add single points of failure in the circuit. For example, decisional tests must be avoided, because the flag bit which commands this test can be also faulted. It is advised to use the infective computation principle: if a fault is detected, the result is changed in a way unpredictable to an adversary (for example, [13] uses this technique). Thus, the following technique will not be implemented:

If parity(in) = parity(out), **then return**(out),
else return($error$)

It will be preferable to implement instead:

return((parity(in) $\oplus$ parity(out)).r $\oplus$ out)

where r is a random value.

The second remark concerns the use of borrow-save representation. This latter enables to make SCFA very difficult to achieve (under the condition of a secure state machine).

At last, the countermeasure described in this paper aims at making also more difficult SEA. It can be done by checking the parity during the ECC unit computations.

7 Conclusion

This paper presents a fault-tolerant elliptic curve cryptoprocessor unit. This resistance is provided by the use of parity preserving logic gates in the critical part of the ECC unit, which is the borrow-save adder. This architecture is protected with high-coverage rate against computational safe-error attack, and the sign-change fault attack seems particularly difficult to perform since borrow-save representation is used. Moreover, standard countermeasures shown in subsection 3.2 can be implemented in addition of our gate-level approach in order to be protected against other possible fault attacks on ECC.

The proposed countermeasure provides a high level of fault detection, but at the expense of an important latency overhead. This overhead can be decreased thanks to the use of optimized parity-preserving logic gates, and other reversible gates, like Toffoli and Peres gates (see [18] for more details). Moreover, some flipflops will be inserted in the critical path of the circuit in order to shorten it. This new architecture is under development. Future works will also consist in generalizing the parity-preserving principle to other ECC unit components, like control logic.

References

[1] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. *IEEE Special Issue on Cryptography and Security*, 96(2):370–382, 2006.

[2] I. Biehl, B. Meyer, and V. Müller. Differential Fault Attacks on Elliptic Curve Cryptosystems. In *Advances in Cryptology – CRYPTO, LNCS*, volume 1880, pages 131–146, 2000.

[3] J. Blömer, M. Otto, and J.-P. Seifert. Sign Change Fault Attacks on Elliptic Curve Cryptosystems. In *Proc. Fault Diagnosis and Tolerance in Cryptography – FDTC, LNCS*, volume 4236, pages 36–52, 2006.

[4] É. Brier and M. Joye. Weierstrass Elliptic Curves and Side-Channel Attacks. In *Proc. Public Key Cryptography – PKC, LNCS*, volume 2274, pages 335–345, 2002.

[5] B. Chevallier-Mames, M. Ciet, and M. Joye. Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity. *IEEE Transactions on Computers*, 53(6):760–768, 2004.

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

[7] J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystem. In *Proc. Cryptographic Hardware and Embedded Systems – CHES, LNCS*, volume 1717, pages 292–302, 1999.

[8] I. Déchène, É. Brier, and M. Joye. Unified Point Addition Formulae for Elliptic Curve Cryptosystems. In *Embedded Cryptographic Hardware: Methodologies and Architectures – Nova Science Publishers*, pages 247–256, 2004.

[9] E. Fredkin and T. Toffoli. Conservative Logic. *International Journal of Theoretical Physics*, 21(3–4):219–253, 1982.

[10] A. Guyot, Y. Herreros, and J.-M. Muller. JANUS, an On-line Multiplier/Divider for Manipulating Large Numbers. In *Proc. IEEE Symposium on Computer Arithmetic*, pages 106–111, 1989.

[11] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[12] M. Joye. Elliptic Curve Cryptosystems in the Presence of Faults. In *Securing Cyberspace: Applications and Foundations of Cryptography and Computer Security*, 2006.

[13] M. Joye, P. Manet, and J.-B. Rigaud. Strengthening Hardware AES Implementations against Fault Attacks. *IET Information Security*, 1(3):106–110, 2007.

[14] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[15] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Advances in Cryptology – CRYPTO, LNCS*, volume 1666, pages 388–397, 1999.

[16] V. Miller. Use of Elliptic Curve in Cryptology. In *Advances in Cryptography – CRYPTO, LNCS*, volume 218, pages 417–426, 1986.

[17] P. L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, 1985.

[18] B. Parhami. Fault-Tolerant Reversible Circuits. In *Proc. IEEE Asilomar Conference on Signals, Systems and Computers – ACSSC*, pages 1726–1729, 2006.

[19] A. Shamir. Method and Apparatus for Protecting Public Key Schemes from Timing and Fault Attack. *United States Patent*, (5991415), 1999.

[20] D. Stebila and N. Thériault. Unified Point Addition Formulae and Side-Channel Attacks. In *Proc. Cryptographic Hardware and Embedded Systems – CHES, LNCS*, volume 4249, pages 354–368, 2006.

[21] N. Takagi and S. Yajima. Modular Multiplication Hardware Algorithms with a Redundant Representation and Their Application to RSA Cryptosystem. *IEEE Transactions on Computers*, 41(7):887–891, 1992.

[22] S. Vanstone. ECC Holds Key to Next-Gen Cryptography. *Technical report, Certicom Corporation*, 2004.

[23] S.-M. Yen and M. Joye. Checking before Output May not be Enough against Fault-Based Cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, 2000.

[24] S.-M. Yen, S. Kim, S. Lim, and S. Moon. RSA Speedup with Chinese Remainder Theorem Immune against Hardware Fault Cryptanalysis. *IEEE Transactions on Computers*, 52(4):461–472, 2003.