A Generic Countermeasure Against Fault Injection Attacks on Asymmetric Cryptography

Using Modular Extension to Provably Protect Elliptic Curve Cryptography in Theory and Practice

Pablo Rauzy, Martin Moreau, Sylvain Guilley, and Zakaria Najm

Telecom ParisTech ; Institut Mines-Telecom ; CNRS LTCI
{*firstname.lastname*}@telecom-paristech.fr

Abstract. Fault injection attacks are a real-world threat to cryptosystems, in particular asymmetric cryptography. In this paper, we focus on countermeasures which guarantee the integrity of the computation result, hence covering most existing and future faults attacks. Namely, we study the *modular extension* protection scheme in previously existing and newly contributed countermeasures on elliptic curve scalar multiplication (ECSM) algorithms. We find that problems undermine existing countermeasures but we are able to solve some of them. We show the genericity of our contributed variant of modular extension countermeasure and formally prove its correctness and security: the fault non-detection probability is inversely proportional to the security parameter. Finally, we implement an ECSM protected with our countermeasure on an ARM Cortex-M4 microcontroller. A systematic fault injection campaign for several values of the security parameter confirms our theoretical prediction and the security of the obtained implementation and provides figures for practical performance.

Keywords: fault injection attack, countermeasure, asymmetric cryptography, elliptic curve cryptography, modular extension.

1 Introduction

Properly used cryptography is a key building block for secure information exchange. Thus, implementation-level hacks must be considered seriously in addition to the threat of cyber-attacks. In particular, fault injection attacks target physical implementations of secured devices in order to induce exploitable errors.

Formal methods. In cryptology, formal methods aim at providing a mathematical / mechanical proof of security, which helps in building trust into proved cryptosystems. However, their use is still limited in the field of fault injection and side channel attacks, as formal methods rely on models and implementations are difficult to model properly.

Asymmetric cryptography. Asymmetric cryptography addresses different needs such as key exchange and digital signature. RSA, Diffie-Hellman, and ElGamal have been used for decades, and elliptic curve cryptography (ECC) algorithms such as ECDSA [20] are more and more deployed. ECC pairing-based cryptography have recently been accelerated in practice and is thus becoming practical [34]. For example, the construction of "pairing-friendly" elliptic curves is an active subject [17]. Homomorphic encryption schemes are getting more practical and are progressively considered viable solutions for some real-world applications requiring strong privacy. All these algorithms use large numbers and take place in mathematical structures such as finite rings and fields. This property enables the use of formal methods but also facilitates attacks.

Fault Attacks. As put forward in the reference book on fault analysis in cryptography [23, Chp. 9], there are three main categories of fault attacks.
1) *Safe-error attacks* consist in testing whether an intermediate variable is dummy (usually introduced against simple power analysis [27]) or not, by faulting it and looking whether there is an effect on the final result.
2) *Cryptosystem parameter alterations* with the goal of weakening the algorithm in order to facilitate key extraction. For example in ECC, invalid-curve fault attacks consist in moving the computation to a weaker curve, enabling the attacker to use cryptanalysis attacks exploiting the faulty outputs.
3) Finally, the most serious attacks belong to the *differential fault analysis* (DFA) category. Often the attack path consists in comparing correct and faulted outputs, like in the well-known BellCoRe attack on CRT-RSA (RSA speeded up using the Chinese Remainder Theorem), or the sign-change fault attack on ECC.

The *BellCoRe attack* [8] on CRT-RSA introduced the concept of fault injection attacks. It is very powerful: faulting the computation even in a very random way yields almost certainly an exploitable result allowing to recover the secret primes of the RSA modulus $N = pq$. This attack is recalled in Sec. A for the sake of completeness.

The *sign-change attack* [7] on ECC consists in changing the sign of an intermediate elliptic curve point in the midst of an elliptic curve scalar multiplication (ECSM). The resulting faulted point is still on the curve so the fault is not detected by traditional point validation countermeasures. Such a fault can be achieved by for instance changing the sign in the double operation of the ECSM algorithm (line 3 of Alg. 1). If the fault injection occurs during the last iteration of the loop, then the final result $\widehat{Q} = \sum_{i=1}^{n-1} k_i 2^i P + k_0 P = -Q + 2k_0 P$, i.e., either $\widehat{Q} = -Q$ or $\widehat{Q} = -Q + 2P$ depending on k_0, which reveals the value of k_0 to the attacker. This process can be iterated to find the other bits of the key, and optimizations exist that trade-off between the number of necessary faulted results and the required exhaustive search.

Both RSA and ECC algorithms continue to be the target of **many** new fault injection attacks: see [3,28,4,5,13] just for some 2014 papers. Besides, this topic is emerging and other new fault attacks will appear sooner or later. Hence, the need for efficient and practical generic countermeasures against fault attacks is obvious. David Wagner from UC Berkeley concurs in [40]: "*It is a fascinating*

```
Input  : $P \in E$, $k = \sum_{i=0}^{n-1} k_i 2^i$ ($n$ is the scalar size in bits, where $k_i \in \{0,1\}$)
Output : $[k]P$
1 $Q \leftarrow \mathcal{O}$
2 for $i \leftarrow n-1$ down to $0$ do
3     $Q \leftarrow 2Q$                                  ▷ ECDBL
4     if $k_i = 1$ then $Q \leftarrow Q + P$             ▷ ECADD
5 return $Q$
```
Algorithm 1: Double-and-add scalar multiplication on elliptic curve E.

research problem to establish a principled foundation for security against fault attacks and to find schemes that can be proven secure within that framework."

Countermeasures. Verifications compatible with mathematical structures can be applied either at computational or at algorithmic level.

Algorithmic protections have been proposed by Giraud [16] (and many others [9,29,25]) for CRT-RSA, which naturally transpose to ECC, as shown in [24]. These protections are implementation specific (e.g., depend on the chosen exponentiation algorithm) and are thus difficult to automate, requiring specialized engineering skills.

Computational protections have been pioneered by Shamir in [36] using *modular extension*, initially to protect CRT-RSA. The idea is to carry out the same computation in two different algebraic structures allowing to check the computation before disclosing its result. For example protecting a computation in $\mathbb{F}_p$ consists in carrying out the same computation in $\mathbb{Z}_{pr}$ and $\mathbb{F}_r$ ($\mathbb{Z}_{pr}$ is the direct product of $\mathbb{F}_p$ and $\mathbb{F}_r$), where r is a small number ($r \ll p$); the computation in $\mathbb{Z}_{pr}$ must match that of $\mathbb{F}_r$ when reduced modulo r, if not an error is returned, otherwise the result in $\mathbb{Z}_{pr}$ is reduced modulo p and returned. The principle of modular extension is sketched in Fig. 1. This method operates at low level (integer arithmetic), thereby enabling countermeasures (and optimizations) to be added on top of it. They are thus easily maintained, which explains why this method is quite popular. Indeed, there is a wealth of variants for CRT-RSA stemming from this idea [1,39,22,6,10,12], as well as a few proofs-of-concept transposing it to ECC [7,2,21]. Despite the nonexistence of literature, the same idea could apply to post-quantum code-based cryptography, pairing, and homomorphic computation for instance. Therefore, our paper focuses on computational countermeasures.

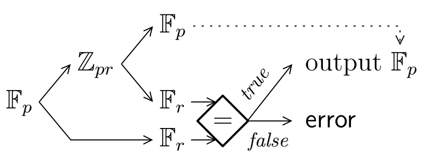

Fig. 1: Sketch of the principle of *modular extension.*

On the one hand, the variety of CRT-RSA countermeasures shows that fault attacks are a threat that is taken seriously by both the academic and the industrial communities. On the other hand, it bears witness to the artisanal way these countermeasures were put together. Indeed, the absence of formal security claims and of proofs added to the necessity of writing implementations by hand results in many weaknesses in existing countermeasures and thus in many attempts to create better ones.

Contributions. We focus on countermeasures which guarantee the integrity of the computation result, hence covering most existing and future faults attacks.

We study the *modular extension* protection scheme in existing countermeasures on elliptic curve scalar multiplication (ECSM) algorithms, namely BOS [7] and BV [2]. We show that BOS is incorrect and BV only "almost correct", i.e., that in some specific cases that we strictly characterize, they do not return the expected result (even in the absence of fault attacks), which may induce a security issue. The flaw in BOS is reminiscent to that provoked artificially by means of injecting *points with low order neighbours* and *bitflip faults* in [14].

We introduce the notion of *test-free algorithms* as a solution, and then use it to propose a new modular extension based countermeasure we call RMGN, that we prove correct and show to be generic (i.e., that may applied to other algorithms than ECSM, like pairings).

We formally study the security of RMGN. Namely, the fault non-detection probability is inversely proportional to the security parameter.

Finally, we implement RMGN on an ARM Cortex-M4 microcontroller and perform a systematic fault injection campaign for several values of the security parameter, which confirms the security of the countermeasure, and provides figures for its practical performance.

2 Elliptic Curve Scalar Multiplication

2.1 Elliptic Curves and the Projective Plane

Definition 1 (Elliptic curve over a finite field). *An elliptic curve is a plane curve over a finite field $\mathbb{F}_p$, which is denoted $E(\mathbb{F}_p)$ (or simply E when the base field is implicit), is composed of a specified point, called "point at infinity" and denoted by $\mathcal{O}$, and of the points (x, y) satisfying an equation of the form $y^2 = x^3 + ax + b$ (known as Weierstrass equation). Alongside with elliptic curve group operations, this set of points form an additive group, where $\mathcal{O}$ is the identity element.*

The points of the curve can be represented in a coordinate system over $\mathbb{F}_p$, the most natural representation being affine. However, in such system, operations on the curves are complicated due to divisions. Thus, we focus on a kind of representation known as *projective*[1], where a third coordinate Z is added, so as to avoid divisions. The equation of the curve thus becomes: $Y^2Z = X^3 + aXZ^2 + bZ^3$, where $X = xZ$ and $Y = yZ$. By convention, $\mathcal{O}$ is represented by $(X : Y : 0)$ in the projective plane. Remark that the Z coordinate is redundant: we can get rid of it by a so-called projective-to-affine transformation, which maps $(X : Y : Z)$ to $(x = X/Z, y = Y/Z)$.

[1] Other projective coordinate systems exist, such as that of Jacobi, but for the sake of simplicity and without loss of generality, we focus on the projective system.

Definition 2 (Curve order). *The order $\#E$ of an elliptic curve E is the number of points on the curve.*

Definition 3 (Point order). *The order $ord(P)$ of a point P on a elliptic curve E is the smallest non-null integer k such that $[k]P = \mathcal{O}$. The maximum value of $ord(P)$ is $\#E$.*

Definition 4 (Generator). *Let P be a point of the curve E. The point P is called a generator of E if $E = \{[k]P, 0 \leq k < \#E\}$, or equivalently if $ord(P) = \#E$.*

Remark 1. The coordinates $(X : Y : Z)$ normally belong to a finite field $\mathbb{F}_p$, but for the purpose of the modular extension countermeasure, we extend the notion of elliptic curve to rings (such as $\mathbb{Z}_{pr}$). For this reason we use the metavariable n (and $\mathbb{Z}_n$) in the following algorithms to represent an integer rather than p or r (and $\mathbb{F}_p$ or $\mathbb{F}_r$) which we use to represent prime numbers.

Computer algebra tools (e.g., MAGMA or SAGE) refuse to handle elliptic curves on $\mathbb{F}_n$ when n is composite. However, in projective coordinate system, computations do not involve divisions, hence ECSM can be computed, as recalled in the next section.

2.2 Regular ECSM

The operations on an elliptic curve are point doubling, point addition, and scalar multiplication which can be built on top of the two first operations. A left-to-right ECSM is already sketched in Alg. 1. But for our analysis, we need to detail exactly how it works internally. This is done in Alg. 2, 3, and 4.

```
  Input  : $P = (X_1 : Y_1 : Z_1) \in E(\mathbb{Z}_n)$
  Output : $(X : Y : Z) = 2P \in E(\mathbb{Z}_n)$
1 if $P$ is $\mathcal{O}$ then return $P$
2 $A = 3(X_1^2 + 2aZ_1(X_1 + Z_1))$
3 $X = 2Y_1Z_1(A^2 - 8X_1Z_1Y_1^2)$
4 $Y = A(12X_1Z_1Y_1^2 - A^2) - 8Z_1^2Y_1^4$
5 $Z = 8Z_1^3Y_1^3$
6 return $(X : Y : Z)$
```

Algorithm 2: Elliptic curve doubling ECDBL(P, n).

Lemma 1 (Correction of $\mathbf{ECSM_{L2R}}$). *Let $Q = ECSM_{L2R}(P, k, n)$, as defined by Alg. 4. Then $Q = [k]P$.*

Proof. Trivial: Alg. 4 is the left to right (L2R) version of the scalar multiplication, and Alg. 2 and 3 implements the correct point doubling and addition equations. □

```
   Input  : $P = (X_1 : Y_1 : Z_1), Q = (X_2 : Y_2 : Z_2) \in E(\mathbb{Z}_n)$
   Output : $(X : Y : Z) = P + Q \in E(\mathbb{Z}_n)$
 1 if $P$ is $\mathcal{O}$ then return $Q$
 2 if $Q$ is $\mathcal{O}$ then return $P$
 3 if $P = -Q$ then return $\mathcal{O}$
 4 if $P = Q$ then return $\mathrm{ECDBL}(P, n)$ ▷ See Alg. 2
 5 $A = Y_2Z_1 - Y_1Z_2$
 6 $B = X_2Z_1 - X_1Z_2$
 7 $C = Z_1Z_2A^2 - (X_1Z_2 + X_2Z_1)B^2$
 8 $X = BC$
 9 $Y = A(X_1Z_2B^2 - C) - Y_1Z_2B^3$
10 $Z = Z_1Z_2B^3$
11 return $(X : Y : Z)$
```
Algorithm 3: Elliptic curve addition $\mathrm{ECADD}(P, Q, n)$.

```
   Input  : $P \in E(\mathbb{Z}_n)$, $k > 0$
   Output : $Q = [k]P \in E(\mathbb{Z}_n)$
 1 $Q = \mathcal{O}$
 2 for $i = \lceil \log_2 k \rceil - 1, \ldots, 0$ do
 3     $Q = \mathrm{ECDBL}(Q, n)$ ▷ See Alg. 2.                Precondition: $Q = [\lfloor k/2^{i+1} \rfloor]P$
 4     if $k_i$ then $Q = \mathrm{ECADD}(Q, P, n)$ ▷ See Alg. 3.   Precondition: $Q = [2\lfloor k/2^{i+1} \rfloor]P$
 5 return $Q$
```
Algorithm 4: Elliptic curve scalar multiplication with left-to-right algorithm $\mathrm{ECSM}_{\mathrm{L2R}}(P, k, n)$.

2.3 Test-Free ECSM

Formal treatment of these algorithms can be made a lot simpler by getting rid of the conditional tests, which also improves the regularity of the computation. These simplifications engender *partial domain correctness* as exposed in [14]. We detail the *test-free* variants of Alg. 2, 3, and 4 in Alg. 5, 6, and 7. Actually, "test-free" refers to the absence of point comparison (line 1 of Alg. 2, and lines 1, 2, 3 & 4 of Alg. 3), not of key value (i.e., at line 4 of Alg. 7).

```
   Input  : $P = (X_1 : Y_1 : Z_1) \in {\mathbb{Z}_n}^3$
   Output : $(X : Y : Z) \in {\mathbb{Z}_n}^3$
1  $A = 3(X_1^2 + 2aZ_1(X_1 + Z_1))$
2  $X = 2Y_1Z_1(A^2 - 8X_1Z_1Y_1^2)$
3  $Y = A(12X_1Z_1Y_1^2 - A^2) - 8Z_1^2Y_1^4$
4  $Z = 8Z_1^3Y_1^3$
5  return $(X : Y : Z)$
```

Algorithm 5: Test-free elliptic curve doubling TF-ECDBL(P, n).

```
   Input  : $P = (X_1 : Y_1 : Z_1), Q = (X_2 : Y_2 : Z_2) \in {\mathbb{Z}_n}^3$
   Output : $(X : Y : Z) \in {\mathbb{Z}_n}^3$
1  $A = Y_2Z_1 - Y_1Z_2$
2  $B = X_2Z_1 - X_1Z_2$
3  $C = Z_1Z_2A^2 - (X_1Z_2 + X_2Z_1)B^2$
4  $X = BC$
5  $Y = A(X_1Z_2B^2 - C) - Y_1Z_2B^3$
6  $Z = Z_1Z_2B^3$
7  return $(X : Y : Z)$
```

Algorithm 6: Test-free elliptic curve addition TF-ECADD(P, Q, n).

```
   Input  : $P \in E(\mathbb{Z}_n)$, $k > 0$
   Output : $Q = (X : Y : Z) \in {\mathbb{Z}_n}^3$
1  $Q = \mathcal{O}$
2  for $i = \lceil \log_2 k \rceil - 1, \ldots, 0$ do
3      $Q =$ TF-ECDBL$(Q, n)$
4      if $k_i$ then $Q =$ TF-ECADD$(Q, P, n)$
5  return $Q$
```

Algorithm 7: Test-free elliptic curve scalar multiplication with left-to-right algorithm TF-ECSM$_{\mathrm{L2R}}(P, k, n)$.

Definition 5 (TF-good scalar). *Let $P \in E(\mathbb{Z}_n)$. Let $k > 0$. The scalar k is said to be TF-good with regard to P and $E(\mathbb{Z}_n)$ if and only if:*

1. *$ord(P) \nmid \lfloor k/2^i \rfloor$, for $\lceil \log_2 k \rceil - 1 \geq i \geq 1$, and for $i = 0$ when $k_0 = 1$,*
2. *$ord(P) \nmid \lfloor k/2^i \rfloor - 1$, for $\lceil \log_2 k \rceil - 2 \geq i \geq 0$ when $k_i = 1$,*
3. *$ord(P) \nmid \lfloor k/2^i \rfloor - 2$, for $\lceil \log_2 k \rceil - 2 \geq i \geq 0$ when $k_i = 1$.*

Remark that $0 < k < ord(P)$ is always TF-good.

This definition of TF-good scalar is relative to the left-to-right ECSM algorithm, but our results are portable to the other variants as well. Interestingly, the same definition would apply to the left-to-right add-always ECSM algorithm [11, §3.1]. However, the definition would differ for other variants; for instance, the case of the right-to-left ECSM algorithm is detailed in Sec. B.2.

Lemma 2 (Partial correctness of TF-ECSM$_{\mathbf{L2R}}$). *Let $P \in E(\mathbb{Z}_n)$, $k > 0$. If k is TF-good as per Def. 5, then TF-ECSM$_{L2R}(P, k, n) =$ ECSM$_{L2R}(P, k, n)$.*

Proof. We want to prove that when k is TF-good, then the test-free versions of the algorithms return the same results as the original versions. We will prove that by showing that it is equivalent to say that k is TF-good and that during the execution none of the conditional tests in the original algorithms are satisfied, and thus can be removed safely.

The only conditional test in ECDBL (line 1 of Alg. 2) checks whether the point given $[\lfloor k/2^{i+1} \rfloor]P$ as argument is $\mathcal{O}$. This condition will never be met when k is TF-good by point 1 of Def. 5.

The same reasoning applies to the first conditional test in ECADD (line 1 of Alg. 3), but with point 2 of Def. 5. Indeed, the value of Q in the ECADD is $[2\lfloor k/2^{i+1} \rfloor]P = [\lfloor k/2^i \rfloor - 1]P$ because $k_i = 1$.

The second conditional test in ECADD (line 2 of Alg. 3) checks whether the point given as argument to ECSM$_{\text{L2R}}$ is $\mathcal{O}$, in which case $ord(P) = 1$, and all three conditions in Def. 5 are violated.

The third conditional test in ECADD (line 3 of Alg. 3) checks if $P = -Q$, that is if $Q + P = \mathcal{O}$. Let us suppose that is the case. It would mean that after the ECADD (if this test was removed), we would be in the situation where the point given as argument to ECDBL is $\mathcal{O}$, which we already have shown to be impossible by point 1 of Def. 5.

The fourth and last conditional test in ECADD (line 4 of Alg. 3) checks if $P = Q$, that is if $Q - P = \mathcal{O}$. Let us suppose that is the case. Before entering ECADD, we know that $k_i = 1$ and that $Q = [\lfloor k/2^i \rfloor - 1]P$. Now, since $Q - P = \mathcal{O}$ it means that $ord(P) \mid \lfloor k/2^i \rfloor - 2$, which contradicts point 3 of Def. 5. □

Proposition 1 (TF-ECSM$_{\mathbf{L2R}}$ extension). *Let $P \in E(\mathbb{Z}_n)$, and $k > 0$. We have:*

1. *if k is TF-good with regard to P and $E(\mathbb{Z}_n)$ then TF-ECSM$_{L2R}(P, k, n) =$ ECSM$_{L2R}(P, k, n)$;*
2. *Otherwise, TF-ECSM$_{L2R}(P, k, n)$ is $\mathcal{O}$.*

Proof. Lem. 2 proves point 1. Let's now prove point 2.

If k is not TF-good, then at least one of the conditional tests will be satisfied.

In TF-ECDBL (Alg. 5), $Z_1 = 0 \implies Z = 0$, so if the conditional test of Alg. 2 (P is $\mathcal{O}$) would be satisfied then the output is $\mathcal{O}$ (recall that $\mathcal{O}$ is $(X : Y : 0)$).

In TF-ECADD (Alg. 6), $Z_1 = 0 \implies Z = 0$, and $Z_2 = 0 \implies Z = 0$, so if the conditional tests on lines 1 (P is $\mathcal{O}$) or 2 (Q is $\mathcal{O}$) of Alg. 3 would be satisfied then the output is $\mathcal{O}$. In the same algorithm, $X_2Z_1 = X_1Z_2 \implies Z = 0$, so if the conditional tests on lines 3 ($P = -Q$) or 4 ($P = Q$) of Alg. 3 would be satisfied then the output is $\mathcal{O}$.

By composition, if k is not TF-good then $\text{TF-ECSM}_{\text{L2R}}(P, k, n)$ is $\mathcal{O}$. $\square$

3 State-of-the-Art on ECSM Protection Against Fault Attacks with Modular Extension

Definition 6 (Correct algorithm). *An algorithm is said correct if it returns the right result when no faults have been injected.*

3.1 BOS

In [7], Blömer, Otto, and Seifert propose a countermeasure based on the modular extension idea of Shamir for CRT-RSA [36]. It is presented in Alg. 8.

```
   Input  : $P \in E(\mathbb{F}_p)$, $k > 0$
   Output : $Q = [k]P \in E(\mathbb{F}_p)$
1  Choose a small prime $r$, a curve $E(\mathbb{F}_r)$, and a point $P_r$ on that curve.
2  Determine the combined curve $E(\mathbb{Z}_{pr})$ and point $P_{pr}$ using the CRT.[2]
3  $(X_{pr} : Y_{pr} : Z_{pr}) = \text{ECSM}(P_{pr}, k, pr)$
4  $(X_r : Y_r : Z_r) = \text{ECSM}(P_r, k, r)$
5  if $(X_{pr} \bmod r : Y_{pr} \bmod r : Z_{pr} \bmod r) = (X_r : Y_r : Z_r)$ then
6      return $(X_{pr} \bmod p : Y_{pr} \bmod p : Z_{pr} \bmod p)$
7  else
8      return error
```

Algorithm 8: ECSM protected with BOS countermeasure $\text{BOS}(P, k, p)$.

An issue with BOS, which is not visible here as we purposedly presented a division-free version of the ECSM algorithm, is that their paper does not address the problem of divisions in $\mathbb{Z}_{pr}$. We will show that it is actually possible to circumvent this problem if necessary in Sec. 4.1. However, there is also a correction and security issue with BOS.

Proposition 2 (BOS is incorrect). *When k is TF-bad[3] with regard to P_r and $E(\mathbb{F}_r)$, BOS returns* error *even in the absence of faults.*

Proof. When k is TF-bad with regard to P_r and $E(\mathbb{F}_r)$, there will be some of the conditional tests in ECDBL and ECADD that will be satisfied in the small computation (line 4 of Alg. 8) but not in the combined one (line 3 of Alg. 8), as Prop. 1 suggests. As a result, the operations carried out in $E(\mathbb{Z}_{pr})$ won't be the same as in $E(\mathbb{F}_r)$ and thus the comparison on line 5 of Alg. 8 will fail (the result in $E(\mathbb{Z}_{pr})$ reduced modulo r will be $\mathcal{O}$, but not the result in $E(\mathbb{F}_r)$). In such cases, the algorithm will return error while the result in $E(\mathbb{F}_p)$ is actually good. $\square$

[2] See Sec. C.1.

[3] We define a TF-bad scalar as a scalar which is not TF-good according to Def. 5.

This behavior can be a *serious security issue* as it reveals information about the scalar (namely: the scalar is TF-bad with respect to a known point P_r and known curve $E(\mathbb{F}_r)$). In signature schemes such as ECDSA, this can allow for cryptanalyses [19]. In particular, when it is possible to measure a side-channel, the TF-good violated condition as well as the value of i can be recovered (see a similar case study where violations are triggered by crafted point and bitflips in [14]). A numerical example where BOS outputs an incorrect result is given in Sec. C.2.

In 2010 Joye patented [21] essentially the same countermeasure except it uses $\mathbb{F}_{r^2}$ and $\mathbb{Z}_{pr^2}$ instead of $\mathbb{F}_r$ and $\mathbb{Z}_{pr}$, which does not address the raised issues.

3.2 BV

In [2], Baek and Vasyltsov propose a countermeasure based on modular extension and point verification. The problem of divisions is explicitly evaded by carrying out computations in projective coordinates. The particularity of this countermeasure, which is presented in Alg. 9, is that instead of computing a sibling ECSM on a smaller curve $E(\mathbb{F}_r)$ to compare with its redundant part in $E(\mathbb{Z}_{pr})$, it only checks whether the point obtained by reducing the result $E(\mathbb{Z}_{pr})$ on modulo r is on the $E(\mathbb{F}_r)$ curve.

```
  Input  : $P \in E(\mathbb{F}_p)$, $k > 0$
  Output : $Q = [k]P \in E(\mathbb{F}_p)$
1 Choose a small random integer $r$.
2 Compute the combined curve $E'(\mathbb{Z}_{pr})$.[4]
3 $(X_{pr} : Y_{pr} : Z_{pr}) = \mathrm{ECSM}(P, k, pr)$
4 if $(X_{pr} \bmod r : Y_{pr} \bmod r : Z_{pr} \bmod r) \in E'(\mathbb{Z}_{pr}) \bmod r$ then
5     return $(X_{pr} \bmod p : Y_{pr} \bmod p : Z_{pr} \bmod p)$
6 else
7     return error
```

Algorithm 9: ECSM protected with BV countermeasure $\mathrm{BV}(P, k, p)$.

While this particularity is interesting in terms of complexity, the check of line 4 of Alg. 9 is weaker than the check of BOS (line 5 of Alg. 8). Indeed, the probability for a random triple $(X : Y : Z) \in {\mathbb{F}_r}^3$ to be on the $E(\mathbb{F}_r)$ curve is $\frac{\#E(\mathbb{F}_r)}{r^3} \approx \frac{1}{r^2}$. Moreover, the same problem BOS had is also present in BV in a worst manner.

Proposition 3 (BV is almost correct). *When k is TF-bad with regard to P and $E'(\mathbb{F}_r)$, BV returns* error *even in the absence of faults in very specific cases.*

Proof. The situation is similar to that of Prop. 2, except that there is no computation on the small $E'(\mathbb{F}_r)$ curve. The modular extension invariant verification done at line 4 of Alg. 9 checks if the result of the ECSM on the curve $E'(\mathbb{Z}_{pr})$ taken modulo r satisfies the curve equation modulo r.

[4] See Sec. C.3.

When k is TF-bad with regard to P and $E'(\mathbb{F}_r)$, most of the time we will have $(X_{pr} \bmod r : Y_{pr} \bmod r : Z_{pr} \bmod r) = (0 : 0 : 0)$, which satisfies the curve equation, thus the good result $(X_{pr} \bmod p : Y_{pr} \bmod p : Z_{pr} \bmod p)$ will be returned.

However, in the case where k violates the first TF-good conditions of Def. 5 when $i < 2$, the curve equation won't be satisfied modulo r by the ECSM result, and BV will return error while the result in $E'(\mathbb{Z}_{pr}) \mod p$ was good. $\square$

This time the correctness problem is negligible in practice. However, note that because the modular extension invariant check is weaker in BV, Prop. 3 would still be true if TF-ECSM was used in BV instead of ECSM (since in BV it would not *a priori* change the computation as $k < ord(P)$ on $E(\mathbb{Z}_{pr})$), while it would not be the case with BOS, which would become correct.

Note that the "almost correctness" of BV comes with a drawback: indeed, when k is TF-bad, a fault occurring before a round i where k violates one of the TF-good conditions of Def. 5 would not be detected as the modular extension invariant would become true while it should not.

3.3 New: RMGN

Here we propose a new countermeasure based on modular extension and test-free variants of elliptic curve algorithms. The particularity of our countermeasure, which is presented in Alg. 10, is its *genericity*. Indeed, the idea of using the test-free variant of the ECSM algorithm is that the same operations will be performed in the small computation (line 3 of Alg. 10) as in the combined one (line 2 of Alg. 10). These operations will be the same as in the nominal computation (without protection) as the arguments are the same. This means that we actually do not care about the "meaning" of the small computation: for example here it is not important that it happens on an elliptic curve. This has the advantage of not being specific to ECSM and hence may readily be applied to more complex computations such as pairings.

```
Input  : $P \in E(\mathbb{F}_p)$, $k > 0$
Output : $Q = [k]P \in E(\mathbb{F}_p)$
1 Choose a small prime $r$.
2 $(X_{pr} : Y_{pr} : Z_{pr}) = \text{TF-ECSM}(P, k, pr)$
3 $(X_r : Y_r : Z_r) = \text{TF-ECSM}(P \bmod r, k, r)$
4 if $(X_{pr} \bmod r : Y_{pr} \bmod r : Z_{pr} \bmod r) = (X_r : Y_r : Z_r)$ then
5     return $(X_{pr} \bmod p : Y_{pr} \bmod p : Z_{pr} \bmod p)$
6 else
7     return error
```

Algorithm 10: TF-ECSM with modular extension protection RMGN(P, k, p).

Proposition 4 (RMGN is correct). *RMGN always returns the correct result.*

Proof. Whether k is TF-bad or not, the result in $E(\mathbb{Z}_{pr})$ reduced modulo r and the result in $E(\mathbb{F}_r)$ will always match during the computation, and in particular when it ends, as Prop. 1 explains. Thus, the modular extension invariant verification done on line 4 of Alg. 10 will always be satisfied in the absence of faults. $\square$

Intuitively, what Prop. 4 says is that RMGN is the correct way to implement modular extension *in general*. However, the correctness of RMGN comes with the same drawback as BV's "almost correctness": it reduces the fault detection probability, albeit in a quantifiable and negligible manner in practice.

Proposition 5 (Probability of TF-bad scalars). *The probability of a scalar k to be TF-bad can be bounded with respect to a point $P \in E(\mathbb{F}_r)$.*

Proof. Let us suppose that $k \neq 0$ (in general, k is of the size of p) and that its bit are uniformly distributed. Let $n = \lceil \log_2 k \rceil$ the size of k and $m = \lceil \log_2 ord(P) \rceil$ the size of $ord(P)$.

Using the same notation as in Def. 5, for all $i < m$ all of the three conditions defining TF-good scalars are met. For each $i \geq m$, there are $(2^{i+1} - 2^i)$ number of size i, of which approximately $\frac{2^{i+1}-2^i}{ord(P)}$ violate one of the TF-good conditions.

For k to be TF-bad it suffices that it exists an i for which one of the TF-good conditions is violated, so the probability of k being TF-bad is:

$$\mathbb{P}_{\text{TF-bad}_P}(k) = 1 - \left(\prod_{m \leq i \leq n} 1 - \frac{\frac{2^{i+1}-2^i}{ord(P)}}{2^{i+1} - 2^i} \right) = 1 - \left(1 - \frac{1}{ord(P)}\right)^{n-m} .$$

For example, with k on 192 bits, we have $\mathbb{P}_{\text{TF-bad}_P}(k) \approx 10^{-8}$ for an $ord(P)$ on 32 bits, while $\mathbb{P}_{\text{TF-bad}_P}(k) \approx 0.5$ for an $ord(P)$ on 8 bits. $\square$

3.4 Conclusion

We can now compare the three protection schemes. Letting apart the problem of divisions in $\mathbb{Z}_{pr}$ that is not addressed by neither BOS nor BV, we have the following characterization:

- BOS is incorrect when k is TF-bad with regard to the small computation, which can be a security issue as it leaks information on k;
- BV is almost correct but has a weaker security against fault attacks when k is TF-bad, and has a weaker integrity check;
- RMGN is always correct even when k is TF-bad with regard to the small computation, which solves the security issue of BOS, but it keeps the (provably negligible in practice) weakness against fault attacks of BV.

We note that when k is TF-good with regard to the small computation, all three countermeasures are correct and secure[5]. Moreover, it follows trivially from

[5] In the sense that the fault detection probability is maximal for the modular extension method.

Def. 5 that it is possible to statically determine if a given scalar is TF-bad with regard to a point and its curve. In addition, it is important to note that in practice with $r \sim 2^{32}$, the problems with TF-bad scalars become anecdotal (see Prop. 5).

4 Formal Security Study of Modular Extension

4.1 Inversions in Direct Products

We will start by addressing the issue of divisions in $\mathbb{Z}_{pr}$. It is actually possible to circumvent this problem in the modular extension setting. Indeed, divisions can be optimized, as expressed in the following proposition.

Proposition 6 (Divisions optimization). *To get the inverse of z in $\mathbb{F}_p$ while computing in $\mathbb{Z}_{pr}$, one has:*
- $z = 0 \bmod r \implies (z^{p-2} \bmod pr) \equiv z^{-1} \mod p$,
- *otherwise* $(z^{-1} \bmod pr) \equiv z^{-1} \mod p$.

Proof. If $z = 0 \mod r$, then z is not invertible in $\mathbb{Z}_{pr}$. However, z^{p-2} exists in $\mathbb{Z}_{pr}$, and $(z^{p-2} \bmod pr) \bmod p = z^{p-2} \bmod p = z^{-1} \bmod p$. Notice that, as p is statically known, a precomputed efficient addition chain can be used.

Otherwise, when $z \neq 0 \mod r$, we have in $\mathbb{Z}_{pr}$ that $z^{-1} = z^{\varphi(pr)-1} = z^{pr-p-r} \mod pr$. Now, $(z^{-1} \bmod pr) \bmod p = z^{-1} \bmod p$ if and only if: $\varphi(p)$ divides $(pr - p - r) - (-1)$. But $(pr - p - r) - (-1) = (p - 1)(r - 1)$, which is indeed a multiple of $\varphi(p) = p - 1$. $\square$

App. D discuss the complexity of inversions as in Prop. 6. An upper-bound for the expected overhead is $(10 \times (1 - \frac{1}{r}) + 384 \times \frac{1}{r})/10 \approx 1 + 10^{-8}$ when r is a 32 bit number, which is negligible in practice.

4.2 Security Analysis

Definition 7 (Fault model). *We consider an attacker who can fault data by randomizing or zeroing any intermediate variable, and fault code by skipping any number of consecutive instructions.*

Definition 8 (Attack order). *We call order of the attack the number of faults (in the sense of Def. 7) injected during the target execution.*

In the rest of this section, we focus mainly on the resistance to first-order attacks on data. Indeed, Rauzy and Guilley have shown in [35] that 1. it is possible to adapt the modular extension protection scheme to resist attack of order D for any D by chaining D repetitions of the final check in a way that forces each repetition of the modular extension invariant verification to be faulted independently, and 2. faults on the code can be formally captured (simulated) by faults on intermediate variables.

Definition 9 (Secure algorithm). *An algorithm is said secure if it is correct as per Def. 6 and if it either returns the right result or an* error *constant when faults have been injected with an overwhelming probability.*

Theorem 1 (Security of the test-free modular extension scheme). *Test-free algorithms protected using the modular extension technique, such as RMGN, are secure as per Def. 9. In particular, the probability of non-detection is inversely proportional to the security parameter r.*

Proof. Faulted results are polynomials of faults. The result of an asymmetric cryptography computing can be written as a function of a subset of the intermediate variables, plus some inputs if the intermediate variables do not suffice to finish the computation. We are interested in the expression of the result as a function of the intermediate variables which are the target of a transient or permanent fault injection. We give the formal name $\widehat{x}$ to any faulted variable x. For convenience, we denote them by $\widehat{x_i}$, $1 \leq i \leq n$, where $n \geq 1$ is the the number of injected faults. The result consists in additions, subtractions, and multiplications of those formal variables (and inputs). Such expression is a multivariate polynomial. If the inputs are fixed, then the polynomial has only n formal variables. We call it $P(\widehat{x_1}, \ldots, \widehat{x_n})$. For now, let us assume that $n = 1$, i.e., that we face a single fault. Then P is a monovariate polynomial. Its degree d is the multiplicative depth of $\widehat{x_1}$ in the result.

A fault is not detected if and only if $P(\widehat{x_1}) = P(x_1) \mod r$, whereas $P(\widehat{x_1}) \neq P(x_1) \mod p$. Notice that the latter condition is superfluous insofar since if it is negated then the effect of the fault does not alter the result in $\mathbb{F}_p$.

Non-detection probability is inversely proportional to r. As the faulted variable $\widehat{x_1}$ can take any value in $\mathbb{Z}_{pr}$, the non-detection probability $\mathbb{P}_{\text{n.d.}}$ is given by:

$$\begin{aligned}\mathbb{P}_{\text{n.d.}} &= \frac{1}{pr-1} \cdot \sum_{\widehat{x_1} \in \mathbb{Z}_{pr} \setminus \{x_1\}} \delta_{P(\widehat{x_1}) \;=\; P(x_1) \mod r} \\ &= \frac{1}{pr-1} \cdot \Big(-1 + p \sum_{\widehat{x_1}=0}^{r-1} \delta_{P(\widehat{x_1}) \;=\; P(x_1) \mod r} \Big). \qquad (1)\end{aligned}$$

Here, $\delta_{\text{condition}}$ is equal to 1 (resp. 0) if the condition is true (resp. false).

Let $\widehat{x_1} \in \mathbb{Z}_r$, if $P(\widehat{x_1}) = P(x_1) \mod r$, then $\widehat{x_1}$ is a root of the polynomial $\Delta P(\widehat{x_1}) = P(\widehat{x_1}) - P(x_1)$ in $\mathbb{Z}_r$. We denote by $\#\text{roots}(\Delta P)$ the number of roots of ΔP over $\mathbb{Z}_r$. Thus (1) computes $(p \times \#\text{roots}(\Delta P) - 1)/(pr-1) \approx \#\text{roots}(\Delta P)/r$.

Study of the proportionality constant. A priori, bounds on this value are broad since $\#\text{roots}(\Delta P)$ can be as high as the degree d of ΔP in $\mathbb{Z}_r$, i.e., $\min(d, r-1)$. However, in practice, ΔP looks like a random polynomial over the finite field $\mathbb{Z}_r$, for several reasons:

- inputs are random numbers in most cryptographic algorithms, such as probabilistic signature schemes,
- the coefficients of ΔP in $\mathbb{Z}_r$ are randomized due to the reduction modulo r.

In such case, the number of roots is very small, despite the possibility of d being large. See for instance [31] for a proof that the number of roots tends to 1 as $r \to \infty$. Interestingly, random polynomials are still friable (i.e., they are clearly not irreducible) in average, but most factors of degree greater than one happen not to have roots in $\mathbb{Z}_r$. Thus, we have $\mathbb{P}_{\text{n.d.}} \gtrsim \frac{1}{r}$, meaning that $\mathbb{P}_{\text{n.d.}} \geq \frac{1}{r}$ but is close to $\frac{1}{r}$. A more detailed study of the theoretical upper bound on the number of roots is available in App. E.

The same law applies to multiple faults. In the case of multiple faults ($n > 1$), then the probability of non-detection generalizes to:

$$
\begin{aligned}
\mathbb{P}_{\text{n.d.}} &= \tfrac{1}{(pr-1)^n} \cdot \sum_{\widehat{x_1},\ldots,\widehat{x_n} \in \mathbb{Z}_{pr}\setminus\{x_1\} \times \ldots \times \mathbb{Z}_{pr}\setminus\{x_n\}} \delta_{P(\widehat{x_1},\ldots,\widehat{x_n}) = P(x_1,\ldots,x_n) \bmod r} \\
&= \tfrac{1}{(pr-1)^n} \cdot \sum_{\widehat{x_2},\ldots,\widehat{x_n} \in \prod_{i=2}^{n} \mathbb{Z}_{pr}\setminus\{x_i\}} \left[\sum_{\widehat{x_1} \in \mathbb{Z}_{pr}\setminus\{x_1\}} \delta_{P(\widehat{x_1},\ldots,\widehat{x_n}) = P(x_1,\ldots,x_n) \bmod r} \right] \\
&= \tfrac{1}{(pr-1)^n} \cdot \sum_{\widehat{x_2},\ldots,\widehat{x_n} \in \prod_{i=2}^{n} \mathbb{Z}_{pr}\setminus\{x_i\}} \left[p \times \#\text{roots}(\varDelta P) - 1 \right] \\
&= \tfrac{1}{(pr-1)^n} \cdot (pr-1)^{n-1} \left[p \times \#\text{roots}(\varDelta P) - 1 \right] \\
&= \frac{p \times \#\text{roots}(\varDelta P) - 1}{pr - 1}.
\end{aligned}
$$

Therefore, the probability not to detect a fault when $n > 1$ is identical to that for $n = 1$. Thus, we also have $\mathbb{P}_{\text{n.d.}} \approx \frac{1}{r}$ in the case of multiple faults of the intermediate variables[6]. $\square$

Examples can be found in App. F that illustrate the security property: indeed, $\mathbb{P}_{\text{n.d.}}$ is inversely proportional to r, with a proportionality constant which depends on the specific algorithm. The purpose of the next section is to show that the product $\mathbb{P}_{\text{n.d.}} \times r$ is constant in practice. Moreover, we explicit this constant for ECSM computations.

4.3 Modular Extension the Right Way

Here we recap how to correctly implement the modular extension countermeasure in order to achieve maximum security.

Vocabulary. We call *nominal* computation the original unprotected computation over $\mathbb{F}_p$. We call *small* computation the computation performed in a smaller field $\mathbb{F}_r$. We call *combined* computation the same computation lifted into the direct product $\mathbb{Z}_{pr}$ of $\mathbb{F}_p$ and $\mathbb{F}_r$.

[6] Note that this study does not take correlated faults into account.

Test-free algorithms. Protecting a computation using a modular extension based countermeasure means that this computation will be run twice for comparison; and thus the operation performed in the combined and in the small computation must be the equivalent. When there are tests depending on the data in $\mathbb{F}_p$ in the nominal computation, the conditions of these tests may not be satisfied at the same time in the small and in the combined computations, which would make the modular extension invariant check fail while no faults have been injected (as we have seen in BOS countermeasure for example). This problem can be circumvented by using test-free version of the computation. Of course this may cause the test-free computation to only be correct for a part of the domain of the nominal computation. Hopefully, the part of the domain for which the computation becomes incorrect can be statically determined and is negligible in practice for ECSM (and does not exists for CRT-RSA which nominal computation is naturally test-free).

Conditions on the security parameter. The size r of the mathematical structure underlying the small computation is the security parameter of a modular extension countermeasure. Several conditions have to be met to obtain maximum security and the best performance:

- r must be co-prime with p, but as p is a larger prime this should never be a problem;
- r must be prime itself, in order to avoid the maximum possible division problems in the combined computation (performance), and for $\mathbb{F}_r$ to be a field, thus maximizing the chances of $ord(P)$ to be big enough in $E(\mathbb{F}_r)$ (security);
- r should be large enough for the non-detection probability to be sufficiently low, but at the same time should remain small enough to keep the overhead of the countermeasure reasonable (we have seen that for CRT-RSA and ECSM, a value of r on 32 bits is a good option);
- r may be static, but as pointed out in [2], it helps against against side-channel analyses if it is randomly selected at runtime;
- r should not be public information contrary to what is said in [7], as it would give an attacker the opportunity to forge input values which breaks the countermeasure (e.g., a point P such that $ord(P \bmod r)$ is very low in the case of ECSM, or a message which is a multiple of r in CRT-RSA[7]).

5 Practical Case Study with RMGN

In order to practically validate our theoretical results, we have implemented RMGN (Alg. 10) ARM Cortex-M4 microcontroller (specifically an STM32).

[7] We wish to thank David Vigilant for his insight on chosen-message attack on some protected CRT-RSA.

5.1 Setup

The code is written in C and use the GMP library[8]. We used the P-192 elliptic curve from NIST [38, D.1.2.1]. Parameter values are listed below:

Field characteristic	$p =$	`0xfffffffffffffffffffffffffffffffeffffffffffffffff`
Curve equation	$a =$	`0xfffffffffffffffffffffffffffffffefffffffffffffffc`
coefficients	$b =$	`0x64210519e59c80e70fa7e9ab72243049feb8deecc146b9b1`
Point coordinates	$X_P =$	`0x188da80eb03090f67cbf20eb43a18800f4ff0afd82ff1012`
	$Y_P =$	`0x07192b95ffc8da78631011ed6b24cdd573f977a11e794811`
Point order	$ord(P) =$	`0xffffffffffffffffffffffff99def836146bc9b1b4d22831`

Figure 2 shows an architectural overview of the electromagnetic fault injection (EMFI) analysis platform we used for our experiments. EMFI has recently emerged as an efficient non-invasive fault attack, which is able to perturb a circuit through its package. The platform includes a signal generator able to generate pulses of 1.5 ns width amplified by a broadband class A amplifier (400 MHz, 300 Watt max), and an electromagnetic (EM) probe. An oscilloscope and a data timing generator are also present, so that we can precisely (with 1 ps precision) control the delay before the injection. All experiments have been performed at a settled spatial location of the EM probe relative to the ARM microcontroller: a fixed position and a fixed angular orientation. A boundary-scan (also known as JTAG) probe has been used to dump internal registers and memory contents after injection (for attack analysis purpose only).

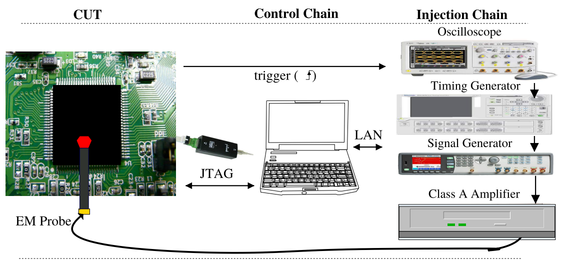

Fig. 2: EMFI platform.

We manually explored the effect of different width and power of the EM pulse, and chose values which maximize the faulting success rate. Then, we manually tuned the delay before the injection happens in order to maximize the probability of obtaining an exploitable fault for each value of r.

5.2 Method

In order to assess our theoretical results, we performed multiple attack campaigns with different values for r. The practical results allowed us to verify our theoretical prediction that the probability of non-detection $\mathbb{P}_{\text{n.d.}}$ to be inversely

[8] We used the `mini-gmp` implementation for easy portability onto the ARM microcontroller.

proportional to r (see Sec. 4.2). At the same time we were able to measure the cost of the countermeasure and confirm that the size of r is a security parameter that trades speed off for security.

– The value $r = 1$ basically means that there is no countermeasure, since $\mathbb{F}_r = \mathbb{F}_1 = \{0\}$. It helps verify that the platform is indeed injecting faults effectively, i.e., that most of the fault injection attempts are successful.
– The small values of r (on 8 to 16 bits) aim at verifying that the probability of detection / non-detection follow our theoretical prediction.
– The values of r on 32 and 64 bits represent realistic values for an operational protection.

Each value of r is chosen to be the largest prime number of its size. That is, if n is the size of r in bits, then r is the largest prime number such that $r < 2^n$.

5.3 Security Results

The table presented in Tab. 1 shows the security assessment of the RMGN countermeasure. For each value of r (lines of the table) we ran and injected random faults in approximately[9] 1000 ECSM $[k]P$ using a random 192-bit k. In total, the execution of the tests we present took approximately 6 hours. The results of our attack campaign are depicted in the last four columns.

– Correct results for which there is no error detection are simply fault injections without effect (*true negatives*).
– Correct results for which an error is detected are *false positives*, and should be minimized. Those false positive alarms are annoyances, as they warn despite no secret is at risk security-wise.
– The incorrect results for which an error is detected (*true positives*) should appear with probability $1 - \frac{1}{r}$.
– The incorrect results for which there is no error detection are *false negatives*, and should really be minimized: otherwise, the countermeasure is bypassed without notice and sensitive information may leak.

Table 1: RMGN security assessment results.

r value	r size (bit)	Positives (%) true	Positives (%) false	Negatives (%) true	Negatives (%) false
1	1	0.00	0.00	2.74	97.3
251	8	63.7	0.00	2.56	33.8
1021	10	89.1	0.00	2.96	7.95
2039	11	98.8	0.00	0.00	1.18
4093	12	97.6	0.00	1.91	0.48
65521	16	97.8	0.00	2.21	0.00
4294967291	32	97.2	0.00	2.81	0.00
18446744073709551557	64	99.8	0.00	0.21	0.00

[9] A bit less in practice: a few attempts were lost due to communication errors between our computer and the JTAG probe `gdb-server`.

Once renormalized to remove the true negatives, the last column of Tab. 1 (false negatives) represents the non-detection probability $\mathbb{P}_{\text{n.d.}}$. The relationship between r and $\mathbb{P}_{\text{n.d.}}$ is plotted in Fig. 3. The experimental results are the dots with error bars (representing plus/minus one sigma), and match the theoretical curve in blue color. The asymptotical equivalent, namely $r \mapsto 96/r$, is superimposed in red color, and is a valid approximation for $r \gtrsim 2000$, which is reasonable since practical values of r are $\approx 2^{32}$.

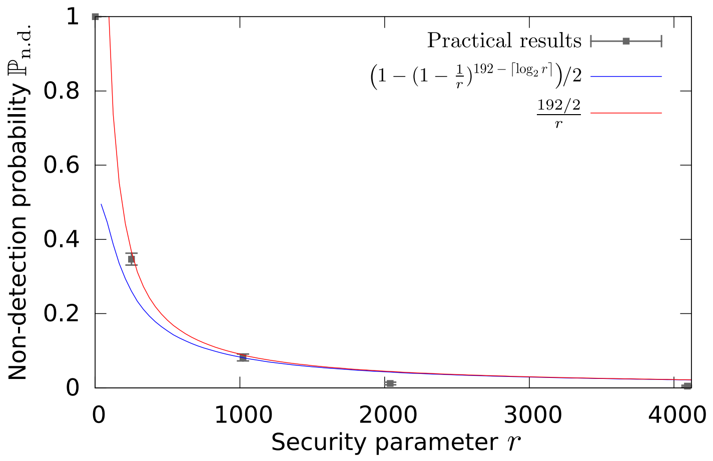

Fig. 3: Relationship between $\mathbb{P}_{\text{n.d.}}$ and r.

Proposition 7 ($\mathbb{P}_{\text{n.d.}}$ proportionality factor for RMGN). *In the case of RMGN on curve P-192, the proportionality constant is* ≈ 96.

Lemma 3. *During TF-ECSM (Alg. 7), if the coordinates of the intermediate point becomes multiples of r, they stay so until the end.*

Proof. Let $P = (X_1 : Y_1 : Z_1)$ and $Q = (X_2 : Y_2 : Z_2)$. When the scalar k satisfies one of the TF-bad conditions at step i, depending on the conditional test that would have been satisfied in the non test-free version of the algorithms, we have in TF-ECDBL:

– for test P is $\mathcal{O}$ (Alg. 2 line 1), result is $(0 : -27X_1^6 : 0)$;

and in TF-ECADD (when $k_i = 1$):

– for test P is $\mathcal{O}$ (Alg. 3 line 1), result is $((X_1Z_2)^4 : Y_1Z_2^4(Y_1^3 - 2X_1^3) : 0)$;
– for test Q is $\mathcal{O}$ (Alg. 3 line 2), result is $(-(X_2Z_1)^4 : Z_1^4X_2^3Y_2 : 0)$;
– for test $P = -Q$ (Alg. 3 line 3), result is $(0 : 8Y_1^3Z_1^5 : 0)$;
– for test $P = Q$ (Alg. 3 line 4), result is $(0 : 0 : 0)$.

Now, according to the elliptic curve doubling and addition equations (see Alg. 5 and 6), we can see that if any of this case appear at step i then at step $i + 2$ all the coordinate of the intermediate point will be 0.

In the combined computation on the curve $E(\mathbb{Z}_{pr})$, this translate to coordinates being null modulo r when k is TF-bad with regard to P on the small curve $E(\mathbb{F}_r)$. □

Proof. Using Lem. 3, we can now prove Prop. 7. If a fault occurs before step i where k satisfies one of the TF-bad conditions, then the fault is not detected. Indeed in these cases, both the result of the ECSM on $E(\mathbb{Z}_{pr})$ modulo r and the result on $E(\mathbb{F}_r)$ are equal to $(0 : 0 : 0)$.

This happens with probability $\frac{1}{2}(1 - (1 - \frac{1}{ord(P)})^{192 - \lceil \log_2 ord(P) \rceil})$, i.e., the probability of having a TF-bad scalar (see Prop. 5), with the $\frac{1}{2}$ factor to accounts for the faulting to happen before step i, where it is likely to be absorbed by a subsequent product with a multiple of r.

If we approximate the order of the point P on the $E(\mathbb{F}_r)$ curve by r, we have $\frac{1}{2}(1 - (1 - \frac{1}{r})^{192 - \lceil \log_2 r \rceil}) = \frac{(192 - \lceil \log_2 r \rceil)/2}{r} + O(\frac{1}{r})$. In practice, we can safely remove the "$-\lceil \log_2 r \rceil$" part as $\log_2 r$ will be of negligible value, especially once divided by 2. Thus we have $\mathbb{P}_{\text{n.d.}} \approx \frac{192/2}{r}$. □

As Tab. 1 and Fig. 3 show, practical values of r are sufficiently large for the latter equality to be true, and thus for the security to be highly efficient.

5.4 Performance Results

The table presented in Tab. 2 shows the cost of the entanglement countermeasure in terms of speed[10]. For each value of r (lines of the table) we list the execution time of the ECSM computation over $\mathbb{Z}_{pr}$, of the one over $\mathbb{F}_r$, of the test (comprising the extraction modulo r from the result of the computation over $\mathbb{Z}_{pr}$ and its comparison with the result of the computation over $\mathbb{F}_r$), and eventually the overhead of the countermeasure.

Table 2: RMGN performance results.

r value	r size (bit)	time (ms) $\mathbb{Z}_{pr}$	$\mathbb{F}_r$	test	overhead
1	1	683	24	$\ll$1	$\times$1.04
251	8	883	91	$\ll$1	$\times$1.43
1021	10	899	100	$\ll$1	$\times$1.46
2039	11	902	197	$\ll$1	$\times$1.61
4093	12	903	197	$\ll$1	$\times$1.61
65521	16	883	189	$\ll$1	$\times$1.56
4294967291	32	832	172	$\ll$1	$\times$1.47
18446744073709551557	64	996	246	$\ll$1	$\times$1.82

In the unprotected implementation, the ECSM computation over $\mathbb{F}_p$ took 683 ms (which naturally corresponds to the 683 ms over $\mathbb{Z}_{pr}$ when $r = 1$ as shown in Tab. 2, except that there is no need for the 24 ms needed by the computation over $\mathbb{F}_r$ which is mathematically trivial, but not optimized by `gcc`). We can see that when r is on 32 bits, the alignment with `int` makes `mini-gmp` faster, resulting in the protected algorithm running for 1004 ms, incurring a factor of only ≈ 1.47 in the run time compared to the unprotected algorithm. This is a

[10] Note that we compiled the code with `gcc -O0` option.

particularly good performance result. Indeed, in the context of digital signature, for instance ECDSA [20], an alternative to the verification by entanglement is the mere verification of the signature. However, the verification in ECDSA incurs an overhead of about $\times 4.5$ (measured with `openssl speed ecdsa` for P-192), indeed, in ECDSA, the signature verification is much more complex than the signature generation. Moreover, the curve P-192 that we use is among the smallest standardized curves, and the performance factor is directly tied to the increase in the size of the ring in which the computations are performed: when $\mathbb{F}_p$ grows, the countermeasure gets cheaper as $\frac{\log_2(pr)}{\log_2(p)}$ will be smaller.

Finally, we underline the advantage of choosing r the largest prime smaller than a given power of two, so as to increase the detection probability at given cost in terms of code speed.

6 Conclusion and Perspectives

In this paper, we have studied how to efficiently protect asymmetric cryptography algorithms, and in particular elliptic curve scalar multiplications (ECSM), against fault injection attacks. We have focused on countermeasures which guarantee the integrity of the computation result, hence covering most existing and future faults attacks.

We have reviewed the state of the art of the *modular extension* protection scheme in existing countermeasures for ECSM algorithms, namely BOS [7] and BV [2]. We have shown that BOS is incorrect and BV only "almost correct", i.e., that in some specific cases that we have strictly characterized, they do not return the expected result (even in the absence of fault attacks), which may induce a security issue.

We have introduced the notion of *test-free algorithms* as a solution, and then we have used it to propose a new modular extension based countermeasure we call RMGN, that we have proven correct and shown to be generic (i.e., that may applied to other algorithms than ECSM, like pairings).

We have formally studied the security of our proposed RMGN countermeasure, and proven that the fault non-detection probability is inversely proportional to the security parameter.

Finally, we implement RMGN on an ARM Cortex-M4 microcontroller and perform a systematic fault injection campaign for several values of the security parameter, which confirms the security of the countermeasure, and provides figures for its practical performance.

To our best knowledge, this is the first ECSM implementation to be provably protected against fault injection attacks. We used it to show that the cost of the RMGN countermeasure is extremely reasonable: **with a 32-bit value for the security parameter, the code is less than** 1.5 **times slower**. Our fault injection campaign revealed that it is also very efficient: using the same 32-bit security parameter, **100% of the fault injections were detected**.

A natural extension would be the application of the RMGN countermeasure to pairing. In this case, some computations must be carried out in *field extensions*, typically with embedding degree $m = 12$.

Finally, the security parameter r can be chosen randomly at execution time. As already pointed out in [2], this can make a natural protection against side-channel analyses. The formalization of this nice side-effect of the modular extension protection scheme would be welcomed.

References

1. C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures. In B. S. Kaliski, Jr., Ç. K. Koç, and C. Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2002.
2. Y.-J. Baek and I. Vasyltsov. How to Prevent DPA and Fault Attack in a Unified Way for ECC Scalar Multiplication - Ring Extension Method. In E. Dawson and D. Wong, editors, *Information Security Practice and Experience*, volume 4464 of *Lecture Notes in Computer Science*, pages 225–237. Springer Berlin Heidelberg, 2007.
3. G. Barthe, F. Dupressoir, P. Fouque, B. Grégoire, and J. Zapalowicz. Synthesis of Fault Attacks on Cryptographic Implementations. In G. Ahn, M. Yung, and N. Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1016–1027. ACM, 2014.
4. J. Blömer, R. Gomes Da Silva, P. Gunther, J. Krämer, and J.-P. Seifert. A Practical Second-Order Fault Attack against a Real-World Pairing Implementation. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on*, pages 123–136, Sept 2014. Busan, Korea.
5. J. Blömer, P. Günther, and G. Liske. Tampering Attacks in Pairing-Based Cryptography. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on*, pages 1–7, Sept 2014. Busan, Korea.
6. J. Blömer, M. Otto, and J.-P. Seifert. A new CRT-RSA algorithm secure against bellcore attacks. In S. Jajodia, V. Atluri, and T. Jaeger, editors, *ACM Conference on Computer and Communications Security*, pages 311–320. ACM, 2003.
7. J. Blömer, M. Otto, and J.-P. Seifert. Sign Change Fault Attacks on Elliptic Curve Cryptosystems. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography*, volume 4236 of *Lecture Notes in Computer Science*, pages 36–52. Springer Berlin Heidelberg, 2006.
8. D. Boneh, R. A. DeMillo, and R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *Proceedings of Eurocrypt'97*, volume 1233 of *LNCS*, pages 37–51. Springer, May 11-15 1997. Konstanz, Germany. DOI: 10.1007/3-540-69053-0_4.
9. A. Boscher, R. Naciri, and E. Prouff. CRT RSA Algorithm Protected Against Fault Attacks. In D. Sauveron, C. Markantonakis, A. Bilas, and J.-J. Quisquater, editors, *WISTP*, volume 4462 of *Lecture Notes in Computer Science*, pages 229–243. Springer, 2007.
10. M. Ciet and M. Joye. Practical fault countermeasures for chinese remaindering based RSA. In *Fault Diagnosis and Tolerance in Cryptography*, pages 124–131, Friday September 2nd 2005. Edinburgh, Scotland.

11. J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. K. Koç and C. Paar, editors, *CHES*, volume 1717 of *LNCS*, pages 292–302. Springer, 1999.
12. E. Dottax, C. Giraud, M. Rivain, and Y. Sierra. On Second-Order Fault Analysis Resistance for CRT-RSA Implementations. In O. Markowitch, A. Bilas, J.-H. Hoepman, C. J. Mitchell, and J.-J. Quisquater, editors, *WISTP*, volume 5746 of *Lecture Notes in Computer Science*, pages 68–83. Springer, 2009.
13. N. El Mrabet, J. J. Fournier, L. Goubin, and R. Lashermes. A survey of fault attacks in pairing based cryptography. *Cryptography and Communications*, pages 1–21, 2014.
14. J. Fan, B. Gierlichs, and F. Vercauteren. To Infinity and Beyond: Combined Attack on ECC Using Points of Low Order. In B. Preneel and T. Takagi, editors, *CHES*, volume 6917 of *LNCS*, pages 143–159. Springer, 2011.
15. H. L. Garner. Number Systems and Arithmetic. *Advances in Computers*, 6:131–194, 1965.
16. C. Giraud. An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis. *IEEE Trans. Computers*, 55(9):1116–1120, 2006.
17. A. Guillevic and D. Vergnaud. Genus 2 Hyperelliptic Curve Families with Explicit Jacobian Order Evaluation and Pairing-Friendly Constructions. In M. Abdalla and T. Lange, editors, *Pairing-Based Cryptography — Pairing 2012*, volume 7708 of *Lecture Notes in Computer Science*, pages 234–253. Springer Berlin Heidelberg, 2013.
18. D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.
19. N. Howgrave-Graham and N. P. Smart. Lattice attacks on digital signature schemes. *Des. Codes Cryptography*, 23(3):283–290, 2001.
20. D. Johnson, A. Menezes, and S. Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001.
21. M. Joye. Fault-resistant calculations on elliptic curves, Sept. 15 2010. EP Patent App. EP20,100,155,001 ; `http://www.google.com/patents/EP2228716A1?cl=en`.
22. M. Joye, P. Paillier, and S.-M. Yen. Secure evaluation of modular functions, 2001.
23. M. Joye and M. Tunstall. *Fault Analysis in Cryptography.* Springer LNCS, March 2011. `http://joye.site88.net/FAbook.html`. DOI: 10.1007/978-3-642-29656-7 ; ISBN 978-3-642-29655-0.
24. D. Karaklajic, J. Fan, J. Schmidt, and I. Verbauwhede. Low-cost fault detection method for ECC using montgomery powering ladder. In *Design, Automation and Test in Europe, DATE 2011, Grenoble, France, March 14-18, 2011*, pages 1016–1021. IEEE, 2011.
25. S.-K. Kim, T. H. Kim, D.-G. Han, and S. Hong. An efficient CRT-RSA algorithm secure against power and fault attacks. *J. Syst. Softw.*, 84:1660–1669, October 2011.
26. Ç. K. Koç, T. Acar, and B. S. Kaliski, Jr. Analyzing and comparing montgomery multiplication algorithms. *Micro, IEEE*, 16(3):26–33, Jun 1996.
27. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
28. R. Lashermes, M. Paindavoine, N. El Mrabet, J. J. Fournier, and L. Goubin. Practical Validation of Several Fault Attacks against the Miller Algorithm. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on*, pages 115–122, Sept 2014. Busan, Korea.

29. D.-P. Le, M. Rivain, and C. H. Tan. On double exponentiation for securing RSA against fault analysis. In J. Benaloh, editor, *CT-RSA*, volume 8366 of *Lecture Notes in Computer Science*, pages 152–168. Springer, 2014.
30. A. Lenstra, H. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.
31. V. Leont'ev. Roots of random polynomials over a finite field. *Mathematical Notes*, 80(1-2):300–304, 2006.
32. M. McLoone, C. McIvor, and J. V. McCanny. Coarsely integrated operand scanning (CIOS) architecture for high-speed Montgomery modular multiplication. In O. Diessel and J. Williams, editors, *Proceedings of the 2004 IEEE International Conference on Field-Programmable Technology, Brisbane, Australia, December 6-8, 2004*, pages 185–191. IEEE, 2004.
33. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, October 1996.
34. M. Naehrig, R. Niederhagen, and P. Schwabe. New software speed records for cryptographic pairings. In M. Abdalla and P. S. Barreto, editors, *Progress in Cryptology – LATINCRYPT 2010*, volume 6212 of *Lecture Notes in Computer Science*, pages 109–123. Springer-Verlag Berlin Heidelberg, 2010. Updated version: `http://cryptojedi.org/papers/#dclxvi`.
35. P. Rauzy and S. Guilley. Countermeasures Against High-Order Fault-Injection Attacks on CRT-RSA. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on*, pages 68–82, Sept 2014. Busan, Korea.
36. A. Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks, November 1999. US Patent Number 5,991,415; also presented at the rump session of EUROCRYPT '97 (May 11–15, 1997, Konstanz, Germany).
37. U.S. Department of Commerce, National Institute of Standards and Technology. Recommended Elliptic Curves For Federal Government Use, July 1999. `http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf`.
38. U.S. Department of Commerce, National Institute of Standards and Technology. FIPS PUB 186-4, FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION: Digital Signature Standard (DSS), July 2013. `https://oag.ca.gov/sites/all/files/agweb/pdfs/erds1/fips_pub_07_2013.pdf`.
39. D. Vigilant. RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks. In E. Oswald and P. Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2008.
40. D. Wagner. Cryptanalysis of a provably secure CRT-RSA algorithm. In V. Atluri, B. Pfitzmann, and P. D. McDaniel, editors, *ACM Conference on Computer and Communications Security*, pages 92–97. ACM, 2004.

A Unprotected CRT-RSA and the BellCoRe Attack

The *BellCoRe attack* [8] on CRT-RSA (RSA optimized using the Chinese Remainder Theorem) introduced the concept of fault injection attacks. It is very powerful: faulting the computation even in a very random way yields almost certainly an exploitable result allowing to recover the secret primes of the RSA modulus $N = pq$. Indeed, in the CRT-RSA algorithm, which is described in Alg. 11, most of the time is spent in the exponentiation algorithm. If the intermediate variable S_p (resp. S_q) is returned faulted as $\widehat{S_p}$ (resp. $\widehat{S_q}$), then the attacker gets an erroneous signature $\widehat{S}$, and is able to recover q (resp. p) as $\gcd(N, S - \widehat{S})$. For any integer x, $\gcd(N, x)$ can only be either 1, p, q, or N. In Alg. 11, if S_p is faulted (i.e., replaced by $\widehat{S_p} \neq S_p$), then $S - \widehat{S} = q \cdot ((i_q \cdot (S_p - S_q) \bmod p) - (i_q \cdot (\widehat{S_p} - S_q) \bmod p))$, and thus $\gcd(N, S - \widehat{S}) = q$. If S_q is faulted (i.e., replaced by $\widehat{S_q} \neq S_q$), then $S - \widehat{S} \equiv (S_q - \widehat{S_q}) - (q \bmod p) \cdot i_q \cdot (S_q - \widehat{S_q}) \equiv 0 \bmod p$ because $(q \bmod p) \cdot i_q \equiv 1 \mod p$, and thus $S - \widehat{S}$ is a multiple of p. Additionally, the difference $(S - \widehat{S})$ is not a multiple of q. So, $\gcd(N, S - \widehat{S}) = p$.

```
Input  : Message $M$, key $(p, q, d_p, d_q, i_q)$
Output : Signature $M^d \bmod N$
1 $S_p = M^{d_p} \bmod p$            ▷ Intermediate signature in $\mathbb{Z}_p$
2 $S_q = M^{d_q} \bmod q$            ▷ Intermediate signature in $\mathbb{Z}_q$
3 $S = \mathrm{CRT}(S_p, S_q)$       ▷ Recombination in $\mathbb{Z}_N$
4 return $S$
```

Algorithm 11: Unprotected CRT-RSA.

Since then, other attacks on CRT-RSA have been found, including as recently as last year, when Barthe et al. [3] exposed two new families of fault injections on CRT-RSA: *"almost full" linear combinations of p and q*, and *"almost full" affine transforms of p or q*. Both target intermediate variables of the Montgomery multiplication algorithm (namely Coarsely Integrated Operand Scanning, or CIOS [26,32]) used to implement the exponentiations of the CRT-RSA computation, and both leads to attacks based on the Lenstra-Lenstra-Lovász (LLL) lattice basis reduction algorithm [30].

B TF-Good Scalars for different ECSM Algorithms

We detail in this section the conditions for a scalar to be TF-good for unregular ECSMs, namely L2R & L2R add-always in Sec. B.1, R2L & R2L add-always in Sec. B.2, a sliding window "Non-Adjacent Form" in Sec. B.3, and a regular ECSM, namely the Montgomery ladder in Sec. B.4.

B.1 L2R and L2R add-always [11, §3.1]

The L2R algorithm has already been given in Alg. 4. The conditions for the scalar k to be TF-good are listed in Definition 5.

The L2R add-always is a more costly variant, protected against simple power attacks. It is given in Alg. 12.

It can be seen in Alg. 12 that the point Q_1 is dummy, hence does not impact the computation, even if conditional branches are not taken. Hence a TF-good scalar with respect to left-to-right add-always and left-to-right share the same conditions.

```
Input   : $P \in E(\mathbb{Z}_n)$, $k > 0$
Output  : $Q = [k]P \in E(\mathbb{Z}_n)$
1 $Q_0 = \mathcal{O}$
2 $Q_1 = P$ ▷ Dummy variable, which can be initialized to whatever value
3 for $i = \lceil \log_2 k \rceil - 1, \ldots, 0$ do
4     $Q_0 = \mathrm{ECDBL}(Q_0, n)$ ▷ See Alg. 2
5     $Q_{1-k_i} = \mathrm{ECADD}(Q_{1-k_i}, P, n)$ ▷ See Alg. 3
6 return $Q_0$
```

Algorithm 12: Elliptic curve scalar multiplication with left-to-right add-always algorithm $\mathrm{ECSM}_{\text{L2R-AA}}(P, k, n)$.

```
Input   : $P \in E(\mathbb{Z}_n)$, $k > 0$
Output  : $Q = [k]P \in E(\mathbb{Z}_n)$
1 $Q_0 = \mathcal{O}$
2 $Q_1 = P$
3 for $i = 0, \ldots, \lceil \log_2 k \rceil - 1$ do
4     if $k_i$ then $Q_0 = \mathrm{ECADD}(Q_0, Q_1, n)$ ▷ See Alg. 3
5     $Q_1 = \mathrm{ECDBL}(Q_1, n)$ ▷ See Alg. 2
6 return $Q_0$
```

Algorithm 13: Elliptic curve scalar multiplication with right-to-left algorithm $\mathrm{ECSM}_{\text{R2L}}(P, k, n)$.

B.2 R2L and R2L add-always

The R2L ECSM algorithm is described in Alg. 13.

We have the following pre-conditions:

- $Q_0 = [k \mod 2^i]P$: at line 4 of Alg. 13.
- $Q_1 = [2^i]P$: at line 5 of Alg. 13.

A scalar k is said TF-good using the right-to-left (R2L) ECSM algorithm if tests in ECADD and ECDBL are not taken.

Regarding ECDBL, this means that input Q_1 /is $\mathcal{O}$ at each iteration i. Hence, for all $0 \leq i < \lceil \log_2 k \rceil$, $ord(P) \not| 2^i$.

Regarding ECADD, this means that at every iteration i such that $k_i = 1$:

1. Input Q_0 satisfies Q_0 /is $\mathcal{O}$, thus $ord(P) \not| k \mod 2^i$.
2. Input Q_1 satisfies Q_0 /is $\mathcal{O}$, which is already covered by the condition: $\forall 0 \leq i < \lceil \log_2 k \rceil, ord(P) \not| 2^i$.
3. Inputs Q_0 and Q_1 satisfy $Q_0 + Q_1$ /is $\mathcal{O}$, that is:

$$\begin{aligned} & ord(P) \not| (2^i + (k \mod 2^i) \\ & \iff ord(P) \not| (k \mod 2^{i+1}) \quad \text{(recall that by hypothesis, } k_i = 1) \end{aligned}$$

4. Inputs Q_0 and Q_1 satisfy $Q_0 - Q_1$ /is $\mathcal{O}$, i.e., $ord(P) \not| (2^i - (k \mod 2^i)$.

Thus, a point P is TF-good if all four conditions are satisfied, for all $0 \leq i < \lceil \log_2 k \rceil$:

1. $ord(P) \nmid 2^i$,
2. $ord(P) \nmid (k \mod 2^i)$ if $k_i = 1$,
3. $ord(P) \nmid (k \mod 2^{i+1})$ if $k_i = 1$,
4. $ord(P) \nmid 2^i - (k \mod 2^i)$ if $k_i = 1$.

The right-to-left add-always ECSM is similar to Alg. 13, except that a dummy point is added to balance the branch depending on k_i bits. Thus, the notion of TF-good point for R2L and R2L-add-always is the same.

B.3 Left-to-Right sliding window NAF scalar multiplication

A *non-adjacent form* (NAF) of a positive integer k is an expression $k = \sum_{i=0}^{l-1} k_i 2^i$ where $k_i \in \{-1, 0, 1\}$, $k_{l-1} \neq 0$, and no two consecutive digits k_i are nonzero. The *length* of the NAF is l.

Theorem 2 ([18, Theorem 3.29]). *Let k be a positive integer.*

- *k has a unique NAF denoted $NAF(k)$ or $(k_{l-1}, \ldots, k_0)_{NAF}$.*
- *$NAF(k)$ has the fewest nonzero digits of any signed digit representation of k.*
- *l is at most one more than the length of the binary representation of k.*
- *The average density of nonzero digits among all NAFs of length n is approximatively* $1/3$.

We recall the L2R sliding window NAF scalar multiplication in Alg. B.4.

It is hard to provide a closed form expression to decide whether a scalar k is good or bad with respect to the multiplication of a point P by k with ECSM algorithm L2R_wNAF. However, such test can be implemented as an algorithm (see Alg. 15).

The idea behind Alg. 15 is to test the conditions for the test-free version of ECADD and ECDBL to work.

In Alg. 14, the code between lines 1 and 4 precomputes $[i]P$ for $i = 3, 5, \ldots, m$. It can easily be checked that the conditions are enumerated as follows:

- At line 2, $ord(P) \neq 1$;
- At line 4, P' is not $\mathcal{O}$, $[i-4]P, [i-2]P, [i]P$ is not $\mathcal{O}$ (for $i = 3, 5, \ldots, m$).

This is clearly equivalent to have the test implemented at line 4 of Alg. 15.

Then, at line 9 of Alg. 14, the ECDBL calls no test provided the point $Q = [(k_{l-1}, \ldots, k_{i+1})_{\mathrm{NAF}}]P$ to be doubled is not $\mathcal{O}$. This is reflected at line 9 of Alg. 15.

The ECDBL at line 17 has a test similar to that at line 9. The presence of test can be tested by the check at line 16 of Alg. 15. In this test, the notation $2^j(k_{l-1}, \ldots, k_{i+1})_{\mathrm{NAF}}$ is short for $(k_{l-1}, \ldots, k_{i+1}, \underbrace{0, \ldots, 0}_{j \text{ zeros}})_{\mathrm{NAF}}$. Actually, the index j goes up to $i - s + 1$, because this last test is required by the next instructions.

Between lines 18 and 21 of Alg. 14, one ECADD is computed. Notice that it is either an addition or a subtraction. But the tests are the same in $Q \pm P_u$, namely $Q, \pm P_u, Q \pm P_u, Q \mp P_u$ is $\mathcal{O}$. One needs to check that:

```
Input  : $P \in E(\mathbb{Z}_n)$, $w \geq 2$, $k = (k_{l-1}, \ldots, k_0)_{\mathrm{NAF}} > 0$
Output : $Q = [k]P \in E(\mathbb{Z}_n)$
1  $m \leftarrow 2(2^w - (-1)^w)/3 - 1$
2  $P' \leftarrow \mathrm{ECDBL}(P, n)$ ▷ See Alg. 2
3  for $i = 3$ to $m$ by 2 do
4      $P_i \leftarrow \mathrm{ECADD}(P_{i-1}, P')$ ▷ See Alg. 3
5  $Q \leftarrow P$
6  $i \leftarrow l - 2$
7  while $i \geq 0$ do
8      if $k_i = 0$ then
9          $Q = \mathrm{ECDBL}(Q, n)$ ▷ See Alg. 2
10         $i \leftarrow i - 1$
11     else
12         $s \leftarrow \max(i - w + 1, 0)$
13         while $k_s = 0$ do
14             $s \leftarrow s + 1$
15         $u \leftarrow (k_i, \ldots, k_s)_{\mathrm{NAF}}$
16         for $j = 1$ to $i - s + 1$ do
17             $Q \leftarrow \mathrm{ECDBL}(Q, n)$ ▷ See Alg. 2
18         if $u > 0$ then
19             $Q \leftarrow \mathrm{ECADD}(Q, P_u, n)$ ▷ See Alg. 3
20         if $u < 0$ then
21             $Q \leftarrow \mathrm{ECADD}(Q, -P_{-u}, n)$ ▷ See Alg. 3
22         $i \leftarrow s - 1$
23 return $Q$
```

Algorithm 14: Elliptic curve scalar multiplication with left-to-right sliding window NAF algorithm $\mathrm{ECSM}_{\mathrm{L2R\_wNAF}}(P, k, n)$.

```
Input   : $P \in E(\mathbb{Z}_n)$, $w \geq 2$, $k = (k_{l-1}, \ldots, k_0)_{\mathrm{NAF}} > 0$
Output : True if $k$ is TF-good, or False otherwise
1  $m \leftarrow 2(2^w - (-1)^w)/3 - 1$
2  $P' \leftarrow \mathrm{ECDBL}(P, n)$ ▷ See Alg. 2
3  for $i = 3$ to $m$ by $2$ do
4      if $ord(P) \mid i$ then return False
5  $Q \leftarrow P$
6  $i \leftarrow l - 2$
7  while $i \geq 0$ do
8      if $k_i = 0$ then
9          if $ord(P) \mid (k_{l-1}, \ldots, k_{i+1})_{NAF}$ then return False
10         $i \leftarrow i - 1$
11     else
12         $s \leftarrow \max(i - w + 1, 0)$
13         while $k_s = 0$ do
14             $s \leftarrow s + 1$
15         for $j = 0$ to $i - s + 1$ do
16             if $ord(P) \mid 2^j (k_{l-1}, \ldots, k_{i+1})_{NAF}$ then return False
17         if $ord(P) \mid (k_{l-1}, \ldots, k_{i+1}, \pm k_i, \ldots, \pm k_s)_{NAF}$ then return False
18         $i \leftarrow s - 1$
19 return True
```

Algorithm 15: Test whether a scalar k is TF-good or TF-bad with respect to the ECSM of a point P using the left-to-right sliding window NAF algorithm $\mathrm{ECSM}_{\mathrm{L2R\_wNAF}}$.

- P_u is not $\mathcal{O}$, which has already been done at line 4 of Alg. 15.
- Q is not $\mathcal{O}$, which is the test for $j = i - s + 1$ at line 16 of Alg. 15.
- As already mentioned, checking an addition or a subtraction imply the same tests, which can be summarized as: $ord(P) \mid 2^{i-s+1}(k_{l-1}, \ldots, k_{i+1})_{\text{NAF}} + \pm(k_i, \ldots, k_s)_{\text{NAF}}$, as is tested at line 17 of Alg. 15.

B.4 Montgomery Powering Ladder

The Montgomery powering ladder is a regular ECSM, described in Alg. 16.

```
  Input  : $P \in E(\mathbb{Z}_n)$, $k > 0$
  Output : $Q = [k]P \in E(\mathbb{Z}_n)$
1 $Q_0 = \mathcal{O}$
2 $Q_1 = P$
3 for $i = \lceil \log_2 k \rceil - 1, \ldots, 0$ do
4     $Q_{1-k_i} = \text{ECADD}(Q_0, Q_1, n)$ ▷ See Alg. 3
5     $Q_{k_i} = \text{ECDBL}(Q_{k_i}, n)$ ▷ See Alg. 2
6 return $Q_0$
```

Algorithm 16: Elliptic curve scalar multiplication with Montgomery powering ladder algorithm $\text{ECSM}_{\text{Mont}}(P, k, n)$.

We have the following pre-conditions:

- $Q_0 = [\lfloor k/2^{i+1} \rfloor]P$ and $Q_1 = [\lfloor k/2^{i+1} \rfloor + 1]P$: at line 4 of Alg. 16.
- $Q_{k_i} = [\lfloor k/2^{i+1} \rfloor + k_i]P$: at line 5 of Alg. 16.

A scalar k is said TF-good using the Montgomery powering ladder algorithm if tests in ECADD and ECDBL are not taken.

Regarding ECADD, this means that at every iteration i such that $\lceil \log_2 k \rceil > i \geq 0$:

1. Input Q_0 satisfies Q_0 is not $\mathcal{O}$, thus $ord(P) \nmid \lfloor k/2^{i+1} \rfloor$.
2. Input Q_1 satisfies Q_1 is not $\mathcal{O}$, thus $ord(P) \nmid \lfloor k/2^{i+1} \rfloor + 1$.
3. Inputs Q_0 and Q_1 satisfy $Q_0 \neq -Q_1$, i.e., $Q_0 + Q_1 = [2\lfloor k/2^{i+1} \rfloor + 1]P$ is not $\mathcal{O}$, thus $ord(P) \nmid 2\lfloor k/2^{i+1} \rfloor + 1$. But,
$$2\lfloor k/2^{i+1} \rfloor + 1 = \begin{cases} \lfloor k/2^i \rfloor + 1 & \text{if } k_i = 0, \\ \lfloor k/2^i \rfloor & \text{if } k_i = 1, \end{cases}$$
hence this condition is already covered by the first two ones.
4. Inputs Q_0 and Q_1 satisfy $Q_0 \neq Q_1$. In Montgomery powering ladder, $Q_1 - Q_0 = P$ at every iteration i, thus we must have $P \neq \mathcal{O}$, i.e., $ord(P) \neq 1$, which is also already covered by the two first conditions.

Regarding ECDBL, this means that input Q_{k_i} is not $\mathcal{O}$ at each iteration i. This is equivalent to $ord(P) \nmid \lfloor k/2^{i+1} \rfloor + k_i$, which is also already covered by the two first conditions of ECADD.

So, to summarize, a scalar is TF-good when used in conjunction with a Montgomery powering ladder if, for all i such that $\lceil \log_2 k \rceil > i \geq 0$:

1. $ord(P) \nmid \lfloor k/2^{i+1} \rfloor$, and
2. $ord(P) \nmid \lfloor k/2^{i+1} \rfloor + 1$.

C Details on the Existing Countermeasures

C.1 BOS Combined Curve and Point

Here we detail the computation behind line 2 of Alg. 8. For the combined curve $E(\mathbb{Z}_{pr})$ (we denote A_n and B_n the equation parameters of the curve on $\mathbb{Z}_n$):

– $A_{pr} = \mathrm{CRT}(A_p, A_r)$,
– $B_{pr} = \mathrm{CRT}(B_p, B_r)$.

For the point P_{pr}:

– $X_{pr} = \mathrm{CRT}(X_p, X_r)$,
– $Y_{pr} = \mathrm{CRT}(Y_p, Y_r)$,
– $Z_{pr} = \mathrm{CRT}(Z_p, Z_r)$.

Where $\mathrm{CRT}(U_a, U_b)$ denote the CRT recombination in $\mathbb{Z}_{ab}$ of $U_a \in \mathbb{F}_a$ and $U_b \in \mathbb{F}_b$.

It is possible to avoid these computations by choosing $U_r = U_p \bmod r$ in order to have $U_{pr} = U_p$ for all the variables listed above. However, it will be less secure than choosing security-optimized U_r values.

C.2 BOS Numerical Example of Incorrect Result

Here we give a toy example for which BOS returns an incorrect result. We chose an unrealistic, very small r as it allows to verify the computations quickly.

The nominal elliptic curve $E(\mathbb{F}_p)$ (P-192) has the following parameters:

$$\begin{aligned}
p &= \texttt{0xfffffffffffffffffffffffffffffffeffffffffffffffff},\\
A &= p - 3,\\
B &= \texttt{0x64210519e59c80e70fa7e9ab72243049feb8deecc146b9b1},\\
X_P &= \texttt{0x188da80eb03090f67cbf20eb43a18800f4ff0afd82ff1012},\\
Y_P &= \texttt{0x07192b95ffc8da78631011ed6b24cdd573f977a11e794811}.
\end{aligned}$$

The small curve $E(\mathbb{F}_r)$:

$$\begin{aligned}
r &= \texttt{0x7},\\
A_r &= \texttt{0x4},\\
B_r &= \texttt{0x1},\\
X_{P_r} &= \texttt{0x6},\\
Y_{P_r} &= \texttt{0x4}.
\end{aligned}$$

The combined curve $E(\mathbb{Z}_{pr})$ obtained as explained in Sec. C.1:

$$\begin{aligned}
pr &= \texttt{0x6fffffffffffffffffffffffffffffff8fffffffffffffff9},\\
A_{pr} &= \texttt{0x6fffffffffffffffffffffffffffffff8fffffffffffffff6},\\
B_{pr} &= \texttt{0x264210519e59c80e70fa7e9ab72243047feb8deecc146b9af},\\
X_{P_{pr}} &= \texttt{0x3188da80eb03090f67cbf20eb43a187fdf4ff0afd82ff100f},\\
Y_{P_{pr}} &= \texttt{0x307192b95ffc8da78631011ed6b24cdd273f977a11e79480e}.
\end{aligned}$$

Given these parameters, Alg. 8 is correct for all k such that $0 < k \leq 8$. Actually, $P_r = (X_{P_r} : Y_{P_r} : 1)$ is a point of order 8 on $E(\mathbb{F}_r)$, hence $[8]P_r = \mathcal{O}$.

However, the results start to be incorrect when $k > 8$. For example:

– $[9]P$ on $E(\mathbb{F}_p)$ is equal to

  $$(\texttt{0x818a4d308b1cabb74e9e8f2ba8d27c9e1d9d375ab980388f}, \\ \texttt{0x1d1aa5e208d87cd7c292f7cbb457cdf30ea542176c8e739}),$$

  and no conditional tests are satisfied during the computation.

– $[9]P_r$ on $E(\mathbb{F}_r)$ is equal to $(\texttt{0x6}, \texttt{0x4})$, and one conditional test is satisfied, namely P is $\mathcal{O}$ (line 1 of Alg. 2).

– $[9]P_{pr}$ on $E(\mathbb{Z}_{pr})$ is equal to

  $$(\texttt{0x3818a4d308b1cabb74e9e8f2ba8d27c9b1d9d375ab980388c}, \\ \texttt{0x401d1aa5e208d87cd7c292f7cbb457cdb30ea542176c8e735}),$$

  which matches $[9]P$ on $E(\mathbb{F}_p)$ modulo p, but is equal to $(\texttt{0x0}, \texttt{0x0}) \neq (\texttt{0x6}, \texttt{0x4})$ modulo r.

As a result, error will be returned while no error nor fault attacks actually occurred.

C.3 BV Combined Curve and Point

Here we detail the computation behind line 2 of Alg. 9.

In their paper [2], Baek and Vasyltsov use the following curve equation (in Jacobian projective coordinate) for the curve $E(\mathbb{F}_p)$: $Y^2Z = X^3 + AXZ^4 + BZ^6$. To obtain the combined curve $E'(\mathbb{Z}_{pr})$, a value B' is computed first: $B' = y^2 + py - x^3 - ax \bmod pr$, where $(x : y : 1)$ are the Jacobian projective coordinates of P. Then, the equation of the curve $E'(\mathbb{Z}_{pr})$ is: $Y^2Z + pYZ^3 = X^3 + AXZ^4 + B'Z^6$.

Notice that the check at line 4 of Alg. 9 is based on this Jacobian projective coordinate equation[11], only taken modulo r.

The same point P is used.

D Complexity of Modular Inverse in Direct Product

The modular inverse of a number can be efficiently obtained thanks to an extended Euclid algorithm [33, Algorithm 2.107 at §2.4, p. 67]. The complexity of this algorithm is quadratic in the size in bits of the modulo (i.e., it is $O(\log_2(p))$ when the ring is $\mathbb{Z}_p$), like the modular multiplication (see [33, Table 2.8 in Chap. 2, page 84]). However, in practice, for moduli of cryptographic size (i.e., $192 \leq \log_2(p) \leq 521$ for ECC, see [37]) the duration of a division lasts from 4 to 10 times the duration of a multiplication[12].

[11] In the original paper [2], there is a typo in Alg. 2 and the equation reads $Y^2 + pYZ^3 = X^3 + AXZ^4 + B'Z^6$, missing a Z.

[12] As benchmarked by OpenSSL version 1.0.1f `BN_mod_mul` and `BN_mod_inverse` functions.

Prop. 6 shows that divisions can also be implemented efficiently in $\mathbb{Z}_{pr}$, provided the division exists. If not, an exponentiation $z \mapsto z^{p-2}$ is necessary. Assuming that the binary representation of p consists of as many ones as zeros and that the exponentiation is done with a double-and-add algorithm, the cost of $z \mapsto z^{p-2}$ is about $\frac{3}{2}\lfloor \log_2 p \rfloor$, that is 288 multiplications when p is a 192-bit prime number.

However, multiples of r occur only with probability $\frac{1}{r}$ in computations. Thus, an upper-bound for the expected overhead is $(10\times(1-\frac{1}{r})+384\times\frac{1}{r})/10 \approx 1+10^{-8}$ when r is a 32 bit number, which is negligible in practice.

E Theoretical Upper-Bound for #roots

It is interesting to study the theoretical upper bound on the number of roots in practical cases. Leont'ev proved in [31] that if P is a random polynomial in $\mathbb{F}_p$ then $\#\text{roots}(P) \sim \text{Poisson}(\lambda = 1)$, i.e., $\mathbb{P}(\#\text{roots}(P) = k) = \frac{1}{ek!}$. In the case of $\mathit{\Delta} P \bmod r$, we know that there is always at least one root, when $\widehat{x_1} = x_1$, so we can rewrite $\mathit{\Delta} P(\widehat{x_1}) = P(\widehat{x_1}) - P(x_1) = R(\widehat{x_1}) \cdot a(\widehat{x_1} - x_1)$, where a is some constant, and R is indeed a random polynomial of degree $r-2$, owing to the modular reduction of $\mathit{\Delta} P$ by r. So we know that $\#\text{roots}(\mathit{\Delta} P) = 1 + \#\text{roots}(R)$, hence $\mathbb{P}(\#\text{roots}(\mathit{\Delta} P) = k) = \mathbb{P}(\#\text{roots}(R) = k-1)$, which is 0 if $k = 0$ and $\frac{1}{e(k-1)!}$ otherwise. We want the maximum value of k which has a "plausible" probability, let us say that is 2^{-p}, e.g., 2^{-256}. Since the values of a Poisson distribution of parameter $\lambda = 1$ are decreasing, we are looking for k such that: $\mathbb{P}(\#\text{roots}(R) = k-1) = \frac{1}{e(k-1)!} \leq 2^{-256}$. This would suggest that $k \gtrsim 58$.

This result means that $\mathbb{P}_{\text{n.d.}}$ is predicted to be at most $\frac{57}{r}$, with r being at least a 32-bit number, i.e., that $\mathbb{P}_{\text{n.d.}}$ is at maximum $\approx 2^{-26}$, and that this worst-case scenario has a probability of $\approx 2^{-256}$ of happening, in theory.

Remark 2. Note that we do not take into account the probability of TF-bad k. However, the probability of k being TF-bad can be bounded with respect to a point $P \in E(\mathbb{F}_r)$, as explained in Prop. 5 (Sec. 4.2).

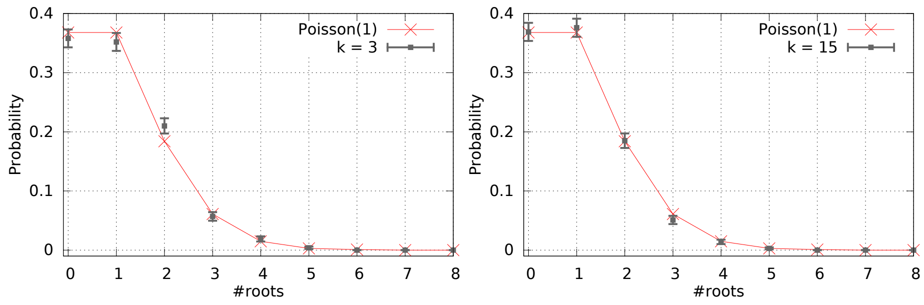

Fig. 4: #roots probability for ECSM $[k]G$.

Fig. 4 shows typical number of roots (obtained with SAGE) for practical cases in ECC, and compare them to the theoretical predictions. In this figure, we chose values of k of the form $2^j - 1$, which maximize the number of operations, and thus the size and degree of the resulting ΔP polynomials. For each value of k, we expressed the polynomial ΔP corresponding to the ECSM $[k]G$, and did so for a thousand random G. We then plotted for $i = 0$ to 8 the number of $[k]G$ for which $\#\text{roots}(\Delta P) = i + 1$ divided by 1000, that is the estimated probability $\hat{\mathbb{P}}(\#\text{roots}(\Delta P) = i + 1)$. Let us denote by Z the Boolean random variable which is equal to one if ΔP has a $(i + 1)$ roots, and zero otherwise. Our estimation of $\hat{\mathbb{P}}(\#\text{roots}(\Delta P) = i + 1)$ is thus the expectation of $\frac{1}{1000} \sum_{j=1}^{1000} Z_j$. This random variable follows a binomial distribution, of mean $p = \mathbb{P}(\#\text{roots}(\Delta P) = i + 1)$ and variance $p(1 - p)/1000$. The later values are used for the error bars ($[p - \sqrt{p(1-p)/1000}, p + \sqrt{p(1-p)/1000}]$).

The two graphs in Fig. 4 correspond to two corner-cases:

1. $k = 3 = (11)_2$: the number of roots is small because the polynomial degree is small (it is 13). (recall that $\#\text{roots}(P)$ cannot exceed the degree of P).
2. $k = 15 = (1111)_2$: the number of roots is also small, but this times because the result of Leont'ev applies. Indeed, the degree is 7213, thus the polynomial is much more random-looking.

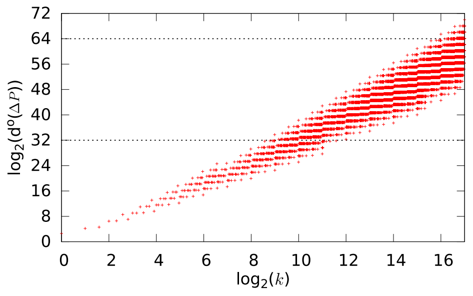

Fig. 5: Degree of the polynomial ΔP against the value of k (in log-log scale).

Actually, it is computationally hard to count the roots of polynomials of degree greater than 7213. But it can be checked that the degree of the polynomials is growing exponentially with k. This is represented in Fig. 5, where we see that the degree is about equal to $k^{3.25}$ (of course, when k has a large Hamming weight, as in $(11\ldots1)_2$, the degree is larger than when k is hollow, as in $(10\ldots0)_2$). In particular, the polynomial ΔP reaches degree 2^{32} (resp. 2^{64}) when k has about 10 (resp. 18) bits. Thus, modulo r (recall Eqn. (1)), the polynomial ΔP has maximal degree as long as the fault is injected before the last 10 (resp. 18) elliptic curve operations when r fits on 32 bits (resp. 64 bits).

F Examples of $\mathbb{P}_{\text{n.d.}}$

Example 1 ($\mathbb{P}_{n.d.}$ for CRT-RSA). From Thm. 1's proof, we can derive that for proven CRT-RSA countermeasures such as [1,39,35], we have $\mathbb{P}_{\text{n.d.}} = \frac{1}{r}$. Indeed, in CRT-RSA, the computation mainly consists of two exponentiations. In an exponentiation, ΔP takes on the form $\widehat{m}^k \cdot m^{d-k} - m^d = (\widehat{m}^k - m^k) \cdot m^{d-k}$. Assuming the message $m \neq 0$, we have $\#\text{roots}(\Delta P) = 1$ (that is $\widehat{m} = m \mod r$),

hence a non-detection probability of $\frac{1}{r}$ (in the case RSA is computed with CRT). Otherwise, after the Garner recombination [15] $\mathit{\Delta}P$ is of the form $m^{d_q} + q \cdot (i_q \cdot (m^{d_p-k}\widehat{m}^k - m^{d_q}) \bmod p) - m^{d_q} + q \cdot (i_q \cdot (m^{d_p} - m^{d_q}) \bmod p) = q \cdot (i_q \cdot (m^{d_p-k}\widehat{m}^k - m^{d_p}))$, if the fault is on the p part; or $m^{d_q-k}\widehat{m}^k + q \cdot (i_q \cdot (m^{d_p} - m^{d_q-k}\widehat{m}^k) \bmod p) - m^{d_q} + q \cdot (i_q \cdot (m^{d_p} - m^{d_q}) \bmod p) = (m^{d_q-k}\widehat{m}^k - m^{d_q}) + q \cdot (i_q \cdot (m^{d_q-k}\widehat{m}^k - m^{d_q}))$, if it is on the q part.
We conclude that $\#\text{roots}(\mathit{\Delta}P)$ is still 1 in both cases and thus that $\mathbb{P}_{\text{n.d.}} = \frac{1}{r}$.

It can be noticed that this result had been used in most previous articles dealing with fault protection on CRT-RSA without being formally proved. So, we now formally confirm those results were indeed correct.

Example 2 (Fault non-detection probability greater than $\frac{1}{r}$). Let us assume the computation $P(a, b, c) = (a + b) \cdot (b + c)$. If a single fault strikes b, then the polynomial $\mathit{\Delta}P$ is equal to $P(a, \hat{b}, c) - P(a, b, c) \mod r$. Its degree is equal to 2, and has 2 distinct roots provided $b \neq -(a + c)/2 \mod r$, or 1 double root otherwise. Thus, in the general case where the nominal inputs satisfy $b \neq -(a + c)/2 \mod r$ (which occurs also with probability $\frac{1}{r}$), the non-detection probability is $\frac{2}{r}$. Namely, the $2p - 1$ values of $\hat{b} \in \mathbb{Z}_{pr}$ causing an undetected fault are $b + kr$, with $k \in \{1, \ldots, p - 1\}$, and $-(a + c)/2 + lr$, with $l \in \{0, \ldots, p - 1\}$.

G The "enreda" Compiler

We implemented the generic modular extension protection scheme in a compiler called enreda[13].The compiler takes an algorithm and generate an equivalent protected algorithm.

G.1 Input Language

The input language of enreda is a "while language". This ensures that the language is generic enough to express all the algorithms one would want to, while keeping the language simple enough for formal study. This language, which BNF can be seen on Fig. 6, is used to describe the algorithms that we want enreda to protect by applying the entanglement protection scheme.

Algorithm descriptions start with the declarations of the structures (`Fields` and `Rings`) and of the variables, which are members of these structures. Then the algorithm itself is stated using an imperative style, i.e., a list of statements that can be assignments of expressions to variables, conditional branching or looping. An expression can be a number or a reference to a variable, or the sum of two expressions, or their product, or a projection of an expression into a given structure that has to be compatible with the type of the expression. A condition can be an equality or an inequality between expressions, or the negation of the condition, or the conjunction or disjunction of expressions. A structure is said to be compatible with the type of an expression if either of the following is true:

[13] Code will be available with the non-anonymous version of the paper.

```
Algo   ::= Decl* Stmt
Decl   ::= "Field" Struct ";"
         | "Ring" Struct "=" Struct ("*" Struct)* ";"
         | Struct Var ";"
Stmt   ::= "skip"
         | Var ":=" Expr
         | "if" Cond "then" Stmt+ "else" Stmt+ "end"
         | "while" Cond "do" Stmt+ "end"
         | "return" Var
         | Stmt ; Stmt
Expr   ::= "(" Expr ")"
         | Expr "+" Expr
         | Expr "-" Expr
         | Expr "*" Expr
         | Expr "^" Expr
         | Expr "[" Struct "]"
         | Var
         | <int>
Cond   ::= "(" Cond ")"
         | Cond "/\" Cond
         | Cond "\/" Cond
         | "!" Cond
         | Expr "=" Expr
         | Expr "!=" Expr
Struct ::= <uident>
Var    ::= <lident>
```

Fig. 6: enreda's input language.

– the type of the expression is $\mathbb{Z}$,
– the structure is $\mathbb{Z}$,
– both are the same field $\mathbb{Z}_p$,
– the type of the expression is $\mathbb{Z}_p$ and the structure is $\mathbb{Z}_{\prod_i p_i}$, where $\exists i, p_i = p$,
– the type of the expression is $\mathbb{Z}_{\prod_i p_i}$ and the structure is $\mathbb{Z}_p$, where $\exists i, p_i = p$,
– the type of the expression is $\mathbb{Z}_{\prod_i p_i}$ and the structure is $\mathbb{Z}_{\prod_j q_j}$, where $\forall i, \exists j, p_i = q_j$.

G.2 Typing and Semantics

The typing environment Γ of an algorithm written in enreda's input language is obtained by judging its types and variables declarations (i.e., the `Decl` part in the BNF provided on Fig. 6). Typing judgments are described in Fig. 7.

$$\frac{\text{“}\mathtt{Field\ F}\text{”}}{\Gamma \vdash \mathtt{F} : p}\ \text{field } (p \text{ is a fresh prime number})$$

$$\frac{\text{“}\mathtt{Ring\ R\ =\ F}_1\ \mathtt{*\ F}_2\ \mathtt{*\ \dots\ *\ F}_n\text{”} \quad \Gamma \vdash \mathtt{F}_1 : p_1 \quad \Gamma \vdash \mathtt{F}_2 : p_2 \ \dots\ \Gamma \vdash \mathtt{F}_n : p_n}{\Gamma \vdash \mathtt{R} : \prod_{n=1}^{i} p_i}\ \text{ring}$$

$$\frac{\text{“}\mathtt{S\ v}\text{”} \quad \Gamma \vdash \mathtt{S} : s}{\Gamma \vdash \mathtt{v} : s}\ \text{variable}$$

Fig. 7: Typing judgments.

Let a be an algorithm expressed in enreda's input language. Let Γ be a typing of algorithm a. Let m be a memory with the argument variables set to their input values and all other variables to the default value zero.

The semantic of a is $[\![\mathsf{a}]\!]_\Gamma\ m$, defined by:

$$\begin{aligned}
[\![\mathtt{skip}]\!]_\Gamma &= m \mapsto m\\
[\![\mathsf{v} := \mathsf{e}]\!]_\Gamma &= m \mapsto m\{\mathsf{v}\ \leftarrow\ [\![\mathsf{e}]\!]_{expr(\Gamma,m)} \bmod \Gamma[\mathsf{v}]\}\\
[\![\mathtt{if\ c\ t\ e}]\!]_\Gamma &= \text{if } [\![\mathsf{c}]\!]_{cond(\Gamma,m)} \text{ then } [\![\mathsf{t}]\!]_\Gamma \text{ else } [\![\mathsf{e}]\!]_\Gamma\\
[\![\mathtt{while\ c\ d}]\!]_\Gamma &= \text{if } [\![\mathsf{c}]\!]_{cond(\Gamma,m)} \text{ then } [\![\mathsf{d}\ ;\mathtt{while\ c\ d}]\!]_\Gamma \text{ else } [\![\mathtt{skip}]\!]_\Gamma\\
[\![\mathtt{return\ v}]\!]_\Gamma &= m[\mathsf{v}]\\
[\![\mathtt{s\ ;\ a}]\!]_\Gamma &= [\![\mathsf{a}]\!]_\Gamma \circ [\![\mathsf{s}]\!]_\Gamma, \text{ where } f \circ g \text{ is } f \circ g \text{ if } g \text{ returns a memory,}\\
&\quad\ \text{and } g \text{ otherwise;}
\end{aligned}$$

where $[\![\mathsf{e}]\!]_{expr(\Gamma,m)}$ is defined by:

$$\begin{aligned}
[\![\mathsf{e}_1 + \mathsf{e}_2]\!]_{expr(\Gamma,m)} &= [\![\mathsf{e}_1]\!]_{expr(\Gamma,m)} + [\![\mathsf{e}_2]\!]_{expr(\Gamma,m)}\\
[\![\mathsf{e}_1 - \mathsf{e}_2]\!]_{expr(\Gamma,m)} &= [\![\mathsf{e}_1]\!]_{expr(\Gamma,m)} - [\![\mathsf{e}_2]\!]_{expr(\Gamma,m)}\\
[\![\mathsf{e}_1 * \mathsf{e}_2]\!]_{expr(\Gamma,m)} &= [\![\mathsf{e}_1]\!]_{expr(\Gamma,m)} \times [\![\mathsf{e}_2]\!]_{expr(\Gamma,m)}\\
[\![\mathsf{e}_1\ \hat{}\ \mathsf{e}_2]\!]_{expr(\Gamma,m)} &= [\![\mathsf{e}_1]\!]_{expr(\Gamma,m)}{}^{[\![\mathsf{e}_2]\!]_{expr(\Gamma,m)}}\\
[\![\mathtt{e\ [S]}]\!]_{expr(\Gamma,m)} &= [\![\mathsf{e}]\!]_{expr(\Gamma,m)} \bmod \Gamma[\mathsf{S}]\\
[\![\mathsf{v}]\!]_{expr(\Gamma,m)} &= m[\mathsf{v}] \bmod\ \Gamma[\mathsf{v}]\\
[\![\mathsf{n}]\!]_{expr(\Gamma,m)} &= n;
\end{aligned}$$

and $[\![\mathtt{c}]\!]_{cond(\Gamma,m)}$ is defined by:

$$\begin{aligned}
[\![\mathtt{c}_1 \mathbin{\mathtt{/\backslash}} \mathtt{c}_2]\!]_{cond(\Gamma,m)} &= [\![\mathtt{c}_1]\!]_{cond(\Gamma,m)} \wedge [\![\mathtt{c}_2]\!]_{cond(\Gamma,m)}\\
[\![\mathtt{c}_1 \mathbin{\mathtt{\backslash/}} \mathtt{c}_2]\!]_{cond(\Gamma,m)} &= [\![\mathtt{c}_1]\!]_{cond(\Gamma,m)} \vee [\![\mathtt{c}_2]\!]_{cond(\Gamma,m)}\\
[\![\mathtt{!\,c}]\!]_{cond(\Gamma,m)} &= \neg\, [\![\mathtt{c}]\!]_{cond(\Gamma,m)}\\
[\![\mathtt{e}_1 \mathtt{=} \mathtt{e}_2]\!]_{cond(\Gamma,m)} &= [\![\mathtt{e}_1]\!]_{expr(\Gamma,m)} = [\![\mathtt{e}_2]\!]_{expr(\Gamma,m)}\\
[\![\mathtt{e}_1 \mathtt{\,!=} \mathtt{e}_2]\!]_{cond(\Gamma,m)} &= [\![\mathtt{!\,(e}_1 \mathtt{=} \mathtt{e}_2\mathtt{)}]\!]_{cond(\Gamma,m)}.
\end{aligned}$$

G.3 Correctness of the Transformation

Let `a` be an algorithm. We define the enreda transformation $\langle a\rangle_r$ for any field $\mathbb{Z}_r$. Let Γ be a typing of algorithm `a`. We define $\langle\Gamma\rangle_r$ as $\Gamma' \uplus \Gamma_r$, where, $\forall$ $\mathtt{v} \in Dom(\Gamma)$,

- $\Gamma_r[\langle\mathtt{v}\rangle_v] = r$, $\Gamma_r[\mathbb{Z}_r] = r$, and
- $\Gamma'[\mathtt{v}] = \Gamma[\mathtt{v}] \times r$.

Let m be a memory with the argument variables set to their input values and all other variables to the default value zero. We define $\langle m\rangle_r$ as $m' \uplus m_r$, where $\forall$ $\mathtt{v} \in Dom(m)$,

- $m_r[\langle\mathtt{v}\rangle_v] = m[\mathtt{v}] \bmod r$, and
- $m'[\mathtt{v}] = \mathrm{CRT}(m[\mathtt{v}], m_r[\langle\mathtt{v}\rangle_v])$.

The enreda transformation of `a` is $\langle\mathtt{a}\rangle_{r,\Gamma}$, defined by:

$$\begin{aligned}
\langle \mathtt{v := e}\rangle_{r,\Gamma} &= \langle\mathtt{v}\rangle_v \ \mathtt{:=}\ \langle\mathtt{e}\rangle_e \ \mathtt{;}\ \mathtt{v}\ \mathtt{:=}\ \langle\mathtt{e}\rangle_E\ \mathtt{;}\\
\langle \mathtt{return\ v}\rangle_{r,\Gamma} &= \mathtt{if\ v\ [}\mathbb{Z}_r\mathtt{]\ =\ }\langle\mathtt{v}\rangle_v\\
&\ \mathtt{then\ return\ v\ [}\Gamma[\mathtt{v}]\mathtt{]\ ;}\\
&\ \mathtt{else\ return\ }\mathsf{error}\mathtt{\ ;}\\
\langle \mathtt{s\ ;\ a}\rangle_{r,\Gamma} &= \langle\mathtt{s}\rangle_{r,\Gamma}\ \langle\mathtt{a}\rangle_{r,\Gamma},
\end{aligned}$$

where $\langle\mathtt{e}\rangle_e$ is defined by:

$$\begin{aligned}
\langle\mathtt{e}_1 \mathtt{+} \mathtt{e}_2\rangle_e &= \langle\mathtt{e}_1\rangle_e \ \mathtt{+}\ \langle\mathtt{e}_2\rangle_e\\
\langle\mathtt{e}_1 \mathtt{-} \mathtt{e}_2\rangle_e &= \langle\mathtt{e}_1\rangle_e \ \mathtt{-}\ \langle\mathtt{e}_2\rangle_e\\
\langle\mathtt{e}_1 \mathtt{*} \mathtt{e}_2\rangle_e &= \langle\mathtt{e}_1\rangle_e \ \mathtt{*}\ \langle\mathtt{e}_2\rangle_e\\
\langle\mathtt{e}_1 \,\hat{}\, \mathtt{e}_2\rangle_e &= \langle\mathtt{e}_1\rangle_e \ \hat{}\ \langle\mathtt{e}_2\rangle_e\\
\langle\mathtt{e\ [S]}\rangle_e &= \langle\mathtt{e}\rangle_e\\
\langle\mathtt{v}\rangle_e &= \langle\mathtt{v}\rangle_v\\
\langle\mathtt{n}\rangle_e &= \mathtt{n},
\end{aligned}$$

and $\langle\mathtt{e}\rangle_E$ is defined by:

$$\begin{aligned}
\langle\mathtt{e}_1 \mathtt{+} \mathtt{e}_2\rangle_E &= \langle\mathtt{e}_1\rangle_E \ \mathtt{+}\ \langle\mathtt{e}_2\rangle_E\\
\langle\mathtt{e}_1 \mathtt{-} \mathtt{e}_2\rangle_E &= \langle\mathtt{e}_1\rangle_E \ \mathtt{-}\ \langle\mathtt{e}_2\rangle_E\\
\langle\mathtt{e}_1 \mathtt{*} \mathtt{e}_2\rangle_E &= \langle\mathtt{e}_1\rangle_E \ \mathtt{*}\ \langle\mathtt{e}_2\rangle_E\\
\langle\mathtt{e}_1 \,\hat{}\, \mathtt{e}_2\rangle_E &= \langle\mathtt{e}_1\rangle_E \ \hat{}\ \langle\mathtt{e}_2\rangle_E\\
\langle\mathtt{e\ [S]}\rangle_E &= \langle\mathtt{e}\rangle_E\ \mathtt{[}\langle\mathtt{S}\rangle_s\mathtt{]}\\
\langle\mathtt{v}\rangle_E &= \mathtt{v}\\
\langle\mathtt{n}\rangle_E &= \mathtt{n}.
\end{aligned}$$

Proposition 8 (Correctness of the enreda transformation). *The transformation is* correct *if at all time during the execution the invariant defining the transformation of the memory holds, and when a value is returned, it is either the same as in the original program.*

Formally, let `a` *be an algorithm,* Γ *be a typing of algorithm* `a`*, and* r *a random prime.* $\forall$ *initial memory* m*, for each statement* `s` *in* `a`*, we have either*

$$\begin{tikzcd} m \arrow[r, "{[\![s]\!]_\Gamma}"] \arrow[d, "\langle\cdot\rangle_r"'] & m' \arrow[d, "\langle\cdot\rangle_r"] \\ \langle m\rangle_r \arrow[r, dashed, "{[\![\langle s\rangle_{r,\Gamma}]\!]_{\langle\Gamma\rangle_r}}"'] & \langle m'\rangle_r \end{tikzcd}$$ *during the execution, or*

$$\begin{tikzcd} m \arrow[r, "{[\![s]\!]_\Gamma}"] \arrow[d, "\langle\cdot\rangle_r"'] & v \arrow[d, equal] \\ \langle m\rangle_r \arrow[r, dashed, "{[\![\langle s\rangle_{r,\Gamma}]\!]_{\langle\Gamma\rangle_r}}"'] & v' \end{tikzcd}$$ *when the algorithm terminates.*

We have that the enreda transformation is correct.

Proof. The proof is done by structural induction on algorithm `a`.

Empty program. $[\![\langle \mathtt{skip}\rangle_{r,\Gamma}]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r = \langle m\rangle_r$
and $[\![\mathtt{skip}]\!]_\Gamma\ m = m$, in which case the first diagram trivially holds.

Assignments. For assignments we have

$$\begin{aligned}
&[\![\langle \mathtt{v} := \mathtt{e}\rangle_{r,\Gamma}]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r\\
&= [\![\langle \mathtt{v}\rangle_v := \langle \mathtt{e}\rangle_e \;;\; \mathtt{v} := \langle \mathtt{e}\rangle_E]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r\\
&= [\![\mathtt{v} := \langle \mathtt{e}\rangle_E]\!]_{\langle\Gamma\rangle_r} \circ [\![\langle \mathtt{v}\rangle_v := \langle \mathtt{e}\rangle_e]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r\\
&= [\![\mathtt{v} := \langle \mathtt{e}\rangle_E]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r\{\langle \mathtt{v}\rangle_v \leftarrow [\![\langle \mathtt{e}\rangle_e]\!]_{expr(\Gamma,m)} \bmod \langle\Gamma\rangle_r[\langle \mathtt{v}\rangle_v]\}\\
&= \langle m\rangle_r\{\mathtt{v} \leftarrow [\![\langle \mathtt{e}\rangle_E]\!]_{expr(\langle\Gamma\rangle_r,\langle m\rangle_r)} \bmod \langle\Gamma\rangle_r[\mathtt{v}]\}\\
&\qquad \{\langle \mathtt{v}\rangle_v \leftarrow [\![\langle \mathtt{e}\rangle_e]\!]_{expr(\langle\Gamma\rangle_r,\langle m\rangle_r)} \bmod \langle\Gamma\rangle_r[\langle \mathtt{v}\rangle_v]\}\\
&= \langle m\rangle_r\{\mathtt{v} \leftarrow [\![\langle \mathtt{e}\rangle_E]\!]_{expr(\langle\Gamma\rangle_r,\langle m\rangle_r)} \bmod \Gamma[\mathtt{v}] \times r\}\\
&\qquad \{\langle \mathtt{v}\rangle_v \leftarrow [\![\langle \mathtt{e}\rangle_e]\!]_{expr(\langle\Gamma\rangle_r,\langle m\rangle_r)} \bmod r\}
\end{aligned}$$

and $[\![\mathtt{v} := \mathtt{e}]\!]_\Gamma\ m = m\{\mathtt{v} \leftarrow [\![\mathtt{e}]\!]_{expr(\Gamma,m)} \bmod \Gamma[\mathtt{v}]\}$,
which satisfies the first diagram too.

Returns. When returning a value we have

$$\begin{aligned}
&[\![\langle \mathtt{return\ v}\rangle_{r,\Gamma}]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r\\
&= [\![\mathtt{if\ v\,[}\mathbb{Z}_r\mathtt{]\ =\ }\langle \mathtt{v}\rangle_v\\
&\qquad \mathtt{then\ return\ v\,[}\Gamma[\mathtt{v}]\mathtt{]\ ;}\\
&\qquad \mathtt{else\ return\ error\ ;}]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r\\
&= (\text{if } [\![\mathtt{v\,[}\mathbb{Z}_r\mathtt{]\ =\ }\langle \mathtt{v}\rangle_v]\!]_{cond(\langle\Gamma\rangle_r,\langle m\rangle_r)}\\
&\qquad \text{then } [\![\mathtt{return\ v\,[}\Gamma[\mathtt{v}]\mathtt{]}]\!]_{\langle\Gamma\rangle_r}\\
&\qquad \text{else } [\![\mathtt{return\ error}]\!]_{\langle\Gamma\rangle_r})\langle m\rangle_r\\
&= (\text{if } [\![\mathtt{v\,[}\mathbb{Z}_r\mathtt{]}]\!]_{expr(\langle\Gamma\rangle_r,\langle m\rangle_r)} = [\![\langle \mathtt{v}\rangle_v]\!]_{expr(\langle\Gamma\rangle_r,\langle m\rangle_r)}
\end{aligned}$$

$$\begin{aligned}
&\quad \text{then } [\![\texttt{return v [}\mathit{\Gamma}[\texttt{v}]\texttt{]}]\!]_{\langle \mathit{\Gamma}\rangle_r}\\
&\quad \text{else } [\![\texttt{return error}]\!]_{\langle \mathit{\Gamma}\rangle_r})\langle m\rangle_r\\
&= (\text{if } \langle m\rangle_r[\texttt{v}] \bmod r = \langle m\rangle_r[\langle \texttt{v}\rangle_v]\\
&\quad \text{then } [\![\texttt{return v [}\mathit{\Gamma}[\texttt{v}]\texttt{]}]\!]_{\langle \mathit{\Gamma}\rangle_r}\\
&\quad \text{else } [\![\texttt{return error}]\!]_{\langle \mathit{\Gamma}\rangle_r})\langle m\rangle_r\\
&= [\![\texttt{return v [}\mathit{\Gamma}[\texttt{v}]\texttt{]}]\!]_{\langle \mathit{\Gamma}\rangle_r}\langle m\rangle_r \text{ by induction}\\
&= \langle m\rangle_r[\texttt{v}] \bmod \mathit{\Gamma}[\texttt{v}]\\
&= m[\texttt{v}]
\end{aligned}$$

and $[\![\texttt{return v}]\!]_{\mathit{\Gamma}}\ m = m[\texttt{v}]$, which satisfies the second diagram.

Sequential composition. For the sequential composition we have
$[\![\langle \texttt{s ; a}\rangle_{r,\mathit{\Gamma}}]\!]_{\langle \mathit{\Gamma}\rangle_r}\langle m\rangle_r = [\![\langle \texttt{a}\rangle_{r,\mathit{\Gamma}}]\!]_{\langle \mathit{\Gamma}\rangle_r} \circ [\![\langle \texttt{s}\rangle_{r,\mathit{\Gamma}}]\!]_{\langle \mathit{\Gamma}\rangle_r}\langle m\rangle_r$,
and $[\![\texttt{s ; a}]\!]_{\mathit{\Gamma}}\ m = [\![\texttt{a}]\!]_{\mathit{\Gamma}} \circ [\![\texttt{s}]\!]_{\mathit{\Gamma}}\ m$

If $[\![\texttt{s}]\!]_{\mathit{\Gamma}}\ m$ returns a value, then by induction we know that $[\![\langle \texttt{s}\rangle_{r,\mathit{\Gamma}}]\!]_{\langle \mathit{\Gamma}\rangle_r}\langle m\rangle_r$ does return the same value. In that case by definition of $\circ$ it works.
If $[\![\texttt{s}]\!]_{\mathit{\Gamma}}\ m$ returns a memory m', then by induction we know that $[\![\langle \texttt{s}\rangle_{r,\mathit{\Gamma}}]\!]_{\langle \mathit{\Gamma}\rangle_r}\langle m\rangle_r$ does return the memory $\langle m'\rangle_r$. In that case by induction on the structure of a it works. □