Combined Implementation Attack Resistant Exponentiation

Jörn-Marc Schmidt[1], Michael Tunstall[2], Roberto Avanzi[3], Ilya Kizhvatov[4], Timo Kasper[3], and David Oswald[3]

[1] Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, A–8010 Graz, Austria
joern-marc.schmidt@iaik.tugraz.at
[2] Department of Computer Science, University of Bristol
Merchant Venturers Building, Woodland Road
Bristol BS8 1UB, United Kingdom
tunstall@cs.bris.ac.uk
[3] Ruhr-University Bochum
Horst Görtz Institute for IT Security
Bochum, Germany
{roberto.avanzi, timo.kasper, david.oswald}@rub.de
[4] University of Luxembourg
Computer Science and Communications Research Unit
Luxembourg
ilya.kizhvatov@uni.lu

Abstract. Different types of implementation attacks, like those based on side channel leakage and active fault injection, are often considered as separate threats. Countermeasures are, therefore, often developed and implemented accordingly. However, Amiel et al. showed that an adversary can successfully combine two attack methods to overcome such countermeasures. In this paper, we consider instances of these combined attacks applied to RSA and elliptic curve-based cryptosystems. We show how previously proposed countermeasures may fail to thwart these attacks, and propose a countermeasure that protects the variables in a generic exponentiation algorithm in the same scenario.

Keywords: Combined Implementation Attacks, Countermeasures, Infective Computation, RSA, ECC.

1 Introduction

In order to provide security on an embedded microprocessor, several different threats have to be taken into account. In particular, countermeasures need to be included to prevent implementation attacks, which try to benefit from examining a device and its behavior.

Implementation attacks can be classed as either passive or active. Passive attacks collect information to overcome the security of a device. Such information can be the time a computation takes [1], the power consumption [2], or

electromagnetic emanations [3]. This information can enable a successful attack, if it correlates with secret data used within computations.

Active attacks try to influence the behavior of a device and determine sensitive information by examining the effects. The active attack we will be considering in this paper is fault analysis, where an attacker attempts to stress a device when it is executing a cryptographic algorithm so that an error is made during the computation [4,5].

The two threat scenarios described above are often considered separately, and many different methods have been considered to protect public key primitives from either side channel or fault analysis. In recent years, some examples of unified countermeasures that protect against both threats at the same time have been proposed for elliptic curve based cryptosystems using ring extensions [6] and for RSA when computed using the Chinese Remainder Theorem [7]. However, in this paper we consider generic exponentiation algorithms.

In 2007, Amiel et al. [8] presented a combined attack on a side channel resistant implementation of the square and multiply algorithm. In this paper, we review and generalize such combined implementation attacks. Furthermore, we propose a method of computing an exponentiation that is resistant to both side channel and fault analysis as well as their combination. The idea is somewhat similar to the idea of *infective computation* [9], where faults in otherwise redundant functions will affect the result of an algorithm.

In this paper, we present a countermeasure that diffuses cryptographic keys and masks intermediate values during computation. As a result, tampering with these values randomizes the computation. Some of the techniques described extend the hardening methods proposed in [10], and demonstrate their applicability when several attack methods are combined.

The remainder of this paper is organized as follows: In Section 2 we lay the background, recalling the necessary facts about exponentiation and scalar multiplication, and describe our attack model. Section 3 is devoted to combined attacks: we described previous attacks, generalize them, and describe a few countermeasures from the literature. Our countermeasure is described in Section 4, for RSA and curve-based cryptosystems. Finally, in Section 5 we conclude.

2 Preliminaries

In this section we define the algorithms that we will be considering as targets and the type of fault an attacker is able to induce.

2.1 Generic Side Channel Resistant Exponentiation

The simplest algorithm for computing an exponentiation is the square and multiply algorithm. This is where an exponent is read left-to-right bit-by-bit, a zero results in a squaring operation being performed, and a one results in a squaring operation followed by a multiplication with the original basis.

Algorithm 1 Left-to-Right Square and Multiply Algorithm

Require: $x \in \mathbb{G}$, $\nu \geq 1$, ℓ the binary length of ν (i.e. $2^{\ell-1} \leq \nu < 2^{\ell}$)
Ensure: $z = x^{\nu}$
1: Construct a random semigroup $\mathbb{H}$, an injective map $\chi : \mathbb{G} \to \mathbb{H}$ satisfying (1).
2: $R_0 \leftarrow \chi(1)$ // *where* 1 *is the neutral element of* $\mathbb{G}$
3: $R_1 \leftarrow \chi(x)$
4: $k \leftarrow 0$
5: $j \leftarrow \ell - 1$
6: **while** $j \geq 0$ **do**
7: $\quad R_0 \leftarrow R_0 \circ R_k$
8: $\quad k \leftarrow k \oplus \texttt{bit}(\nu, j)$
9: $\quad j \leftarrow j - \neg k$
10: **end while**
11: $z \leftarrow \chi^{-1}(R_0)$ // *the value held in* R_0 *is mapped back to* $\mathbb{G}$
12: **return** z

Algorithm 1 is a square and multiply algorithm where the exponentiation is computed only using one form of the group operation, denoted $\circ$. In other words, an attacker will not be able to distinguish whether the operands are distinct or equal elements by observing a side channel. This is referred to as side channel atomicity [11].

In Algorithm 1 the input is an element x in a (multiplicatively written) group $\mathbb{G}$ and a positive ℓ-bit integer $\nu = (\nu_{\ell-1}, \ldots, \nu_0)$; the output is the element $z = x^{\nu}$ in $\mathbb{G}$. We also have an injective function $\chi : \mathbb{G} \to \mathbb{H}$ of $\mathbb{G}$ into a semigroup $\mathbb{H}$, where both $\mathbb{H}$ and χ can be constructed at run time in a random manner. We denote the semigroup operation in $\mathbb{H}$ by $\circ$ as well, and we assume that, in practice, it is performed in a very similar way to the operation in $\mathbb{G}$.

It is assumed that the representation of the elements of $\mathbb{H}$ looks "randomized" with respect to the representation of the original elements of $\mathbb{G}$. Being injective, the function χ is, of course, invertible, but we also require that the inverse function be *efficiently* computable and satisfy the property that

$$\chi^{-1}\big(\chi(a) \circ \chi(b)\big) = a \circ b \quad \text{for all } a, b \in \mathbb{G} \ . \tag{1}$$

The neutral element of $\mathbb{G}$ is not necessarily mapped to a neutral element of $\mathbb{H}$ (which we do not even assume to exist—although in many practical examples it will). The input x is, therefore, blinded at the beginning of the algorithm by transferring the computation to $\mathbb{H}$ using χ. From (1) we get that $\chi^{-1}(y^n) = x^n$ with $y = \chi(x)$, i.e. pulling back the result will yield the correct result. We do not assume, *a priori*, that the function χ is deterministic (i.e. two distinct applications of χ to an element $x \in \mathbb{G}$ could yield different elements of $\mathbb{H}$).

The function $\texttt{bit}(\beta, i)$ returns the i-th bit of β.

RSA: One use of such an algorithm is to implement RSA, where all computations usually take place in the group $\mathbb{Z}_n^*$ (i.e. the set of elements of the ring $\mathbb{Z}_n$ that are invertible with respect to multiplication, where we use the notation

$\mathbb{Z}_n$ as a shorthand for $\mathbb{Z}/n\mathbb{Z}$), and the security of the algorithm is based on the problem of factoring the product of two large primes. Let p and q be such primes and $n = p\,q$ to be their product. The public key e is an integer coprime to $\varphi(n) = (p-1)(q-1)$, where φ is Euler's totient function, and its corresponding secret key d is $d = e^{-1} \bmod \varphi(n)$.

Due to this construction, $m = (m^e)^d \bmod n$ holds for any $m \in \mathbb{Z}_n^*$, which provides the basis for the system. An encrypted message $c = m^e \bmod n$ can be decrypted by computing $m = c^d \bmod n$, a signature $s = m^d \bmod n$ can be verified by checking that $m = s^e \bmod n$ holds.

In computing a modular exponentiation for RSA (using Algorithm 1) $\mathbb{Z}_n$ is extended by multiplying n by some random value r_2, so that the computation takes place modulo some multiple of n (this was first suggested by Shamir [12]). To map R_0 and R_1 we can, for instance, just add some random multiple of n to 1 and x, i.e. in this case $\chi(x) = x + \texttt{random()} \cdot n \in \mathbb{Z}_{nr_2}$, where `random()` returns some random integer.

Thus, the assumption that the representation of R_0 and R_1 will be to randomized with respect to those of 1 and x is fulfilled. Note that the image of $\mathbb{Z}_n^*$ in $\mathbb{Z}_{nr_2}$ will not necessarily be a group, but the value held in R_0 mapped back to $\mathbb{Z}_n^*$ by reducing R_0 modulo n will be correct by (1).

Elliptic Curve Cryptography: An elliptic curve is usually given, for cryptographic purposes, by its equation. The most common form is the *Weierstraß model*, which, for curves defined over prime fields, takes the form $y^2 = x^3 + a\,x + b$ where a and b are constants. Other models are also possible, notably the *Edwards form* $x^2 + y^2 = c^2\,(1 + d\,x^2\,y^2)$ where c and d are constants. In order to apply Algorithm 1 we need to construct a group using the curve: this group $\mathbb{G}$ is just the set of points on the curve (with one additional element, the point at infinity being the neutral element if the curve is given in the Weierstraß model).

The fundamental operation on cryptosystems designed around elliptic curves is the scalar multiplication in a prime order subgroup of the rational point group of an elliptic curve over a finite field: given a point P and an integer (referred to as a scalar) ν, compute $\nu \cdot P = P + P + \cdots + P$ (ν summands). In Algorithm 1 the operation $\circ$ is the addition of two points. The embedding in a larger semigroup will be discussed in detail in § 4.2.

Further Countermeasures: In Algorithm 1, and the two specific examples given above, we can note that the exponent ν is not modified by the algorithm. Typically, one would modify the exponent at the beginning of the algorithm such that the bits of the exponent vary from one execution to the next but the correct result is always produced. For example, for RSA we could define

$$\nu' = \nu + w\,\varphi(n)\,,$$

using the notation above. We define w as a random integer of a suitable bit length that is multiplied by the order of the exponent's group (see [13] for a detailed

explanation). An equivalent expression for elliptic curves is $\nu' = \nu + w\,\#(\mathbb{G})$ where $\#(\mathbb{G})$ is the number of points on the curve.

In this paper we assume that the exponent (or scalar) is blinded, so we do not consider attacks where an attacker is required to derive an exponent through repeated observations of the same exponent, e.g. as it is the case for differential attacks. We consider that an attacker is obliged to derive the exponent from one observation.

2.2 Fault Attack Model

There are several known mechanisms for injecting faults into microprocessors that vary from a glitch in the power supply [4] to using laser light [14]. The following models for the faults that can be created by this method will be used to evaluate the potential for a fault attack to allow a side channel attack to be conducted.

Data randomization: the adversary could change data to a random value. However, the adversary does not control the random value and the new value of the data is unknown to the adversary.

Resetting data: the adversary could force data to the blank state, i.e., reset the bits of a given byte, or bytes, of data back to all zeros or all ones depending on the logical representation.

Modifying opcodes: the adversary could change the instructions executed by the chip's CPU. Additional effects to those given above could include the removal of functions or the breaking of loops.

These three types of attack cover everything that an attacker could hope to do to an implementation of an algorithm. When describing the effectiveness of our countermeasure, we will consider the above models for first-order faults, i.e. an adversary can inject at most one fault during each execution of the algorithm. However, the attacks described in this paper will be based on a fault model derived from the experiments detailed in [8], where it is shown that a fault can be used to affect the bit length of a given variable. In a secure microprocessor the manipulation of a value with a large bit length is typically conducted using a coprocessor. These values will have to be loaded into the coprocessor from a given location of memory over a bus that will only be able to transfer a certain number of bits in one clock cycle. We shall thus consider faults that tamper with this loading process to produce one of the following effects:

1. That a variable is partially or entirely prevented from loading. A smaller amount of data or no data at all is loaded in the coprocessor. The possible effects are that this register is partially or completely zeroed.
2. The memory location from which the data has to be loaded has been tampered with, resulting in the loading of wrong data in the register—this data can be zero in realistic scenarios (for instance, if the location is outside the memory effectively implemented in the embedded device).

Note that there are different ways of achieving these goals (one being the modification of instructions being performed, as noted above). These effects have been shown to be possible experimentally in [8].

3 Combined Attacks

In this section, we present the existing work on combined attacks and generalize the combined attacks. We then describe why many recent countermeasures designed to thwart side channel and faults attacks do not fully protect an exponentiation algorithm.

3.1 Previous Work

The term *Passive and Active Combined Attacks* (PACA) was introduced by Amiel et al. [8]. In this work, the authors target a classical side channel resistant implementation of RSA, such as the algorithm described in Section 2—with the notation of § 2.1, i.e. the operation is an integer multiplication.

The authors note that fault resistance can be added to the implementation by placing a classical checking routine that verifies the result at the end of the computations by raising it to the public exponent e modulo n and comparing the result to the initial message m. In the following paragraph the combined implementation attack of [8] is described using Algorithm 1.

Let us consider attacks on RSA, where the operation is a simple multiplication. If a fault is injected to bypass Line 2 of Algorithm 1, which leaves the R_0 uninitialized, which would mean (on many architectures) that R_0 would be set to 0. After a successful fault, the operation in Line 7 of Algorithm 1 would be either $R_0 \cdot R_0 = 0 \cdot 0$ or $R_0 \cdot R_1 = 0 \cdot R_1$ ($R_1 \neq 0$). The authors report that these two patterns can be identified in a power consumption trace. So a classical SPA attack could be applied, and the knowledge of the sequence of squaring operations and multiplications would lead to the recovery of the private exponent d. Note that the invocation of a fault checking routine *after* the exponentiation has taken place will not affect the attack at all.

Along with the combined attack, a countermeasure was proposed in [8] under the name of Detect and Derive, but it does not prevent combined implementation attacks, as we will describe below.

3.2 Generalisation to Elliptic Curve Scalar Multiplication

As described above, combined implementation attacks on a side channel resistant modular exponentiation are based on making operations distinguishable in a side channel. Faults causing different effects could be used to cause a difference in a side channel, where the possible range is platform-dependent. For example, in Algorithm 1 a fault reducing R_0 to some small value (i.e. setting the most significant bytes of R_0 to 0 an implementation of an side channel attack based

on this is described in [15]) will also suffice on platforms where multiplication of short operands is distinguishable from that of long operands.

In order to demonstrate that the same attack can be applied to elliptic curve scalar multiplication, we consider a recent explicit addition formula described in [16]. The curve is given in Edwards form, its equation is therefore $x^2 + y^2 = c^2\,(1 + d\,x^2\,y^2)$ where c and d are constants. One can often take $c = 1$ and d small for performance reasons. Three coordinates are used, $(X_1 : Y_1 : Z_1)$ with $X_1, Y_1, Z_1 \neq 0$, to represent the point $(Z_1/X_1, Z_1/Y_1)$ on the Edwards curve. These coordinates are called *inverted Edwards coordinates*.

For any two input points given as $(X_1 : Y_1 : Z_1)$, $(X_2 : Y_2 : Z_2)$, their sum $(X_3 : Y_3 : Z_3)$ is computed by the following sequence of operations:

$$\begin{gathered} A = Z_1 \cdot Z_2 \;\;;\;\; B = d \cdot A^2 \;\;;\;\; C = X_1 \cdot X_2 \;\;;\;\; D = Y_1 \cdot Y_2 \;\;;\;\; E = CD \;\;; \\ H = C - D \;\;;\;\; I = (X_1 + Y_1) \cdot (X_2 + Y_2) - C - D \;\;; \\ X_3 = c \cdot (E + B) \cdot H \;\;;\;\; Y_3 = c \cdot (E - B) \cdot I \;\;;\;\; Z_3 = A \cdot H \cdot I \;\;. \end{gathered}$$

This is a strongly unified formula for addition of any two points on the curve, i.e., it works if the two inputs are equal. This makes, in theory, the group operations indistinguishable when observed using a side channel.

Now, assume the coordinate of one input is faulted, such that it is, for example, zeroed. Then, one of the multiplications to obtain A, C, or D will be a multiplication by zero. However, if we are doubling a point, i.e. we are adding a point to itself, *both* inputs to this multiplication will be zero and will be visible in a side channel. This would allow an attacker to distinguish between an addition and a doubling operation and derive the scalar used.

All other aspects of the scalar multiplication are essentially the same as in an RSA exponentiation and similar faulting scenarios can be envisioned.

3.3 Countermeasures

Despite the implementation described in [8], recent works on countermeasures for public-key algorithms do not always consider the possibility of a combined attack. As a consequence, many of the recent countermeasures are vulnerable to combined attacks. Here we briefly list some of these countermeasures and discuss their resistance to combined implementation attacks.

We begin with the countermeasure Detect and Derive from [8], which is based on the principle of infective computation introduced in [9]. The basic idea of Detect and Derive is to modify (infect) the secret exponent d with some check value that is dependent on the input variables that should not be modified by a fault (e.g. R_0 in Algorithm 1). The bits of the exponent are then recovered on-the-fly using the value held in R_0. One would expect that, whenever R_0 is modified by a fault, the bits of d would be recovered incorrectly and any information recovered using a side channel would be incorrect. However, the Detect and Derive countermeasure only protects the exponent from being manipulated. This is because the value held in R_0 has to be computed *before* the check value P_1.

So, if R_0 is modified by a fault *before* the computation of the check value, d will still be correct and may be visible in a side channel.

Other countermeasures, such as those proposed by Kim et al. [7,17], and the improvements described in [18], are also vulnerable to combined implementation attacks, as they do not change the core exponentiation algorithm. This is because, in our scenario, checking the correct result has been computed at the end of the exponentiation is not sufficient to prevent a combined implementation attack.

Algorithm 2 Atomic double modular exponentiation from [19]

Require: An element $m \in \mathbb{Z}_N$, a chain $\omega(a, b)$ s.t. $a \leq b$, a modulus N
Ensure: The pair of modular powers $(m^a \bmod N, m^b \bmod N)$
1: $R_{(0,0)} \leftarrow 1; R_{(0,1)} \leftarrow m; R_{(1,0)} \leftarrow m$
2: $\gamma \leftarrow 1; \mu \leftarrow 1; i \leftarrow 0$
3: **while** $i < n$ **do**
4: $\quad t \leftarrow \omega_i \wedge \mu; v \leftarrow \omega_{i+1} \wedge \mu$
5: $\quad R_{(0,\gamma\oplus t)} \leftarrow R_{(0,\gamma\oplus t)} \cdot R_{((\mu\oplus 1),\gamma\oplus\mu)} \bmod N$
6: $\quad \mu \leftarrow t \vee (v \oplus 1); \gamma \leftarrow \gamma \oplus t$
7: $\quad i \leftarrow i + \mu + \mu \wedge (t \oplus 1)$
8: **end while**
9: **return** $(R_{\gamma\oplus 1}, R_\gamma)$

The atomic double modular exponentiation of Rivain [19], which is depicted in Algorithm 2, is also potentially vulnerable to a combined implementation attack. We can observe that the value $R_{(1,0)}$ in this algorithm is conditionally used as a multiplier in Line 5. If this value is set to a small value by a fault in Line 1, a side channel trace should reveal the addition chain $\omega(a, b)$ and, therefore, the exponents a and b.

The only countermeasure that we are aware of which will resist a combined implementation attack is described in [20], where an algorithm is presented that is based on the Montgomery Ladder [21]. We can note that this countermeasure provides a method of protecting the intermediate variables of an algorithm. However, it does not protect the exponent from being modified and the integrity of the exponent needs to be verified separately[5]. In the following section we describe a generic countermeasure that also protects the exponent.

4 A Novel Countermeasure against Combined Attacks

In order to protect exponentiation algorithms against combined implementation attacks, we propose a countermeasure, based on *infective computation*, that implicitly checks whether an adversary has tampered with intermediate variables

[5] The same holds for variants [22,23] of Giraud's method [20].

in each loop iteration. If a fault is injected at any point in the algorithm, the remaining bits of the exponent are randomized, so that no exploitable side channel information is leaked.

Our countermeasure involves two ideas. Firstly, all intermediate computations are performed over an extension $\mathbb{H}$ of the group $\mathbb{G}$, as described in Section 2. For the transformation a function like the χ function introduced in § 2.1 is used. The computations take place in a subset of $\mathbb{H}$ and the result is mapped back to $\mathbb{G}$ at the end of the algorithm. The extension $\mathbb{H}$ is based on a random number r_2 that is generated during an initialization phase. In $\mathbb{H}$, we define a checkable property which a "legal" (i.e., correct) value fulfills, but which is violated by a fault with a high probability. An intermediate value is accepted as legal, if it is equal to 0 mod r_2. If r_2 is sufficiently large, a tampered value will violate this condition with a very high probability. This checkable property is preserved by multiplications and additions, and thus holds for all cryptographic primitives we consider.

Secondly, a method that checks the validity of the intermediate values and that immediately randomizes the algorithm when it detects a fault is required. We propose to encode the exponent d in an initialization phase and decode it during the computation. The correct decoding depends on the defined property of the intermediate values. Therefore, the exponent is divided into ℓ blocks of W bits: $d \rightarrow [d^{(0)}, \ldots, d^{(\ell-1)}]$. A function ψ is applied to each block of d in the initialization phase, yielding ℓ encoded blocks $[\tilde{d}^{(0)}, \ldots, \tilde{d}^{(\ell-1)}]$:

$$[\tilde{d}^{(0)}, \ldots, \tilde{d}^{(\ell-1)}] \leftarrow [\psi_0(d^{(0)}), \ldots, \psi_0(d^{(\ell-1)})] \ .$$

During the exponentiation, the encoding is removed by applying the inverse function ψ^{-1}. This is done iteratively, i.e., one block $\tilde{d}^{(j)}$ is decoded just before its bits are processed during the exponentiation. Naturally, the inversion ψ^{-1} must be efficient and provide sufficient diffusion capabilities in case of a fault. ψ^{-1} delivers the correct decoding, i.e., $\hat{d}$ equals d, if, and only if, a check value α (i.e. the parameter of the function ψ) is equal to zero:

$$[\hat{d}^{(0)}, \ldots, \hat{d}^{(\ell-1)}] \leftarrow [\psi_\alpha^{-1}(\tilde{d}^{(0)}), \ldots, \psi_\alpha^{-1}(\tilde{d}^{(\ell-1)})] \ .$$

The link between the property of legal intermediate values and the correct decoding of the exponent must, therefore, be established by computing α in such a way that it equals zero only if the check condition is fulfilled. The actual realization depends on the ring or group used.

In the following, we describe this property and the corresponding check functions in greater detail for RSA exponentiation in the ring $\mathbb{Z}_n$ and for scalar multiplication in the point group of an elliptic curve. We shall also discuss the security of the protected algorithms and estimate the performance impact of our countermeasures.

4.1 RSA

A possible solution to provide a checkable property is to make the intermediate values a multiple of a random, non-trivial idempotent factor i, i.e., an element

for which $i^2 = i$ holds but $i \neq 0$ or 1. The idea of using idempotent elements for constructing a code was first introduced by Proudler [24]. His so-called AN code was analyzed and extended for fault detection purposes in [10,25]. In contrast to the previous suggestions, we use the properties of the idempotent not as error detection code for the whole algorithm, but implicitly check intermediates in each step of the algorithm.

In this section, we detail the construction of an idempotent element and its integration into the exponentiation algorithm. Let r_2 be a random value, coprime to the RSA modulus n. The idempotent element i is an integer modulo $n\,r_2$ with the property that $i \equiv 1 \pmod{n}$ and $i \equiv 0 \pmod{r_2}$. Using the Chinese Remainder Theorem (CRT) we can easily find

$$i = (r_2^{-1} \bmod n) \cdot r_2 \ .$$

However, this expression is only of use if r_2 is invertible modulo n, but since n is assumed to be the product of large primes, and r_2 is chosen to be comparatively small, this is not a problem. According to the defined properties of i and the CRT, $i = i^2 \bmod n\,r_2$ holds.

In the following, all intermediate values are elements of $\mathbb{Z}_{n\,r_2}$. We define the embedding of $\mathbb{Z}_n^*$ into $\mathbb{Z}_{n\,r_2}$ as follows

$$\chi \ : \ \mathbb{Z}_n^* \to \mathbb{Z}_{n\,r_2} \ , \ x \to i \cdot x \bmod n\,r_2 \ .$$

Besides, $\chi(x) \cdot \chi(y) = i \cdot x \cdot i \cdot y = i \cdot x \cdot y = \chi(x \cdot y)$ holds. This is thus an instance of the function χ introduced in § 2.1.

For encoding the exponent d, the block length W is chosen to be smaller than the binary length of r_2, i.e., $2^W < r_2$. Then, $\psi : \mathbb{Z}_{r_2} \times \mathbb{Z}_{r_2} \to \mathbb{Z}_{r_2}$ is defined as:

$$\psi_\alpha(d^{(j)}) = (\alpha + n)^{-1} \cdot d^{(j)} \bmod r_2$$

The inverse $\psi^{-1} : \mathbb{Z}_{r_2} \times \mathbb{Z}_{r_2} \to \mathbb{Z}_{r_2}$ is given as:

$$\psi_\alpha^{-1}(\tilde{d}^{(l)}) = (\alpha + n) \cdot \tilde{d}^{(l)} \bmod r_2$$

Following the above construction of the idempotent element i, and the χ function, each legal value equals 0 mod r_2, since $i = 0 \bmod r_2$. This also holds for the sum of two intermediate values R_0 and R_1, i.e., $R_0 + R_1 = 0 \bmod r_2$, because of the way we chose our particular χ. Checking the validity of the intermediates can, therefore, be conducted by applying the ψ^{-1} operation to $\alpha = R_0 + R_1$. The computation of n^{-1} can be done together with r_2^{-1} in the initialization phase.

Algorithm 3 shows the whole countermeasure. It comprises exponent blinding, i.e., the addition of a multiple r_1 of the group order $\varphi(n)$. We define the function `random`$(1, 2^\lambda - 1)$ as returning a random integer $\in \{1, \ldots, 2^\lambda - 1\}$. The blinding is required since an adversary can always recover one bit of information, as shown in the previous section[6].

[6] The discussion of the bit length governed by λ is beyond the scope of this paper. The interested reader is referred to [26].

Algorithm 3 Resistant Left-to-Right Square and Multiply Algorithm

Require: $d = (d_{t-1}, \ldots, d_0)_2$, $m \in \mathbb{Z}_n$, n, block length W
Ensure: $m^d \bmod n$

$r_1 \leftarrow \texttt{random}(1, 2^\lambda - 1)$
$r_2 \leftarrow \texttt{random}(1, 2^\lambda - 1)$
$i \leftarrow (r_2^{-1} \bmod n) \cdot r_2$
$R_0 \leftarrow i \cdot 1 \bmod n\, r_2$ *// equals* $\chi(1)$
$R_1 \leftarrow i \cdot m \bmod n\, r_2$ *// equals* $\chi(m)$
$d \leftarrow d + r_1 \cdot \varphi(n)$
$[\tilde{d}^{(0)}, \ldots, \tilde{d}^{(\ell-1)}] \leftarrow [\psi_0(d^{(0)}), \ldots, \psi_0(d^{(\ell-1)})]$ *// encode the ℓ blocks of d*
$k \leftarrow 0$
$j \leftarrow \texttt{bit-length}(\tilde{d})$-1
while $j \geq 0$ **do**
  $R_0 \leftarrow R_0 \cdot R_k \bmod n\, r_2$
  if $(R_0 = 0)$ or $(R_1 = 0)$ **then**
    $[\tilde{d}^{(0)}, \ldots, \tilde{d}^{(\ell-1)}] \leftarrow [1, \ldots, 1]$ *// diffuse exponent*
  end if
  $\hat{d} \leftarrow \psi^{-1}_{(R_0 + R_1 \bmod r_2)}(\tilde{d}^{(\lfloor j/W \rfloor)})$ *// decode the current block*
  $k \leftarrow k \oplus \texttt{bit}(\hat{d}, j \bmod W)$ *// use only the desired bit*
  $j \leftarrow j - \neg k$
end while
$c \leftarrow R_0 \bmod n$
return c

4.2 Elliptic Curve Exponentiation

In this section we describe a variant of our countermeasure for elliptic curve exponentiation, i.e., scalar multiplication. We want to emphasize the fact that this countermeasure also works for hyperelliptic curves with almost no changes, and can be adapted to any group whose group operations can be described as a sequence of finite field additions, subtractions and multiplications—for example, Trace Zero Varieties [27].

Some popular countermeasures against fault attacks on elliptic curve cryptosystems over prime fields work by "enlarging" the group in which computations are performed (for example, see [28]). In these countermeasures one does not work on a curve $E(\mathbb{F}_p)$, one considers a curve $E'(\mathbb{Z}_n)$ with $n = p\,r$, where r is another prime, such that the coefficients of E', when reduced mod p are just the coefficients of E. This amounts to computing in the group $E(\mathbb{F}_p) \times E''(\mathbb{F}_r)$, where E'' is the elliptic curve over $\mathbb{F}_r$ obtained by reducing the coefficients of E' modulo r. For these countermeasures the group order of $E''(\mathbb{F}_r)$ needs to be known.

One intuitive attempt to adapt the countermeasure of § 4.1 would be to try to work in such a product of elliptic curve groups. There, the role of our idempotent element i would be played by a non-trivial point I on $E'(\mathbb{Z}_{p\,r})$ that projects to points equal to their doubles on $E(\mathbb{F}_p)$ and $E''(\mathbb{F}_r)$—these must be then the

neutral group elements, and thus such a point I does not exist apart from 0 in $\mathbb{Z}_{p\,r}$.

Instead, we opt for a more straightforward adaptation of the method from § 4.1. As before, given the definition field $\mathbb{F}_p$ of the curve E, we choose a random r_2 (that does not even need to be a prime) and embed the field $\mathbb{F}_p$ into the ring $\mathbb{Z}_{p\,r_2}$. The idempotent i and the functions χ, ψ are defined as in the RSA case, but with p in place of n.

We assume that we use an inversion-free coordinate system for elliptic curves that can represent the neutral element $\mathcal{O}$ and treat it like any other point. Any point P on the elliptic curve then has a representation with s components $(\pi_1, \ldots, \pi_s)$, where $\pi_1, \ldots, \pi_s \in \mathbb{Z}$.

We observe that $\chi(x) \cdot \chi(y) = i \cdot x \cdot i \cdot y = i \cdot x \cdot y = \chi(x \cdot y)$ and $\chi(x) + \chi(y) = i \cdot x + i \cdot y = i \cdot (x + y) = \chi(x + y)$ holds. In other words, this particular χ is a homomorphism of rings without unit. A group operation on a curve can, therefore, be split into a sequence of field operations, and these are just multiplications, additions and subtractions, but not field inversions[7]. Hence, all computations we need to perform on $\mathbb{Z}_{p\,r_2}$ are well defined. We can, therefore, transfer (using the map i) the computation to the larger ring and reduce modulo p at the end of the computation. This will return the same result as would be produced if all computations had been done in $\mathbb{F}_p$. The validity check, that is, the computation of $\tilde{\alpha}$, is performed by summing over all components of the two intermediate points $R_0 = (\rho_{0,1}, \ldots, \rho_{0,s})$, $R_1 = (\rho_{1,1}, \ldots, \rho_{1,s})$.

The protected scalar multiplication algorithm is given in Algorithm 4.

4.3 Security Considerations

In this section we discuss the effects of the possible fault attacks described in § 2.2, in order to show that our method prevents an adversary from successfully applying a combined implementation attack.

RSA: The secret exponent in Algorithm 3 is blinded by a random value r_1 and the intermediate variables by r_2. As noted in § 2.1, side channel analysis is not directly possible. A random manipulation of r_1 or r_2 in Lines 1 and 2 does not improve the situation for an adversary, since both values are randomly generated. Erasing r_1 in Line 1 totally, or in parts, or manipulating Line 6 will, at most, remove the exponent blinding. However, the computation is still blinded by r_2. A modification of r_2 cannot remove the blinding, since reducing the size of r_2 will result in a (partial) erasure of the exponent because of the modular reduction in ψ. A partial erasure of r_2 that does not influence its size does not help, since the exponent is still blinded by r_1.

Setting any other value to zero is detected in Line 12, a manipulation in Lines 3-5 or 11 will be detected. If a fault produces a random value it will not be

[7] Except for a final field inversion to convert the final result to affine coordinates, but this latter operation is done outside the part of the scalar multiplication that needs to be protected.

Algorithm 4 Resistant Left-to-Right Binary Double and Add Algorithm

Require: $d = (d_{t-1}, \ldots, d_0)_2$, $P \in E(\mathbb{F}_p)$, block length W, curve-order $\#E(\mathbb{F}_p)$
Representation of $P = (\pi_1, \ldots, \pi_s)$
Representation of neutral element $\mathcal{O} = (o_1, \ldots, o_s)$
Ensure: $d \cdot P \in E(\mathbb{F}_p)$
1: $r_1 \leftarrow$ `random`$(1, 2^\lambda - 1)$
2: $r_2 \leftarrow$ `random`$(1, 2^\lambda - 1)$
3: $i \leftarrow (r_2^{-1} \bmod p) \cdot r_2$
4: $R_0 = (\rho_{0,1}, \ldots, \rho_{0,s}) \leftarrow (i \cdot o_1, \ldots, i \cdot o_s)$
5: $R_1 = (\rho_{1,1}, \ldots, \rho_{1,s}) \leftarrow (i \cdot \pi_1, \ldots, i \cdot \pi_s)$
6: $d \leftarrow d + r_1 \cdot \#E(\mathbb{F}_p)$
7: $[\tilde{d}^{(0)}, \ldots, \tilde{d}^{(\ell-1)}] \leftarrow [\psi_0(d^{(0)}), \ldots, \psi_0(d^{(\ell-1)})]$ *// encode the ℓ blocks of d*
8: $k \leftarrow 0$
9: $j \leftarrow$ `bit-length`$(\tilde{d})$-1
10: **while** $j \geq 0$ **do**
11: $R_0 \leftarrow R_0 + R_k \in E(\mathbb{Z}_{pr_2})$
12: **if** some selected coordinates of R_0 or R_1 are 0 **then**
13: $[\tilde{d}_0, \ldots, \tilde{d}_{t-1}] \leftarrow [1, \ldots, 1]$ *// diffuse scalar*
14: **end if**
15: $\hat{d} \leftarrow \psi^{-1}_{\sum_{\nu=1}^{s}(\rho_{0,\nu}+\rho_{1,\nu})}(\tilde{d}^{(\lfloor j/W \rfloor)})$ *// decode the current block*
16: $k \leftarrow k \oplus$ `bit`$(\hat{d}, j \bmod W)$ *// use only the desired bit*
17: $j \leftarrow j - \neg k$
18: **end while**
19: $C \leftarrow R_0 \bmod p$ *// component-wise modular reduction*
20: **return** `affine`(C) *// return the affine representation of C*

detected with a probability of $\frac{1}{r_2}$. Any detection will result in the exponent being diffused. In addition, a fault in R_0 is automatically remasked when multiplied with R_1. Manipulating the computation of the initial calculation of ψ (Line 7) is not an option, since it would also destroy the exponent.

The counter value j in Line 9 or in Line 17 of the while loop does not influence the vulnerability of the algorithm to combined attacks, but has to be secured to prevent fault analysis, by having a redundant counter where, for example, we have a counter j and its complement $\bar{j}$. One would then need to verify that the bits of $j \oplus \bar{j}$ consists entirely of ones.

Manipulating k in Line 8 or Line 16 can reveal at most one bit of information in the presence of classical countermeasures that suppress the output. Since we consider that exponent blinding, this does not increase the chances of an adversary recovering the exponent, which also hold for a given exponent (Line 15). The instructions after the while loop do not contain any valuable information that would be of interest to an attacker.

In summary, an adversary has only a chance of $\frac{1}{r_2}$ to retrieve more than one bit information during a computation.

Elliptic Curve Cryptography Algorithm 4 is mostly an adaption of Algorithm 3 and, in general, similar considerations hold. The probability of success for an adversary is also $\frac{1}{r_2}$, which also holds if a register is only partially zeroed. In fact, all the coordinates of R_0 and R_1 are supposed to be equivalent to 0 (mod r_2) in the case of a non-faulted computation. This also applies to the sum of R_0 and R_1, but, in general, their sum will be a random value if the computation has been faulted. Hence, the scalar will be randomized from that point on with probability $1 - \frac{1}{r_2}$.

Zero Coordinates. Passive attacks that use vanishing coordinates, as first described in [29], are discussed here in the context of combined implementation attacks.

When considering inverted Edwards coordinates there are no points with a zero coordinate. So in Line 12 we can simply verify if a coordinate is equal to zero and then diffuse the scalar.

If we are using a different coordinate system, such as affine, projective, Jacobian, etc. we may have points with zero coordinates. Most of these coordinate systems are based on the Weierstraß form of an elliptic curve and we can, therefore, assume that first two coordinates are typically multiples of the affine coordinates x and y.

Manipulating a point will, in effect, transfer the computation to a different curve [30]. If the attacker erases a coordinate, the corresponding modified point will not be detected at Line 12. Even with unified addition and doubling formulae, the attacker can, therefore, distinguish an addition from a doubling operation, because, in the first case, a multiplication with only one zeroed input will be performed in the formula. In the second case, the two inputs will be the same allowing one bit to be derived.

Blinding the scalar by adding a random multiple of the order of the curve is typically considered a good countermeasure, making the observed bit useless. However, this is not entirely correct if we consider commonly used prime fields where there are lots of consecutive zeros[8].

Also, in this case we may want to ensure that there are no points with zero coordinates on the original curve. Points with a vanishing y-coordinate are not a concern: since they have order two, they cannot appear in the cryptographically relevant group of large prime order. To avoid points with a vanishing x-coordinate it is enough to translate the curve along the x-axis (the explicit formulae may have to be adapted, usually in a straightforward way). In Line 12 it will then suffice to test if any coordinate is zero to prevent this attack scenario. Other, more sophisticated randomization methods for the case of hyperelliptic curves are discussed in [31].

Other Attacks. There are, of course, other types of attacks. For example, sign-change attacks would not be detectable by our form of point blinding by field

[8] The impact of the use of these fields is beyond the scope of this paper. Again, the interested reader is referred to [26] for a discussion of this topic.

enlargement. However, these attacks work by reconstructing the scalar from the least significant bit onwards, hence scalar blinding is enough to thwart them, even if the scalar is quite close to a power of two.

4.4 Performance Analysis

The incorporation of the protection mechanisms into the proposed algorithms involves additional computations which naturally affect performance.

RSA: We compare the overhead of our proposal to the implementation of a masked square and multiply algorithm.

For the initialization step, $r_2^{-1} \bmod n$ and $n^{-1} \bmod r_2$ has to be calculated. Both values can be calculated simultaneously in one run of an extended Euclid algorithm (EEA). This would be particularly attractive since one operand is just about one machine word long, so the EEA would begin with a division, and then continue with two single precision operands, leading to several possible optimizations, that can also be combined with a binary approach. But, from a trace of a GCD computation one can in theory reconstruct the original operands "backwards" or at least elicit some information on r_2 – this is even worse if a binary approach is used [**?**]. In order do avoid possible side-channel leakage of the EEA, Fermat's little theorem can be used for the computation of the inverses. Following this approach, we first raise the (small) value r_2 to the power of $\varphi(n) - 1 = (p-1) \cdot (q-1) - 1$. (Since it is required for the exponent blinding, $\varphi(n)$ is available. We can also compute this power modulo p and q separately and then reconstruct the value using the Chinese Remainder Theorem.) For the result $v = r_2^{-1} \pmod n$ holds $v \cdot r_2 = 1 + u \cdot n$ for some integer value u. We can assume $v < n$ and $u < r_2$, and u can be computed by an exact division $u = (v \cdot r_2 - 1)/n$. This exact division requires only multiplications by a (truncated) inverse of n modulo a small power of two (for instance 2^w where w is the bit size of a machine word), that can be quickly precomputed [**?**]. If this inverse is precomputed and stored, the cost of the second inversion is approximately that of two multiplications of a full operand by a single precision one.

The computation of i and the exponent blinding requires one multiplication each. The initialization of the infective computation requires ℓ multiplications in $\mathbb{Z}_{r_2}$, its computation in the while loop requires a multiplication in $\mathbb{Z}_{r_2}$ per iteration. Furthermore, a conditional branch is required in each run of the loop.

Elliptic Curve Cryptography: Assuming the size of r_2 to be about one machine word, as recommended for the RSA countermeasure, the relative slowdown of the field multiplications is higher than that imposed on a computation of RSA. For example, implementing a 256-bit curve on a 32-bit embedded CPU requires 8 words of storage per operand, masked operands require however 9 words. For primes of general form and interleaved Montgomery multiplication, the complexity of a field operation increases from $2t^2 + t$ to $2(t+1)^2 + (t+1)$ CPU

multiplications, which for operands of $t = 8$ words amounts to an increase of 26% in runtime.

An increase in runtime can also be expected from scalar blinding, if a random factor with the size of one machine word is considered. For instance, going from 8-word to 9-word scalars will increase the running time for the scalar multiplication by 12.5%, so a total increase of about 40% can be expected. The cost for setup and scalar diffusion is, in comparison, negligible.

For the computation of inverses, the same considerations as for the RSA case apply (only, in this case $\varphi(n) = n - 1$).

5 Conclusions

In this paper we show that the many countermeasures proposed in the literature do not consider the threat posed by combining different implementation attacks. These algorithms are, therefore, potentially insecure if not implemented carefully.

We present the first countermeasure that protects all the variables of an exponentiation against such combined implementation attacks. Our countermeasure is based on an infective computation strategy that checks the correctness of the intermediate values in each iteration. An adversary cannot learn more than one bit of the exponent during one computation, which does not endanger its security since an exponent will typically be blinded. As we demonstrate, our countermeasure can be applied to algorithms based on RSA or curve-based cryptosystems.

6 Acknowledgments

The work described in this paper has been supported in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II and EPSRC grant EP/F039638/1 “Investigation of Power Analysis Attacks”.

The information in this document is provided as is, and no guarantee or warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information as its sole risk and liability.

References

1. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Koblitz, N., ed.: CRYPTO ’96. Number 1109 in LNCS, Springer (1996) 104–113
2. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In Wiener, M., ed.: CRYPTO ’99. Volume 1666 of LNCS., Springer (1999) 388–397
3. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, Naccache, D., Paar, C., eds.: CHES 2001. Volume 2162 of LNCS., Springer (2001) 251–261
4. Anderson, R.J., Kuhn, M.G.: Tamper resistance — a cautionary note. In: Second Usenix Workshop on Electronic Commerce. (1996) 1–11

5. Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In Jr., B.S.K., Çetin Kaya Koç, Paar, C., eds.: CHES 2002. Volume 2523 of LNCS., Springer (2003) 2–12
6. Baek, Y.J., Vasyltsov, I.: How to prevent DPA and fault attack in a unified way for ECC scalar multiplication — ring extension method. In Dawson, E., Wong, D.S., eds.: ISPEC 2007. Volume 4464 of LNCS., Springer (2007) 225–237
7. Kim, C.H., Quisquater, J.J.: How can we overcome both side channel analysis and fault attacks on RSA-CRT? In Breveglieri, L., Gueron, S., Koren, I., Naccache, D., Seifert, J.P., eds.: FDTC 2007, IEEE Computer Society (2007) 21–29
8. Amiel, F., Villegas, K., Feix, B., Marcel:, L.: Passive and active combined attacks: Combining fault attacks and side channel analysis. In Breveglieri, L., Gueron, S., Koren, I., Naccache, D., Seifert, J.P., eds.: FDTC 2007, IEEE Computer Society (2007) 92–102
9. Yen, S.M., Kim, S., Lim, S., Moon, S.J.: RSA speedup with residue number system immune against hardware fault cryptanalysis. In Kim, K., ed.: ICISC 2001. Volume 2288 of LNCS., Springer (2002) 397–413
10. Gaubatz, G., Sunar, B.: Robust finite field arithmetic for fault-tolerant public-key cryptography. In Breveglieri, L., Koren, I., Naccache, D., Seifert, J.P., eds.: FDTC 2006. Volume 4236 of LNVS., Springer (2006) 196–210
11. Chevallier-Mames, B., Ciet, M., Joye, M.: Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. IEEE Transactions on Computers **53**(6) (2004) 760–768
12. Shamir, A.: Improved method and apparatus for protecting public key schemes from timing and fault attacks. US Patent 5991415 (1999)
13. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks — Revealing the Secrets of Smart Cards. Springer-Verlag (2007)
14. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The sorcerer's apprentice guide to fault attacks. Proceedings of the IEEE **94**(2) (2006) 370–382
15. Courrége, J.C., Feix, B., Roussellet, M.: Simple power analysis on exponentiation revisited. In D. Gollmann, J.-L. Lanet, J.I.C., ed.: CARDIS 2010. Volume 6035 of LNCS., Springer (2010) 65–79
16. Bernstein, D.J., Lange, T.: Inverted Edwards coordinates. In Boztas, S., Lu, H., eds.: AAECC 2007. Volume 4851 of LNCS., Springer (2007) 20–27
17. Kim, C.H., Quisquater, J.J.: Fault attacks for CRT based RSA: New attacks, new results, and new countermeasures. In Sauveron, D., Markantonakis, C., Bilas, A., Quisquater, J.J., eds.: WISTP 2007. Volume 4462 of LNCS., Springer (2007) 215–228
18. Dottax, E., Giraud, C., Rivain, M., Sierra, Y.: On second-order fault analysis resistance for CRT-RSA implementations. In Markowitch, O., Bilas, A., Hoepman, J.H., Mitchell, C.J., Quisquater, J.J., eds.: WISTP 2009. Volume 5746 of LNCS., Springer (2009) 68–83
19. Rivain, M.: Securing RSA against fault analysis by double addition chain exponentiation. In Fischlin, M., ed.: CT-RSA 2009. Volume 5473 of LNCS., Springer (2009) 459–480
20. Giraud, C.: An RSA implementation resistant to fault attacks and to simple power analysis. IEEE Transactions on Computers **12**(4) (2006) 241–245
21. Joye, M., Yen, S.M.: The Montgomery powering ladder. In Jr., B.S.K., Çetin Kaya Koç, Paar, C., eds.: CHES 2002. Volume 2523 of LNCS., Springer (2003) 291–302

22. Boscher, A., Naciri, R., Prouff, E.: CRT RSA algorithm protected against fault attacks. In Sauveron, D., Markantonakis, C., Bilas, A., Quisquater, J.J., eds.: WISTP 2007. Volume 4462 of LNCS., Springer (2007) 229–243
23. Fumaroli, G., Vigilant, D.: Blinded fault resistant exponentiation. In Breveglieri, L., Koren, I., Naccache, D., Seifert, J.P., eds.: FDTC 2006. Volume 4236 of LNCS., Springer (2006) 62–70
24. Proudler, I.K.: Idempotent AN codes. In: IEE Colloquium on Signal Processing Applications of Finite Field Mathematics, IEEE (1989) 8/1–8/5
25. Medwed, M., Schmidt, J.M.: A generic fault countermeasure providing data and program flow integrity. In Breveglieri, L., Gueron, S., Koren, I., Naccache, D., Seifert, J.P., eds.: FDTC 2008, IEEE (2008) 68–73
26. Smart, N., Oswald, E., Page, D.: Randomised representations. IET Proceedings on Information Security **2**(2) (2008) 19–27
27. Lange, T.: Trace zero subvarieties of genus 2 curves for cryptosystems. Journal of the Ramanujan Mathematical Society **19**(1) (2004) 15–33
28. Blömer, J., Otto, M., Seifert, J.P.: Sign change fault attacks on elliptic curve cryptosystems. In Breveglieri, L., Koren, I., Naccache, D., Seifert, J.P., eds.: FDTC 2006. Volume 4236 of LNCS., Springer (2006) 36–52
29. Goubin, L.: A refined power analysis attack on elliptic curve cryptosystems. In Desmedt, Y., ed.: PKC 2003. Volume 2567 of LNCS., Springer (2003) 199–210
30. Biehl, I., Meyer, B., Müller, V.: Differential fault attacks on elliptic curve cryptosystems. In Bellare, M., ed.: CRYPTO 2000. Volume 1880 of LNCS., Springer (2000) 131–146
31. Avanzi, R.M.: Countermeasures against differential power analysis for hyperelliptic curves. In Walter, C., Çetin Kaya Koç, Paar, C., eds.: CHES 2003. Volume 2779 of LNCS., Springer (2004) 77–88
32. Acimez, O., Gueron, S., Seifert, J.P.: New branch prediction vulnerabilities in OpenSSL and necessary software countermeasures. In: 11^{th} IMA International Conference on Cryptography and Coding. Volume 4887 of LNCS. (2007) 185–203
33. Jebelean, T.: An algorithm for exact division. Journal of Symbolic Computation **15**(2) (February 1993) 169–180