

# A New CRT-RSA Algorithm Secure Against Bellcore Attacks

Johannes Blömer  
Paderborn University  
D-33095 Paderborn, Germany  
bloemer@upb.de

Martin Otto<sup>\*</sup>  
Paderborn University  
PaSCo Graduate School  
D-33095 Paderborn, Germany  
martinmo@upb.de

Jean-Pierre Seifert  
Infineon Technologies  
Secure Mobile Solutions,  
SMS IC  
D-81609 Munich, Germany  
Jean-Pierre.Seifert@infineon.com

## ABSTRACT

In this paper we describe a new algorithm to prevent fault attacks on RSA signature algorithms using the Chinese Remainder Theorem (CRT-RSA). This variant of the RSA signature algorithm is widely used on smartcards. Smartcards on the other hand are particularly susceptible to fault attacks like the one described in [7]. Recent results have shown that fault attacks are practical and easy to accomplish ([21], [17]). Therefore, they establish a practical need for fault attack protected CRT-RSA schemes. Starting from a careful derivation and classification of fault models, we describe a new variant of the CRT-RSA algorithm. For the most realistic fault model described, we rigorously analyze the success probability of an adversary. Thereby, we prove that our new algorithm is secure against the Bellcore attack. Only once in the analysis do we need to refer to a plausible number theoretic assumption.

## Categories and Subject Descriptors

B.8.1 [Reliability, Testing, and Fault-Tolerance]: fault attacks; C.3 [Special-Purpose and Application-based Systems]: smartcards; D.4.6 [Security and Protection]: Cryptographic controls

## General Terms

Algorithms, Security

## Keywords

RSA, cryptanalysis, faults attacks, Bellcore attack, smartcards, Chinese Remainder Theorem

<sup>\*</sup>Supported by the PaSCo Graduate School, Paderborn University.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CCS'03, October 27–30, 2003, Washington, DC, USA.  
Copyright 2003 ACM 1-58113-738-9/03/0010 ...\$5.00.

## 1. INTRODUCTION

Smartcards play an important role in modern cryptography. Smartcards are used to compute digital signatures, most notably digital signatures based on RSA. Since speed is still an issue with modern smartcards, enhancements have been adopted to the plain RSA signature algorithm. The most common enhancement is the computation of an RSA signature using the Chinese Remainder Theorem (CRT). We will refer to this variant of RSA as CRT-RSA. With CRT-RSA one can expect a speed-up by a factor of 4 compared to plain RSA. However, smartcards are not as tamper-resistant as one may wish. Hence side-channel attacks like fault, power, and timing attacks, on smartcards have attracted a lot of attention. Among the side-channel attacks, fault attacks seem to be easiest to realize [2]. In particular, CRT-RSA proved to be susceptible to fault attacks. In [7] an extremely simple attack on CRT-RSA is described. Named the *Bellcore attack*, this attack reveals the secret factorization of the RSA modulus  $N$  by introducing a single fault resulting in a signature that is correct modulo one of the secret prime factors of  $N$ , but faulty modulo the other prime factor. This attack is particularly devastating because the type of fault induced is irrelevant.

Several types of countermeasures against fault attacks have been described, e.g. to compute a signature twice and compare the two results or to verify the result with the public key before output. However, these two countermeasures are too costly to be of practical interest. A more sophisticated software countermeasure has been proposed by Shamir (see [20]). He suggested to check intermediate results modulo a small integer. This approach will be described later in more detail. Of course, general (usually randomized) schemes that enhance the security of RSA can also prevent fault attacks or at least make them harder to realize. The most prominent of these randomization schemes is OAEP [4]. Most smartcard certification authorities, however, require that a smartcard implements a pure RSA signature algorithm that is secure without using OAEP or similar schemes.

Although several software countermeasures against fault attacks have been proposed (see [7], [24]), none of these proposals was based on an explicitly formulated and justified fault model. Accordingly, no proper security analysis of the various schemes could be given. Hence, in this paper starting from a careful derivation and classification of fault models, we describe a new variant of the CRT-RSA algorithm. For the most realistic fault model described, we analyze whether

a Bellcore-type attack on the algorithm can be successful. In [2] it was shown that a fault attack can basically change the value of any variable used in a smartcard algorithm. Hence, for any variable used in our scheme we analyze whether a Bellcore-type attack can be mounted by changing the value of that particular variable. We prove that for all variables the success probability of a Bellcore-type fault attack is negligible. Only once do our proofs rely on a plausible number theoretic assumption.

Our scheme borrows ideas from Shamir’s countermeasure against fault attacks as well as the idea of infective computations from [24]. Shamir suggests choosing a small prime  $t$  of about 32 bits to compute  $S_p = m^d \bmod pt$  and  $S_q = m^d \bmod qt$  and check whether  $S_p \equiv S_q \pmod t$  before combining them with the CRT. As is easily seen, Shamir’s scheme only protects the signature computations modulo the two secret prime factors of the RSA modulus  $N$ . It leaves the CRT-combination step to obtain the final signature modulo  $N$  unprotected. Furthermore, as observed in [24], Shamir’s algorithm has a single point of failure when it checks if  $S_p \equiv S_q \pmod t$ . Usually, such a comparison relies on the zero flag only. A single point of failure means that once this point is successfully attacked, a smartcard may output a defective signature that can be used to recover secrets.

Infective computation, as introduced in [24], means that any error introduced by a fault attack propagates through the computation. In particular, in CRT-RSA a faulty signature will always be faulty modulo both prime factors, thereby preventing a Bellcore attack. Unfortunately, for realistic parameters the infective computation proposed in [24] can be broken. In particular, the method proposed in [24] restricts the pairs of public/secret exponents in RSA to pairs, that basically can be broken by Wiener’s small secret exponent attacks ([5]).

The scheme proposed in this paper extends Shamir’s idea to protect every single computation step of the signature algorithm, including the CRT combination. We achieve this by using two small integers  $t_1$  and  $t_2$  to compute  $S_p = m^d \bmod pt_1$  and  $S_q = m^d \bmod qt_2$ . These values are combined to  $S \bmod Nt_1t_2$  via the CRT. This combination with a larger modulus allows to use infective computation steps afterwards. These infective steps ensure that an error will cause the final signature to be false modulo both primes  $p$  and  $q$ . Infective computations not only avoid single points of failures. They also allow a card to continue its computation, even if a fault is detected. Hence, our scheme renders mechanism like security resets or error messages pointless. This is an important feature of our scheme, since error messages or security resets may leak important and useful information to an adversary (see for example [14], [6] for more details).

Finally, unlike the scheme proposed in [24], our algorithm works with any RSA key, no restriction on the key space applies. To prove security, we present a rigorous analysis of our scheme.

To describe and classify fault models, we start from the most powerful adversary imaginable, i.e. an adversary that can change any specific bit at any specific time of the algorithm execution. However, we make use of several hardware features employed on realistic smartcards to argue that the effects of such a precise intrusion are not completely predictable for an adversary. The hardware mechanisms we are referring to include randomized clocks, memory encryption

/ decryption schemes, and randomized address scrambling (see [18], [9]). Like randomized schemes, these hardware features try to make fault attacks harder by randomizing the effects of a fault attack in a manner that can not be controlled or predicted by an adversary.

Based on the effects of randomized clocks, memory encryption/decryption, and randomized address scrambling we argue that even a very powerful adversary can not hope to reset a bit value at a specific location and time. Instead all he can hope is that with a certain probability he will change the value of a particular variable used by the algorithm. Furthermore, we argue that an adversary can only hope to change the value of a targeted variable to some random value. We also describe some intermediate fault models, in which the adversary is able to attack specific byte or bit values. We extend the analysis of our CRT-RSA scheme to these models as well. We show that in these models the new CRT-RSA scheme sometimes offers an even better security, i.e. knowing the effect of an attack exactly helps defending against this attack. In our analysis, we only look at fault attacks. Combinations with other side channel attacks like timing or power attacks are not regarded.

The paper is organized as follows: After stating the preliminaries in Section 2, we formulate a careful classification of fault models in Section 3. We propose a new algorithm in Section 4 that will be proven to be secure against the Bellcore attack in Section 5. Section 6 concludes the paper.

## 2. PRELIMINARIES

Throughout the paper, we will use the following notation. We assume that an RSA key scheme consists of two primes  $p$  and  $q$  that form the RSA modulus  $N = p \cdot q$ . We denote the public RSA key as  $e$  and the private RSA key as  $d$ , satisfying  $e \cdot d \equiv 1 \pmod{\varphi(N)}$ . Here,  $\varphi(N)$  denotes Euler’s totient function. The function  $l(n)$  will be used to denote the binary length of an integer  $n$ . We will use  $a \mid b$  to denote that an integer  $a$  divides  $b$ , and  $a \nmid b$  to denote the converse.

Usually, smartcards compute RSA signatures  $S(m) := m^d \bmod N$  using the CRT-RSA scheme. Here, the two signature parts  $S_p := m^{d \bmod (p-1)} \bmod p$  and  $S_q := m^{d \bmod (q-1)} \bmod q$  are computed first. They are combined using the Chinese Remainder Theorem (CRT) as  $S(m) := \text{CRT}(S_p, S_q) \bmod N$ . On average, this scheme is four times faster than the direct computation via a single exponentiation, cf. [12].

The major exploit of fault attacks on smartcards performing CRT-RSA signatures is an attack first presented in [7] (and named the ”Bellcore attack”). Here it is assumed that an adversary induces an error that causes  $S_p$  to be defective while  $S_q$  is computed correctly (or vice versa of course). If the defective CRT-combination  $S = \text{CRT}(S_p, S_q) \bmod N$  is disclosed, the scheme is completely broken as  $\text{gcd}(S^e - m, N) = q$ .

As we investigate errors on various variables, we use the following convention: For random errors on a specific variable  $x$ , we write  $f(x) = x + e(x)$ , where  $e(x)$  is the error dependent on  $x$ . For random errors,  $e(x) \in [-x, 2^{l(x)} - 1 - x]$ . In some scenarios,  $e(x)$  can be specified in greater detail, e.g. for random byte errors,  $e(x) = b \cdot 2^k$  with  $|b| \in \mathbb{Z}_{2^8}$  a random byte and  $0 \leq k < l(x) - 7$ . For single bit errors,  $|b| = 1$  and  $0 \leq k < l(x)$ . We always assume a uniform distribution on the errors.

All parts of our analysis can be rigorously proven. Only

the analysis for one single variable relies on the following heuristically justified assumption.

ASSUMPTION 1. For an RSA modulus  $N = p \cdot q$ ,  $d$  a secret key,  $m \in \mathbb{Z}_N$  a given message,  $t \ll p$  a prime,  $e(\cdot)$  a random error as defined above, the value

$$\alpha := \left( m^d \operatorname{div} (pt + e(pt)) \right) \bmod t$$

can be seen as a random variable uniformly distributed in  $\mathbb{F}_t$ .

Clearly,  $\alpha$  only has a chance to fulfill the assumption correctly if  $2^{l(pt)}$  is a multiple of  $t$ . However, the assumption is justified for other cases as well, because the distance to the uniform distribution is negligibly small as  $t \ll 2^{l(pt)}$ .

### 3. DEVELOPMENT OF PRACTICALLY APPROVED FAULT MODELS

There has been a large number of different fault attacks in the literature. They differ in the power to locate and time an attack, in the number of bits affected, in the effect of an attack (the fault type), in the probability of the implied effect of an attack, and in prior work that has to be applied to the card in order to mount the attack, cf. among others [7], [16], [22], [14], [15], [1], [3], [8], [23], [2], [21]. However, the characterization of the used fault models has been simple and insufficient to derive usable frameworks for a satisfactory analysis. Therefore, we present a characterization of the different parameters needed to fully describe all known types of fault attacks. This leads to a proper mathematical formulation of errors induced by such attacks. The derived fault models are motivated by smartcards as they are used today.

- For **control on the fault location** we define the three classes "no control", "loose control" (a selected variable can be targeted) and "complete control" (selected bits can be attacked).
- For **control on the timing** we also define the three classes "no control", "loose control" (an error is induced in a block of a few operations) and "precise control" (the exact time can be met).
- For **the number of bits attacked**, we differentiate between a "single faulty bit", "few faulty bits" (e.g. a byte) or a "random number of faulty bits" (bounded by the length of the attacked variable).
- The **fault type** describes the character of the fault as it manifests itself in the chip. This parameter has appeared in the literature as the "fault model". We will break with this tradition as a reasonable description of a fault model must contain more than just the type of the fault. Fault types include the classic fault types, namely the *stuck at fault* (saf), where bits permanently keep their value from the point of a successful attack, the *bit flip fault* (bf), where bits are flipped to their complementary value and the *random fault* (rf), where bits are changed to a random value, possibly the one they already had. In addition to these classic fault types, we also have the fault type derived from recent work in [21]. Their approach enables them to set a single specific (targeted) bit to 1 or reset that bit to 0. This is the *bit set or reset fault* (bsr).

- Attacks mounted also have a **certain probability** associated with them. Usually an attack is not certain to be successful, it is only so most of the time. Therefore, any effect as well as the control on location and timing might require a probability or even a distribution to be completely described. For example, some physical attacks might have a greater probability of resetting a bit than of setting that bit (see [8], [6]). No control on the location implies that a specific location is expected to be hit with a certain probability  $1/(\text{number of locations})$ .

To derive reasonable fault models, we combine parameter settings known from actual attacks with hardware countermeasures in effect on the card. As we are guided by practical considerations, we always assume the most powerful adversary, i.e. the adversary presented in [21] and [17] able to use (bsr) faults. This adversary can select any specific area on the card at a precise time (on *his* clock) and set the stored bit to any specific value. We also consider smartcards, that may have several hardware countermeasures (see [18]). The architecture of a modern smartcard is sketched in Figure 1. In these realistic scenarios, the effective power of the adversary is reduced significantly, e.g. knowledge about the location of the induced fault need not imply knowledge about the position of the bit within an algorithm. Similarly, precise timing need not imply knowledge of the actual step performed at that time. The following five fault models gradually improve the strength of the card's countermeasures.

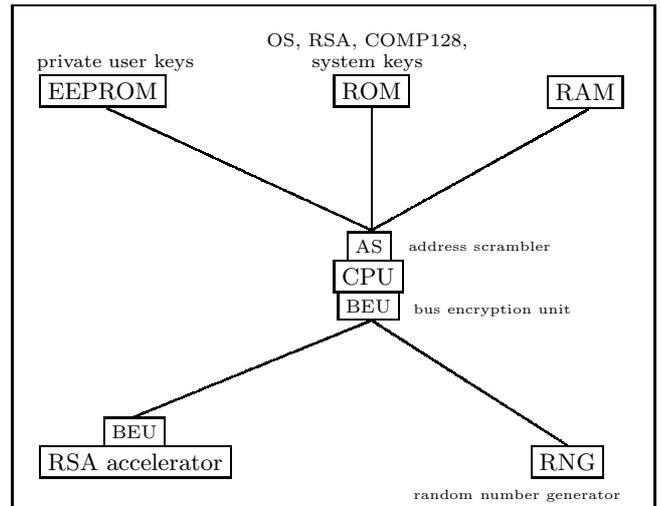


Figure 1: architectural sketch of a modern high-end smartcard

#### Fault Model #1: Precise Bit Errors.

*Parameter setting.* For this strong fault model, we assume that the adversary has precise control on both timing and location. This means that the adversary knows the attacked bit as well as the attacked operation. Note that an attack usually happens before the variable is used in a line of an algorithm. We assume that only a single bit is affected. This resembles the (bsr) fault type that is achieved by attacks described in [21] or [17] on RAM or EEPROM of an unprotected smartcard.

*Mathematical model.* This attack can be modeled as an addition or subtraction of a single bit, i.e. a variable  $X$  is changed to  $X' = X \pm 2^k$  for  $0 \leq k \leq l(X) - 1$ .

*Motivation from the real world.* Although high-end smartcards implement sophisticated hardware countermeasures, many smartcards currently used are either too old or too cheap to do so. Hence, this fault model is a realistic one. It assumes the strongest adversary and the weakest card. Since algorithms secure in this fault model are secure in the weaker models as well, it is a particularly interesting model.

### Fault Model #2: Precise Byte Errors.

*Parameter setting.* In this scenario, we assume that the timing is precise. Hence, a specific operation can be targeted. However, control on location is loose, i.e. the number of bits affected can only be bounded by a block of few bits (we assume a byte). We allow any fault type in this model.

*Mathematical model.* The attack can be modeled as an addition or subtraction of an unknown error byte at a known position, i.e. a variable  $X$  is changed to  $X' = X \pm b \cdot 2^k$  for a known  $0 \leq k \leq l(X) - 8$  and an unknown  $b \in \mathbb{Z}_{28}$ .

*Motivation from the real world.* This model is motivated by the fact that the strong adversary's power is reduced on smartcards if encryption of the data is used. Usually, all data stored in EEPROM and RAM is encrypted [18]. Hence, if an error is induced into memory, the CPU will see a random block of data. The same model is derived if the bus lines are attacked.

### Fault Model #3: Unknown Byte Errors.

*Parameter setting.* In this scenario, we assume loose control on both timing and location. The loose control on location means that a certain variable can be targeted but the number of bits affected can only be bounded by a block of few bits (usually a byte). In addition, loose control on timing means that the attacker can only affect the variable within a specific time frame that usually contains several instructions. The exact instruction affected by the attack is unknown. Hence, the attacker does not know for sure which byte of the variable is currently used by the algorithm. We allow any fault type in this model.

*Mathematical model.* This attack can be modeled as an addition or subtraction of an error byte, i.e. a variable  $X$  is changed to  $X' = X \pm b \cdot 2^k$  for an unknown  $0 \leq k \leq l(X) - 8$  and an unknown  $b \in \mathbb{Z}_{28}$ .

*Motivation from the real world.* This model is motivated by the fact that attacks on EEPROM and RAM with address scrambling (cf. [18]) will not allow to specify when the attacked block is requested by the CPU. Encryption of the memory ensures that a faulty bit affects a whole block of data.

### Fault Model #3': Unknown Byte Errors in Unknown Variables.

*Parameter setting.* This model assumes loose control on location, once again a whole byte is affected, and no control on timing. Due to the latter it is unknown at which exact time within the program the attack is mounted. It is even unknown, which variable is faulty.

*Mathematical model.* We model this type of fault as a variable dependent error, i.e. a variable  $X$  is changed to  $X' = X \pm b \cdot 2^k$  for an unknown  $0 \leq k \leq l(X) - 8$  and an unknown  $b \in \mathbb{Z}_{28}$ . Note that due to the unprecise timing,

the attacked variable  $X$  is also unknown (to some degree).

*Motivation from the real world.* The strong adversary's power is effectively reduced to this model if the smartcard uses memory encryption in RAM and EEPROM. This causes any bit fault to affect a whole block of data. In addition, some smartcards use a randomized clock (cf. [9]). In this case, the attacker knows that a successful attack will change a block of data. But he does not know the exact time of the change within the algorithm. Therefore the attacker does not know the position of the block as it is used in the CPU.

### Fault Model #4: Random Errors.

*Parameter setting.* In this fault model, we assume that the adversary has no control on the location of a fault and only a loose timing, i.e. he can target an interval of some operations. This interval may have been derived from other sources of information, for example from the power profile of the card (see [2]). The number of affected bits is unknown.

*Mathematical model.* We model this uncertainty on the number of affected bits by a random fault. We assume that for a given variable  $X$ , the uniformly distributed random value  $f(X) \in [0, 2^{l(X)} - 1]$  is used by the algorithm. In this model, any fault may result in any faulty value.

*Motivation from the real world.* This scenario is motivated by strong high-end smartcards completely armed with countermeasures. Memory encryption, address scrambling and a randomized clock imply that any error induced into memory or the CPU at a vague point will leave the attacker at most with the information that a certain variable is faulty. It therefore enforces a very weak adversary.

### Reaction Of The Smartcard

A smartcard may react in various ways to an attack: Unprotected cards will fail to notice the attack and output a faulty result, leaving them vulnerable to the Bellcore attack. More sophisticated smartcards may have countermeasures that alter the result to some random value or use detection mechanisms that report an error to the user. The error output may either depend on the kind of error or be unspecific.

Modern high-end smartcards may have additional hardware countermeasures (see [6]), that might successfully fight some of these attacks and react with a complete security reset. This reaction of the card may already be set off before an attack successfully induced an error.

### Implications For Our Analysis

So far, no software countermeasures against fault model #1 are known (see Section 6). Lukily, the models #1 and #2 are unlikely in the real world, as all trustworthy smartcards are fully armed with a variety of countermeasures. This causes a precise location of an affected bit or byte to be unrealistic.

We will therefore concentrate to analyze security against fault models #3 and #4. The two models #3 and #3' are equivalent, as model #3' usually only means that the faulty variable  $X$  is from a small set of possible variables. If the attack hits every variable within this set with a sufficiently high probability, a specific variable is expected to be hit after a relatively low number of attacks. Hence, both scenarios can be modeled in the same way (see [6]).

Our analysis will show that only a negligible number of faults result in an output that potentially leaks valuable

information to an adversary. Hence, the smartcard needs no other means of reaction to errors such as error messages or security resets. We will therefore not consider any such reaction mechanisms in the following.

## 4. THE NEW APPROACH

The drawbacks of Shamir's small prime verification countermeasure explained in the introduction show that better methods are needed. As Shamir's basic idea is very promising, we extend this countermeasure to the whole CRT-RSA computation. We also use infective computations as introduced by [24] to eliminate the single point of failure of a checking step.

ALGORITHM 2 (INFECTIVE CRT-RSA).

**Input.** A message  $m \in \mathbb{Z}_N$

**Output.**  $Sig := m^d \bmod N$  or a random number in  $\mathbb{Z}_N$

**In Memory.**  $p \cdot t_1, q \cdot t_2, N, N \cdot t_1 \cdot t_2, d_p, d_q, t_1, t_2, e_{t_1}$  and  $e_{t_2}$

- 1 Let  $S_p := m^{d_p} \bmod p \cdot t_1$
- 2 Let  $S_q := m^{d_q} \bmod q \cdot t_2$
- 3 Let  $S := \text{CRT}(S_p, S_q) \bmod N \cdot t_1 \cdot t_2$
- 4 Let  $c_1 := (m - S^{e_{t_1}} + 1) \bmod t_1$
- 5 Let  $c_2 := (m - S^{e_{t_2}} + 1) \bmod t_2$
- 6 Let  $Sig := S^{c_1 \cdot c_2} \bmod N$
- 7 output Sig

The basic algorithm for CRT-RSA consists of three steps, the computation of the two parts  $S_p$  and  $S_q$  and their combination to the signature  $S$  using the CRT. We modify the computation of all three values implementing a variant of Shamir's idea. Then we introduce a detection mechanism that is not required to be error free in order to prevent a fault attack on the whole smartcard. If an error was induced at any step of the algorithm, this countermeasure will change the final result in a way unpredictable to an adversary. The resulting algorithm looks extremely simple, but it proves to be very effective in the most practical attack model assuming the strongest adversary.

As a precomputation step that can be done for any smartcard at production time, generate a valid RSA key with  $(N, e)$ ,  $N = p \cdot q$ , as the public key and  $d$  as the corresponding private key satisfying  $e \cdot d \equiv 1 \pmod{\varphi(N)}$ .

Additionally, select two integers  $t_1$  and  $t_2$  of sufficiently large bitlength to withstand exhaustive search (see Section 5.5 for concrete suggestions) which must satisfy several conditions in order to allow a secure scheme:

1.  $t_1$  and  $t_2$  must be coprime
2.  $\gcd(d, \varphi(t_1)) = 1$  and  $\gcd(d, \varphi(t_2)) = 1$
3.  $t_1$  and  $t_2$  are squarefree
4.  $t_i \equiv 3 \pmod 4$  for  $i \in \{1, 2\}$
5.  $t_2 \nmid X = pt_1 \cdot ((pt_1)^{-1} \bmod qt_2)$

Note that the use of two small moduli instead of a single one has already been described by Shamir in [20], but for a different use. Let  $d_p := d \bmod \varphi(p \cdot t_1)$ ,  $d_q := d \bmod \varphi(q \cdot t_2)$ . Afterwards, compute two corresponding public keys  $e_{t_1}$  and  $e_{t_2}$  such that  $d \cdot e_{t_i} = 1 \pmod{\varphi(t_i)}$ . Store  $p \cdot t_1, q \cdot t_2, N, N \cdot t_1 \cdot t_2, d_p, d_q, t_1, t_2, e_{t_1}$  and  $e_{t_2}$  on the smartcard. It is easy to see that the algorithm computes the correct signature if no error occurs.

Let us briefly comment on the five conditions for the small primes  $t_i$ : Condition 1 is needed to ensure that the CRT

combination of  $S_p$  and  $S_q$  works, because it requires the two moduli to be coprime. Condition 2 is required to ensure that the small keys  $e_{t_i}$  ( $i = 1, 2$ ) can be generated. Condition 3 must hold, because otherwise the equation  $m^{d \cdot e_{t_i}} \equiv m \pmod{t_i}$  may not hold. This condition can be further relaxed. However, we suggest choosing primes for both  $t_i$ . Condition 4 ensures a good resistance against attacks on the exponents  $d_p$  and  $d_q$ . The security analysis will further explain this condition. Finally, Condition 5 provides security against attacks on the CRT combination (see the analysis of attacks on line 3 below). Section 4.1 will show that choosing both  $t_i$  as strong primes is a useful recommendation.

### 4.1 Efficiency Of The New Algorithm

Now we will show that the new algorithm is indeed an efficient algorithm to compute RSA signatures. The efficiency depends on the keys  $t_i$ . The additional costs compared to the plain CRT-RSA method are mainly an increased number of operations in lines 1 and 2 due to the larger moduli. This increases the size of the exponent and of the intermediate results. These have to be computed in a larger group now. The additional costs of the CRT (due to a larger modulus) and the costs of the two modular exponentiations modulo  $t_1$  and  $t_2$  (in lines 4 and 5) do not contribute to the overall costs significantly. If quadratic time complexity is assumed for the modular exponentiations in  $\mathbb{Z}_{pt_1}$  and  $\mathbb{Z}_{qt_2}$ , the savings compared to the plain CRT-RSA scheme is lowered to 1/3 instead of 1/4 (for  $l(t_i) \in \{60, 80\}$ ) of the cost of the plain CRT-RSA exponentiation. This is still an acceptable speedup.

For the key generation process, which is usually performed on the card at production time, we can efficiently find suitable candidates for  $t_1$  and  $t_2$ . First, a valid RSA key pair  $((e, N), d)$  is chosen, with  $N = p \cdot q$ . Then the two small moduli  $t_1$  and  $t_2$  are generated. We would like to emphasize the fact that neither the algorithm nor the small moduli impose any restrictions on the choice of the main RSA key, no special structure or generation process is required. Hence, any valid RSA key pair can be used for  $((e, N), d)$ .

We suggest choosing both  $t_i$  as different strong primes, i.e.  $(t_i - 1)/2$  are also primes. As  $t_i \ll p, q$  this obviously satisfies Conditions 1, 3, and 4. Condition 2 is not satisfied for a fixed  $t_i$  in a fraction of about  $1/t_i$  cases. Therefore given  $l(t_i)$  large enough, this probability is negligible. For randomly chosen  $t_i$ , the values  $pt_1$  and  $qt_2$  are independent, therefore  $(pt_1)^{-1} \bmod qt_2$  is uniformly distributed in  $\mathbb{Z}_{qt_2}$ . As  $t_2 \nmid pt_1$ , this means that the probability that a chosen  $t_2$  does not satisfy Condition 5 is at most  $1/t_2$ . Hence, we expect very few strong prime choices. Since the density of strong primes is conjectured to be asymptotically  $D \cdot x / \log^2(x)$  ([13]), the task of finding suitable  $t_i$  is easy. Here  $D \approx 0.6601618$  is the twin prime constant.

Note that it is also possible to use a modified CRT combination that can handle the case  $t_1 = t_2$ . Hence, it suffices to choose a single  $t$ . We choose two in order to use the standard CRT.

## 5. SECURITY ANALYSIS OF THE PROPOSED COUNTERMEASURE

### 5.1 Undetectable Errors

For the security analysis, we need to investigate the probability of any induced error to circumvent our countermea-

sure and result in an undetectable error. Note that we are only concerned with errors that cause the final signature to be correct modulo  $p$  but false modulo  $q$  (or vice versa), in which case the classic Bellcore attack can be applied. Otherwise, no exploits of specific errors in a faulty CRT-RSA signature are known yet. In our analysis, we do not look at combinations with other side channel attacks like timing or power attacks.

The checking mechanism in lines 4 and 5 is done via a small modulus, hence, undetectable errors are introduced into the system. However, we will show that the number of these errors is negligibly small. Therefore, they pose no threat to the security of the system.

An error will slip by lines 4 and 5 undetected if it is eliminated by the modular reduction. If  $S'$  is a faulty value for  $S$ , then if  $S' = S + k \cdot t_1 \cdot t_2$ ,  $k \in \mathbb{Z}$ , both modular reductions in lines 4 and 5 will fail to detect this error and set  $c_1 = c_2 = 1$ . For other values of  $S'$ , it is  $S' \neq S \bmod t_i$  for  $i = 1$  or  $i = 2$ . Hence  $S'^{e_{t_i}} \neq m \bmod t_i$ , which in turn forces  $m - S'^{e_{t_i}} > 0 \bmod t_i$ . Therefore, it is  $c_i \neq 1$ . Note that this observation is independent of the type of error.

However, in order to analyze the scheme, we need to define our fault model. For our analysis, we assume again the strongest adversary attacking the best protected card. We will only analyze in detail the most interesting/practical scenario, the random error fault model #4, and show its security against the Bellcore attack. Results for the byte error fault models #2, #3 yield even better results. See the appendix for details on attacks using the other fault models. The random error fault model #4 assumes that an attacked value  $x$  is replaced by a random bit string  $f(x) = x + e(x)$  as defined in Section 2. For simplicity, we only assume a single fault. However, as long as multiple errors are uncorrelated, the results are the same. Note that it is virtually impossible to correlate attacks on smartcards that use hardware countermeasures.

The two special inputs  $m = 0$  and  $m = 1$  will prove dangerous to the algorithm in the presence of some faults. Later, we will describe several enhancements to our scheme such that also these two messages cannot be exploited in a fault attack.

In the following analysis of attacks on each variable, the term *success probability* always refers to the success probability for a random fault induced into the attacked variable to result in an undetectable error such that the Bellcore attack can be applied.

## 5.2 Attacks On Lines 1 And 2

### • Attack on the stored variable $d_p$

*The success probability for an attack on  $d_p$  is at most  $3/t_1$  for messages  $m \not\equiv \pm 1 \bmod t_1$ . This probability is taken over the errors. A fraction of at most  $3/t_1$  of all messages satisfies  $m \equiv \pm 1 \bmod t_1$ . The message  $m = 1$  is secure. For other messages  $m \equiv \pm 1 \bmod t_1$ ,  $m \neq 1$ , the success probability is at least  $1/2$ .*

*Comment.* Although this analysis seems to prove the algorithm insecure, this is not the case. Any adversary capable of constructing malicious instances of messages, i.e. an  $m \equiv \pm 1 \bmod t_1$ , needs to know  $t_1$ . But this parameter is secret. Hence, the adversary has no information on how to construct  $m$ . Therefore, the best he can do is to randomize the inputs. And since

the number of malicious messages is less than  $3/t_1$ , this is a secure situation.

PROOF. Let  $m \not\equiv \pm 1 \bmod t_1$ . Given  $m$ , an error  $e(d_p)$  leads to an undetectable error if

$$m^{e(d_p)} \equiv 1 \bmod t_1. \quad (1)$$

We need to analyze how many  $e(d_p)$  exist at most with (1). Consider  $\gcd(e(d_p), t_1 - 1)$ . Since  $t_1$  is a strong prime, we get  $\gcd(e(d_p), t_1 - 1) \in \{1, 2, (t_1 - 1)/2, t_1 - 1\}$ .

Any  $e(d_p)$  with  $\gcd(e(d_p), t_1 - 1) \in \{1, 2\}$  can be written as  $e(d_p) = 2^l \cdot b$ ,  $b$  odd and  $\gcd(b, t_1 - 1) = 1$ . For these  $e(d_p)$ , (1) implies  $m^{2^l} \equiv 1 \bmod t_1$ . Next, since  $t_1$  is a strong prime, the equation  $x^{2^l} \equiv 1 \bmod t_1$  has only the solutions  $x \equiv \pm 1 \bmod t_1$ . We conclude that for  $m \not\equiv \pm 1$  and  $\gcd(e(d_p), t_1 - 1) \in \{1, 2\}$ , no error will be undetectable.

Hence it remains to bound the number of  $e(d_p)$  with  $\gcd(e(d_p), t_1 - 1) \in \{(t_1 - 1)/2, t_1 - 1\}$  and  $m^{e(d_p)} \equiv 1 \bmod t_1$ . The worst case for  $m$  is if  $m^{(t_1 - 1)/2} \equiv 1 \bmod t_1$ , in which case any  $e(d_p)$  with the property that  $e(d_p)$  is a multiple of  $(t_1 - 1)/2$  leads to an undetectable fault. Since  $e(d_p) \in [-d_p, 2^{l(d_p)+1} - 1]$ , the number of  $e(d_p)$  with  $\gcd(e(d_p), t_1 - 1) \in \{(t_1 - 1)/2, t_1 - 1\}$  is a fraction of at most  $2/(t_1 - 1) < 3/t_1$  of all possible  $e(d_p)$ .

Let us now determine the number of messages  $m \equiv \pm 1 \bmod t_1$ . As the messages are in  $\mathbb{Z}_N$ , there are at most  $2 \cdot \lfloor N/t_1 \rfloor + 2$  messages satisfying the condition  $m \equiv \pm 1 \bmod t_1$ . This is a fraction of less than  $3/t_1$  of all possible messages.

Now let  $m \equiv 1 \bmod t_1$ . If  $m = 1$ , then  $m^{e(d_p)} = 1$  and  $S_p = 1$ . Hence, the error has no effect. Otherwise, every fault will cause an undetectable error, because  $m^{e(d_p)} \equiv 1 \bmod t_1$  independent of the error  $e(d_p)$ .

Let  $m \equiv -1 \bmod t_1$ . The probability that a random fault causes an undetectable error is at least  $1/2$ , since every even  $e(d_p)$  yields  $m^{e(d_p)} \equiv 1 \bmod t_1$ . In addition, if  $e(d_p)$  is invertible modulo  $(t_1 - 1)$ , the same considerations as above apply. This increases the success probability further.  $\square$

### • Attack on the stored variable $pt_1$

*The success probability for an attack on  $pt_1$  is at most  $2/t_1$ . This result is based on Assumption 1. The probability is taken over random choices of the error.*

PROOF. If the modulus is randomly changed to  $pt'_1 = pt_1 + e(pt_1)$ , write  $m^d = \alpha_0 \cdot (pt_1 + e(pt_1)) + \alpha_1$  with  $\alpha_1 < pt_1 + e(pt_1)$ . The correct result  $S_p$  is now  $S_p = \alpha_0 \cdot e(pt_1) + \alpha_1 \bmod pt_1$ , while the faulty result  $S'_p$  is  $\alpha_1$ . An undetectable error happens, if  $S_p \equiv S'_p \bmod t_1$ , hence if  $\alpha_0 \cdot e(pt_1) + \alpha_1 \equiv \alpha_1 \bmod t_1$ . This is equivalent to  $\alpha_0 \cdot e(pt_1) \equiv 0 \bmod t_1$ .

As  $t_1$  is a prime,  $t_1$  has to divide at least one of the two factors. Hence, we need to compute the probability of  $0 \equiv e(pt_1) \bmod t_1$  and of  $0 \equiv \alpha_0 = m^d \operatorname{div} (pt_1 + e(pt_1)) \bmod t_1$ . As  $e(pt_1)$  is a uniformly distributed integer in a contiguous interval and  $\alpha_0$  is uniformly distributed by assumption 1, the success probability is

at most  $1/t_1$  for each factor, and altogether at most  $2/t_1$ . This probability is taken over random choices of the error.  $\square$

- **Attack on  $m$  or the exponentiation's intermediate variable**

The success probability for an attack during the exponentiation is at most  $2/t_1$  for messages  $m \not\equiv 0 \pmod{t_1}$ . This probability is taken over the errors. For messages  $m \equiv 0 \pmod{t_1}$ , all faults yield an undetectable error. A fraction of at most  $1/t_1$  of all messages  $m$  satisfies  $m \equiv 0 \pmod{t_1}$ .

*Comment.* There are many possible ways to compute  $m^{d_p} \pmod{pt_1}$ . Algorithm 3 presents a timing and simple power attack secure version of the well-known square-and-multiply algorithm (cf. [10], [11]). The result holds for other exponentiation algorithms as well.

Again, some messages are malicious, but similar to the reasoning before, the adversary can gain no advantage from this fact as he cannot choose  $m$  accordingly. Attacks on  $y_0$  and  $y_1$  resemble the same situation if they get incorporated into the computation at all. If the modulus  $pt_1$  is attacked during the exponentiation, the resulting scenario is equivalent to that analyzed for a global attack on  $pt_1$  above.

ALGORITHM 3 (MODULAR EXPONENTIATION).

**Input.** A message  $m \in \mathbb{Z}_N$ , a key  $d \geq 3$ , a modulus  $pt_1$

**Output.**  $m^d \pmod{pt_1}$

```

1  Let  $y := m^2 \pmod{pt_1}$ 
2  For  $i$  from  $l(d) - 2$  downto 1 do
3    Let  $y_0 := y$ 
4    Let  $y_1 := y \cdot m \pmod{pt_1}$ 
5    Let  $y := y_{d_i}^2 \pmod{pt_1}$ 
6  Let  $y_0 := y$ 
7  Let  $y_1 := y \cdot m \pmod{pt_1}$ 
8  Let  $y := y_{d_i} \pmod{pt_1}$ 
9  output  $y$ 

```

*PROOF.* If Algorithm 3 is attacked at the time when  $i = l$ , and the intermediate value  $y$  is altered, we have  $S'_p = (y + e(y))^{2^{l-1}} \cdot m^w$  for  $w = \sum_{i=1}^{l-1} d_i \cdot 2^i$ . Hence messages  $m \equiv 0 \pmod{t_1}$  lead to  $S'_p \equiv S_p \pmod{t_1}$  and to an undetectable error in line 4 of Algorithm 2. There are at most  $1/t_1$  of all possible messages satisfying this condition.

For messages  $m \not\equiv 0 \pmod{t_1}$ , we analyze the probability of  $(y + e(y))^{2^{l-1}} \equiv (y)^{2^{l-1}} \pmod{t_1}$ . First consider the case  $(y + e(y)) \equiv 0 \pmod{t_1}$ . This implies  $(y)^{l-1} \equiv 0 \pmod{t_1}$  as well. Since  $y$  is of the form  $m^x$  for some  $x$ , this in turn implies  $m \equiv 0 \pmod{t_1}$ , which is impossible.

From now on we assume  $y + e(y) \not\equiv 0 \pmod{t_1}$ . Then  $(y + e(y))^{2^{l-1}} \equiv (y)^{2^{l-1}} \pmod{t_1}$  implies  $1 = (y/(y + e(y)))^{2^{l-1}} \pmod{t_1}$ . Since  $t_1 \equiv 3 \pmod{4}$ , this is equivalent to  $1 = (y/(y + e(y)))^2$ , which in turn implies  $\pm 1 = (y/(y + e(y)))$ . For any fixed  $y$ , there are exactly two choices of  $e(y)$  that satisfy this equality. Hence, in case  $m \not\equiv 0 \pmod{t_1}$  we can bound the success probability by  $2/t_1$ .  $\square$

- **Attack on the result  $S_p$**

The success probability for an attack on  $S_p$  is at most  $1/t_2$  for messages  $m \neq 0, 1$ . Again, the probability is taken over the error.

This case will be analyzed while considering attacks on the CRT combination in line 3.

### 5.3 Attacks On Line 3

Line 3, the CRT combination, may also be successfully attacked. We assume that  $S = S_p + X \cdot (S_q - S_p) \pmod{N \cdot t_1 \cdot t_2}$  with  $X = pt_1 \cdot ((pt_1)^{-1} \pmod{qt_2})$ . Here  $X$  is a precomputed value stored on the smartcard.

- **Attack on the result  $S$ , and on the two addends  $S_p$  and  $X \cdot (S_q - S_p)$**

The success probability for an attack on  $S$ ,  $S_p$  or  $X \cdot (S_q - S_p)$  is at most  $1/(t_1 \cdot t_2)$ . The probability is taken over random errors only, it is independent from the chosen message.

*PROOF.* If  $S$  is attacked directly, then  $S' = S + e(S)$ . This would circumvent the countermeasure iff  $e(S) \equiv 0 \pmod{t_i}$  for both  $i$ . Because both  $t_i$  are different primes, this means that  $e(S) \equiv 0 \pmod{t_1 \cdot t_2}$  must hold. As the error  $e(S)$  comes from a contiguous interval, this probability is at most  $1/(t_1 \cdot t_2)$ . The same result holds for attacks on the two summands  $S_p$  and  $X \cdot (S_q - S_p)$ .  $\square$

- **Attack on  $X$**

The success probability for an attack on  $X$  is at most  $1/(t_1 \cdot t_2)$  for all but a fraction of  $2/\min(t_1, t_2)$  messages. This probability is taken over random choices for the error.

*Comment.* An attack on  $X$  will result in  $S' = S + e(X) \cdot (S_q - S_p)$ , which may be an undetectable error if  $t_1 \cdot t_2 | e(X) \cdot (S_q - S_p)$ . The probability for this requirement is at most  $1/\min(t_1, t_2)$ . A detailed analysis of this case can be found in Appendix A. Moreover, the error  $e(X)$  also needs to be a multiple of either  $p$  or  $q$  in order to apply the Bellcore attack.

As with attacks on  $d_p$ , the adversary has no information on how to construct a message that yields  $S_q - S_p$  to be a multiple of  $t_1, t_2$  or both. His best choice is to choose random messages, which only gives him a negligible success probability. Hence, this attack is not promising to an adversary.

- **Attack on  $S_p$  or  $S_q$  or  $(S_q - S_p)$**

The success probability for an attack on  $S_p$  or  $S_q$  is less than  $1/t_2$ . This probability is taken over random choices of the error only.

*PROOF.* If an adversary attacks either  $S_p$  or  $S_q$  in the second summand, the output of the CRT combination is  $S' = S + e(S_p) \cdot X$  (or  $e(S_q)$  respectively). This causes an undetectable error if  $t_1 \cdot t_2 | e(S_p) \cdot X$ . As  $t_1$  and  $t_2$  are primes, this means that both  $t_i$  have to divide at least one factor. As  $t_1$  always divides  $X = pt_1 \cdot ((pt_1)^{-1} \pmod{qt_2})$  and  $t_2$  never divides  $X$  by Condition 5, this may only happen if  $t_2 | e(S_p)$ . Because  $e(S_p)$  is uniformly distributed over an interval of consecutive numbers, the success probability is at most  $1/t_2$ . The same reasoning holds for  $(S_q - S_p)$ .  $\square$

*Comment.* If  $t_2$  is not chosen carefully to prevent  $t_2|X$ , the success probability is increased with the probability to meet  $t_2|X$ . This is independent from the error, therefore any error  $e(S_p) \neq 0$  would be harmful. This explains Condition 5 of the condition list for selecting  $t_1$  and  $t_2$ .

## 5.4 Attacks On Lines 4 – 6

We also need to investigate the possibilities to attack the detection mechanism, lines 4 - 6. But attacks on the computation of  $c_i$  are in vain unless another successful attack has been carried out already. If a random error into a correct  $c_i$  is induced, it is  $c_i \neq 1$  and the final signature will look like a random value. The same consideration applies to line 6.

### Excluding the two messages $m = 0$ and $m = 1$ .

The analysis shows that choosing  $m \in \{0, 1\}$  leads to a malicious message as  $m \equiv 0, 1 \pmod{t_i}$  in these cases as well. The choice  $m = 0$  or  $m = 1$  is useful for an adversary in attacks on  $m$ , on the values  $S_p$  and  $S_q$ , and on the intermediate results of the exponentiation. Therefore, these two messages must be treated separately. For all other cases, the adversary's ability to create malicious messages implies knowledge about  $t_i$ . As we assume these parameters to be secret, the adversary has no better choice than to choose  $m$  at random. This leaves him with a success probability of at most  $1/t_i$ . Hence,  $t_i$  is a security parameter and can be chosen large enough to effectively prevent efficient attacks.

Now let us explain several methods dealing with the case  $m \in \{0, 1\}$ . The first method is to use padding schemes. In fact, almost any padding scheme, deterministic or randomized, will ensure that  $m = 0$  and  $m = 1$  will either not be signed at all or will only be signed with negligible probability. However, as explained in the introduction, most smartcard certification authorities require that a smartcard implements a pure RSA signature algorithm that is secure without using OAEP or similar padding schemes.

To avoid padding schemes, one can modify the message used in lines 1 and 2 in the following way: In line 1 one uses the message  $m_p := m + r_1 \cdot p$  and in line 2 the message  $m_q := m + r_2 \cdot q$ . Here,  $1 < r_i < t_i$ ,  $i = 1, 2$  are fixed numbers. Obviously, it should hold that  $r_1 \cdot p \pmod{t_1} \notin [-2, \dots, 2]$ , and for line 2 equivalently. In this way, the algorithm actually computes  $m^d \pmod{N}$ . This blinding technique is also useful against other side channel attacks.

fault attack on	probability of the attack
line 1	$3/\min(t_1, t_2)$
line 2	$3/\min(t_1, t_2)$
line 3	$2/\min(t_1, t_2)$
line 4	0 in our fault model
line 5	0 in our fault model
line 6	0 in our fault model

**Table 1.** Summarizing the success probabilities of a fault attack adversary

**Summarizing the results.** Table 1 shows the most successful attack scenarios on each line of Algorithm 2. Summarizing the results of this section, the probability to induce an error that can fool our countermeasure and still break the system by the Bellcore attack is negligibly small if the bitlength of  $t_1$  and  $t_2$  is large enough. Additionally, in the real world various randomization strategies are applied on the card to counteract other side-channel attacks. These

measures show that malicious messages, which have been shown to exist for some attacks, are virtually impossible to create.

## 5.5 Further Security Considerations

Note that disclosure of most intermediate variables can be used to break the system. E.g. if our countermeasure prevents a Bellcore attack on a faulty  $S_p$  using  $c_1 \neq 1$ ,  $c_2 = 1$  and  $c_1$  is revealed, then  $\gcd(m^e - \text{Sig}^{c_1}, N) = p$ . This also implies that the bitlength of the parameters  $t_i$  must be large enough to defend against a brute force search on  $c_i$  from lines 4 and 5 of Algorithm 2.

The length of the two parameters  $t_i$  should be as small as possible to ease the cost of computation, but it must be large enough to guarantee security. Section 5.1 shows that the most promising attacks succeed with a probability of at most  $3/\min(t_1, t_2)$ . Hence, both  $t_1$  and  $t_2$  must be large enough to ensure that attacking the scheme succeeds only with negligible probability. The definition of "small" and "negligible" will have to be adapted to the actual implementation of a system using our algorithm. If we assume a very high level of security, we will demand a security of  $2^{80}$ , i.e.  $l(t_i) > 80$ . Less conservative security considerations may allow to reduce this bound. Practical applications may only need to guarantee the security of the signature key for a small time like 2 years — today's credit cards incorporate the same security feature. In cases like these,  $l(t_i) = 60$  seems to be secure (the SETI@home project as one of the largest open attacks achieved about  $2^{61}$  operations [19]). If less powerful attacks are assumed, this level might be lowered even further.

## 6. CONCLUSIONS

The fault models described in Section 3 show that the adversaries known in the literature can be described with few parameters. This description leads to a common model with proper mathematical formulations. The models always assume quite natural the most powerful adversary known. However, the power of the adversary is gradually reduced due to countermeasures in effect on the attacked smartcard.

The proposed algorithm develops known ideas into a form that can be proven to be secure within the presented framework with respect to the Bellcore attack. The only problem still unsolved by software mechanisms poses the fault model #1 (precise bit error attacks on an unprotected smartcard). If the power of the adversary is not reduced by hardware or software means, the adversary may perform a successful oracle attack circumventing the proposed countermeasure: As attacks like those described in [21] or [17] allow to set any specific bit to any specific value, we just need  $l(x)$  steps to determine the bit pattern, and hence the value, of any parameter  $x$ . We guess a bit value, set that bit and verify whether the final result and therefore the guess is correct or not. This reveals any attacked parameter and only uses the knowledge whether a fault occurred or not. No specific value is needed. No efficient software countermeasure protecting against such a fault can withhold that information from the adversary. Therefore, we insist that all currently proposed CRT-RSA implementations are broken by this attack. To prevent this fault model #1, smartcards must fight the cause of error rather than the effect on computation to reduce the power of the adversary significantly. Luckily, various but not all hardware manufacturers of cryptographic

devices such as smartcard ICs have been aware of the importance of protecting their chips against intrusion. To do so they use carefully developed logic families, sensors, filters, regulators, etc. We are also investigating possible software countermeasures that use random bits to alter the parameters. As oracle attacks need to test several identical runs of an algorithm, this will effectively reduce the power of the adversary. We will elaborate this idea and present an enhanced algorithm in a different paper shortly.

## 7. REFERENCES

- [1] R. Anderson and M. Kuhn. Tamper resistance – a cautionary note. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 1 – 11, Oakland, California, November 18-21 1996. USENIX Association.
- [2] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault attacks on RSA with CRT: Concrete results and practical countermeasures. In *Workshop on Cryptographic Hardware and Embedded Systems 2002 (CHES 2002)*, Hotel Sofitel, San Francisco Bay (Redwood City), USA, August 13–15 2002.
- [3] F. Bao, H. Deng, R. Y. Jeng, D. Narasimhalu, A. and T. Ngair. Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults. In B. Christianson, B. Crispo, M. Lomas, and M. Roe, editors, *Security Protocols*, volume 1362 of *Lecture Notes in Computer Science*, pages 115–124. Springer-Verlag, 1998.
- [4] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in cryptology—EUROCRYPT '94 (Perugia)*, Lecture Notes in Computer Science, pages 92–111. Springer, Berlin, 1995.
- [5] J. Blömer and A. May. personal communication, 2002.
- [6] J. Blömer and J.-P. Seifert. Fault based cryptanalysis of the Advanced Encryption Standard (AES). In *Seventh International Financial Cryptography Conference (FC 2003) (Gosier, Guadeloupe, FWI January 27-30)*, 2003.
- [7] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults. In W. Fumy, editor, *Advances in Cryptology - EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 1997.
- [8] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptology*, 14(2):101–119, 2001.
- [9] C. Clavier, J.-S. Coron, and N. Dabbous. Differential power analysis in the presence of hardware countermeasures. In *Cryptographic Hardware and Embedded Systems – Proceedings of CHES 2000, Worcester, MA, USA*, volume 1965 of *Lecture Notes in Computer Science*, pages 252–263. Springer-Verlag, 2000.
- [10] J.-S. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *Proceedings of Cryptographic Hardware and Embedded Systems (CHES'99)*, volume 1717 of *Lecture Notes in Computer Science*, page 292 ff. Springer-Verlag, 1999.
- [11] J.-S. Coron, P. Kocher, and D. Naccache. Statistics and secret leakage. In *Proceedings of Financial Cryptography*, volume 1962 of *Lecture Notes in Computer Science*, page 157 ff. Springer-Verlag, 2000.
- [12] C. Couvreur and J. Quisquater. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronic Letters*, 18(21):905–907, 1982.
- [13] G. Hardy and J. Littlewood. Some problems of 'Partitio Numerorum' III: On the expression of a number as a sum of primes. In *Acta Mathematica*, volume 44, pages 1–70, 1922.
- [14] M. Joye, J.-J. Quisquater, S.-M. Yen, and M. Yung. Observability analysis: Detecting when improved cryptosystems fail. In B. Preneel, editor, *Topics in Cryptology – CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 17–29, San Jose, CA, USA, February 18–22, 2002, February 2002. Springer-Verlag.
- [15] B. Kaliski, Jr. and M. Robshaw. Comments on some new attacks on cryptographic devices. Bulletin 5, RSA Laboratories, July 1997.
- [16] I. Peterson. Chinks in digital armor – exploiting faults to break smart-card cryptosystems. *Science News*, 151(5):78–79, 1997.
- [17] J.-J. Quisquater and D. Samyde. Eddy current for magnetic analysis with active sensor. In *Proceedings of Esmart 2002 3rd edition. Nice, France*, September 2002.
- [18] W. Rankl and W. Effing. *Smart Card Handbook*. John Wiley & Sons, 2 edition, 2000.
- [19] T. SETI@home project. Current total statistics, June 28th 2002.
- [20] A. Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks, 1999. US Patent No. 5,991,415, Nov. 23, 1999.
- [21] S. Skorobogatov and R. Anderson. Optical fault induction attacks. In *Workshop on Cryptographic Hardware and Embedded Systems 2002 (CHES 2002)*, Hotel Sofitel, San Francisco Bay (Redwood City), USA, August 13 - 15, 2002, 2002.
- [22] S.-M. Yen and M. Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, September 2000.
- [23] S.-M. Yen, S. Kim, S. Lim, and S. Moon. A countermeasure against one physical cryptanalysis may benefit another attack. In K. Kim, editor, *Information Security and Cryptology - ICISC 2001*, volume 2288 of *Lecture Notes in Computer Science*, page 414 ff., 4th International Conference Seoul, Korea, December 6-7, 2001. Proceedings, 2001. Springer-Verlag.
- [24] S.-M. Yen, S. Kim, S. Lim, and S. Moon. RSA speedup with residue number system immune against hardware fault cryptanalysis. In K. Kim, editor, *Information Security and Cryptology - ICISC 2001*, volume 2288 of *Lecture Notes in Computer Science*, page 397 ff., 4th International Conference Seoul, Korea, December 6-7, 2001. Proceedings, 2001. Springer-Verlag. (journal version in *IEEE Trans. on Comp.*, April 2003).

## APPENDIX

### A. PROVING THAT $S' = S + E(X) \cdot (S_Q - S_P)$ IS HARMLESS

Assume that an adversary attacks the CRT combination  $S = S_p + X \cdot (S_q - S_p) \bmod N t_1 t_2$  with  $X = p t_1 \cdot ((p t_1)^{-1} \bmod q t_2)$ .  $X$  is assumed to be precomputed and stored on the card. A random error induced into  $X$  will result in a faulty value  $S'$  instead of  $S$ :

$$\begin{aligned} S' &= S_p + X \cdot (S_q - S_p) + e(X) \cdot (S_q - S_p) \bmod N t_1 t_2 \\ &= S + e(X) \cdot (S_q - S_p) \bmod N t_1 t_2, \end{aligned}$$

with  $e(X) \in [-X, 2^{l(X)} - 1 - X]$ . The additional term is the induced error. The countermeasure of Algorithm 2 will fail to detect this fault iff the addend is a multiple of both  $t_1$  and  $t_2$ , i.e. if  $t_1 \cdot t_2 | e(X) \cdot (S_q - S_p) \bmod N t_1 t_2$  because both  $t_1$  and  $t_2$  are different primes. The latter property also implies that at least one of the factors must be a multiple of  $t_1$  and one (possibly the same) a multiple of  $t_2$ .

As we consider the security independent from the adversary's choices for  $m$ , we first assume that neither  $t_1$  nor  $t_2$  divides  $(S_q - S_p)$ . As  $e(X)$  is an equally distributed value from a consecutive interval, and  $t_1$  and  $t_2$  may be seen as independent values, the probability for  $t_1 | e(X)$  and  $t_2 | e(X)$  is at most  $1/(t_1 \cdot t_2)$ .

For the message dependent question whether any of the primes  $t_i$  divides  $(S_q - S_p)$ , let  $S_q := c$  be fixed first (with  $0 \leq c < q t_2$ ). In this case, there are  $p t_1$  integers in  $[c - p t_1 + 1, c]$ . Of these numbers, only multiples of  $t_1$  are counted. Hence, there are at most  $\lfloor (p t_1)/t_1 \rfloor = \lfloor p \rfloor$  many such integers. Therefore, the probability of getting such an integer is  $\leq p \cdot 1/(p t_1) = 1/t_1$ . If we now count the overall number of possible integers for all choices of  $c$ , we determine

$$\begin{aligned} \Pr[(S_q - S_p) = k \cdot t_1] &= \sum_{c=0}^{q t_2 - 1} \Pr[(S_q - S_p) = k \cdot t_1 | S_q = c] \cdot \Pr[S_q = c] \\ &= \sum_{c=0}^{q t_2 - 1} \Pr[(c - S_p) = k \cdot t_1 \text{ for some } k] \cdot \frac{1}{q t_2} \\ &\leq \frac{1}{q t_2} \cdot q t_2 \cdot \frac{1}{t_1} = \frac{1}{t_1}. \end{aligned}$$

As the same consideration holds for  $t_2$ , we have a maximum of  $2/\min(t_1, t_2)$  messages where the probability that a random error is not detected is significantly higher than  $1/(t_1 \cdot t_2)$ .

### B. UNDETECTABLE BYTE ERRORS

Similar to the analysis in Section 5.1, results for induced byte faults according to the byte error fault models #2 and #3 can be stated. This models an attack on a variable  $x$  as  $f(x) = x + b \cdot 2^k$  with  $|b| \in \mathbb{Z}_{2^8}$ ,  $0 \leq k < l(x) - 7$ . All probabilities stated in the following will be over random choices of errors  $e(x) = b \cdot 2^k$  with random  $b$  and  $k$ . Here,  $b$  will always denote a random byte value, that can be either positive or negative. The analysis of a byte error attack is completely analogous to the analysis for random errors presented in Section 5. The results of the analysis of byte errors are shown in Table 2. The displayed results are all

better than the results for random errors as analyzed in Section 5. This is not surprising, since the special structure of the induced error, i.e.  $b < 2^8$  eliminates some possible attacks. These attacks require that the greatest common divisor of  $b$  and  $t_i - 1$  is large, which is impossible for byte errors. Although, the following two cases yield worse results, because they cannot be based on Assumption 1:

- **attack on the stored variable  $p t_1$**   
If a random byte fault is induced into  $p t_1$ , such that  $p t_1$  is changed to  $p t_1 + b \cdot 2^k$ , an undetectable error requires that  $m^d \bmod (p t_1 + b \cdot 2^k) \equiv 0 \bmod t_1$ .
- **attack on  $m$  or the exponentiation's intermediate variable**

Any random byte fault induced during the exponentiation that causes an intermediate value  $y$  of Algorithm 3 to be changed into  $y + b \cdot 2^k$  must fulfill the equation  $b \cdot 2^k \equiv -2y \bmod t_1$  in order to induce an undetectable error. For messages  $m \equiv 0 \bmod t_1$ , all faults yield an undetectable error.

*Comment.* Both cases described above require the adversary to be able to construct malicious messages in order to be practical. However, the adversary has no information about  $t_1$ , which is needed to construct a malicious message. Therefore, his best choice is to choose random  $m$  for input. In this case, his success probability is negligible.

The two special messages  $m = 0$  and  $m = 1$  need to be excluded from the set of possible inputs. Here, the same considerations as in Section 5 apply.

fault attack on	probability of the attack
line 1	$3/t_1$
line 2	$3/t_2$
line 3	$1/(t_1 \cdot t_2)$
lines 4 – 6	0 in our fault model

**Table 2.** Summarizing the success probabilities of a fault attack adversary for byte faults

### C. UNDETECTABLE BIT ERRORS

The analysis of bit errors is completely analogous to the analysis in section 5. Here, the precise bit error fault model #1 is considered, where a variable  $x$  is changed to  $f(x) = x \pm 2^k$  with  $0 \leq k < l(x)$ . The results are shown in Table 3. Note that similar to the results in Appendix B, attacks on  $p t_1$  require that  $m^d \bmod (p t_1 \pm 2^k) \equiv 0 \bmod t_1$  and attacks on  $m$  or the exponentiation's intermediate variable require that  $\pm 2^k \equiv -2y \bmod t_1$  in order to induce undetectable errors. An adversary cannot construct such messages unless he knows  $t_i$ . As these values are secret, his chance of successfully choosing a random  $m$  that satisfies any of these conditions is negligible.

fault attack on	probability of the attack
line 1	$3/t_1$
line 2	$3/t_2$
line 3	$1/(t_1 \cdot t_2)$
lines 4 – 6	0 in our fault model

**Table 3.** Summarizing the success probabilities of a fault attack adversary for bit faults