# On the Importance of Eliminating Errors in Cryptographic Computations*

Dan Boneh
dabo@cs.stanford.edu

Richard A. DeMillo
rad@bellcore.com

Richard J. Lipton
rjl@cs.princeton.edu

**Abstract**

We present a model for attacking various cryptographic schemes by taking advantage of random hardware faults. The model consists of a black-box containing some cryptographic secret. The box interacts with the outside world by following a cryptographic protocol. The model supposes that from time to time the box is affected by a random hardware fault causing it to output incorrect values. For example, the hardware fault flips an internal register bit at some point during the computation. We show that for many digital signature and identification schemes these incorrect outputs completely expose the secrets stored in the box. We present the following results: (1) The secret signing key used in an implementation of RSA based on the Chinese Remainder Theorem (CRT) is completely exposed from a *single* erroneous RSA signature, (2) for non-CRT implementations of RSA the secret key is exposed given a large number (e.g. 1000) of erroneous signatures, (3) the secret key used in Fiat-Shamir identification is exposed after a small number (e.g. 10) of faulty executions of the protocol, and (4) the secret key used in Schnorr's identification protocol is exposed after a much larger number (e.g. 10000) of faulty executions. Our estimates for the number of necessary faults are based on standard security parameters such as a 1024 bit modulus, and a $2^{-40}$ identification error probability. Our results demonstrate the importance of preventing errors in cryptographic computations. We conclude the paper with various methods for preventing these attacks.

**Keywords:** Hardware faults, Cryptanalysis, RSA, CRT, Fiat-Shamir identification, Schnorr identification, Public key systems, Identification protocols.

## 1 Introduction

Direct attacks on the famous RSA cryptosystem seem to require that one factor the modulus. Therefore, it is interesting to ask whether there are attacks that avoid this. The answer is yes: the first was an attack due to Kocher [13] based on timing. Kocher observed that the secret key can be obtained by precisely measuring the *time* that operations took. This allows one to attack the system without directly factoring the modulus. More powerful attacks, due to Kocher et al. [14], show how to obtain the secret key by measuring a device's *power* consumption during decryption.

We present another type of attack that also avoids directly factoring the modulus. We essentially use the fact that from time to time the hardware or software performing the computations *may* introduce errors. We show that erroneous cryptographic values (e.g. erroneous RSA signatures) jeopardize security by enabling an attacker to expose secret information. We describe some environments where

*This is an expanded version of an earlier paper that appeared in Eurocrypt '97 [7].

the attack may apply. Within these environments it is appropriate to implement defenses against the attack as discussed in Section 4.

**Certificate Authority.** A certificate authority (CA) issues certificates to various entities. During certificate generation, the CA uses its private key to sign the data contained in the certificate [17]. The CA's private key is highly guarded since anyone possessing the private key can issue fake certificates. Suppose that during certificate generation a rare computer error on the CA's machine (hardware or software) results in a certificate containing an erroneous CA signature. We show that such invalid certificates can completely expose the CA's private key. At the extreme, a *single* erroneous certificate is sufficient to recover the CA's private key. Note that typically the user is alerted whenever an invalid certificate is received, at which point the user could try to exploit this certificate.

**Web server.** A web server uses a secret key to authenticate itself to a web browser and to establish a secure session with the browser. Suppose that during key exchange, a rare computer error on the web server causes it to miscalculate. The resulting value sent to the browser can completely expose the server's private key.

**Smartcard.** Smartcards are typically used to authenticate their owners and sign certain contracts on behalf of their owners. As before, a glitch in the smartcard's processor may cause it to send an erroneous value to the outside world. These values expose the secret keys stored on the card.

**Obfuscated keys.** Several software products contain an embedded secret key. The secret key is "hidden" in the software so that it is supposedly hard to extract from the executable. For example, several software audio players running on desktop computers contain a secret key used to defend against music piracy. The embedded key is used to decrypt encrypted music sent to the user. To extract the embedded key, an attacker may add a single malicious instruction to the decryption code, thus causing the decryption process to malfunction. The invalid decryptions produced expose the secret key embedded in the player.

One may wonder whether hardware or software errors are a concern. After all, most of the hardware and software used in every day life appears to be reliable. Nevertheless, several scenarios may enable an adversary to collect and possibly cause faults. We group these into three categories.

**Latent faults.** Latent errors are hardware or software bugs that are difficult to catch. As an example, consider Intel's floating point division bug [11]. A crypto library using a faulty floating point unit to speed up multi-precision arithmetic may, on rare occasions, generate incorrect values. Similarly, latent *software* bugs in the multi-precision package are also likely to produce incorrect values.

**Transient faults.** Transient faults are random hardware glitches that cause the processor to miscalculate. These may be caused by power glitches, high temperature, static electricity, etc. A transient error that takes place during signature generation will result in an invalid signature.

**Induced faults.** When an adversary has physical access to a device she may try to purposely induce hardware faults. For instance, one may attempt to attack a tamper-resistant device by deliberately causing it to malfunction. See the discussion by Anderson and Kuhn [1] for examples of tampering with tamper resistant devices. Fortunately, most smartcards have built in sensors to detect various forms of tampering. It is likely that the cost of inducing useful faults is higher than the potential gains.

## 1.1 The attack model

Throughout the paper our model consists of a black box interacting with the outside world according to a predefined protocol. The black box contains secret keys that are inaccessible to the outside world. For example, a CA may be viewed as a black box that issues certificates on demand. The CA's private key is stored inside the box. The adversary's goal is to interact with the black box and extract the secret keys stored in it using only the values output by the box. The assumption is that, on rare occasions, errors within the box machinery (either hardware or software) cause it to output incorrect values. The attacks described in the paper show how these values enable an adversary to deduce the secret keys stored inside the box.

The attack described in Section 2.2 is the most powerful and is capable of dealing with arbitrary errors. Other attacks in the paper assume more "hardware-like" errors. We refer to these more specialized errors as *register faults.* The idea is as follows: suppose that at some point during a computation (such as signature generation) a temporary value stored in a register is corrupted. More precisely, one bit in the register flips between the time the value is loaded onto the register and the time it is read out of the register. Typically, the bit flip results from a premature power drain on one of the register cells. Hence, a flip from a '1' to a '0' is more likely than the reverse, but we do not make use of this fact. We will show that the secret keys used in several cryptographic schemes are completely exposed in the presence of register faults.

## 1.2 Summary of results

Our attack is effective against several cryptographic schemes such as the RSA system and Rabin signatures [19] as well as several identification protocols. As expected, the effectiveness of the attack depends on the exact implementation of each of these schemes. We briefly review the results:

- For public key systems we present the following results:

  **RSA+CRT.** For an implementation of RSA based on the Chinese Remainder Theorem (CRT) we show that given *one* erroneous RSA signature one can efficiently factor the RSA modulus with high probability. The same approach can also be used to attack Rabin's signature scheme. Our attack shows that one invalid signature along with a valid signature on the same message are sufficient for factoring the modulus. A later improvement due to Lenstra [15] shows that an invalid signature along with the original message to be signed are sufficient.

  **RSA.** Register faults can be used to attack other implementations of the RSA system though many more erroneous signatures are required. When an $n$ bit RSA modulus is used the number of required faults is $O(n)$.

- For identification schemes we show the following:

  **Fiat-Shamir.** A few erroneous executions of the Fiat-Shamir identification protocol [8] enable an adversary to recover the private key of the party trying to authenticate itself. When a single execution of the protocol has security $2^{-t}$ we require $O(t)$ erroneous executions. Furthermore, in case the prover is a smartcard the adversary mounts the attack by inducing a register fault while the card is waiting for a challenge. Thus, precise timing of the induced register fault is not necessary.

**Schnorr.** Similar results hold for Schnorr's identification protocol [20] though a larger number of erroneous executions is necessary. When an $n$ bit modulus is used the number of executions is $O(n \log n)$. Both attacks use faults that corrupt the prover while it is waiting for a challenge from the verifier.

Since the initial publication of our results several authors devised faults based attacks on other cryptographic systems. Biham and Shamir [5] presented elegant and novel attacks on DES. Some of their techniques can be used to recover the secret key of a *totally unknown cipher*. Anderson and Kuhn [2] used a different fault model to obtain attacks against symmetric ciphers. Bao et al. [3] devised fault attacks against DSS and several other signature schemes. Joye and Quisquater [12] noted that the CRT attacks (described in the next section) can also be mounted against several elliptic curve systems. Finally, Zheng and Matsumoto [22] showed how faults in the random number generator can be used to attack systems.

It is important to emphasize that the attacks described in this paper are currently theoretical. We are not aware of any published results physically experimenting with this type of attack. The purpose of these results is to demonstrate the danger that hardware or software bugs pose to various cryptographic systems. In conjunction with Kocher's work our results show that a pure mathematical analysis of a cryptographic algorithm is insufficient. One must also analyze the actual implementation to ensure it does not leak timing or power information and never outputs faulty values.

There are many ways to prevent attacks based on hardware faults. The simplest solution is to ensure the black-box verifies the values it computes before sending them out to the outside world. In protocols where the black-box has to keep some state (such as in identification protocols) our results show the importance of protecting the registers storing the state information using error detection bits (e.g. CRC). Preventing errors is crucial in many areas unrelated to cryptography. For instance, special precautions are taken to ensure error-free computations in core memories of large computers [16], in computers onboard satellites crossing the Van Allen belt, and many other embedded control systems. Scientists working in these areas may not be aware that their techniques are also critical for securing cryptographic implementations. We discuss methods for preventing errors in cryptographic computations in Section 4.

We note that FIPS [9] publication 140-1 suggests that hardware faults may compromise the security of a module. Our results explicitly demonstrate the extent of damage caused by such faults. We give algorithms that show how certain faults can expose sensitive security information. FIPS 140-1 also specifies a list of self tests a module should apply to itself. Our results suggest that these tests may be insufficient and full verification of computed values is necessary.

## 2 RSA's vulnerability to hardware faults

We are now ready to describe the various attacks. We begin by describing RSA's vulnerability to hardware faults.

### 2.1 The RSA system

Let $N = pq$ be a product of two large primes each $n/2$ bits long. To sign a message $x \in \mathbb{Z}_N$ using RSA one computes $S = x^d \bmod N$ where $d$ is a secret signing exponent[1]. The computationally expensive

[1]Note that for simplicity we assume the message $x$ is an integer in the range 1 to $N$. Usually one first hashes the message and then pads the result to an integer in that range [18].

part of signing using RSA is the modular exponentiation of $x$. For efficiency most implementations exponentiate as follows: using repeated squaring they first compute $S_1 = x^d \bmod p$ and $S_2 = x^d \bmod q$. They then use the Chinese remainder theorem (CRT) to construct the signature $S = x^d \bmod N$. This last CRT step takes negligible time compared to the two exponentiations. It is done by computing $S = aS_1 + bS_2 \bmod N$ for some predefined constants $a, b \in \mathbb{Z}_N$.

Exponentiation using CRT is much faster than repeated squaring modulo $N$. To see this observe that $S_1 = x^d \bmod p = x^{d \bmod p-1} \bmod p$. Usually $d$ is of order $N$ while $d \bmod p-1$ is of order $p$. Consequently, computing $S_1$ requires half as many multiplications as computing $S$ directly. In addition, intermediate values during the computation of $S_1$ are only half as big — they are in the range $[1, p]$ rather than $[1, N]$. When quadratic time multiplication is used, multiplying two numbers in $\mathbb{Z}_p$ takes $1/4$ the time as multiplying elements in $\mathbb{Z}_N$. Hence, computing $S_1$ takes $1/8$ the time of computing $S$ directly. Computing both $S_1$ and $S_2$ takes $1/4$ the time of computing $S$ directly. Thus, CRT exponentiation is *four times* faster than direct exponentiation. This is why RSA with CRT is the preferred method for generating RSA signatures [17, p. 613].

### 2.2 An attack on "RSA–CRT"

We show that RSA with CRT is especially susceptible to software or hardware errors. The attack enables us to factor the modulus $N$. The attack is based on obtaining two signatures of the same message. One signature is the correct one; the other is a faulty signature.

Let $x \in \mathbb{Z}_N$ be a message and let $S = x^d \bmod N$ be a valid RSA signature of $x$. Let $\hat{S}$ be a faulty signature. Recall that $S$ is computed by first computing $S_1$ and $S_2$. Similarly, $\hat{S}$ is computed by first computing $\hat{S}_1$ and $\hat{S}_2$. Suppose that during the computation of $\hat{S}$ an error occurs during the computation of only *one* of $\hat{S}_1, \hat{S}_2$. Without loss of generality, suppose a hardware fault occurs during the computation of $\hat{S}_1$ (i.e. $S_1 \neq \hat{S}_1 \bmod p$) but no fault occurs during the computation of $\hat{S}_2$ (i.e. $\hat{S}_2 = S_2$). Then $S = \hat{S} \bmod q$, but $S \neq \hat{S} \bmod p$. Therefore,

$$\gcd(S - \hat{S}, N) = q$$

and so $N$ can be easily factored. That's it.

We see that using one faulty signature and one correct signature the modulus $N$ can be efficiently factored. The above attack works under a very general fault model. It makes no difference what type of error or how many errors occur in the computation of $S_1$. All we rely on is the fact that faults occur in the computation modulo only one of the primes. To obtain both a correct signature and a faulty signature of the *same* message an attacker can query the black-box on the same message multiple times. Since standard signature formats (e.g. PKCS1) do not involve any randomness, the same $x$ will be fed through the signing engine every time.

Based on our results Arjen Lenstra [15] observed that one faulty signature of a known message $x$ is sufficient. There is no need to obtain a valid signature as well. For completeness we describe Lenstra's improvement here. Let $S = x^d \bmod N$. Let $\hat{S}$ be a faulty signature obtained under the same model as above, that is $S = \hat{S} \bmod q$ but $S \neq \hat{S} \bmod p$. Then $x = \hat{S}^e \bmod q$ but $x \neq \hat{S}^e \bmod p$, where $e$ is the public exponent used to verify the signature, i.e., $S^e = x \bmod N$. It now follows that

$$\gcd(x - \hat{S}^e, N) = q$$

Lenstra's improvement shows that as long as the entire signed message $x$ is known, a single interaction with the black-box resulting in an invalid signature $\hat{S}$ is sufficient for factoring the modulus.

### 2.2.1 Attacks on other systems using the CRT

The attack on Chinese remainder theorem implementations applies to other cryptosystems as well. For instance, the same attack applies to Rabin's signature scheme [19]. A Rabin signature of a number $x \bmod N$ is the modular square root of $x$. When the extraction of square roots modulo a composite makes use of the CRT the same attack as above applies. Other attacks on systems using the CRT are described in [12].

## 2.3 An attack on RSA without CRT

In the previous section we observed that RSA–CRT is susceptible to hardware or software errors. In this section we show that using register faults it is possible to attack other implementations of RSA as well. The attack is not as practical as attacks on RSA–CRT. Nevertheless, it illustrates the vulnerability of non CRT implementations.

Let $N$ be an $n$-bit RSA composite and $d$ a secret exponent. The exponentiation function $x \longrightarrow x^d \bmod N$ can be computed using the following algorithm (we let $d = d_{n-1}d_{n-2}\dots d_1d_0$ be the binary representation of $d$):

**Algorithm 1:**

**init** $y \leftarrow x \quad ; \quad z \leftarrow 1$.

**main** For $k = 0, \dots, n-1$.
 If $d_k = 1$ then $z \leftarrow z \cdot y \pmod N$.
 $y \leftarrow y^2 \pmod N$.

Output $z$.

When the above algorithm is used, several faulty signatures are sufficient to recover the secret key $d$. Here faulty signatures refer to signatures obtained in the presence of register faults (see Section 1.1). The attack uses erroneous signatures of *random* messages in $\mathbb{Z}_N$ (as opposed to chosen messages). Furthermore, the attacker need not obtain the correct signature of any of the messages nor does he need to obtain multiple signatures of the same message.

The attack proceeds as follows: the attacker asks the black-box to sign messages $M_1, M_2, \dots, M_l$. The attacker collects the responses until she has sufficiently many erroneous signatures $\hat{S}_i$. The pairs $\langle M_i, \hat{S}_i\rangle$ are then used to deduce the secret signing key $d$. We assume that for each pair $\langle M_i, \hat{S}_i\rangle$ a *single* register fault occurs during the computation of $\hat{S}_i$. The fault occurs at a random iteration during the exponentiation algorithm and flips one bit of the value stored in the variable $z$. The following result was the starting point of our research on fault based cryptanalysis.

**Theorem 2.1** *Let $N$ be an $n$-bit RSA modulus. For any $1 \le m \le n$, given $(n/m)log2n$ pairs $\langle M_i, \hat{S}_i\rangle$, the secret exponent $d$ can be extracted from a black-box implementing the above exponentiation algorithm with probability at least $\frac{1}{2}$. The probability is over the location of the register faults and the random messages $M_i \in \mathbb{Z}_N$. The algorithm's running time is dominated by the time it takes to perform $O((2^m n^3 \log^2 n)/m^2)$ full modular exponentiations mod $N$.*

**Remark:** Taking $m = \log 2n$ shows that the secret $d$ can be recovered using $n$ faults and $\tilde{O}(n^4)$ modular exponentiations. With $m = 1$ the secret $d$ can be found using $n \log n$ faults and $\tilde{O}(n^3)$ exponentiations.

**Proof** Let $M \in \mathbb{Z}_N$ be a message to be signed. Suppose that at a single random point during the exponentiation algorithm on input $M$ one of the bits of the register $z$ is flipped. We denote the resulting erroneous signature by $\hat{S}$. We show that an ensemble of such erroneous signatures enables one to recover the secret exponent $d$.

Let $l = (n/m)log2n$ and let $M_1, \ldots, M_l \in \mathbb{Z}_N$ be a set of random messages. Let $S_i = M_i^d \bmod N$ be the correct signature on $M_i$. Let $\hat{S}_i$ be an erroneous signature of $M_i$. We are given $\hat{S}_i$ but do not know $S_i$. By assumption, a register fault occurs at exactly one point during the computation of $\hat{S}_i$. For each faulty signature, $\hat{S}_i$, let $k_i$ denote the value of $k$ at the time at which the fault occurred (recall $k$ is the counter used in the exponentiation algorithm). We may sort the messages so that $0 \le k_1 \le k_2 \le \ldots \le k_l < n$. The time at which the faults occur is chosen uniformly (among the $n$ iterations) and independently at random. It follows that given $l$ such faults, with probability at least half, $k_{i+1} - k_i < m$ for all $i = 1, \ldots, l-1$. To see this observe that the probability that no fault occurs in a specific interval of width $m$ is $\left(1 - \frac{m}{n}\right)^l < \frac{1}{2n}$. Since there are at most $n$ such intervals the probability that all of them contain a fault is at least $1 - n \cdot \frac{1}{2n} = \frac{1}{2}$. Note that since we do not know where the faults occur, the values $k_i$ are unknown to us.

Let $d = d_{n-1} \ldots d_1 d_0$ be the bits of the secret exponent $d$. We recover a block of these bits at a time starting with the MSBs. Suppose we already know bits $d_{n-1}d_{n-2} \ldots d_{k_i}$ for some $i$. Initially $i = l + 1$ indicating that no bits are known. We show how to expose the bits of $d$ in positions $k_i - 1,\ k_i - 2,\ \ldots,\ k_{i-1}$. To simplify the notation let $a = k_i$ and $c = k_{i-1}$. To expose the block of bits $d_{a-1}d_{a-2} \ldots d_{c+1}d_c \in \{0,1\}^{a-c}$ we intend to try all possible bit vectors until the correct one is found. Since even the length of the block, namely $a - c$, is unknown we try all possible lengths. The attack algorithm works as follows:

1. For all lengths $r = 1, 2, 3 \ldots, m$ do:
2. For all candidate $r$-bit vectors $u = u_{a-1}u_{a-2} \ldots u_{a-r}$ do:
3. Set $w = \sum_{j=a}^{n-1} d_j 2^j + \sum_{j=a-r}^{a-1} u_j 2^j$. In other words, $w$ matches the bits of $d$ and the bits of $u$ at all bit positions that are already exposed and is zero everywhere else.
4. Test if the current candidate bit vector $u$ is correct by checking if one of the erroneous signatures $\hat{S}_j$ for $j = 1, \ldots, l$ satisfies

$$\exists b \in \{0, \ldots, n\} \text{ s.t. } \left(\hat{S}_j \pm 2^b M_j^w\right)^e = M_j \pmod N$$

   Recall that $e$ is the public signature verification exponent. The $\pm$ means that the condition is satisfied if it holds with either a plus or minus.

5. If a signature satisfying the above condition is found, output $u_{a-1}u_{a-2} \ldots u_{a-r}$ and stop. At this point we know that $k_{i-1} = c = a - r$ and $d_{a-1}d_{a-2} \ldots d_{a-r} = u_{a-1}u_{a-2} \ldots u_{a-r}$. Hence, $r$ more bits of $d$ are exposed.

We show that the condition at step (4) is satisfied by the correct candidate $u_{a-1}u_{a-2} \ldots u_c$. To see this recall that $\hat{S}_{i-1}$ is obtained from a fault at the $c = k_{i-1}$'st iteration. That is, at the $k_{i-1}$'st iteration the value of $z$ was changed to $\hat{z} \leftarrow z \pm 2^b$ for some $b$ (corresponding to a register fault on the bit in position $b$). A simple property of Algorithm 1 is that just before the fault took effect we had $z = M_{i-1}^{d_c d_{c-1} \ldots d_0} \bmod N$. By definition of $w$ it follows that $S_{i-1} = z \cdot M_{i-1}^w \bmod N$. Since no faults occurred in the remaining iterations, replacing $z$ by $\hat{z}$ produces an erroneous signature $\hat{S}_{i-1}$ satisfying:

$$\hat{S}_{i-1} = \hat{z} \cdot M_{i-1}^w = (z \pm 2^b) M_{i-1}^w = S_{i-1} \pm 2^b M_{i-1}^w \pmod N$$

When in step (4) the erroneous signature $\hat{S}_{i-1}$ is corrected (by adding $2^b M_{i-1}^w$) it properly verifies when raised to the public exponent $e$. Consequently, when the correct candidate $u$ is tested, the faulty signature $\hat{S}_{i-1}$ guarantees that it is accepted.

To bound the running time of the algorithm we bound the number of times the condition of step (4) is executed. Each invocation of Step (4) requires $n \cdot l$ modular exponentiations. Working through the loops in steps (1) and (2) we see that the total number of modular exponentiations is at most

$$n \cdot l \cdot \left[ \sum_{r=1}^{n-k_l} 2^r + \sum_{r=1}^{k_l - k_{l-1}} 2^r + \ldots + \sum_{r=1}^{k_2 - k_1} 2^r + \sum_{r=1}^{k_1} 2^r \right] \leq nl \left[ l \cdot \sum_{r=1}^{m} 2^r \right] \leq 2nl^2 2^m$$

The first inequality follows from the fact that $k_i - k_{i-1} < m$ for all $i$. Plugging in the value for $l$ we see that the total run time is dominated by the time it takes to perform $O((2^m n^3 \log^2 n)/m^2)$ modular exponentiations.

We still need to show that a wrong candidate $u'$ will not pass the test of step (4). This is done in the following lemma. The lemma shows that, when the encryption/decryption exponents $\langle e, d \rangle$ are chosen at random, and the messages $M_1, \ldots, M_l \in Z_N$ are random, a wrong candidate $u'$ will pass the test with negligible probability.

**Lemma 2.2** *Let $c > 0$ be a fixed constant. For all $n$-bit RSA moduli $N = pq$ at least one of the following claims hold:*

1. *The probability that a wrong candidate $u'$ passes the test of Step (4) is less than $1/n^c$. The probability is over the random choice of messages $M_i \in \mathbb{Z}_N$ given to the attack algorithm and the random choice of the decryption exponent $d$.*
2. *There is a uniform polynomial time (in $n$ and $2^m$) algorithm for factoring $N$.*

**Proof** We show an algorithm that factors all RSA moduli $N$ for which part (1) is false. The algorithm works as follows: it picks a random exponent $d$ and random messages $M_1, \ldots, M_l \in \mathbb{Z}_N$. It then computes erroneous signatures $\hat{S}_i$ of the $M_i$ by using the exponentiation algorithm (Algorithm 1) to compute $M_i^d \bmod N$ and deliberately simulating a random register fault at a random iteration. Let $\langle M_i, \hat{S}_i \rangle_{i=1}^l$ be the resulting set of faulty signatures. We show there is a polynomial time (in $n$ and $2^m$) algorithm that given this data succeeds in factoring $N$ with probability at least $1/n^c$.

Suppose the attack algorithm were given $\langle M_i, \hat{S}_i \rangle_{i=1}^l$ as input. By assumption, with probability at least $1/n^c$, at some point during the algorithm a signature $\hat{S}_v$ will incorrectly cause the wrong candidate $u'$ to be accepted in Step (4). That is, $\hat{S}_v \pm 2^b M_v^w = S_v \bmod N$ even though $\hat{S}_v$ was generated by a different fault (here $w$ is defined as in step (3) using the bits of $u'$). We know that $\hat{S}_v = S_v \pm 2^{b_1} M_v^{w_1}$ for some $b_1, w_1$ with $w_1 \neq w$. The pair $b_1, w_1$ correspond to the actual location of the fault during the computation of $\hat{S}_v$. Then,

$$S_v \pm 2^{b_1} M_v^{w_1} = S_v \pm 2^b M_v^w \pmod N$$

Rearranging terms we get $M_v^{w-w_1} = \pm 2^{b_1 - b} \pmod N$. In other words, $M_v$ must be a root of a polynomial of the form

$$x^{w-w_1} = a \pmod N \tag{1}$$

for some constant $a$. Recall that the message $M_v$ is chosen independently of the fault location, i.e. independently of $w, w_1$ and $a$. It follows that a random $x \in \mathbb{Z}_N$ must satisfy $x^{w-w_1} = a \bmod N$ with non-negligible probability. We show that consequently $\gcd(w - w_1, \varphi(N))$ is very large enabling us to factor $N$. First, we bound the number of roots of $x^{w-w_1} = a \bmod N$. Define $A_p = \gcd(w - w_1, p-1)$ and $A_q = \gcd(w - w_1, q-1)$. The number of roots of the polynomial $x^{w-w_1} = a \bmod N$ is exactly $\alpha = A_p \cdot A_q$. Hence, the probability that $\hat{S}_v$ causes the wrong $u'$ to be accepted is $\alpha/N$. Note that $\gcd(w - w_1, \varphi(N)) = \alpha$.

To bound the probability that a wrong candidate $u'$ is accepted throughout the algorithm we count the number of pairs $w, w_1$. The value of $w_1$ is essentially the prefix of $d$ from the MSBit to the fault location. Since we have $l$ faulty signatures there are $l$ possible values for $w_1$. The values of $w$ are the ones tested in Step (4). There are at most $l \cdot 2^m$ possible values. Hence, there are $l^2 2^m$ possible values for $w - w_1$. Let $\bar{\alpha}$ be the maximum value of $\alpha$ over all pairs $w, w_1$. The probability that a wrong candidate is ever accepted is at most $l^2 2^m \cdot \frac{\bar{\alpha}}{N}$.

By assumption, part (1) is false. Hence, with probability at least $1/n^c$ (over the choice of $d$) we have that $l^2 2^m \cdot \frac{\bar{\alpha}}{N} > 1/n^c$. Let $\mathcal{A}$ be the event that $l^2 2^m \cdot \frac{\bar{\alpha}}{N} > 1/n^c$. When $\mathcal{A}$ occurs there exists a pair $w, w_1$ such that $\gcd(w - w_1, \varphi(N)) = \alpha > N/(l^2 2^m n^c)$. The factoring algorithm factors $N$ by trying all pairs $w, w_1$ until the correct one is found. For each pair it computes $\gcd(N, g^{t(w-w_1)/2} - 1)$ for a random $g \in \mathbb{Z}_N$ and all $t \in [1, \ldots, l^2 2^m n^c]$. Once $t(w - w_1)$ is a multiple of $\varphi(N)$ the algorithm will factor $N$ with probability $\frac{1}{2}$. Hence, when the event $\mathcal{A}$ occurs the algorithm factors $N$ in polynomial time with probability $\frac{1}{2}$. Since $\Pr[\mathcal{A}] > 1/n^c$ repeating this process $n^c$ times will factor $N$ with constant probability. $\square$

**Remark 1:** If one allows the attacker to obtain both the erroneous and correct signature of each message $M_i$ then the running time of the attack algorithm can be improved. The test at step (4) can be simplified to

$$\exists b \in \{0, \ldots, n\} \text{ s.t. } \hat{S}_j \pm 2^b M_j^w = S_j \pmod N$$

thus saving the need for an RSA encryption on every invocation of the test.

**Remark 2:** The messages $M_i$ used by the attack algorithm were assumed to be random elements of $\mathbb{Z}_N$. This was necessary for the proof of Lemma 2.2. However, it should be clear that heuristically almost any set of messages $\{M_i\}$will make the attack algorithm succeed in exposing the private key $d$. In particular, one can use elements of $\mathbb{Z}_N$ that are formated according to the PKCS1 standard [18]. Similarly, the decryption exponent $d$ was assumed to be random. Again, the attack is certain to work for *any* valid $d$. In particular, it will work for a $d$ that correspond to a low public exponent $e$, e.g. $e = 65537$.

## 3 Attacks on identification protocols

We now turn our attention to attacks on identification protocols. Throughout we describe a scenario in which a prover Alice is authenticating herself to a verifier Bob. At setup time Alice publishes some public information (public accreditation information) and keeps certain values secret (secret accreditation information). Whenever she wishes to authenticate herself to Bob she proves knowledge of the secret information. She does so by engaging Bob in a zero-knowledge proof of knowledge [8]. We show that for several classic identification protocols, the presence of register faults on Alice's machine enables Bob to completely extract Alice's secret accreditation information.

### 3.1 The Fiat-Shamir identification scheme

We begin by discussion the Fiat-Shamir [8] identification scheme. Alice and Bob first agree on an $n$-bit modulus $N$ which is a product of two large primes, and a security parameter $t$. A typical value for $t$ is $t = 10$. At setup time Alice chooses her secret accreditation information as a set of random invertible elements $s_1, \ldots, s_t \bmod N$. Her public accreditation information is the square of these numbers $v_1 = s_1^2, \ldots, v_t = s_t^2 \bmod N$. To authenticate herself to Bob they engage in the following protocol:

1. **Commitment:** Alice picks a random $r \in \mathbb{Z}_N^*$ and sends $z = r^2 \bmod N$ to Bob.
2. **Challenge:** Bob picks a random subset $S \subseteq \{1, \ldots, t\}$ and sends the subset to Alice.
3. **Response:** Alice computes $y = r \cdot \prod_{i \in S} s_i \bmod N$ and sends $y$ to Bob.
4. **Verify:** Bob verifies Alice's response by checking that $y^2 = z \cdot \prod_{i \in S} v_i \pmod N$. The protocol completes successfully if the response verifies, and fails otherwise.

The probability that an imposter who does not know the secret information succeeds in fooling Bob is $2^{-t}$. Typically, the protocol is repeated a small number of times (e.g. four times) to reduce the probability of error. Using $t = 10$ and iterating the protocol four times results in an error probability of $2^{-40}$.

For the purpose of authentication one may implement Alice's role in a tamper resistant device. The device contains the secret information and is used by Alice to authenticate herself to various parties. We show that using register faults one can extract the secret $\langle s_1, \ldots, s_t \rangle$ from the device. We use register faults that occur while the device is waiting for a challenge from the outside world.

**Theorem 3.1** *Let $N$ be an $n$-bit modulus and $t$ the predetermined security parameter of the Fiat-Shamir protocol. Given $t$ erroneous executions of the protocol one can recover the secret $\langle s_1, \ldots, s_t \rangle$. The algorithm's running time is dominated by the time it takes to perform $O(nt + t^2 \log t)$ modular multiplications. The faults are collected over $t$ separate runs of the protocol, each fault being a 1-bit register fault in the variable $r$.*

**Proof** Suppose that due to a register fault, one of the bits of the register holding the value $r$ is flipped while the device is waiting for Bob to send it the challenge set $S$. In this case, Bob receives the correct value $r^2 \bmod N$, however $y$ is computed incorrectly by the device. Due to the fault, the device outputs:

$$\hat{y} = (r + E) \cdot \prod_{i \in S} s_i \pmod N$$

where $E$ is the value added to the register as a result of the fault. Since the fault is a single bit flip we know that $E = \pm 2^b$ for some $b = 0, \ldots, n-1$. Observe that Bob knows the value $\prod_{i \in S} v_i$ and he can therefore compute $(r + E)^2$ using:

$$(r + E)^2 = \frac{\hat{y}^2}{\prod_{i \in S} v_i} \pmod N$$

Since there are only $n$ possible values for $E$, Bob can try all of them until the correct one is found. Using the correct value of $E$ Bob can recover $r$ since he has both $(r + E)^2 \bmod N$ and $z = r^2 \bmod N$. Indeed,

$$r = \frac{(r + E)^2 - r^2 - E^2}{2E} = \frac{\frac{\hat{y}^2}{\prod_{i \in S} v_i} - z - E^2}{2E} \pmod N$$

Bob's ability to discover the secret random value $r$ is the main observation that enables him to attack the system. Using the value of $r$ and $E$ Bob can compute:

$$\prod_{i \in S} s_i = \frac{\hat{y}}{r+E} = \frac{2E \cdot \hat{y}}{\frac{\hat{y}^2}{\prod_{i \in S} v_i} - z + E^2} \pmod{N} \tag{2}$$

We now show that Bob can verify that a candidate value $E$ is correct. Let $T$ be the hypothesized value of $\prod_{i \in S} s_i$ obtained from the above formula. To test if $T$ is correct Bob can verify that the relation $T^2 = \prod_{i \in S} v_i \bmod N$ holds. Usually only one of the possible values for $E$ will satisfy the relation. In such a case Bob correctly obtains the value of $\prod_{i \in S} s_i$.

Even in the unlikely event that two values $E, E'$ satisfy the relation, Bob can still attack the system. Suppose two candidate values $E, E'$ generate two values $T, T', T \neq T'$ satisfying the relation. Clearly $T^2 = (T')^2 \bmod N$. If $T \neq -T' \bmod N$ then Bob can already factor $N$ by computing $\gcd(N, T - T')$. Suppose $T = -T' \bmod N$. Then since one of $T$ or $T'$ must equal $\prod_{i \in S} s_i$ (one of $E, E'$ is the correct fault value) it follows that Bob now knows $\prod_{i \in S} s_i \bmod N$ up to sign. For our purposes this is good enough.

The testing method above enables Bob to check whether a certain value of $E$ is the correct one. By testing all $n$ possible values of $E$ until the correct one is found Bob can determine $\prod_{i \in S} s_i$. Computing $\prod_{i \in S} v_i$ in Equation (2) takes $O(t)$ modular multiplications. Evaluating Equation (2) for all $n$ possible values of $E$ takes time $O(n+t)$ modular multiplications (and inversions). This is the time to determine $\prod_{i \in S} s_i$ for a single set $S$. For $t$ sets we need $O(nt + t^2)$ modular multiplications.

So far we showed that Bob is able to obtain $\prod_{i \in S} s_i$ for arbitrary sets $S$ of his choice. We briefly show that this enables him to quickly recover $\langle s_1, \ldots, s_t \rangle$. The simplest approach is for Bob to obtain $\prod_{i \in S} s_i$ for singleton sets, i.e. sets $S$ containing a single element. If $S = \{k\}$ then $\prod_{i \in S} s_i = s_k$ and hence the $s_i$'s are immediately found. However, it is possible that Alice may refuse to accept singleton sets $S$. In this case Bob can still find the $s_i$'s as follows. We represent a set $S \subseteq \{1, \ldots, t\}$ by its characteristic vector $U \in \{0,1\}^t$, i.e. $U_i = 1$ if $i \in S$ and $U_i = 0$ otherwise. Bob picks sets $S_1, \ldots, S_t$ such that the corresponding set of characteristic vectors $U_1, \ldots, U_t$ form a $t \times t$ full rank matrix over $\mathbb{Z}_2$. Bob then uses the method described above to construct the values $T_i = \prod_{i \in S_i} s_i$ for each of the sets $S_1, \ldots, S_t$. To determine $s_1$ Bob constructs elements $a_1, \ldots, a_t \in \{0, 1\}$ such that

$$a_1 U_1 + \ldots + a_t U_t = (1, 0, 0, \ldots, 0) \pmod{2}$$

These elements can be efficiently constructed since the vectors $U_1, \ldots, U_t$ are linearly independent over $\mathbb{Z}_2$. When all computations are done over the integers we obtain that

$$a_1 U_1 + \ldots + a_t U_t = (2b_1 + 1, 2b_2, 2b_3, \ldots, 2b_t)$$

for some known integers $b_1, \ldots, b_t$ in the range $[1, t]$. Bob can now compute $s_1$ using the formula

$$s_1 = \frac{T_1^{a_1} \cdots T_t^{a_t}}{v_1^{b_1} \cdots v_t^{b_t}} \pmod{N}$$

Recall that the values $v_i = s_i^2 \pmod{N}$ are publicly available. The values $s_2, \ldots, s_t$ can be constructed using the same procedure. This phase of the algorithm requires $O(t^2 \log t)$ modular multiplications.

To summarize, the entire algorithm above makes use of $t$ faults. The running time is dominated by the time it takes to compute $O(nt + t^2 \log t)$ modular multiplications. $\square$

We emphasize that the faults occur while Alice's device is waiting for a challenge from the outside world. Consequently, there is no need to carefully time the induced fault. The adversary knows to induce a fault on Alice's device while it is waiting for a challenge from the outside world.

We described the algorithm above for the case where a register fault causes a *single* bit flip. More generally, the algorithm can be made to handle a small number of bit flips per register fault. However, finding the correct fault value $E$ becomes harder. When a single register fault causes $c$ bits in the register to flip then the algorithm's running time becomes $O(n^c t + t^2 \log t)$ modular multiplications. Essentially, one has to plug in all possible values for $E$ into Equation (2). The number of candidate $E$'s is $O(n^c)$. The rest of the algorithm remains unchanged.

### 3.2 A modification of the Fiat-Shamir scheme

One may suspect that our attack on the Fiat-Shamir scheme is successful due to the fact that the scheme is based on squaring. Recall that Bob was able to compute the random value $r$ chosen by the device since he was given $r^2$ and $(r+E)^2$ where $E$ is the fault value. One may try to modify the scheme and use higher powers. We show that our techniques can be used to attack this modified scheme as well.

The modified scheme uses some publicly known exponent $e$ instead of squaring. As before, Alice's secret key is a set of invertible elements $s_1, \ldots, s_t \bmod N$. Her public key is a set of numbers $v_1 = s_1^e, \ldots, v_t = s_t^e \bmod N$. To authenticate herself to Bob they engage in the following protocol:

1. **Commitment:** Alice picks a random $r$ and sends $z = r^e \bmod N$ to Bob.
2. **Challenge:** Bob picks a random subset $S \subseteq \{1, \ldots, t\}$ and sends the subset to Alice.
3. **Response:** Alice computes $y = r \cdot \prod_{i \in S} s_i \bmod N$ and sends $y$ to Bob.
4. **Verify:** Bob verifies Alice's response by checking that $y^e = r^e \cdot \prod_{i \in S} v_i \pmod N$ .

When $e = 2$ this protocol reduces to the original Fiat-Shamir protocol. Using the methods described in the previous section Bob can obtain the values $L_1 = r^e \bmod N$ and $L_2 = (r+E)^e \bmod N$. As before we may assume that Bob guessed the value of $E$ correctly. Given these two values Bob can recover $r$ by observing that $r$ is a common root of the two polynomials

$$x^e = L_1 \pmod N \quad \text{and} \quad (x+E)^e = L_2 \pmod N$$

Furthermore, $r$ is very likely to be the only common root of the two polynomials. Consequently, when the exponent $e$ is polynomial in $n$ Bob can recover $r$ by computing the GCD of the two polynomials. Once Bob has a method for computing $r$ he can recover the secrets $s_1, \ldots, s_t$ as discussed in the previous section.

### 3.3 Schnorr's identification scheme

The security of Schnorr's identification scheme [20] is based on the hardness of computing discrete log modulo a prime. Alice and Bob first agree on a prime $p$ and an element $g \in \mathbb{Z}_p^*$ of order $q$ (clearly $q$ divides $p-1$). For efficiency reason one typically chooses $q$ to be much smaller than $p$. For instance, $p$ may be 1024 bits long and $q$ only 160 bits long. Alice then chooses her secret accreditation information by choosing a random element $s \in \mathbb{Z}_q$. Her public accreditation information is $y = g^s \bmod p$. To authenticate herself to Bob, Alice engages in the following protocol:

1. **Commitment:** Alice picks a random integer $r \in \mathbb{Z}_q$ and sends $z = g^r \bmod p$ to Bob.

2. **Challenge:** Bob picks a random integer $t \in [0, T]$ and sends $t$ to Alice. Here $T < q$ is some upper bound chosen ahead of time.

3. **Response:** Alice sends $u = r + t \cdot s \bmod q$ to Bob.

4. **Verify:** Bob verifies that $g^u = z \cdot y^t \bmod p$. The protocol completes successfully if the response verifies, and fails otherwise.

For the purpose of authentication one may implement Alice's role in a tamper resistant device. The device contains the secret information and is used by Alice to authenticate herself to various parties. We show that using register faults one can extract the secret $s$ from the device. We use register faults that occur while the device is waiting for a challenge from the outside world. Throughout the section $\log x$ denotes logarithm of $x$ to the base $e$ where $e$ is the base of the natural logarithm, $e \approx 2.718$.

**Theorem 3.2** *Suppose $q$ used in Schnorr's protocol is an $n$-bit number. Then given $k = n \log 4n$ erroneous executions of the protocol one can recover the secret $s$ with probability at least $\frac{1}{2}$. The algorithm's running time is dominated by the time to perform $O(n^2 \log n)$ modular multiplications. The faults are collected over $k$ separate runs of the protocol, each fault being a 1-bit register fault in the variable $r$.*

**Proof** Bob wishing to extract the secret information stored in Alice's device first picks a random challenge $t$ in $[0, T]$. The same challenge will be used in all invocations of the protocol. Since the device cannot possibly store all challenges given to it thus far, it cannot possibly know that Bob is always providing the same challenge $t$. The attack enables Bob to determine the value $t \cdot s \bmod q$ from which the secret value $s$ can be easily found. For simplicity we set $x = ts \bmod q$ and assume that $g^x \bmod p$ is known to Bob.

Suppose that due to a register fault, one of the bits of the register holding the value $r$ is flipped while Alice's device is waiting for Bob to send it the challenge $t$. Then, when the third phase of the protocol is executed the device finds $\hat{r} = r \pm 2^i$ in the register holding $r$. Consequently, the device will output $\hat{u} = \hat{r} + x \bmod q$. Suppose $\hat{r} = r + 2^i$. Bob can determine the value of $i$ (the fault position) by trying all possible values $i = 0, \ldots, n-1$ until an $i$ satisfying

$$g^{\hat{u}} = g^{2^i} g^r g^x \pmod{p} \tag{3}$$

is found. Assuming a single bit flip, there is exactly one such $i$. The above identity proves to Bob that $\hat{r} = r + 2^i$ showing that the $i$'th bit of $r$ flipped from a 0 to a 1. Consequently, Bob now knows that indeed that $i$'th bit of $r$ must be 0. Similar logic can be used to handle the case where $\hat{r} = r - 2^i$. In this case Bob can deduce that the $i$'th bit of $r$ is 1.

More abstractly, Bob is given $x + r^{(1)}, \ldots, x + r^{(k)} \bmod q$ for random values $r^{(1)}, \ldots, r^{(k)}$ (recall $k = n \log 4n$). Furthermore, Bob knows the value of one bit in each of $r^{(1)}, \ldots, r^{(k)}$. Obtaining this information requires $O(n^2 \log n)$ modular multiplications since for each of the $k$ faults one must test all $n$ possible values of $i$. Each test requires a constant number of modular multiplications.

We claim that using this information Bob can recover $x$ in time $O(n^2)$. We assume the $k$ faults occur at uniformly and independently chosen locations in the register $r$. Note that this uniformity assumption may or may not be true depending on the cause for these faults. Our attack relies on the randomness of the faults. Assuming the faults occur at random bits of $r$ the probability that at least one fault occurs in every bit position of the register $r$ is at least $1 - n\left(1 - \frac{1}{n}\right)^k \geq 1 - n \cdot e^{-\log 4n} = \frac{3}{4}$.

In other words, with probability at least $\frac{3}{4}$, for every $0 \leq i < n$ there exists an $r^{(i)}$ among $r^{(1)}, \ldots, r^{(k)}$ such that the $i$'th bit of $r^{(i)}$ is known to Bob (we regard the first bit as the LSB).

To recover $x$ Bob first guesses the $\log 8n$ most significant bits of $x$. Later we show that Bob can verify whether his guess is correct. Bob tries all possible $\log 8n$ bit strings until the correct one is found. Let $X$ be the integer that matches $x$ on the most significant $\log 8n$ bits and is zero on all other bits. For now we assume that Bob correctly guessed the value of $X$. Bob recovers the rest of $x$ starting with the LSB. Inductively suppose Bob already knows bits $x_{i-1} \ldots x_1 x_0$ of $x$ (Initially $i = 0$). We show how Bob computes the $i$'th bit of $x$. Let $Y = \sum_{j=0}^{i-1} 2^j x_j$.

Bob determines $x_i$ using $r^{(i)}$. He knows the $i$'th bit of $r^{(i)}$ and the value of $x + r^{(i)} \bmod q$. Let $b$ be the $i$'th bit of $r^{(i)}$. We view $x, X, Y$ and $r^{(i)}$ as integers in the range $[0, q)$. Then assuming $0 \leq x + r^{(i)} - Y - X < q$ we have that:

$$[x]_i \quad = \quad b \oplus \left[(x + r^{(i)}) - Y - X \bmod q\right]_i \tag{4}$$

where for any integer $w$ we use $[w]_i$ to denote the $i$'th bit in the binary representation of $w$. Equation (4) follows from two facts. First observe that the condition $0 \leq x + r^{(i)} - Y - X < q$ implies that the modulo $q$ has no effect. Second, observe that $[x - Y - X]_i = [x]_i$ and that $[x - Y - X]_j = 0$ for all $j < i$. Therefore, the $i$'th bit of $(x - Y - X) + r^{(i)}$ is $[x]_i \oplus [r^{(i)}]_i$ which is simply $[x]_i \oplus b$. Equation (4) immediately follows. Therefore, assuming $0 \leq x + r^{(i)} - X - Y < q$ Bob can easily obtain $x_i$, the $i$'th bit of $x$.

By construction we know that $0 \leq x - X - Y < q/8n$. Hence, the condition $0 \leq x + r^{(i)} - Y - X < q$ will fail only if $r^{(i)} > (1 - \frac{1}{8n})q$. Since $r^{(i)}$ is uniformly chosen in the range $[0, q)$ the probability that the condition is not satisfied is $\frac{1}{8n}$. Since the $r$'s are independent of each other, the probability that the condition is satisfied for all $i = 1, \ldots, n$ is $(1 - \frac{1}{8n})^n > \frac{3}{4}$.

To summarize, we see that for the algorithm to run correctly two events must simultaneously occur. First, all bits of $r$ must be "covered" by faults. Second, all the $r^{(i)}$ must be less than $(1 - \frac{1}{8n})q$. Since each event occurs with probability at least $\frac{3}{4}$, both events happen simultaneously with probability at least $\frac{1}{2}$. Consequently, with probability at least $\frac{1}{2}$, once $X$ is guessed correctly the algorithm requires $O(n)$ modular additions and outputs the correct value of $x$. Of course, once a candidate $x$ is found it can be easily verified using the public data, by testing that $y^t = g^x \bmod p$ (recall that $x$ was defined as $x = st$). Computing $g^x \bmod p$ takes $O(n)$ modular multiplications. There are $O(n)$ possible values for $X$ and hence the running time of this step is $O(n^2)$ modular multiplications. Since the first part of the algorithm takes $O(n^2 \log n)$ modular multiplications it dominates in the overall running time. $\square$

We note that the attack also works when a register fault induces multiple bit flips in the register $r$ (i.e. $\hat{r} = r + \sum_{j=1}^{c} 2^{i_j}$). When an error results in $c$ bits being flipped the location of these errors can be found in time $O(n^c)$. To do so one tries all possible error vectors until one satisfying Equation (3) is found. Once all error vectors are found, the same algorithm as in the proof of Theorem 3.2 can be used to recover $x$.

We also note that the faults we use occur while Alice's device is waiting for a challenge from the outside world. Consequently, the adversary knows at exactly what time the faults should be induced.

## 4 Defending against attacks based on hardware faults

One can envision several methods for defending against the type of attack discussed in the paper. The simplest method is for the device to check the output of the computation before releasing it. Though this extra verification step may reduce system performance, our attack suggests that it is crucial for security reasons. In some systems verifying a computation can be done efficiently (e.g. verifying an RSA signature when the public exponent is 3). In other systems verification appears to be costly (e.g. DSS).

Due to the extreme vulnerability of RSA–CRT checking appears to be necessary whenever it is used. This is especially true for Certification Authorities where a single transient fault could leak the private key. Shamir [21] presented a clever technique for verifying signatures generated by the RSA–CRT method. When the public exponent $e$ is small (e.g. 3) standard verification (i.e. raising the signature to the power of $e$) is still the best way to go. However, for larger values of $e$ Shamir's trick is a clear win. For completeness we describe Shamir's approach. Recall that in RSA–CRT one signs a message $M$ by computing $S_1 = M^d \bmod p$ and $S_2 = M^d \bmod q$. The results are then combined with CRT to build $S$. Shamir suggests picking a small random number $r$ (e.g. 32 bits) and computing $S_1 = M^d \bmod pr$ and $S_2 = M^d \bmod qr$. The overhead in performance is negligible. One then checks that $S_1 \bmod r = S_2 \bmod r$. If the test fails, an error occurred in one of the exponentiations. If the test succeeds the signature $S$ is constructed from $S_1 \bmod p$ and $S_2 \bmod q$. Overall, the performance cost is negligible. The check done modulo $r$ defends against a random error during the exponentiation. Other methods should be used to defend against errors in the CRT step.

Our attack on authentication protocols such as the Fiat-Shamir scheme uses a register fault which occurs while the device is waiting for a response from the outside world. One can not protect against this type of a fault by simply verifying the computation. As far as the device is concerned, it computed the correct output given the input stored in its memory. Therefore, to protect multi-round authentication schemes one must ensure that the internal state of the device can not be affected. Consequently, our attack suggests that for security reasons devices must protect internal memory by adding some error detection bits (e.g. CRC).

Another way to prevent our attack on RSA signatures is to introduce randomness into the signature process. See for instance the system suggested by Bellare and Rogaway [4]. In such schemes RSA is applied to $F(M, r)$ where $F$ is some formatting function and $r$ is a random string. The randomness ensures that the signer never signs the same message twice. Furthermore, given an erroneous signature the verifier does not know the full plain-text $F(M, r)$ that was signed ($r$ is not a part of the message $M$). Consequently, the attack of Section 2.2 cannot be applied to such a system.

## 5 Summary and open problems

We described a general attack which makes use of hardware faults. The attack applies to several cryptosystems. We showed that encryption schemes using Chinese remainder, e.g. RSA and Rabin signatures, are especially vulnerable to this kind of attack. Other implementations of RSA are also vulnerable though many more faults are necessary. The idea of using hardware faults to attack cryptographic protocols applies to authentication schemes as well. For instance, we explained how the Fiat-Shamir and Schnorr identification protocols may be attacked using hardware faults.

Verifying the computation and protecting internal storage using error detection bits defeats attacks based on hardware faults. We hope that this paper demonstrates that these measures are necessary for *security reasons*. Methods of program checking [6] may come in useful when verifying computations in cryptographic protocols. A result of Frankel, Gemmel and Yung [10] could prove useful in this context.

An obvious open problem is whether the attacks described in this paper can be improved. That is, can one mount a successful attack using fewer faults? For instance, can a general implementation of RSA be attacked using significantly fewer faults than $n$, say $\sqrt{n}$? (here $n$ is the size of the modulus). Such a result would significantly improve our Theorem 2.1. Another interesting question is whether an implementation of the Bellare-Rogaway signature scheme [4] based on RSA–CRT can be attacked using a single erroneous signature.

## Acknowledgments

We are grateful to Arjen Lenstra for his helpful comments. We also thank Adi Shamir and Eli Biham for their enthusiasm and research into Differential Fault Analysis. We thank the anonymous referees for their great help in improving the presentation.

## References

[1] R. Anderson, M. Kuhn, "Tamper resistance — a cautionary note", in Proc. of the second USENIX workshop on electronic commerce, pp. 1–11.

[2] R. Anderson, M. Kuhn, "Low Cost Attacks on Tamper Resistant Devices", in proc. of 5th Security Protocols Workshop, LNCS 1361, Springer-Verlag, pp. 125–136, 1997.

[3] F. Bao, R. Deng, Y. Han, A. Jeng, A.D. Narasimhalu, T. Ngair, "Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults", in proc. of 5th Security Protocol Workshop, LNCS 1361, Springer-Verlag, pp. 115–124, 1997.

[4] M. Bellare, P. Rogaway, "The exact security of digital signatures - How to sign with RSA and Rabin", in Proc. Eurocrypt 96, LNCS 1070, Springer-Verlag, pp. 399–416, 1996.

[5] E. Biham, A. Shamir, "Differential fault analysis of secret key cryptosystems", in Proc. of Crypto' 97, LNCS 1294, Springer-Verlag, pp. 513–528, 1997.

[6] M. Blum, H. Wasserman, "Program result checking", in proc. of 35th annual symposium on foundations of computer science, IEEE, pp. 382–392, 1994.

[7] D. Boneh, R. DeMillo, R. Lipton, "On the importance of checking cryptographic protocols for faults", in Proc. of Eurocrypt '97, LNCS 1233, Springer-Verlag, pp. 37–51, 1997.

[8] U. Feige, A. Fiat, A. Shamir, "Zero knowledge proofs of identity", J. of Cryptology, Vol. 1, No. 2, pp. 77–94, 1988.

[9] Federal Information Processing Standards, "Security requirements for cryptographic modules", FIPS publication 140-1, `http://www.nist.gov/itl/csl/fips/fip140-1.txt`.

[10] Y. Frankel, P. Gemmell, M. Yung, "Witness based cryptographic program checking and robust function sharing", in Proc. of 28th annual symposium on theory of computing, ACM, pp. 499–508, 1996.

[11] Analysis of the floating point flaw in the Pentium processor, November, 1994, `http://www.intel.com/procs/support/pentium/fdiv/white11/index.htm`

[12] M. Joye, J.J. Quisquater, "Attacks on systems using chinese remaindering", Tech. Report CG-1996/9, UCL Crypto Group, Belgium, http://www.dice.ucl.ac.be/crypto/techreports.html

[13] P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems", in Proc. of Crypto 96, LNCS 1109, Springer-Verlag, pp. 104–113, 1996.

[14] P. Kocher, J. Jaffe, B. Jun, "Differential Power Analysis: Leaking Secrets", to appear in Crypto '99.

[15] A.K. Lenstra, Memo on RSA signature generation in the presence of faults, manuscript, Sept. 28, 1996. Available from the author, arjen.lenstra@citicorp.com

[16] R. McEliece, "Reliability of Computer Memories", Scientific American, Vol. 252, No. 1, pp. 88-92, January, 1985.

[17] A. Menezes, P. van Oorschot and S. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1996.

[18] Public Key Cryptography Standards (PKCS), No. 1, RSA Cryptography Standard, `http://www.rsa.com/rsalabs/pubs/PKCS/`

[19] M. Rabin, "Digital signatures and public key functions as intractable as factorization", MIT Laboratory for computer science, Technical report MIT/LCS/TR-212, Jan. 1979.

[20] C. Schnorr, "Efficient signature generation by smart cards", J. of Cryptology, Vol. 4, pp. 161–174, 1991.

[21] A. Shamir, "How to check modular exponentiation", Rump session, Eurocrypt 97.

[22] Y. Zheng, T. Matsumoto, "Breaking smartcard implementations of ElGamal signatures and its variants", Rump session, AsiaCrypt '96, preprint available at `http://www.pscit.monash.edu.au/~yuliang/`