On the Importance of Checking Computations

(Extended abstract)

Dan Boneh dabo@bellcore.com
Richard A. DeMillo rad@bellcore.com
Richard J. Lipton* lipton@bellcore.com

Math and Cryptography Research Group, Bellcore,
445 South Street, Morristown NJ 07960

Abstract

We present a theoretical model for breaking various cryptographic schemes by taking advantage of random hardware faults. We show how to attack certain implementations of RSA and Rabin signatures. We also show how various authentication protocols, such as Fiat-Shamir and Schnorr, can be broken using hardware faults.

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 the recent attack based on timing [2]. It was observed that a few bits could be obtained from the *time* that operations took. This would allow one to break the system without factoring.

We have a new type of attack that also avoids directly factoring the modulus. We essentially use the fact that from time to time the hardware performing the computations *may* introduce errors. There are several models that may enable a malicious adversary to collect and possibly cause faults. We give a high level description:

Transient faults Consider a certification authority (CA) that is constantly generating certificates and sending them out to clients. Due to random transient hardware faults the CA might generate faulty certificates on rare occasions. If a faulty certificate is ever sent to a client, that client will be able to break the CA's system and generate fake certificates. Note that on various systems, a client is alerted when a faulty certificate is received.

Latent faults Latent faults are hardware or software bugs that are difficult to catch. As an example, consider the Intel floating point division bug. Such bugs may also cause a CA to generate faulty certificates from time to time.

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. The erroneous values computed by the device enable the adversary to extract the secret stored on it.

*Also at Princeton University. Supported in part by NSF CCR–9304718.

We consider a fault model in which faults are transient. That is, the hardware fault only affects the current data, but not subsequent data. For instance, a bit stored in a register might spontaneously flip. Or a certain gate may spontaneously produce an incorrect value. Note that the change is totally silent: the hardware and the system have no clue that the change has taken place. We assume that the probability of such faults is small so that only a small number of them occur during the computation.

Our attack is effective against several cryptographic schemes such as the RSA system and Rabin signatures [5]. The attack also applies to several authentication schemes such as Fiat-Shamir [3] and Schnorr [6]. As expected, the attack itself depends on the exact implementation of each of these schemes. For an implementation of RSA based on the chinese remainder theorem we show that given *one* faulty version of an RSA signature one can efficiently factor the RSA modulus with high probability. The same approach can also be used to break Rabin's signature scheme. Hardware faults can be used to break other implementations of the RSA system though many more faulty values are required.

In Section 4 we show that the Fiat-Shamir identification scheme [3] is vulnerable to our hardware faults attack. Given a few faulty values an adversary can completely recover the private key of the party trying to authenticate itself. In Section 5 we obtain the same result for Schnorr's identification protocol [6]. Both schemes are suitable for use on smart cards.

It is important to emphasize that the attack described in this paper is currently theoretical. It has not yet been experimented with in the lab. The purpose of these results is to demonstrate the danger that hardware faults pose to various cryptographic protocols. The conclusion one may draw from these results is the importance of verifying the correctness of a computation for *security* reasons. For instance, a smart card using RSA to generate signatures should check that the correct signature has indeed been produced. The same applies to a certification authority using RSA to generate certificates. In protocols where the device has to keep some state (such as in identification protocols) our results show the importance of protecting the registers storing the state information by adding error detection bits (e.g. CRC). We discuss these points in more detail at the end of the paper.

2 Chinese remainder based implementations

2.1 The RSA system

In this section we consider a system using RSA to generate signatures in a naive way. Let $N = pq$ be a product of two large prime integers. To sign a message x using RSA the system computes $x^s \pmod N$ where s is a secret exponent. Here the message x is assumed to be an integer in the range 1 to N (usually one first hashes the message to an integer in that range). The security of the system relies on the fact that factoring the modulus N is hard. In fact, if the factors of N are known then one can easily break the system, i.e., sign arbitrary documents without prior knowledge of the secret exponent.

The computationally expensive part of signing using RSA is the modular exponentiation of the input x. For efficiency some implementations exponentiate as follows: using repeated squaring they first compute $E_1 = x^s \pmod p$ and $E_2 = x^s \pmod q$. They then use the Chinese remainder theorem to compute the signature $E = x^s \pmod N$. We explain this last step in more detail. Let a, b be two precomputed integers satisfying:

$$\begin{cases} a \equiv 1 \pmod p \\ a \equiv 0 \pmod q \end{cases} \quad \text{and} \quad \begin{cases} b \equiv 0 \pmod p \\ b \equiv 1 \pmod q \end{cases}$$

Such integers always exist and can be easily found given p and q. It now follows that

$$E = aE_1 + bE_2 \pmod N$$

Thus, the signature E is computed by forming a linear combination of E_1 and E_2. This exponentiation algorithm is more efficient than using repeated squaring modulo N since the numbers involved are smaller.

2.2 RSA's vulnerability to hardware faults

Our simple attack on RSA signatures using the above implementation enables us to factor the modulus N. Once the modulus is factored the system is considered to be broken. Our attack is based on obtaining two signatures of the same message. One signature is the correct one; the other is a faulty signature. At the end of the section we describe an improvement due to Arjen Lenstra [4] that factors the modulus using just a single faulty signature of a known message M.

Let M be a message and let $E = M^s \bmod N$ be the correct signature of the message. Let $\hat{E}$ be a faulty signature. Recall that E and $\hat{E}$ are computed as

$$E = aE_1 + bE_2 \pmod{N} \quad \text{and} \quad \hat{E} = a\hat{E}_1 + b\hat{E}_2 \pmod{N}$$

Suppose that by some miraculous event a hardware fault occurs only during the computation of *one* of $\hat{E}_1, \hat{E}_2$. WLOG, suppose a hardware fault occurs during the computation of $\hat{E}_1$ but no fault occurs during the computation of $\hat{E}_2$, i.e. $\hat{E}_2 = E_2$. Observe that

$$E - \hat{E} = (aE_1 + bE_2) - (a\hat{E}_1 + b\hat{E}_2) = a(E_1 - \hat{e}_1)$$

Now, if $E_1 - \hat{E}_1$ is not divisible by p then

$$\gcd(E - \hat{E}, N) = \gcd(a(E_1 - \hat{E}_1), N) = q$$

and so N can be easily factored. Notice that if the factors of N are originally chosen at random then it is extremely unlikely that p divides $E_1 - \hat{E}_1$. After all, $E_1 - \hat{E}_1$ can have at most $\log N$ factors.

To summarize, using one faulty signature and one correct one the modulus used in the RSA system can be efficiently factored. We note that the above attack works under a very general fault model. It makes no difference what type of fault or how many faults occur in the computation of E_1. All we rely on is the fact that faults occur in the computation modulo only one of the primes.

Arjen Lenstra [4] observed that, in fact, one faulty signature of a known message M is sufficient. Let $E = M^s \bmod N$. Let $\hat{E}$ be a faulty signature obtained under the same fault as above, that is $E \equiv \hat{E} \bmod q$ but $E \not\equiv \hat{E} \bmod p$. It now follows that

$$\gcd(M - \hat{E}^e, N) = q$$

where e is the public exponent used to verify the signature, i.e. $E^e = M \bmod N$. Thus, using the fact that the message M is known it became possible to factor the modulus given only one faulty signature. This is of interest since most implementations of RSA signatures avoid signing the same message twice using some padding technique. Lenstra's improvement shows that as long as the entire signed message is known, even such RSA/CRT systems are vulnerable to the hardware faults attack.

The attack on chinese remainder theorem implementations applies to other cryptosystems as well. For instance, the same attack applies to Rabin's signature scheme [5]. A Rabin signature of a number $x \bmod N$ is the modular square root of x. The extraction of square roots modulo a composite makes use of CRT and is therefore vulnerable to the attack described above.

3 Register faults

>From here on our attacks are based on a specific fault model which we call *register faults*. Consider a tamper-resistant device. We view the device as composed of some circuitry and a small amount of memory. The circuitry is responsible for performing the arithmetic operations. The memory (registers plus a small on chip RAM) is used to store temporary values.

Our fault model assumes that the circuitry contains no faults. On the other hand, a value stored in a register may be corrupted. With low probability, one (or a few) of the bits of the value stored in some register may flip. We will need this event to occur with sufficiently low probability so that there is some likelihood of the fault occurring exactly once throughout the computation. As before, all errors are transient and the hardware has no clue that the change has taken place.

4 The Fiat-Shamir identification scheme

The Fiat-Shamir [3] identification scheme is an efficient method enabling one party, Alice, to authenticate it's identity to another party, Bob. They first agree on an n-bit modulus N which is a product of two large primes and a security parameter t. Alice's secret key is a set of invertible elements $s_1, \ldots, s_t \bmod N$. Her public key is the square of these numbers $v_1 = s_1^2, \ldots, v_t = s_t^2 \pmod N$. To authenticate herself to Bob they engage in the following protocol:

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

We show that by using hardware faults one can recover Alice's secret keys $s_1, \ldots, s_t$. 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.

Theorem 4.1 *Given t faulty runs of the protocol one can recover the secret s with probability at least half using $O(n^2 t)$ arithmetic operations.*

Proof Suppose that due to a miraculous 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 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:

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

where E is the value added to the register as a result of the fault. Observe that since Bob knows the value $\prod_{i \in S} v_i$ he can compute

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

Since E is a binary number of low weight (i.e. a power of 2 or a sum of a few powers of 2), Bob can guess this value. If E is guessed correctly then Bob can recover r since

$$(r + E)^2 - r^2 = 2E \cdot r + E^2 \pmod{N}$$

and this linear equation in r can be easily solved. Bob's ability to discover the secret random value r is the main observation which enables him to break the system. Using the value of r and E Bob can compute:

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

To summarize, Bob can compute the value $\prod_{i \in S} s_i$ by guessing the fault value E and using the formula:

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

We now argue that Bob can verify that the fault value E was guessed correctly. Let T be the hypothesized value of $\prod_{i \in S} s_i$ obtained from the above formula. To verify this value Bob interacts with the device under normal conditions so that the device works properly. The point is that Bob uses the same set S which he used when the device generated the fault. The device will now correctly produce two values $(r')^2$ and $y' = r' \cdot \prod_{i \in S} s_i$. Now Bob simply checks that $(y')^2 = (r')^2 \cdot T^2$. Usually there is only one low-weight value E satisfying the relation. In such a case Bob correctly obtains the value of $\prod_{i \in S} s_i$.

Even in the unlikely event of two values E, E' satisfying the relation, Bob can still break the system. Observe that the relation $(y')^2 = (r')^2 \cdot T^2$ implies that $T^2 = \prod_{i \in S} s_i^2$. If there are two low-weight values E, E' generating two values $T, T', T \neq T'$ satisfying the relation then clearly $T^2 = (T')^2 \bmod N$. If $T \neq -T' \bmod N$ then Bob can already factor N. 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. Since E is a low binary weight integer Bob can try all possible values for E. For instance, if we assume the fault is a single bit flip then there are only n possible values for E (since in this case $E = 2^k$ for some $1 \leq k \leq n$). By testing all possible values for E until the correct one is found Bob can compute $\prod_{i \in S} s_i$.

Observe that once Bob has a method for computing $\prod_{i \in S} s_i$ for various sets S of his choice, he can easily find $s_1, \ldots, s_t$. The simplest approach is for Bob to construct $\prod_{i \in S} s_i$ 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 the device might 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 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 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 $b_1, \ldots, b_t$. 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.

The procedure above made use of t faults and took $O(n^2 t)$ arithmetic operations. □

We emphasize that the faults occur while the device is waiting for a challenge from the outside world. Consequently, the adversary knows at exactly what time the register faults must be induced.

4.1 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 break 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 the 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. Alice picks a random r and sends $r^e \bmod N$ to Bob.
2. Bob picks a random subset $S \subseteq \{1, \ldots, t\}$ and sends the subset to Alice.
3. Alice computes $y = r \cdot \prod_{i \in S} s_i \bmod N$ and sends y to Bob.
4. Bob verifies Alice's identity 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.

We note that the system can be broken even when a large exponent $e < N$ is used, by using a much larger collection of faults. We give the details in the final version of the paper.

5 Attacking Schnorr's identification scheme

The security of Schnorr's identification scheme [6] is based on the hardness of computing discrete log modulo a prime. Alice and Bob first agree on a prime p and a generator g of Z_p^*. Alice chooses a secret integer s and publishes $y = g^s \bmod p$ as her public key. To authenticate herself to Bob, Alice engages in the following protocol:

1. Alice picks a random integer $r \in [0, p)$ and sends $z = g^r \bmod p$ to Bob.
2. Bob picks a random integer $t \in [0, T]$ and sends t to Alice. Here $T < p$ is some upper bound chosen ahead of time.
3. Alice sends $u = r + t \cdot s \bmod p - 1$ to Bob.
4. Bob verifies that $g^u = z \cdot y^t \bmod p$.

For the purpose of authentication one may implement Alice's role in a tamper resistant device. The device contains the secret information s 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.

Theorem 5.1 *Let p be an n-bit prime. Given $n \log n$ faulty runs of the protocol one can recover the secret s with probability at least $\frac{1}{2}$ using $O(n^2)$ arithmetic operations.*

Proof Bob wishing to extract the secret information stored in the device first picks a random challenge 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 will enable Bob to determine the value $t \cdot s \bmod p$ from which the secret value s can be easily found. For simplicity we set $x = ts \bmod p$ and assume that $g^x \bmod p$ is known to Bob.

suppose that due to a miraculous 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 t. More precisely, when the third phase of the protocol is executed the device finds $\hat{r} = r + 2^i$ in the register holding r. Consequently, the device will output $\hat{u} = \hat{r} + x \bmod p$. Bob can determine the value of i (the fault position) by trying all possible values $i = 0, \ldots, n$ until an i satisfying

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

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 $x + r^{(1)}, \ldots, x + r^{(k)} \bmod p$ for random values $r^{(1)}, \ldots, r^{(k)}$ (recall $k = n \log n$). Furthermore, Bob knows the value of some bit of each of $r^{(1)}, \ldots, r^{(k)}$. We claim that from this information Bob can recover x in time $O(n^2)$. We assume the faults occur at uniformly and independently chosen locations in the register r. It follows that with probability at least $\frac{3}{4}$ a fault will occur in every bit position of the register r. In other words, for every $1 \leq i \leq n$ 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 which 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 $x_{i-1} \ldots x_2 x_1$ of x (Initially $i = 1$). Let $Y = \sum_{j=1}^{i-1} 2^j x_j$. To determine bit x_i Bob uses $r^{(i)}$, of which he knows the i'th bit and the value of $x + r^{(i)}$. Let b be the i'th bit of $r^{(i)}$. Then

$$x_i = b \oplus i\text{'th bit}(x + r^{(i)} - Y - X \bmod p - 1)$$

assuming no wrap around, i.e., $x + r^{(i)} - Y - X < p - 1$. Since $x - X < p/8n$ wrap around will occur only if $r^{(i)} > (1 - \frac{1}{8n})p$. Since the r's are independently and uniformly chosen in the range $[0, p)$ the probability that this doesn't happen in all n iterations of the algorithm is more than $\frac{3}{4}$.

To summarize we see that once X is guessed correctly the algorithm runs in linear time and outputs x with probability at least $\frac{1}{2}$. (The reason for the $\frac{1}{2}$ is that both all bits of r should be "covered" by faults and all r_i should not be too large. Both events are satisfied with probability at least $\frac{1}{2}$.) Of course, once a candidate x is found it can be easily verified using the public data. There are $O(n)$ possible values for X and hence the total running time of the algorithm is $O(n^2)$. □

We note that the attack also works in the case of multiple bit flips of the register r. As long as the number of bit flips is constant, their exact location can be found and used by Bob. We also note that the faults we use occur while the device is waiting for a challenge from the outside world. Consequently, the adversary knows at exactly what time the faults should be induced.

6 Breaking other implementations of RSA

In Section 2.1 we observed that CRT based implementations of RSA can be easily broken in the presence of hardware faults. In this section we show that using register faults it is possible to break other implementations of RSA as well. Let N be n-bit RSA composite and s a secret exponent. The exponentiation function $x \longrightarrow x^s \bmod N$ can be computed using either one of the following two algorithms (we regard bit one as the LSB and bit n as the MSB):

- Algorithm I

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

  main For k = 1, . . . , n.
  If k'th bit of s is 1 then $z \leftarrow z \cdot y \pmod N$.
  $y \leftarrow y^2 \pmod N$.

  Output z.

- Algorithm II

  init $z \leftarrow x$.

  main For $k = n$ down to 1.
  If k'th bit of s is 1 then $z \leftarrow z^2 \cdot x \pmod N$.
  Otherwise, $z \leftarrow z^2 \pmod N$.

  Output z.

For both algorithms given several faulty values one can recover the secret exponent in polynomial time. Here by faulty values we mean values obtained in the presence of register faults. The attack only uses erroneous signatures of *randomly* chosen messages; the attacker need not obtain the correct signature of any of the messages. Furthermore, an attacker need not obtain multiple signatures of the same message. The following result was the starting point of our research on fault based cryptanalysis:

Theorem 6.1 *With probability at least* $\frac{1}{2}$*, the secret exponent* s *can be extracted from a device implementing the first exponentiation algorithm by collecting* $(n/m)log n$ *faults and* $O(2^m \cdot n^4)$ *arithmetic steps, for any* $1 \leq m \leq n$.

Proof We use the following type of faults: let $M \in Z_N$ be a message to be signed. Suppose that at a single random point during the computation of $M^s \bmod N$ a register fault occurs. That is, at a random point in the

computation one of the bits of the register z is flipped. We denote the resulting erroneous signature by $\hat{E}$. We intend to show that an ensemble of such erroneous signatures enables one to recover the secret exponent s. Even if other types of faulty signatures are added to the ensemble, they do not confuse our algorithm.

Let $l = (n/m)logn$ and let $M_1, \ldots, M_l \in Z_N$ be a set of arbitrary messages. Set $E_i = M_i^s \bmod N$ to be the correct signature of M_i. Let $\hat{E}_i$ be an erroneous signature of M_i. A register fault occurs at exactly one point during the computation of $\hat{E}_i$. Let k_i be the value of k (recall k in the counter in algorithm I) at the point at which the fault occurs. Thus, for each faulty signature $\hat{E}_i$ there is a corresponding k_i indicating the time at which the fault occurs. We may sort the messages so that $1 \leq k_1 \leq k_2 \leq \ldots \leq k_l \leq n$. 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 $i = 1, \ldots, m$. Since we do not know where the faults occur, the values k_i are unknown to us.

Let $s = s_n s_{n-1} \ldots s_1$ be the bits of the secret exponent s. We recover a block of these bits at a time starting with the MSBs. Suppose we already know bits $s_n s_{n-1} \ldots s_{k_i}$ for some i. Initially $i = l + 1$ indicating that no bits are known. We show how to recover bits $s_{k_i-1} s_{k_i-2} \ldots s_{k_{i-1}}$. We intend to try all possible bit vectors until the correct one is found. Since even the length of the block we are looking for is unknown, we have to try all possible lengths. The algorithm works as follows:

1. For all lengths $r = 0, 1, 2, 3 \ldots$ do:
2. For all candidate r-bit vectors $u_{k_i-1} u_{k_i-2} \ldots u_{k_i-r}$ do:
3. Define s' to be the integer whose bits match those of s at all known bit positions and are zero everywhere else. In other words, $s' = \sum_{j=k_i}^{n} s_j 2^j$. Similarly, define $u' = \sum_{j=k_i-r}^{k_i-1} u_j 2^j$.
4. Test if the current candidate bit vector is correct by checking if one of the erroneous signatures $\hat{E}_j$, $j = 1, \ldots, l$ satisfies
$$\exists e \in \{0, \ldots, n\} \text{ s.t. } \left(\hat{E}_j \pm 2^e M_j^{s'+u'}\right)^d = M_j \pmod N$$
Recall that d 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 $u_{k_i-1} u_{k_i-2} \ldots u_{k_i-r}$ and stop . At this point we know that $k_{i-1} = k_i - r$ and $s_{k_i-1} s_{k_i-2} \ldots s_{k_{i-1}} = u_{k_i-1} u_{k_i-2} \ldots u_{k_i-r}$.

The condition at step (4) is satisfied by the correct candidate $u_{k_i-1} u_{k_i-2} \ldots u_{k_{i-1}}$. To see this recall that $\hat{E}_{i-1}$ is obtained from a fault at the k_{i-1}'st iteration. That is, at the k_{i-1}'st iteration the value of z was changed to $\hat{z} \leftarrow z \pm 2^e$ for some e. Notice that at this point $E_{i-1} = z M_{i-1}^{s'+u'}$. From that point on no fault occurred and therefore the signature $\hat{E}_{i-1}$ satisfies

$$\hat{E}_{i-1} = \hat{z} M_{i-1}^{s'+u'} = E_{i-1} \pm 2^e M_{i-1}^{s'+u'} \pmod N$$

When in step (4) the signature $\hat{E}_{i-1}$ is corrected it properly verifies when raised to the public exponent. Consequently, when the correct candidate is tested, the faulty signature $\hat{E}_{i-1}$ guarantees that it is accepted.

We still need to show that a wrong candidate will not pass the test. Suppose some signature $\hat{E}_v$ incorrectly causes the wrong candidate u' to be accepted at some point in the algorithm. That is, $\hat{E}_v \pm 2^e M_v^{s'+u'} = E_v \bmod N$ even though $\hat{E}_v$ was generated by a different fault. We know that $\hat{E}_v = E_v \pm 2^{e_1} M_v^{u_1}$ for some e_1, u_1. Therefore,

$$E_v \pm 2^{e_1} M_v^{u_1} \pm 2^e M_v^{s'+u'} = E_v \pmod N$$

In other words, M_v is a root of a polynomial of the form $a_1 x^{u1} + a_2 x^{u2} = 0 \bmod N$ for some a_1, a_2, u_1, u_2. This polynomial has a unique root as long as $u_1 - u_2$ is invertible modulo $N - 1$. Since the message M_v is chosen independently of the fault location (i.e. independently of e_1, u_1) it follows that M_v is a root with probability $1/N$. Since the test is invoked only $2^m n^2$ times this type of mistake will happen only with negligible probability ($2^m n^2 \ll N$).

The test at step (4) takes $O(n^3)$ time and therefore the total running of this procedure is $O(2^m n^4)$. □

7 Protecting against an attack based on hardware faults

One can envision several methods of protection 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 might reduce system performance, our attack suggests that it is crucial for security reasons.

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 effected. 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 the use of random padding. See for instance the system suggested by Bellare and Rogaway [1]. In such schemes the signer appends random bits to the message to be signed. To verify the RSA signature the verifier raises the signature to the power of the public exponent and verifies that the message is indeed a part of the resulting value. The random padding ensures that the signer never signs the same message twice. Furthermore, given an erroneous signature the verifier does not know the full plain-text which was signed. Consequently, our attack cannot be applied to such a system.

8 Summary

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 can be broken using hardware faults.

Verifying the computation and protecting internal storage using parity bits defeats attacks based on hardware faults. We hope that this paper demonstrates that these measures are necessary for *`security reasons`*.

References

[1] M. Bellare, P. Rogaway, “The exact security of digital signatures - How to sign with RSA and Rabin”, in Proc. Eurocrypt 96.

[2] P. Kocher, “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems”, Proc. of Cyrpto 96.

[3] U. Feige, A. Fiat, A. Shamir, “Zero knowledge proofs of identity”, Proc. of STOC 87.

[4] A. Lenstra, “Personal communications”.

[5] 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.

[6] C. Schnorr, “Efficient signature generation by smart cards”, J. Cryptology, Vol. 4, (1991), pp. 161–174.