Cryptanalysis of a Provably Secure CRT-RSA Algorithm

David Wagner*
University of California at Berkeley

ABSTRACT

We study a countermeasure proposed to protect Chinese remainder theorem (CRT) computations for RSA against fault attacks. The scheme was claimed to be provably secure. However, we demonstrate that the proposal is in fact insecure: it can be broken with a simple and practical fault attack. We conclude that the proposed countermeasure is not safe for use in its present form.

Categories and Subject Descriptors

E.3 [**Data**]: Encryption

General Terms

Security

Keywords

Fault attacks, RSA, Chinese remainder theorem, cryptanalysis

1. INTRODUCTION

For decades, cryptographers have analyzed the security of cryptosystems by treating them as mathematical entities and analyzing the properties of the underlying cryptographic algorithm. Recently, though, the community has begun to realize that this approach overlooks some important considerations: to ensure security in practice, one must also consider the properties of the implementations of these algorithms. Strikingly, Boneh, DeMillo, and Lipton showed that when cryptographic computations are faulty, the nominal security of the cryptosystem may evaporate [5]. Moreover, practical experience has shown that, in some important cases, attackers may be able to induce faults at will at arbitrary points in a computation [1, 2, 3], putting security at risk. Results like this have motivated researchers to study how to build cryptosystems that are robust even in the presence of faults.

One such countermeasure was proposed by Blömer, Otto, and Seifert in *ACM CCS 2003* [4]. Their scheme is intended to protect Chinese Remainder Theorem computations, as used in many RSA implementations, against fault attacks. They combine several novel and clever ideas into a countermeasure that, on the surface, looks quite promising. Disappointingly, we show in this paper that their proposal is not secure. We review their scheme (Section 2). Then, we exhibit a simple fault attack against their proposal (Section 3), we outline a few other attacks (Section 4), and we conclude with a discussion of the implications of our results (Section 5–6).

2. REVIEW

Notation. Let $N = pq$ denote a RSA modulus, e a public exponent, and d the corresponding private exponent, so that $ed \equiv 1 \pmod{\varphi(N)}$. Let $\mathrm{CRT}(x, y) \bmod N$ denote the number $z \in \mathbb{Z}/N\mathbb{Z}$ such that $z \equiv x \pmod p$ and $z \equiv y \pmod q$; note that this number is uniquely determined modulo N.

We adopt the notation of Blömer, Otto, and Seifert for describing faults [4]. If x is a value produced during some computation, we write $f(x)$ to represent a faulty value that replaces x during a fault attack. Also, we write $f(x) = x + e(x)$ so that $e(x)$ denotes the error introduced by the fault. We write $l(x)$ for the number of bits needed to represent the variable x, so that $x \in \{0, 1, \ldots, 2^{l(x)} - 1\}$.

RSA, CRT, and fault attacks. Many RSA implementations use the Chinese Remainder Theorem (CRT) to speed up private-key computations. In a standard CRT-RSA algorithm, we compute $m^d \bmod N$ as follows: first, compute

$$S_p \stackrel{\text{def}}{=} m^{d_p} \bmod p, \qquad S_q \stackrel{\text{def}}{=} m^{d_q} \bmod q,$$

where $d_p \stackrel{\text{def}}{=} d \bmod p - 1$ and $d_q \stackrel{\text{def}}{=} d \bmod q - 1$; then, compute

$$S \stackrel{\text{def}}{=} \mathrm{CRT}(S_p, S_q) \bmod N$$

and output S. This provides a fourfold speedup over straightforward exponentiation.

However, Lenstra showed that this use of CRT puts implementations at great risk, if faults can occur during the computation [7]. Suppose that the exponentiation modulo p is erroneous, so that instead of receiving the correct value

*daw@cs.berkeley.edu. This research was supported in part by NSF ITR CCR-0113941.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CCS'04, October 25-29, 2004, Washington, DC, USA.
Copyright 2004 ACM 1-58113-961-6/04/0010 ...$5.00.

$S_p = m^{d_p} \bmod p$, the device instead receives some faulty value $f(S_p)$. In this case, the device will compute the erroneous output $f(S) = \mathrm{CRT}(f(S_p), S_q) \bmod N$. Notice that $f(S)$ is correct modulo q but incorrect modulo p. Since $S^e \equiv m \pmod N$ for a correct computation, we see that $f(S)^e \bmod N$ is $\equiv m \pmod q$ but $\not\equiv m \pmod p$. Consequently, $\gcd(m - f(S)^e \bmod N, N)$ reveals p and thereby allows factoring N. This shows that the standard use of CRT-RSA is totally insecure in the presence of faults.

This motivates the search for some way to secure CRT-RSA against fault attacks, so that we can retain the CRT's speed advantages without its susceptibility to fault attacks. The first countermeasure that might come to mind is to check the correctness of the result of the computation before outputting it, so that erroneous values are not revealed to the attacker. This defeats Lenstra's attack, but it turns out that checking the output for correctness is not enough to stop certain other fault attacks [8]. Consequently, we need a stronger defense, if fault attacks are a realistic threat.

In response, Yen, Kim, Lin, and Moon proposed the appealing notion of infective computation [9]. An infective computation is one where any error introduced by a fault propagates throughout the computation, ultimately ensuring that the final result is randomized in such a way that no useful information can be obtained from any erroneous output of the computation. Consequently, with an infective algorithm, there is no need to check the output for errors; any error will not be useful to the attacker. Yen, Kim, Lin, and Moon proposed a candidate scheme claimed to achieve this property, but it was later shown to be insecure [4]. Consequently, it remained a challenging open problem to find some infective implementation of CRT-RSA.

At *ACM CCS 2003*, Blömer, Otto, and Seifert proposed a CRT-RSA scheme based on the ideas of infective computation. Since they did not give their proposal a name, for this paper we will call it the *BOS algorithm*. We describe the BOS algorithm next.

The BOS algorithm. In the BOS algorithm, we first generate random 80-bit primes t_1, t_2. Next, we set

$$\begin{aligned} d_p &\stackrel{\text{def}}{=} d \bmod \varphi(p \cdot t_1), \\ d_q &\stackrel{\text{def}}{=} d \bmod \varphi(q \cdot t_2), \\ e_{t_1} &\stackrel{\text{def}}{=} d^{-1} \bmod t_1, \\ e_{t_2} &\stackrel{\text{def}}{=} d^{-1} \bmod t_2. \end{aligned}$$

These values are kept secret from the attacker and never disclosed; they will be used to check for faults.

With these definitions, the algorithm is as follows [4]:

Input. *A message* $m \in (\mathbb{Z}/N\mathbb{Z})^*$.
Output. $Sig \stackrel{def}{=} m^d \bmod N$.
In memory. $p \cdot t_1,\ q \cdot t_2,\ N,\ N \cdot t_1 \cdot t_2,\ d_p,\ d_q,$ $t_1,\ t_2,\ e_{t_1},$ *and* e_{t_2}.

1. *Set* $\mathsf{S_p} \stackrel{def}{=} \mathsf{m}^{\mathsf{d_p}} \bmod \mathsf{p} \cdot \mathsf{t_1}$.
2. *Set* $\mathsf{S_q} \stackrel{def}{=} \mathsf{m}^{\mathsf{d_q}} \bmod \mathsf{q} \cdot \mathsf{t_2}$.
3. *Set* $\mathsf{S} \stackrel{def}{=} \mathrm{CRT}(\mathsf{S_p}, \mathsf{S_q}) \bmod \mathsf{N} \cdot \mathsf{t_1} \cdot \mathsf{t_2}$.
4. *Set* $\mathsf{c_1} \stackrel{def}{=} (\mathsf{m} - \mathsf{S}^{\mathsf{e_{t_1}}} + 1) \bmod \mathsf{t_1}$.
5. *Set* $\mathsf{c_2} \stackrel{def}{=} (\mathsf{m} - \mathsf{S}^{\mathsf{e_{t_2}}} + 1) \bmod \mathsf{t_2}$.
6. *Set* $\mathsf{Sig} \stackrel{def}{=} \mathsf{S}^{\mathsf{c_1} \cdot \mathsf{c_2}} \bmod \mathsf{N}$.

Note that if there are no errors, we have $c_1 = c_2 = 1$, and thus the BOS algorithm produces the right result. The intuition is that if there are any errors in the computation, then we will most likely have $c_1 \neq 1$ or $c_2 \neq 1$, which renders *Sig* irreversibly altered and (we hope) useless to an attacker.

Threat models. The authors of the BOS algorithm proposed four different threat models under which a candidate algorithm could be evaluated:

Fault Model #1: The attacker can cause a fault in a single bit. The attacker has full control over the timing and location of the fault.

Fault Model #2: The attacker can cause a fault in a single byte. The attacker has full control over the timing, but may have only partial control over the location of the fault (e.g., which byte is affected), and cannot predict the new faulty value that is introduced.

Fault Model #3: The attacker can cause a fault in a single byte. The attacker has only partial control over the timing and location of the fault, and cannot predict the new faulty value.

Fault Model #4: The attacker can cause a fault in a single variable. The attacker has only partial control over the timing and no control over the location; the targetted variable will be replaced by an entirely new random value not known to the attacker.

Note that these models are listed in terms of decreasing attacker power, so a scheme that is secure against one fault model will be secure against all higher-numbered models as well.

Provable security. In the *ACM CCS 2003* paper proposing this construction, the BOS algorithm was claimed to be provably secure against fault attacks: "we rigorously analyze the success probability of an adversary ... we prove that our new algorithm is secure against the Bellcore attack" [4, p.311]. In particular, the paper claims security against Fault Models #2, #3, and #4.

Unfortunately, although the scheme is claimed to be provably secure, there is no statement of a security theorem and no security reduction or mathematical proof. Instead, the paper systematically enumerates many possible attacks and shows that these attacks do not work. Such results are of considerable interest, but they fall short of a proof of security. There is no assurance that the attacks examined exhaust the space of all possible attacks, so the risk is that other methods not anticipated by the designers of the cryptosystem might be able to defeat the scheme. In the remainder of the paper, we show that further analysis in fact reveals serious attacks on the BOS scheme.

3. A FAULT ATTACK

Our fault attack works as follows. We cause a random transient fault that modifies the value of m as it is being read from memory in line 1 while leaving the value stored in memory unaffected, so that further reads will return the correct value of m.

We will assume that the introduced error pattern $e(m)$ is known or guessable by the attacker. For instance, if we know that the fault will cause a single-byte error in m, then the error pattern will be guessable: there are only $510 \cdot \frac{l(m)}{8}$

Correct computation:	**Incorrect computation:**
1. Set $\mathsf{S_p} \stackrel{\text{def}}{=} \mathsf{m}^{\mathsf{d_p}} \bmod \mathsf{p} \cdot \mathsf{t}_1$.	1. Set $\mathsf{f(S_p)} \stackrel{\text{def}}{=} \mathsf{f(m)}^{\mathsf{d_p}} \bmod \mathsf{p} \cdot \mathsf{t}_1$. *(fault!)*
2. Set $\mathsf{S_q} \stackrel{\text{def}}{=} \mathsf{m}^{\mathsf{d_q}} \bmod \mathsf{q} \cdot \mathsf{t}_2$.	2. Set $\mathsf{S_q} \stackrel{\text{def}}{=} \mathsf{m}^{\mathsf{d_q}} \bmod \mathsf{q} \cdot \mathsf{t}_2$.
3. Set $\mathsf{S} \stackrel{\text{def}}{=} \mathrm{CRT}(\mathsf{S_p}, \mathsf{S_q}) \bmod \mathsf{N} \cdot \mathsf{t}_1 \cdot \mathsf{t}_2$.	3. Set $\mathsf{f(S)} \stackrel{\text{def}}{=} \mathrm{CRT}(\mathsf{f(S_p)}, \mathsf{S_q}) \bmod \mathsf{N} \cdot \mathsf{t}_1 \cdot \mathsf{t}_2$.
4. Set $\mathsf{c}_1 \stackrel{\text{def}}{=} (\mathsf{m} - \mathsf{S}^{\mathsf{e}_{\mathsf{t}_1}} + 1) \bmod \mathsf{t}_1$.	4. Set $\mathsf{f(c_1)} \stackrel{\text{def}}{=} (\mathsf{m} - \mathsf{f(S)}^{\mathsf{e}_{\mathsf{t}_1}} + 1) \bmod \mathsf{t}_1$.
5. Set $\mathsf{c}_2 \stackrel{\text{def}}{=} (\mathsf{m} - \mathsf{S}^{\mathsf{e}_{\mathsf{t}_2}} + 1) \bmod \mathsf{t}_2$.	5. Set $\mathsf{f(c_2)} \stackrel{\text{def}}{=} (\mathsf{m} - \mathsf{f(S)}^{\mathsf{e}_{\mathsf{t}_2}} + 1) \bmod \mathsf{t}_2$.
6. Set $\mathsf{Sig} \stackrel{\text{def}}{=} \mathsf{S}^{\mathsf{c}_1 \cdot \mathsf{c}_2} \bmod \mathsf{N}$.	6. Set $\mathsf{f(Sig)} \stackrel{\text{def}}{=} \mathsf{f(S)}^{\mathsf{f(c_1)} \cdot \mathsf{f(c_2)}} \bmod \mathsf{N}$.

Figure 1: On the left, a correct computation. On the right, a faulty computation.

possible values of $e(m)$ (not all equally likely). We will show that the output from the RSA-CRT computation can be used to factor N if the error pattern $e(m)$ can be predicted.

Let us trace the effect of this fault on the operation of the BOS algorithm. The computation in line 1 will use a faulty value $f(m)$ instead of m in the exponentiation, resulting in an erroneous value $f(S_p) \stackrel{\text{def}}{=} f(m)^{d_p} \bmod p \cdot t_1$. Suppose that the rest of the computation proceeds without further disruption. Line 3 computes $f(S) \stackrel{\text{def}}{=} \mathrm{CRT}(f(S_p), S_q) \bmod N \cdot t_1 \cdot t_2$. Line 4 computes $f(c_1) \stackrel{\text{def}}{=} (m - f(S)^{e_{t_1}} + 1) \bmod t_1$. Line 5 computes $f(c_2) \stackrel{\text{def}}{=} (m - f(S)^{e_{t_2}} + 1) \bmod t_2$, but because $f(S) \equiv S \bmod t_2$, we see that $f(c_2) = c_2 = 1$, so c_2 is unaffected. Line 6 computes $f(\mathit{Sig}) \stackrel{\text{def}}{=} f(S)^{f(c_1) \cdot f(c_2)} \bmod N$. Thus the output produced will be erroneous. We show a trace of the faulty computation in Figure 1 side-by-side with a correct computation.

We claim that this erroneous result $f(\mathit{Sig})$ can be used by an attacker to factor N. Note that $f(c_1) = m - f(m) + 1 \bmod t_1 = 1 - e(m) \bmod t_1$, and $e(m) \neq 0$, hence we can be almost certain that $f(c_1) \neq 1$. Also, $f(c_2) = c_2 = 1$, so the erroneous output is given by $f(\mathit{Sig}) = f(S)^{f(c_1)} \bmod N$. Now $f(S) \equiv S \pmod q$ but $\not\equiv S \pmod p$, so $f(S)^e \equiv m \pmod q$ but $\not\equiv m \pmod p$. If we were given $f(S)$, we could factor N by computing $\gcd(m - f(S)^e \bmod N, N)$, but of course we are not given $f(S)$; instead, we are only given $f(S)^{f(c_1)} \bmod N$.

How can we deal with the extra unknown term in the exponent? If we could take the $f(c_1)$-th root of $f(\mathit{Sig})$ modulo N, we would be done, but of course taking roots modulo N is hard when the factorization of N is unknown. Instead, we exploit a slightly different insight.

Assume, for the moment, that $f(c_1)$ is known. Then we are given $f(c_1)$ where $f(c_1) \neq 1$ and given $f(\mathit{Sig}) = f(S)^{f(c_1)} \bmod N$, where $f(S) \equiv S \pmod q$ but $\not\equiv S \pmod p$. The goal is to factor N. In short, the problem is as follows:

> **Problem 1.**
> **Given:** $m \in (\mathbb{Z}/N\mathbb{Z})^*$, $f(\mathit{Sig}) \in (\mathbb{Z}/N\mathbb{Z})^*$, and $f(c_1) \in \mathbb{Z}$, where $f(\mathit{Sig}) = f(S)^{f(c_1)} \bmod N$ for some $f(S) \in (\mathbb{Z}/N\mathbb{Z})^*$ such that $f(S)^e \equiv m \pmod q$ and $f(S)^e \not\equiv m \pmod p$.
> **Goal:** Factor N.

It turns out that this problem is not hard to solve. Let us define the value $X \stackrel{\text{def}}{=} f(\mathit{Sig})^e \bmod N$. Note that $X = (f(S)^e)^{f(c_1)} \bmod N$, so $X \equiv m^{f(c_1)} \pmod q$ but $X \not\equiv m^{f(c_1)} \pmod p$. Consequently, $\gcd(m^{f(c_1)} - X \bmod N, N)$ discloses a factor of N. In other words, the solution is as follows:

> **Solution to Problem 1.**
> Compute $\gcd(m^{f(c_1)} - f(\mathit{Sig})^e \bmod N, N)$.

Is it realistic to assume that $f(c_1)$ will be known to the attacker? Yes, it is. If the fault is a single-byte fault, then the error pattern $e(m)$ is guessable, since there are not too many possible values of $e(m)$. Moreover, $f(c_1) = 1 - e(m) \bmod t_1$, so knowledge of $e(m)$ reveals $f(c_1)$ if $e(m)$ is negative and greater than $-t_1 + 1$, since then no modular reduction occurs. Let us call the fault *usable* in this case, i.e., when $f(c_1)$ can be predicted (without knowledge of t_1). For a random single-byte fault, it is easy to calculate that

$$\Pr[\text{fault is usable}] = \Pr[-t_1 + 1 < e(m) < 0] \approx \frac{1}{2} l(t_1)/l(N).$$

For instance, for a 1024-bit RSA modulus, a 80-bit prime t_1, and a random single-byte error, a random single-byte fault in m has about a 4% chance of being usable in an attack, and in this case there are about 2550 different possible values of $f(c_1)$.

Thus, our attack works as follows. We induce a transient random single-byte fault somewhere in m as it is read from memory in line 1, and we hope that this fault will be usable. We then observe the output $f(\mathit{Sig})$ of the computation, and for each of the 2550 possible values of $f(c_1)$, we compute $\gcd(m^{f(c_1)} - f(\mathit{Sig})^e \bmod N, N)$ and check whether we have learned a non-trivial factor of N. If not, we repeat the attack, iteratively inducing faults until we succeed. On average, we will need to try about 25 times before factoring N, and the total workload is about $25 \times 2550 \approx 2^{16}$ modular multiplications. If instead of assuming that the attacker can introduce random single-byte faults, we assume that the attacker can inject random faults that affect a single 32-bit word, then the complexity of the attack increases to about 32 faults and about $32 \times 2^{33} = 2^{38}$ modular multiplications.

The attack requires only loose control over the timing and location of the fault. Thus, it fits within Blömer, Otto, and Seifert's Fault Model #3. This shows that the BOS algorithm is not secure against Fault Models #2 or #3, despite the fact that it was claimed to be secure against these kinds of attacks.

If we have more precise control over the timing or location of the fault, we may be able to ensure that the fault occurs somewhere among the low 10 bytes of m as it is being read from memory in line 1, and then the fault will be usable in an attack with probability $\frac{1}{2}$. We see that tighter control over the fault will allow the attack to succeed with fewer trials. In summary, the BOS algorithm is insecure not only in theory, but also quite possibly in practice as well.

4. OTHER FAULT ATTACKS

Earlier, we described in detail one possible attack against the BOS scheme. This is enough to demonstrate that the BOS construction is not secure. However, it may be of interest to see other possible attacks against the BOS scheme. Therefore, in this section we outline two other attack methods. This list is not claimed to be exhaustive.

Long-lived faults. The attack in Section 3 required a transient fault. However, if long-lived or permanent faults are easier to induce, the attack can be modified to this setting as well. Suppose we induce a long-lived fault in the value of m stored in memory between lines 1 and 2 of the BOS algorithm. Then the output will be $f(Sig) = f(S)^{f(c_1)} \bmod N$, where $f(S) \equiv m^d \equiv S \pmod p$ and $f(S) \equiv f(m)^d \not\equiv S \pmod q$. Notice that we will have $f(c_1) = e(m) + 1 \bmod t_1$ and $f(c_2) = 1$; thus we are in a situation similar to that of Section 3. We see that $f(Sig)^e \bmod N$ is $\equiv m^{f(c_1)} \pmod p$ but $\not\equiv m^{f(c_1)} \pmod q$, so $\gcd(f(Sig)^e - m^{f(c_1)} \bmod N, N)$ reveals a factor of N. Also, the value of $f(c_1)$ can be predicted if the fault is usable. In short, an attack with long-lived faults can be made to work in almost exactly the same way as an attack with transient faults.

If we assume that the implementation uses the exponentiation algorithm suggested by the designers of the BOS scheme [4, Algorithm 3], then this attack requires only loose control over the timing and location of the fault. This is because their exponentiation algorithm first copies the value m into a variable y and then never uses m again for the duration of the exponentiation, so the attacker can use any long-lived fault that occurs to m at any point during the execution of line 1 of the BOS algorithm. Therefore, this attack also falls into Fault Model #3.

Random, unpredictable faults. So far we have assumed that error patterns are predictable. We show next that it is also possible to mount a fault attack even if we have very little control over the error introduced. Suppose we can introduce a random fault in the value of N stored in memory before it is used in line 6. Suppose moreover that we can arrange that this fault leaves the 160 most significant bits of N undisturbed: the low $l(N) - 160$ bits can be modified in arbitrary ways, but we require that the high 160 bits remain unchanged. For instance, N might be replaced by the value $f(N) = N + e(N)$, where $e(N)$ is uniformly distributed on $[-2^{l(N)-161}, 2^{l(N)-161}]$ and unknown to the attacker. In this setting, we can break the BOS algorithm.

The attack proceeds in two phases. The first phase learns the values t_1, t_2 using a few faults. The second phase learns the factors of N using a slight modification of Lenstra's attack with a single fault and a chosen input. In more detail:

1. First, run the algorithm with no fault. The BOS algorithm outputs $Sig = S \bmod N$. Then, re-run the algorithm on the same input, this time triggering a random fault in the low $l(N) - 160$ bits of N. This time the algorithm outputs $f(Sig) = S \bmod f(N)$. We know N, $\alpha \stackrel{\text{def}}{=} S \bmod N$, and $\beta \stackrel{\text{def}}{=} S \bmod f(N)$, where S and $f(N)$ are unknown but are known to satisfy $0 \le S < 2^{160}N$ and $|f(N) - N| \le 2^{l(N)-161}$. It turns out that this is enough to deduce the value of S and then t_1 and t_2.

   The math is not too difficult. Note that $S = \alpha + k \cdot N$, for some $0 \le k < 2^{160}$. Thus $\beta - \alpha \equiv S - \alpha \equiv k \cdot (N - f(N)) \pmod{f(N)}$. Since $|k \cdot (N - f(N))| < N/2$, there is a good chance that no overflow or modular reduction occurs when computing $|\beta - \alpha| \bmod f(N)$, and in this case $\gamma = |\beta - \alpha|$ is an integer multiple of k. Thus, after gathering several faulty signatures, we obtain a list of values $\gamma_1, \ldots, \gamma_n$ containing many integer multiples of k. Taking many pairwise gcd's, we find that k (or some small multiple thereof) occurs many times among the values $\gcd(\gamma_i, \gamma_j)$, hence the value k can be easily recovered. This reveals S through the equation $S = \alpha + k \cdot N$, which reveals the product $t_1 \cdot t_2 = (S - Sig)/N$. Finally, we may easily factor the 160-bit product $t_1 \cdot t_2$ to learn the values of t_1 and t_2.

   This is already troubling, because it discloses 160 bits of the private key. Given S and m, note that $S \equiv m^{d_p} \pmod{t_1}$, hence by computing a 80-bit discrete log we can learn $d_p \bmod \varphi(t_1)$. This can be done very efficiently, and it reveals $d \bmod \varphi(t_1)$. Likewise we can learn $d \bmod \varphi(t_2)$. This might be viewed as a certificational weakness. As we shall see next, it can also be leveraged further to fully break the BOS algorithm.

2. Next, we will take advantage of our knowledge of t_2 to mount a classical fault attack. Run the BOS algorithm on an input m that is an multiple integer of t_2, and introduce an arbitrary fault in the value of d_q at any point before line 2 is executed. Since $m \equiv 0 \pmod{d_q}$, we will also have $f(S_q) \equiv 0 \pmod{d_q}$ and thus $f(c_2) = 1$. Line 1 is undisturbed, so $f(c_1) = 1$. We see that the fault goes undetected for this input, and the output reveals the factorization of N via $\gcd(f(Sig)^e - m \bmod N, N)$.

   In practice, whether it is possible to choose an input m that is a multiple of t_2 will depend on the application setting, because the hashing present in most signature schemes may prevent the attack. For instance, with PKCS #1 v1.5 signatures, the value m is (essentially) the hash of the message to be signed, so finding a message whose hash is a multiple of t_2 would require about 2^{80} steps of computation. Similar comments apply to PKCS #1 v2.1 (PSS). If the hashing is performed outside of the cryptographic module, though, this chosen-input attack will apply.

This attack requires only loose control over the timing, location, and error pattern of the fault. As such, this attack lies quite close to Fault Model #4. The main difference is that this attack does need just enough control over the error introduced to ensure that the high 160 bits of N are undisturbed.

5. DISCUSSION

Paradox resolved. This demonstrates that the claims of provable security for the BOS algorithm were incorrect. This might seem to present a paradox: How can a scheme that was proven secure later be found insecure?

One answer is that the proof was flawed. If we examine the intended proof of security, we find an important gap in the reasoning. The proof implicitly assumed that if $c_1 \neq 1$ (indicating that a fault has been detected), then the final output of the algorithm would be randomized enough that no useful information can be gleaned from it. This assumption was never clearly stated or justified in the proof. The concluding discussion does mention that if c_1 becomes known, then a variation of the Lenstra attack allows to break the system [4, § 5.7], and the informal description accompanying the algorithm does claim that if $c_1 \neq 1$, line 6 "will change the final result in a way unpredictable to an adversary" [4, § 4]; however, this claim was never examined or justified in the proof. As we have seen, this assumption turns out to be wrong. There is no paradox.

More fundamentally, in hindsight we can spot structural flaws in the reasoning that was intended to establish the security of the scheme. The security analysis only focuses on a class of attacks known to the authors, and does not attempt to prove anything about the possibility of other attacks. For instance, this comment leads off the security analysis:

> "we need to investigate the probability of any induced error to circumvent our countermeasures and result in an undetectable error. Note that we are only concerned with errors that cause the final signature to be correct modulo p but false modulo q (or vice versa), in which case the classic Bellcore attack can be applied. Otherwise, no exploits of specific errors in faulty CRT-RSA signature are known yet" [4, § 5.1].

Our attacks fall outside the class of attacks they focused on, and thereby illustrate the danger in only demonstrating security against previously known attacks. Of course, any valid proof of security must consider *all* attacks, not just known attacks. In retrospect, then, the problem was that the security analysis did not have the structure of a mathematical proof of security. This highlights the need for claims of provable security to be backed up by complete proofs.

Evaluating potential fixes. It is not clear how to repair the BOS scheme in a way that provides any form of provable security.

After seeing the attack of Section 3, one anonymous referee suggested adding to the original algorithm:

5.1. Pick $\mathsf{r}_1 \in [2, \mathsf{t}_1 - 1]$ *and* $\mathsf{r}_2 \in [2, \mathsf{t}_2 - 1]$ *randomly.*
5.2. Set $\mathsf{c}_1 \stackrel{def}{=} \mathsf{c}_1^{\mathsf{r}_1} \bmod \mathsf{t}_1$ *and* $\mathsf{c}_2 \stackrel{def}{=} \mathsf{c}_2^{\mathsf{r}_2} \bmod \mathsf{t}_2$.

This addition will not impose much performance overhead. However, this augmented scheme is still not secure: it does not prevent the second attack described in Section 4.

Another potential defense is to add a check for correctness at the end of the computation:

6.1. If $\mathsf{Sig}^{\mathsf{e}} \bmod \mathsf{N} \neq \mathsf{m}$, *set* $\mathsf{Sig} \stackrel{def}{=} 0$.

This extension will not affect performance too much if the public exponent e is small. However, depending upon the implementation, the resulting scheme seems likely to remain vulnerable to Yen and Joye's attacks on checking before output [8]. These attacks do require control on the location and/or timing of the fault; we have not tried to evaluate whether it is feasible for current devices to defend against such attacks using hardware countermeasures.

We make no claims about the security of these or other fixes, and we would be extremely reluctant to recommend any of these schemes without further analysis. Even if it were possible to defeat all known attacks, we suspect that focusing on stopping known attacks may not be the most fruitful line of research; it is unclear whether such an approach can provide enough confidence that it will stop attacks not anticipated by the designer. We feel that Yen, et al., and Blömer, et al., had the right idea in seeking provable security, and we leave it as a fascinating open problem to find some CRT-RSA algorithm that can be rigorously proven secure against fault attacks.

Towards sound frameworks for fault attacks. One of the challenges facing researchers in this area is the lack of a sound mathematical framework that captures fault attacks. We sketch one form such a theory might take.

We can consider a cryptographic device as a finite-state machine. Let S denote its state space; for example, the state might include the value of all registers and memory. Introduce a relation $\rightsquigarrow$ on $S \times S$ to model the single-step operational semantics of the device, so that $s \rightsquigarrow s'$ if executing a single step of computation on state s yields the new state s'. Computation can then be viewed as a sequence of steps

$$\langle x, k \rangle = s_0 \rightsquigarrow s_1 \rightsquigarrow \cdots \rightsquigarrow s_n = \langle y \rangle,$$

where the initial state $s_0 = \langle x, k \rangle$ somehow encodes the device's input x and its key k, and the output of the computation is contained in the final state $s_n = \langle y \rangle$.

A fault model specifies a family $\mathcal{F}$ of relations on $S \times S$. Each $\twoheadrightarrow_i \in \mathcal{F}$ describes one type of fault that the attacker can select and models the effect of applying this fault to the state of the device. A fault attack is specified by a sequence of relations $\twoheadrightarrow_1, \ldots, \twoheadrightarrow_n \in \mathcal{F}$. The effect of the faulty computation is then given by the chain

$$\langle x, k \rangle = s_0 \rightsquigarrow s_1 \twoheadrightarrow_1 s_1' \rightsquigarrow s_2 \twoheadrightarrow_2 s_2' \rightsquigarrow \cdots \rightsquigarrow s_n = \langle y \rangle.$$

A scheme is secure if an attacker who can choose tuples $(x, \twoheadrightarrow_1, \ldots, \twoheadrightarrow_n)$ and learn the resulting y cannot learn anything useful. For instance, we might require that anything an attacker can learn by injecting faults while interacting with the device could also have been learned by interacting with the device without faults. The power of the attacker could be restricted by limiting his computational power, restricting the family $\mathcal{F}$ of possible faults, and/or placing an upper bound on the number of faults the attacker may inject. To flesh this out more concretely, it would be necessary to instantiate the relation $\rightsquigarrow$ with some specific semantics (e.g., x86 assembly, JVML), to identify a realistic family $\mathcal{F}$ of faults, and to specify appropriate limits on the power of the attacker.

This is just one possible direction that a theory of fault attacks might take. It is a fascinating research problem to establish a principled foundation for security against fault attacks and to find schemes that can be proven secure within that framework.

6. CONCLUSION

We showed how to cryptanalyze a provably secure CRT-RSA algorithm. To summarize, the BOS algorithm for protecting CRT computations against fault attacks is insecure and should not be used.

7. ACKNOWLEDGEMENTS

I thank Martin Otto and the anonymous reviewers for their helpful comments and feedback.

8. REFERENCES

[1] R. Anderson, M. Kuhn, "Tamper resistance—a cautionary note," *2nd USENIX Workshop on Electronic Commerce*, pp.1–11, 1996.

[2] R. Anderson, M. Kuhn, "Low cost attacks on tamper resistant devices," *1997 Security Protocols Workshop.*

[3] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, C. Whelan, "The sorcerer's apprentice guide to fault attacks," *Workshop on Fault Detection and Tolerance in Cryptography*, June 2004.

[4] J. Blömer, M. Otto, J.-P. Seifert, "A new CRT-RSA algorithm secure against Bellcore attacks," *ACM CCS 2003*, ACM Press, pp.311–320, 2003.

[5] D. Boneh, R.A. DeMillo, R.J. Lipton, "On the importance of checking cryptographic protocols for fault," *EUROCRYPT'97*, Springer-Verlag, LNCS 1233, pp.37–51, 1997.

[6] M. Joye, A.K. Lenstra, J.-J. Quisquater, "Chinese remaindering based cryptosystems in the presence of faults," *Journal of Cryptology*, vol. 12, no. 4, pp.241–245, 1999.

[7] A.K. Lenstra, "Memo on RSA signature generation in the presence of faults," Sept. 1996.

[8] S.-M. Yen, M. Joye, "Checking before output may not be enough against fault-based cryptanalysis," *IEEE Transactions on Computers*, vol. 49, no. 9, pp.96–970, 2000.

[9] S.-M. Yen, S. Kim, S. Lim, S. Moon, "RSA Speedup with Residue Number System Immune against Hardware Fault Cryptanalysis," *ICICS 2001*, Springer-Verlag, LNCS 2288, pp.397–413, 2002.