Memo on RSA signature generation in the presence of faults

Arjen K. Lenstra, Citibank, N.A., September 28 - October 28, 1996

Introduction.

This memo was written after reading newspaper articles about a Bellcore attack on smartcards, because no details about the Bellcore attack were available. In the mean time I have found (cf. [1]) that the first attack described here, on RSA signatures generated using Chinese remaindering, is more efficient and potentially more dangerous than the approach used by the Bellcore researchers: my attack requires a message and only one faulty signature of that message, the Bellcore attack requires one correct and one faulty signature of the same message. The second attack described here, on RSA signatures generated without Chinese remaindering, is probably quite similar to the attack proposed by the Bellcore researchers (cf. [1]), though I have not seen any of the details of their approach in this case. The Bellcore researchers have developed similar attacks against the Fiat-Shamit, Schnorr, and Guillou-Quisquater identification schemes and the ElGamal public key system (cf. [1]).

RSA signatures with Chinese remaindering.

Let $n=pq$ be a composite RSA-modulus, with public and secret exponents e and d. The signature $S(m)$ of a message m equals $m^d \bmod n$. Thus, $S(m)^e \bmod n$ is again equal to m modulo n. It is well known that $S(m)$ can be computed by computing m^d both modulo p and q, and by combining the two results using the Chinese remainder algorithm.

If a fault occurs in the course of the computation of the signature, the resulting value $S'(m)$ will most likely not satisfy the congruence $S'(m)^e \equiv m \bmod n$. If, however, the fault occurred only during the computation of m^d modulo p, and if the exponentiation modulo q and the application of the Chinese remainder algorithm were carried out correctly, then the resulting faulty signature $S'(m)$ satisfies $S'(m)^e \equiv m \bmod q$, but the same congruence modulo p does not hold. Therefore, q divides $S'(m)^e-m$, but p does not divide $S'(m)^e-m$, so that a factor of n may be discovered by the recipient of the faulty signature $S'(m)$ by computing the greatest common divisor of n and $S'(m)^e-m$.

This implies that, even if there is the tiniest probability that an error occurs during RSA signature generation, the generator of an RSA-signature must make sure that each signature generated is indeed correct. An inefficient way to do this is by generating the signature twice and checking that the same signature was found twice. This solution assumes that an identical arithmetic error does not occur twice. Since e is usually much smaller than d, a more efficient way to proceed is to check the correctness of the signature in the same way the verifier would do it, namely by checking that $S(m)^e \bmod n$ is equal to m modulo n. Even if an error occurs twice, it is exceedingly unlikely that this test will be satisfied. Note that this possible weakness only occurs if RSA signatures are generated using the method sketched above. But even if $m^d \bmod n$ is computed directly modulo n, the secret key might leak if an error is made during the computation; see below.

Obviously, faulty decryption of RSA-encrypted messages using the Chinese remainder theorem can lead to decrypted messages m' for which $m'-m$ and n (where m is

the correctly decrypted message) have a non-trivial factor in common. RSA-decryptions should therefore not be shared before their correctness is verified.

Arithmetic faults of this type may be triggered in devices with relatively low level security specifications if they are subjected to exceptional physical circumstances (abnormal voltage, heat, radiation): according to [2], the specifications of devices with security level 3 or higher require that they shut off and/or self-destroy if they are put in exceptional circumstances.

Remark. Note that the attack works if a fault is made in the data operated upon, or in the order or type of instructions that are carried out, or both, as long as the code works again properly for the Chinese remainder operation. The time-window of one of the modular exponentiations is sufficiently large that they may indeed be successfully targeted; it is unclear, however, if the circuitry will again function faultlessly afterwards, and correctly perform the remainder of the computation. If the result of the first exponentiation is stored at some particular location that is not used for any other purposes, then this attack can also be carried out by permanently damaging the wiring of that location so that its value will never be retrieved correctly.

RSA signatures without Chinese remaindering.
Assume that we compute RSA signatures without the use of the Chinese remainder theorem and that a small error is made during the computation of the signature $\mathbf{S(m)}$ of message $\mathbf{m}$. Can the secret exponent $\mathbf{d}$ be recovered? If only one faulty signature of some message is available, there does not seem to be much one can do. Let us assume that we may request faulty signatures $\mathbf{S'(m)}$ of as many different and known messages $\mathbf{m}$ as we would like to have, and that all $\mathbf{S'(m)}$'s have some particular type of fault that is randomly selected from some collection of faults. Can we find $\mathbf{d}$, and if so, how many faulty signatures do we need to find $\mathbf{d}$? Let $\mathbf{e}$ again be the corresponding public exponent.

Let the collection of (faulty signature, message) pairs received be $(\mathbf{S'(m_1)}, \mathbf{m_1})$, $(\mathbf{S'(m_2)},\mathbf{m_2})$, ..., $(\mathbf{S'(m_k)}, \mathbf{m_k})$. If each $\mathbf{S'(m_i)}$ would be the (correct) signature of $\mathbf{m_i+E_i}$, i.e., the original message plus some random and presumably small error pattern $\mathbf{E_i}$, then we would be able to break RSA in general if we could find $\mathbf{d}$ from the collection of corrupted signatures. Since we consider that to be unlikely, we assume that the collection of faulty signatures received looks different, in particular we assume that they are not all $\mathbf{d}$th powers (of a known value).

So, let us assume that the error does not occur right away at the beginning, corrupting only the message (but leaving the computation intact), but that some type of error occurs in the course of the computation. We analyze what the effect of a single error may be, where we restrict ourselves to an error that affects a single register value once; we do not consider the possible effects of corruption of values that may be hardwired or that are stored in non-volatile memory, such as $\mathbf{d}$ and $\mathbf{n}$. We also assume that the proper instructions are carried out in the right order. Thus, we assume that the correct code operates on values $\mathbf{d}$ and $\mathbf{n}$ that remain correct, but that one of the other values during the computation gets corrupted, presumably by a value of small Hamming weight. For an attack based on single bit distortions of $\mathbf{d}$, see [3].

To study this effect, we first have to consider how the exponentiation is carried out. There are essentially two methods to do this, the first using the bits of **d** from right to left (least to most significant), the second one using them from left to right:

1. **s=m; r=1; for i=1 to #d { if (bit i of d is on) r *=s; s *=s;} output r**

and

2. **r=1; for i=#d downto 1 { r *= r; if (bit i of d is on) r *= m;} output r**

In (1) we consider a single corruption of either **r** or **s** in the course of the computation, in (2) we consider a single corruption of either **m** or **r** in the course of the computation. We refer to the four different possibilities as (1r), (1s), (2m), and (2r). We assume that $\mathbf{d=(d_2*2^j)+d_1}$, that the corresponding public exponent is **e**, and that the single corruption takes place during iteration **j**.. We also assume that the computation from that point on is based on the corrupted value, i.e., once **s** gets corrupted (in (1s)), all subsequent powers of **s** are also corrupted, and are powers of the corrupted **s**. (For an attack where only a single **s** gets corrupted by a single bit, and where all subsequent powers are again correct, see [3].) The attack is based on the observation that (again assuming (1s)) if the leading **j** bits of **d** are known and **k** is large enough, then there should be an $\mathbf{m_i}$ for which the correct signature divided by $\mathbf{m_i\text{^}(d_2*2^j)}$ is equal to the corrupted signature divided by its 'corrupted part' $\mathbf{(m_i\text{^}(2^j)+E)\text{^}d_2}$ for some small error **E**. Thus the **e**th powers of these values are also equal, so that the **e**th power of the correct signature can be replaced by $\mathbf{m_i}$. This equality can be used to confirm a guess for the **j**th bit of **d**, once the first **j–1** bits of **d** are known, by trying sufficiently many small error patterns **E**. We now describe this approach in somewhat more detail.

Let **#d** denote the number of bits of **d**. As a result of the corruption and based on our assumption, the $\mathbf{S(m_i)}$'s look as follows, where in each case $\mathbf{E_i}$ is an error pattern, most likely of small Hamming weight (and everything is done modulo **n**):

(1r): $\mathbf{(m_i\text{^}(d_2*2^j))(m_i\text{^}d_1+E_i)}$
(1s): $\mathbf{((m_i\text{^}(2^j)+E_i)\text{^}d_2)(m_i\text{^}d_1)}$
(2m): $\mathbf{(m_i\text{^}(d_2*2^j))(m_i+E_i)\text{^}d_1}$
(2r): $\mathbf{((m_i\text{^}d_2+E_i)\text{^}(2^j))(m_i\text{^}d_1)}$

Given a sufficiently large **k** (with either all $\mathbf{S(m_i)}$ of type (1r) or (1s), or all of type (2m) or (2r)) we may be able to retrieve **d**. Omitting all details, and with a lot of handwaving, this goes as follows (assuming (1r) or (1s); (2m) or (2r) goes similarly). Assume that the first **f** and the last **l** bits of **d** are already known (namely $\mathbf{d_f}$ and $\mathbf{d_l}$). We guess the next bit of $\mathbf{d_f}$ and append it to $\mathbf{d_f}$ resulting in $\mathbf{d_f'}$ of length **f+1**. For all **E**'s of low Hamming weight and all available (and unused) $\mathbf{S'(m_i)}$'s (some of them of type (1s), and some of which might even have the correct **j=#d–f–1**), compute

$$\mathbf{\left(S'(m_i)/((m_i\text{^}(2^{\#d-f-1}))+E)\text{^}d_f'\right)^e}$$

and compare the result to

$$(S(m_i)/(m_i\text{^}(d_f\text{'}*2^{\#d-f-1})))^e = m_i/(m_i\text{^}(e*d_f\text{'}*2^{\#d-f-1})).$$

If a match is found, it is likely that we were using the correct guess $\mathbf{d_f'}$, the correct **E**, and that we were comparing with a type (1s) fault. The 'correct' $\mathbf{S'(m_i)}$ can now be marked as used. If no match is found, guess the other $\mathbf{d_f'}$, and try again. Similarly, we may try to find new bits of $\mathbf{d_l}$, hoping that we have values of type (1r). If none of the $\mathbf{S'(m_i)}$'s had the right **j=#d–f–1** or **j=l+1**, we may simply progress with a guess for either $\mathbf{d_f'}$ or $\mathbf{d_l'}$, until we find a match at **f+u** or **l+u** for a small value of **u** (trying all guesses for all small **u**'s). If no match is found, the '**j**-gap' in the collection of $\mathbf{S'(m_i)}$'s is too large, or the error pattern was too large, and more $\mathbf{S'(m_i)}$'s need to be collected.

How large does **k** need to be? If we do not allow gaps between the **j**'s and insist on a match at each step we need **k** on the order of **(#d)log(#d)**. If we allow rather large gaps (i.e., large **u**'s) and make sure that we try all possible error patterns, then a **k** of the same order of magnitude as **#d** suffices. For instance, for **#d**=1024, the maximal gap for **k**=1024 can be expected to be about 7 (for 512 it is only 4, however).

As mentioned above, many details would have to be filled in in order to obtain a complete description of this 'attack'. Given how 'realistic' our attack scenario is, however, providing these details is probably not worth the trouble. In circumstances where this attack is considered to be realistic, signatures should be checked for correctness before they are transmitted. Note that our attack scenario is about just as realistic as the scenarios from [4] and [5].

Run time. For each fixed **u** the run time of the above approach is polynomial in the number of error patterns allowed, and in all other relevant variables. Note that a similar approach can be used to retrieve **d** if more than a single corruption takes place during the computation; the run time remains polynomial as long as the number of corruptions is bounded by a fixed constant.

References.

1. Dan Boneh[1], personal communication, October 1996.
2. Security Requirements for Cryptographic Modules, FIPS PUB 140-1.
3. Feng Bao, Robert Deng, Yongfei Han, Albert Jeng, Teow Hin Nagir, Desai Narasimhalu, A new attack to RSA on tamperproof devices, manuscript, October 23, 1996. Available from the first author (baofeng@iss.nus.sg).
4. Eli Biham, Adi Shamir, A new cryptanalytic attack on DES, draft, October 18, 1996.
5. Jean-Jacques Quisquater, Short cut for exhaustive search using fault analysis: applications to DES, MAC, keyed hash function, identification protocols, ..., draft, October 23, 1996.

[1] Dan Boneh is one of the authors of the original Bellcore paper *Cryptanalysis in the presence of hardware faults* by Boneh, DeMillo, and Lipton; a draft of this paper exists, but is not for distribution. **Addition**, October 30, 1996: an extended abstract of a Bellcore paper titled *On the importance of checking computations* by the same authors may now be obtained from the first author (dabo@bellcore.com).