PSS Is Secure against Random Fault Attacks

Jean-Sébastien Coron and Avradip Mandal

University of Luxembourg

Abstract. A fault attack consists in inducing hardware malfunctions in order to recover secrets from electronic devices. One of the most famous fault attack is Bellcore's attack against RSA with CRT; it consists in inducing a fault modulo p but not modulo q at signature generation step; then by taking a gcd the attacker can recover the factorization of $N = pq$. The Bellcore attack applies to any encoding function that is deterministic, for example FDH. Recently, the attack was extended to *randomized* encodings based on the ISO/IEC 9796-2 signature standard. Extending the attack to other randomized encodings remains an open problem.

In this paper, we show that the Bellcore attack cannot be applied to the PSS encoding; namely we show that PSS is provably secure against random fault attacks in the random oracle model, assuming that inverting RSA is hard.

Keywords: Probabilistic Signature Scheme, Provable Security, Fault Attacks, Bellcore Attack.

1 Introduction

RSA [14] is still the most widely used signature scheme in practical applications. To sign a message m with RSA, the signer first applies an encoding function μ to m, and then computes the signature $\sigma = \mu(m)^d \bmod N$. The signature is verified by checking that $\sigma^e = \mu(m) \mod N$. For efficiency reasons RSA signatures are often computed using the Chinese Remainder Theorem (CRT); in this case the signature is first computed modulo p and q separately:

$$\sigma_p = m^d \mod p\ , \qquad \sigma_q = m^d \mod q$$

and then σ_p and σ_q are combined by CRT to form the signature σ.

Boneh, DeMillo and Lipton showed that RSA signatures computed with CRT can be vulnerable to fault attacks [3]. If the attacker can induce a fault when σ_q is computed while keeping the computation of σ_p correct, one obtains:

$$\sigma_p = m^d \mod p\ , \qquad \sigma_q \neq m^d \mod q$$

and the resulting faulty signature σ satisfies

$$\sigma^e = m \mod p\ , \qquad \sigma^e \neq m \mod q\ .$$

Therefore, given one faulty signature σ, the attacker can recover the factorization of N by computing $\gcd(\sigma^e - m \bmod N, N) = p$. This attack actually applies to any deterministic RSA encoding, *e.g.* Full Domain Hash (FDH) [2] with $\sigma = H(m)^d \mod N$.

More generally, the attack applies to any probabilistic scheme where the random used to generate the signature is sent along with the signature, *e.g.* as in the Probabilistic Full Domain Hash (PFDH) encoding [6] where the signature is $\sigma \| r$ with $\sigma = H(m \,\|\, r)^d \bmod N$. In that case, given the faulty value of σ and knowing r, the attacker can still factor N by computing $\gcd(\sigma^e - H(m \,\|\, r) \bmod N, N) = p$.

However, if the random r is not given to the attacker along with the signature σ then the Bellcore attack is thwarted. This is the case for signatures of the form $\sigma = \mu(m, r)^d \bmod N$ where the random r is only recovered when verifying the signature, as in PSS [2]. To recover r one needs a *correct* signature; from a faulty signature, the attacker cannot retrieve r nor infer $\mu(m, r)$ in order to compute $\gcd(\sigma^e - \mu(m, r) \bmod N, N) = p$, unless r is short enough to be guessed by exhaustive search. Note that obtaining another correct signature for m would not help the attacker since with high probability a different random r' would be used to generate this signature.

Recently, it was shown how to extend Bellcore's attack to a large class of randomized RSA encoding schemes [7]. The extended attack was illustrated with the ISO/IEC 9796-2 standard [11]. ISO/IEC 9796-2 is originally a deterministic encoding scheme but often used in combination with message randomization, as in the EMV standard [8]. The ISO/IEC 9796-2 encoded message has the form

$$\mu(m) = \mathtt{6A}_{16} \,\|\, m[1] \,\|\, H(m) \,\|\, \mathtt{BC}_{16}$$

where $m = m[1] \,\|\, m[2]$ is split into two parts. The authors of [7] showed that if the randomness introduced into $m[1]$ is not too large (*e.g.* less than 160 bits for a 2048-bit RSA modulus), then a single faulty signature allows to factor N as in the original Bellcore attack. The attack is based on Coppersmith's technique for finding small roots of polynomial equations [5], which is based on the LLL algorithm [12].

However, extending the attack to other randomized RSA signatures remains an open problem. In particular, it is natural to ask whether the Bellcore attack could apply to PSS [2], the most popular RSA-based signature scheme. In this paper, we show that the Bellcore attack cannot be extended to PSS; namely we show that PSS is provably secure against random fault attacks in the random oracle model, assuming that inverting RSA is hard.

More precisely, we consider an extended model of security in which the attacker, in addition to the regular signing oracle, has access to a faulty signature oracle; that is, the attacker can request faulty signatures either modulo p or modulo q. For a faulty signature modulo q, the signer first generates the correct value modulo p:

$$\sigma_p = \mu(m, r)^d \mod p$$

but generates a random σ_q modulo q. With CRT the signer then computes σ' such that $\sigma' = \sigma_p \mod p$ and $\sigma' = \sigma_q \mod q$, and returns the faulty signature

σ' to the adversary. Our result is that PSS is still secure under this extended notion of security, in the random oracle model, assuming that inverting RSA is hard.

2 Security Model

We recall the definition of a signature scheme.

Definition 1 (signature scheme). *A signature scheme* ($\texttt{Gen}, \texttt{Sign}, \texttt{Verify}$) *is defined as follows:*

- The key generation algorithm $\texttt{Gen}$ *is a probabilistic algorithm which given* 1^k*, outputs a pair of matching public and private keys,* (pk, sk)*.*

- The signing algorithm $\texttt{Sign}$ *takes the message* M *to be signed, the public key* pk *and the private key* sk*, and returns a signature* $x = \texttt{Sign}_{sk}(M)$*. The signing algorithm may be probabilistic.*

- The verification algorithm $\texttt{Verify}$ *takes a message* M*, a candidate signature* x' *and* pk*. It returns a bit* $\texttt{Verify}_{pk}(M, x')$*, equal to one if the signature is accepted, and zero otherwise. We require that if* $x \leftarrow \texttt{Sign}_{sk}(M)$*, then* $\texttt{Verify}_{pk}(M, x) = 1$*.*

In the *existential unforgeability under an adaptive chosen message attack* scenario, the forger can dynamically obtain signatures of messages of his choice and attempts to output a valid forgery. A *valid forgery* is a message/signature pair (M, x) such that $\texttt{Verify}_{pk}(M, x) = 1$ whereas the signature of M was never requested by the forger.

In the following, we consider an extended model of security in which the attacker, in addition to the regular signing oracle, has access to a faulty signature oracle; that is, the attacker can request faulty signatures either modulo p or modulo q. For a faulty signature modulo q, the signer first generates the correct value modulo p:

$$\sigma_p = \mu(m, r)^d \mod p$$

and generates a random σ_q modulo q. With CRT the signer then computes σ' such that $\sigma' = \sigma_p \mod p$ and $\sigma' = \sigma_q \mod q$, and returns the faulty signature σ' to the adversary. This is actually equivalent to first computing a correct signature σ:

$$\sigma = \mu(m, r)^d \mod N$$

and then generating a random u modulo q and computing the faulty signature:

$$\sigma' = \sigma + u \cdot p \mod N$$

Formally, we consider the following scenario between a challenger and an attacker. Our scenario applies to any RSA based signature scheme in which a signature σ is computed as $\sigma = \mu(m, r)^d \mod N$ for some (randomized) encoding function $\mu(m, r)$.

Setup: the challenger generates an RSA modulus $N = p \cdot q$, a public exponent e such that $\gcd(e, \phi(N)) = 1$ and a private exponent d such that $e \cdot d = 1 \mod \phi(N)$. The challenger sends (N, e) to the adversary.

Queries: the adversary can make regular signature queries to the challenger. In this case, given a message m, the challenger generates a random r and output the (correct) signature:

$$\sigma = \mu(m,r)^d \mod N$$

Additionally, the attacker can make faulty signature queries. For every such query, the attacker specifies whether the fault should be modulo p or modulo q. For a faulty signature modulo q, the challenger first generates a random r and computes the correct signature:

$$\sigma = \mu(m,r)^d \mod N$$

Then the challenger generates a random u modulo q, and computes:

$$\sigma' = \sigma + u \cdot p \mod N$$

and sends σ' to the attacker. The challenger proceeds similarly if a faulty signature modulo p is requested.

Forgery: eventually the attacker must output a forgery, that is a message signature pair (m, x) such that $\texttt{Verify}_{pk}(m, x) = 1$ whereas the signature of m was never requested by the forger, neither as a regular signature query nor in a faulty signature query.

This completes the description of the attack scenario. As usual, we say that a signature scheme is (t, ε)-secure if no adversary running in time t can output a forgery with probability better than ε.

The PSS scheme was proven secure in the random oracle model [1], and our security proof with faulty signatures is also in the random oracle model. It is well known that a security proof in the random oracle model does not necessarily imply that a scheme is secure in the real world (see [4]). Although it is always better to have a security proof in the standard model, we think that it is still better to have a proof in the random oracle model than no proof at all.

2.1 Why Random Faults?

In our security model we have assumed that when a faulty signature σ' is obtained, it has the uniform distribution modulo p (or modulo q). This could be seen as a very strong assumption; namely in practice the faults might have a completely non-random distribution. Consider for example a fault attack inducing the values of the registers to be set to zero. This gives $\sigma_p = 0$ and recovering p is then straightforward: simply compute $\gcd(\sigma', N) = p$. To prevent from this attack we could assume that when a fault occurs the value σ_p still has enough min-entropy.

In the following we argue that 1) the random fault assumption is almost unavoidable if we want to obtain a security proof and 2) such assumption might actually be reasonable in practice.

Assume that a fault gives a random $\sigma_p \mod p$ but with the k most significant bits set to 0, for some small integer k. That is, the attacker can obtain a list of faulty signatures σ'_i such that the corresponding $\sigma'_{i,p} = \sigma'_i \mod p$ satisfy:

$$0 \leq \sigma'_{i,p} < \frac{p}{2^k} \tag{1}$$

for all $1 \leq i \leq n$, where n is the number of faulty signatures. We show how to recover p, using an attack similar to [13]. With LLL [12], the attacker computes a short vector $(u_1, \ldots, u_n)$ such that:

$$\sum_{i=1}^{n} u_i \cdot \sigma'_i = 0 \mod N$$

This implies:

$$\sum_{i=1}^{n} u_i \cdot \sigma'_{i,p} = 0 \mod p$$

Since from (1) the $\sigma'_{i,p}$ are small modulo p, if the u_i's are small enough, then the equality will hold not only modulo p but also over $\mathbb{Z}$:

$$\sum_{i=1}^{n} u_i \cdot \sigma'_{i,p} = 0$$

This gives a vector $(u_1, \ldots, u_n)$ that is orthogonal in $\mathbb{Z}$ to the unknown vector $(\sigma'_{1,p}, \ldots \sigma'_{n,p})$. It is shown in [13] that by generating sufficiently many such vectors, one can recover the unknown vector $(\sigma'_{1,p}, \ldots \sigma'_{n,p})$ and eventually p.

Note that this attack applies to any RSA-based signature scheme with CRT, not only to PSS. This attack shows it is *not* enough for σ_p to have min-entropy, as only a few bits of entropy loss compared to the uniform distribution enable to recover p. Therefore, if we want to obtain a security proof, it seems necessary to assume that σ_p is uniformly distributed modulo p.

Actually the random fault assumption might be reasonable in practice. Namely to prevent probing attacks, the data being transmitted in the memory bus inside the micro-processor is usually encrypted. Therefore, the content of a register after a fault attack could still be some encrypted value, so it can be reasonable to model this register value as uniformly random.

3 PSS Is Secure against Random Fault Attacks

3.1 The PSS Scheme

We recall the definition of the PSS scheme [2]. The scheme uses three hash functions $h : \{0,1\}^* \to \{0,1\}^{k_1}$, $g_1 : \{0,1\}^{k_1} \to \{0,1\}^{k_0}$ and $g_2 : \{0,1\}^{k_1} \to \{0,1\}^{k-k_0-k_1-1}$, where k, k_0 and k_1 are parameters.

Key Generation: generate a k-bit RSA modulus $N = pq$, and a random exponent $e \in \mathbb{Z}^*_{\phi(N)}$. Generate d such that $e \cdot d = 1 \mod \phi(N)$. The public-key is (N, e); the private key is (N, d).

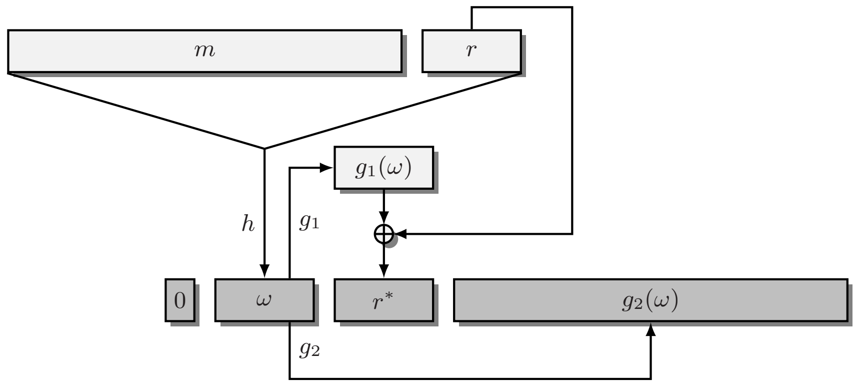

Fig. 1. PSS: the components of the image $y = 0\|\omega\|r^*\|g_2(\omega)$ are darkened. The signature of m is $y^d \mod N$.

Signature generation: given a message m, do the following:

1. $r \leftarrow \{0,1\}^{k_0}$
2. $\omega \leftarrow h(m\|r)$
3. $r^* \leftarrow g_1(\omega) \oplus r$
4. $y \leftarrow 0\|\omega\|r^*\|g_2(\omega)$
5. Return $\sigma = y^d \mod N$

Signature Verification: given a message m and a signature σ, do the following:

1. Let $y = \sigma^e \mod N$
2. Parse y as $0\|\omega\|r^*\|\gamma$. If the parsing fails return 0.
3. $r \leftarrow r^* \oplus g_1(\omega)$
4. If $h(m\|r) = \omega$ and $g_2(\omega) = \gamma$ return 1.
5. else return 0.

3.2 Security Proof

We first give an intuition of the proof. We denote by $\mu(m,r)$ the PSS encoding scheme, that is $\mu(m,r) = 0\|\omega\|r^*\|g_2(\omega)$ where $\omega = h(m\|r)$ and $r^* = g_1(\omega) \oplus r$.

We receive as input a challenge (N, e, η) and we must output $\eta^d \mod N$. In the original PSS security proof [2], when receiving a signature query, the simulator generates a random α modulo N such that $\alpha^e \mod N$ can be written as $0\|\omega\|s\|t$. The simulator generates a random r of k_0 bits. Then it lets $h(m,r) = \omega$, $g_1(\omega) = s \oplus r$ and $g_2(\omega) = t$. Therefore we have that $\mu(m,r) = (\alpha^e \mod N)$. The simulator can then return α as a signature for m. When receiving a hash query for $h(m,r)$, the simulator generates a random α modulo N such that $\eta \cdot \alpha^e$ can be written as $0\|\omega\|s\|t$; it then proceeds as previously. In this case we have $\mu(m,r) = (\eta \cdot \alpha^e \mod N)$. Therefore a forgery for $\mu(m,r)$ enables to compute $\eta^d \mod N$.

One can see that if there is no collision on the randoms r used for signature generation, and no collision on the values ω, then the simulation is perfect. Then given a forgery σ' for some message m', with high probability we have that $\mu(m', r') = (\eta \cdot \alpha^e \mod N)$ for some known α. Therefore from $\sigma' = \mu(m', r')^d \mod N$ one can compute $\eta^d \mod N$ as required and solve the RSA challenge.

In our extended model of security, we must additionally simulate a faulty signature oracle. To do this, one could first generate as previously a random α modulo N such that $\alpha^e \mod N$ can be written as $0\|\omega\|s\|t$. The simulator generates a random r of k_0 bits. Then it lets $h(M, r) = \omega$, $g_1(\omega) = s \oplus r$ and $g_2(\omega) = t$, so that again $\mu(m, r) = (\alpha^e \mod N)$. Then instead of returning the correct signature α, the simulator could generate a random u modulo q, and output the faulty signature:

$$\alpha' = \alpha + u \cdot p \mod N \tag{2}$$

Obviously our simulator cannot do this, because it does not know the prime factors p and q. Instead we show that the distribution of α' is statistically close to uniform in $\mathbb{Z}_N$; therefore, the simulator can simply return a random $\alpha' \in \mathbb{Z}_N$.

Since RSA is a permutation, instead of considering the distribution of α', one can consider the distribution of $y' = \alpha'^e \mod N$. From (2) we have:

$$y' = y + v \cdot p \mod N$$

where v is uniformly distributed modulo q and y is uniformly distributed in $[0, 2^{k-1}[$. The following lemma shows that the distribution of y' is statistically close to uniform in $\mathbb{Z}_N$.

Lemma 1. *Let $N = pq$ be a k-bit modulus where p and q are $k/2$-bit, and let y be a random integer such that $0 \leq y < 2^{k-1}$. Let v be a random integer modulo q. Then the distribution of $y' = y + v \cdot p \mod N$ is ϵ-statistically close to uniform modulo N, with $\epsilon = \frac{4}{2^{k/2}}$*

Proof. We consider a fixed $a \in \mathbb{Z}_N$ and we provide an estimate of $\Pr[y' = a]$. For this we consider the solutions of the equation:

$$a \equiv y + v \cdot p \mod N \tag{3}$$

We have that for every $v \in [0, q)$, there exists a unique $y \in [0, N[$ which satisfies the above relation. However we are only interested in the y's in the range $[0, 2^{k-1}[$. We have that for each $i \in [1, q]$, the pair:

$$(v = q - i,\ y = a + ip \mod N)$$

is a solution of (3) iff

$$a + ip \mod N < 2^{k-1} \tag{4}$$

Depending on the choice of a, there are actually either $\lfloor \frac{2^{k-1}}{p} \rfloor$ or $\lfloor \frac{2^{k-1}}{p} \rfloor + 1$ many i values which satisfy relation (4). Hence there are $\lfloor \frac{2^{k-1}}{p} \rfloor$ or $\lfloor \frac{2^{k-1}}{p} \rfloor + 1$ many solutions to congruence (3) such that $y < 2^{k-1}$. Since y and v are random integers in the range $[0, 2^{k-1})$ and $[0, q)$ respectively, this gives:

$$\left\lfloor \frac{2^{k-1}}{p} \right\rfloor \cdot \frac{1}{2^{k-1}} \cdot \frac{1}{q} \leq \Pr[y' = a] \leq \left(\left\lfloor \frac{2^{k-1}}{p} \right\rfloor + 1 \right) \cdot \frac{1}{2^{k-1}} \cdot \frac{1}{q}$$

We write $\lfloor \frac{2^{k-1}}{p} \rfloor = c$, which gives $p \cdot c < 2^{k-1} < p \cdot c + p$. We obtain:

$$\begin{aligned}
\Pr[y' = a] &\geq \frac{c}{2^{k-1}q} = \frac{1}{N} \cdot \frac{pc}{2^{k-1}} = \frac{1}{N} \cdot \left(1 - \frac{2^{k-1} - pc}{2^{k-1}}\right) \\
&> \frac{1}{N} \cdot \left(1 - \frac{p}{2^{k-1}}\right) \qquad (\text{as } 2^{k-1} < pc + p) \\
&> \frac{1}{N} \cdot \left(1 - \frac{2p}{N}\right) \qquad (\text{as } 2^{k-1} > \frac{N}{2}) \\
&= \left(1 - \frac{2}{q}\right) \cdot \frac{1}{N}
\end{aligned}$$

Similarly, we have:

$$\Pr[y' = a] \leq \left(1 + \frac{2}{q}\right) \cdot \frac{1}{N}$$

This gives:

$$\left(1 - \frac{2}{q}\right) \cdot \frac{1}{N} \leq \Pr[y' = a] \leq \left(1 + \frac{2}{q}\right) \cdot \frac{1}{N}$$

for all $a \in [0, N)$. This implies that the distribution of y' is $\frac{4}{2^{k/2}}$-statistically close to uniform modulo N as $q > 2^{k/2-1}$. □

Lemma 1 shows that it is sufficient for our simulator to return a random α' modulo N as the faulty signature. In other words, instead of first generating a random $y \in [0, 2^{k-1})$, then a random v modulo q, then $y' = y + v \cdot p$ and finally $\alpha' = y'^d \mod N$, the simulator can simply output a random α' modulo N, and such output will be statistically indistinguishable from a faulty signature.

However to this faulty signature α' corresponds a correct signature α such that:

$$\alpha = \alpha' - u \cdot p \mod N$$

where u is randomly distributed modulo q. Equivalently letting $y' = \alpha'^e \mod N$ there exists a corresponding value y with:

$$y = y' - v \cdot p \mod N \tag{5}$$

where v is randomly distributed modulo q such that y can be written as:

$$y = 0\|\omega\|s\|t = \mu(m, r)$$

This implicitly defines $h(m, r) = \omega$, $g_1(\omega) = s \oplus r$ and $g_2(\omega) = t$ for the simulation of random oracles h, g_1 and g_2.

Since our simulator does not know p, it cannot compute y in equation (5) and therefore our simulator does not known the corresponding values of ω, s and t; therefore our simulator cannot answer the corresponding h queries, g_1 queries and g_2 queries if such queries are made by the attacker. Intuitively for h-queries it is sufficient that the set of r values is exponentially large; for this the parameter k_0 must be large enough. For g_1 and g_2 queries we must show that the adversary has a negligible probability of querying ω. This is shown in the following lemma: we show that given a faulty signature α' (or equivalently $y' = \alpha'^e \mod N$) the distribution of ω has enough variability, if the parameter k_1 is sufficiently large. This implies that ω does not need to be computed, and therefore the factorization of N is not needed for our simulation.

Lemma 2. *Let $N = pq$ be a k-bit modulus where p and q are $k/2$-bit, and let y be a random integer such that $0 \leq y < 2^{k-1}$. Let v be a random integer modulo q, and let $y' = y + v \cdot p \mod N$. Write $y = 0\|\omega\|x$ where ω is k_1-bit and x is $k - k_1 - 1$ bits. Given y', for any ω' of k_1-bit we have:*

$$\Pr[\omega = \omega' | y'] \leq \frac{8}{2^{\min(k_1, k/2)}}$$

Proof. We have that:

$$\Pr[\omega = \omega' | y'] = \frac{\#(y,v) \textit{ pairs, s.t. } y' = y + v \cdot p \mod N \textit{ and } y = 0\|\omega'\|x}{\#(y,v) \textit{ pairs, s.t. } y' = y + v \cdot p \mod N \textit{ and } 0 \leq y < 2^{k-1}}$$

For a fixed v, the value $y \mod N$ gets fixed by the relation $y' = y + v \cdot p \mod N$. Moreover at least $\lfloor \frac{q}{2} \rfloor$ of the possible v values give $y \mod N$ in the desired range between 0 and 2^{k-1}. Hence the denominator of the above fraction can be lower bounded by $\lfloor \frac{q}{2} \rfloor$.

We have that for a fixed y', the value of y is fixed modulo p; hence for a fixed ω' with $y = 0\|\omega'\|x$, the value of x is also fixed modulo p. As x is $k - k_1 - 1$-bit, over $\mathbb{Z}$ there can be at most $\lceil \frac{2^{k-k_1-1}}{p} \rceil$ many possible x values. Hence the numerator of the above fraction can be upper bounded by $\lceil \frac{2^{k-k_1-1}}{p} \rceil$.

Hence we have,

$$\Pr[\omega = \omega' | y'] \leq \frac{\lceil \frac{2^{k-k_1-1}}{p} \rceil}{\lfloor \frac{q}{2} \rfloor} < \frac{\frac{2^{k-k_1-1}}{2^{k/2-1}} + 1}{2^{k/2-2}} = \frac{2^{k-k_1-1} + 2^{k/2-1}}{2^{k-3}} < \frac{8}{2^{\min(k_1, k/2)}}$$

□

Formally, we obtain the following theorem:

Theorem 1. *Assume that no algorithm can invert RSA in time t' with probability better than ε'. Then the signature scheme* $\text{PSS}[k_0, k_1]$ *is $(t, q_h, q_g, q_s, q_{fs}, \varepsilon)$ secure, where*

$$\begin{aligned} t(k) &= t'(k) - [q_s(k) + q_g(k) + q_h(k) + 1] \cdot k_0 \cdot \Theta(k^3) \\ \varepsilon(k) &= \varepsilon'(k) + (q_s + q_{fs} + 1) \cdot (q_s + q_{fs} + q_h) \cdot 2^{-k_0} + 8 \cdot q_g \cdot q_{fs} \cdot 2^{-\min(k_1, k/2)} \\ &\quad + (q_h + q_s + q_{fs}) \cdot (q_h + q_g + q_s + q_{fs} + 1) \cdot 2^{-k_1} \\ &\quad + q_h \cdot q_{fs} \cdot 2^{-k_0} + 4 \cdot q_{fs} \cdot 2^{-k/2} \end{aligned}$$

Here the attacker can make at most q_h, q_g, q_s, q_{fs} number of h queries, g queries, signature queries and fault signature queries respectively.

Proof. We use a simulator which behaves in exactly same way as in original PSS security proof [2], in addition it answers fault queries with a uniformly random integer modulo N. Now if the attacker is successful against our simulator then we break the RSA challenge (N, e, η) as in the original paper.

We must show that any attacker which is successful against the original attack scenario will be successful against our simulator. For that, we use a sequence of games. We start with $\texttt{Game}_0$, which is exactly the attack scenario, which requires to know the factorization of N. Then we progressively modify the game, so that eventually knowledge of the factorization of N is not needed anymore. We denote by S_i the event that the attacker succeeds in $\texttt{Game}_i$.

$\texttt{Game}_0$: this is the attack scenario. We answer signature queries as specified in the signature generation algorithm, using the private exponent d. We simulate the faulty signature queries by first generating a correct signature σ and then computing $\sigma' = \sigma + u \cdot p \mod N$ for a random u modulo q. In the following for simplicity we only consider faulty signatures modulo q; faulty signatures modulo p are simulated in exactly the same way.

$\texttt{Game}_1$: we abort if there is a collision for ω at Step 2 of the signature generation algorithm, or if the random r used during signature generation has already appeared before. We call this event A_1. More precisely event A_1 happens if one of the following is true:

- The random r used in a signature oracle or faulty signature oracle query collides with either 1) the r used in a previous signature oracle or faulty signature oracle query or 2) the r used in a previous h oracle query.
- The h function output in a signature oracle or faulty signature oracle query collides with either 1) the h function outputs in previous signature oracle or faulty signature oracle queries or 2) with a previous h oracle query output or 3) a previous g oracle query input.
- The h oracle query output collides with either 1) a h function output in previous signature oracle or faulty signature oracle query or 2) a previous h oracle query output or 3) a previous g oracle query input.

We obtain:

$$\Pr[A_1] \leq (q_s + q_{fs}) \cdot (q_s + q_{fs} + q_h) \cdot 2^{-k_0} + (q_h + q_s + q_{fs}) \cdot (q_h + q_g + q_s + q_{fs}) \cdot 2^{-k_1}$$

and:

$$|\Pr[S_1] - \Pr[S_0]| \leq \Pr[A_1]$$

$\texttt{Game}_2$: we construct a similar simulator as in the original PSS security proof [2]; however to deal with faulty signature queries we continue to use the factorization of N.

The simulator receives as input a challenge η and must output $\eta^d \mod N$. When receiving a signature query, the simulator generates a random α modulo N such that $\alpha^e \mod N$ can be written as $0\|\omega\|s\|t$. The simulator generates a random r of k_0 bits. Then it lets $h(m, r) = \omega$, $g_1(\omega) = s \oplus r$ and $g_2(\omega) = t$.

When receiving a hash query for $h(m, r)$, the challenger generates a random α modulo N such that $\eta \cdot \alpha^e \mod N$ can be written as $0\|\omega\|s\|t$; it then defines $h(m, r) = \omega$, $g_1(\omega) = s \oplus r$ and $g_2(\omega) = t$ as previously. The queries to g_1 and g_2 are simulated by returning a random value for every new input.

To simulate the faulty signature oracle, one first generates as above a random α modulo N such that $\alpha^e \mod N$ can be written as $0\|\omega\|s\|t$. The simulator generates a random r of k_0 bits. Then it lets $h(m, r) = \omega$, $g_1(\omega) = s \oplus r$ and $g_2(\omega) = t$. Then instead of returning α, the simulator generates a random u modulo q, and outputs:

$$\alpha' = \alpha + u \cdot p \mod N \tag{6}$$

In $\texttt{Game}_2$ we abort as in $\texttt{Game}_1$, and additionally in the following case: while generating a random α modulo N such that $\alpha^e \mod N$ can be written as $0\|\omega\|s\|t$ during signature or faulty signature queries (and similarly for $h(m, r)$ queries), we stop after trying $k_0 + 1$ times. This adds $(q_h + q_s + q_{fs}) \cdot 2^{-k_0}$ in the error term:

$$|\Pr[S_2] - \Pr[S_1]| \le (q_h + q_s + q_{fs}) \cdot 2^{-k_0}$$

$\texttt{Game}_3$: we abort if the attacker makes a query for $g(\omega)$ where ω was used in a faulty signature for message m and random r, while the attacker has not made a query to $h(m, r)$ before. We define this event as A_3. As all the query answers are simulated independently, from Lemma 2 this gives:

$$|\Pr[S_3] - \Pr[S_2]| \le \Pr[A_3] \le q_g \cdot q_{fs} \cdot \frac{8}{2^{\min(k_1, k/2)}}$$

$\texttt{Game}_4$: we abort if the attacker makes a query for $h(m, r)$ where r was used to generate a faulty signature with ω, while the attacker has not made a query before to $g(\omega)$. In this case the attacker's view is independent from r, which gives:

$$|\Pr[S_4] - \Pr[S_3]| \le q_h \cdot q_{fs} \cdot 2^{-k_0}$$

$\texttt{Game}_5$: we abort if the attacker makes a query for $h(m, r)$ where r was used to generate a faulty signature, or if the attacker makes a query for $g(\omega)$ where ω was used in a faulty signature. $\texttt{Game}_5$ is the same as $\texttt{Game}_4$ since for a faulty signature m with random r and ω, either the attacker starts with a $h(m, r)$ query or it starts with a $g(\omega)$ query.

$$\Pr[S_5] = \Pr[S_4]$$

$\texttt{Game}_6$: we change the way the faulty signature oracle is simulated. Instead of first generating α and then α' as in equation (6), we first generate a uniformly random α' and then a random u modulo q such that $\alpha^e \mod N$ can be written as $0\|\omega\|s\|t$. From Lemma 1 we have:

$$|\Pr[S_6] - \Pr[S_5]| \le q_{fs} \cdot \frac{4}{2^{k/2}}$$

$\texttt{Game}_7$: since we do not answer the queries for $h(m, r)$ where r was used to generate a faulty signature, and the queries for $g(\omega)$ where ω was used in a faulty signature, we do not need to compute ω. Therefore, we do not need to compute a random u modulo q such that $\alpha^e \mod N$ can be written as $0\|\omega\|s\|t$. Therefore we do not need to know the factorization of N anymore, and we have:

$$\Pr[S_7] = \Pr[S_6]$$

Finally, if the adversary outputs a forgery with probability at least ε in $\texttt{Game}_0$, then the adversary must output a forgery with probability at least $\varepsilon - |\Pr[S_7] - \Pr[S_0]|$ in $\texttt{Game}_7$. As in the original PSS security proof, from this forgery we can solve the RSA challenge with probability at least:

$$\varepsilon' = \varepsilon - |\Pr[S_7] - \Pr[S_0]| - 2^{-k_1}$$

Combining the previous inequalities, we get (6). □

4 PSS-R Is Secure against Fault Attacks

In PSS-R or PSS with message recovery the goal is to save bandwidth such that the message is recoverable from the signature; hence it is not necessary to send the message separately.

4.1 The PSS-R Scheme

We recall the definition of the PSS-R scheme [2]. The scheme uses three hash functions $h : \{0,1\}^* \to \{0,1\}^{k_1}$, $g_1 : \{0,1\}^{k_1} \to \{0,1\}^{k_0}$ and $g_2 : \{0,1\}^{k_1} \to \{0,1\}^{k-k_0-k_1-1}$, where k, k_0 and k_1 are the parameters.

Key Generation: generate a k-bit RSA modulus $N = pq$, and a random exponent $e \in \mathbb{Z}^*_{\phi(N)}$. Generate d such that $e \cdot d = 1 \mod \phi(N)$. The public-key is (N, e); the private key is (N, d).

Signature generation: given a message m, do the following:

1. $r \leftarrow \{0,1\}^{k_0}$
2. $\omega \leftarrow h(M\|r)$
3. $r^* \leftarrow g_1(\omega) \oplus r$
4. $m^* \leftarrow g_2(\omega) \oplus m$
5. $y \leftarrow 0\|\omega\|r^*\|m^*$
6. Return $\sigma = y^d \mod N$

Message Recovery: given a signature σ, do the following:

1. Let $y = \sigma^e \mod N$
2. Parse y as $0\|\omega\|r^*\|m^*$. If the parsing fails return REJECT.
3. $r \leftarrow r^* \oplus g_1(\omega)$
4. $m \leftarrow m^* \oplus g_2(\omega)$
5. If $h(m\|r) = \omega$ return m.
6. else return REJECT.

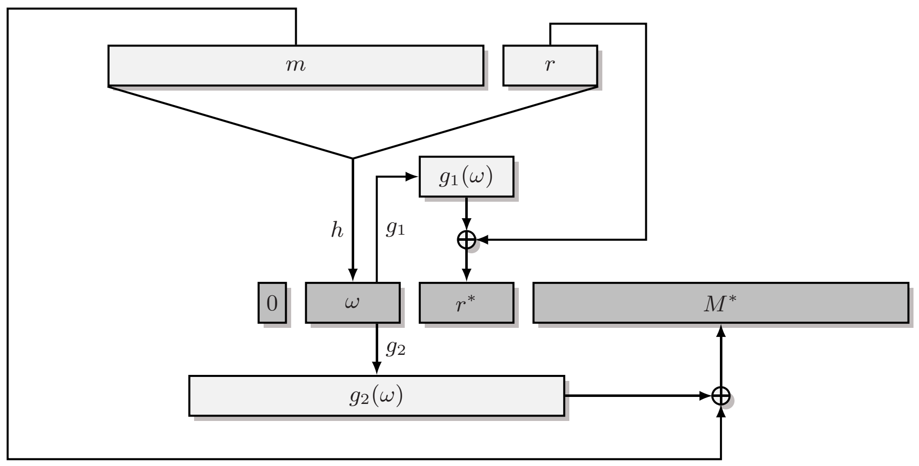

Fig. 2. PSS-R: Components of image $y = 0\|\omega\|r^*\|M^*$ are darkened. The signature of M is $y^d \mod N$.

4.2 Security Proof

Theorem 2. *Assume that no algorithm can invert RSA in time t' with probability better than ε'. Then the signature scheme* PSS-R$[k_0, k_1]$ *is* $(t, q_h, q_g, q_s, q_{fs}, \varepsilon)$ *secure, where:*

$$\begin{aligned}
t(k) &= t'(k) - [q_s(k) + q_g(k) + q_h(k) + 1] \cdot k_0 \cdot \Theta(k^3) \\
\varepsilon(k) &= \varepsilon'(k) + (q_s + q_{fs} + 1) \cdot (q_s + q_{fs} + q_h) \cdot 2^{-k_0} + 8 \cdot q_g \cdot q_{fs} \cdot 2^{-\min(k_1, k/2)} \\
&\quad + (q_h + q_s + q_{fs}) \cdot (q_h + q_g + q_s + q_{fs} + 1) \cdot 2^{-k_1} \\
&\quad + q_h \cdot q_{fs} \cdot 2^{-k_0} + 4 \cdot q_{fs} \cdot 2^{-k/2}
\end{aligned}$$

Here the attacker can make at most q_h, q_g, q_s, q_{fs} number of h queries, g queries, signature queries and fault signature queries respectively.

Proof. The proof of this theorem is very similar to that of Theorem 1 and hence is omitted.

5 Conclusion

We obtain from the previous theorems that unless the attacker is making more fault oracle queries than hash oracle queries, one gets the same security bound as in the original PSS proof without fault oracle. We note that in practice fault queries are usually more expensive than hash queries, since those hash queries can be made offline when a concrete hash function is used.

In [6] a better security bound was given for PSS (without fault oracle). It was shown that the random size k_0 could be taken as small as $\log_2 q_s$, where q_s is the maximum number of signature queries; with $q_s = 2^{30}$ this gives $k_0 = 30$ bits. However with a fault oracle one cannot take such a small k_0, since in this case the random r could be recovered by exhaustive search and the Bellcore attack would still apply.

In summary. any parameters chosen according to the bounds in the original PSS paper [2] give the same level of security against fault attacks. One can take $k = 1024$, $k_0 = k_1 = 128$ as in [2].

References

1. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Proceedings of the First Annual Conference on Computer and Communications Security. ACM, New York (1993)
2. Bellare, M., Rogaway, P.: *The Exact security of digital signatures: How to sign with* RSA *and Rabin*. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (1996)
3. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults. Journal of Cryptology, Springer-Verlag 14(2), 101–119 (2001)
4. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. In: STOC 1998. ACM, New York (1998)
5. Coppersmith, D.: Small solutions to polynomial equations, and low exponent vulnerabilities. Journal of Cryptology 10(4), 233–260 (1997)
6. Coron, J.S.: Optimal security proofs for PSS and other signature schemes. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 272–287. Springer, Heidelberg (2002)
7. Coron, J.-S., Joux, A., Kizhvatov, I., Naccache, D., Paillier, P.: Fault Attacks on Randomized RSA Signatures with Partially Unknown Messages. In: CHES 2009, pp. 444–456 (2009), `http://www.jscoron.fr/publications.html`
8. EMVIntegrated circuit card specifications for payment systems, Book 2. Security and Key Management. Version 4.2 (June 2008), `http://www.emvco.com`
9. Goldwasser, S., Micali, S., Rivest, R.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal of computing 17(2), 281–308 (1988)
10. IEEE P1363a, Standard Specifications For Public Key Cryptography: Additional Techniques, `http://www.manta.ieee.org/groups/1363`
11. ISO/IEC 9796-2:2002 Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: Integer factorization based mechanisms (2002)
12. Lenstra, A., Lenstra Jr., H., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen 261, 513–534 (1982)
13. Nguyên, P.Q., Stern, J.: Cryptanalysis of a fast public key cryptosystem presented at SAC '97. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, p. 213. Springer, Heidelberg (1999)
14. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public key cryptosystems. CACM 21 (1978)