Fault Attacks on Randomized RSA Signatures

Jean-Sébastien Coron	extsuperscript{1}, Antoine Joux	extsuperscript{2}, David Naccache	extsuperscript{3}, and Pascal Paillier	extsuperscript{4}

	extsuperscript{1} Université du Luxembourg
6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg, Luxembourg
jean-sebastien.coron@uni.lu

	extsuperscript{2} DGA and Université de Versailles
UVSQ PRISM 45 avenue des États-Unis, F-78035, Versailles CEDEX, France
antoine.joux@m4x.org

	extsuperscript{3} École normale supérieure
Département d’informatique, Groupe de Cryptographie
45, rue d’Ulm, F-75230 Paris CEDEX 05, France
david.naccache@ens.fr

	extsuperscript{4} Gemalto, Cryptography & Innovation
6, rue de la Verrerie, F-92447 Meudon sur Seine, France
pascal.paillier@gemalto.com

Abstract. Fault attacks exploit hardware malfunctions or induce them to recover secret keys embedded in a secure device such as a smart card. In the late 90’s, Boneh, DeMillo and Lipton [6] and other authors introduced fault-based attacks on CRT-RSA which allow the attacker to factor the signer’s modulus when the message padding function is deterministic. Since then, extending fault attacks to randomized signature schemes (e.g. PSS [4]) has remained an open problem given the impracticality of recovering the randomness used to generate the signature (the randomness is only recovered when verifying a correct signature). This paper successfully extends fault attacks to a large class of randomized RSA signatures. Our attack techniques rely on Coppersmith’s algorithm for finding small roots of multivariate polynomial equations. We illustrate the practicality of our approach by attacking several randomized versions of the ISO/IEC 9796-2 encoding standard. Our practical experiments show that a 2048-bit modulus can be factored in less than a minute given one faulty signature containing about 160 random bits and an unknown 160-bit hash value.

Keywords: Fault attacks, digital signatures, RSA, Coppersmith’s theorem, ISO/IEC 9796-2.

1 Introduction

1.1 Background

RSA [23] is undoubtedly the most popular digital signature scheme and is implemented in the vast majority of secure embedded devices. To sign a message \( m \) with RSA, the signing procedure applies an encoding (a.k.a. padding) function \( \mu \) to \( m \), and then computes the signature \( \sigma = \mu(m)^d \mod N \). To verify the signature, the receiver checks that

\[
\sigma^e = \mu(m) \mod N .
\]

(1)

As shown by Boneh, DeMillo and Lipton [6], RSA implementations are vulnerable to fault attacks, especially when the Chinese Remainder Theorem (CRT) is implemented; in this case the device computes

\[
\sigma_p = m^d \mod p , \quad \sigma_q = m^d \mod q
\]

and the signature \( \sigma \) is computed from \( \sigma_p \) and \( \sigma_q \) via Chinese Remaindering. Assuming the attacker is able to induce a fault when \( \sigma_q \) is computed while keeping the computation of \( \sigma_p \) correct, one gets

\[
\sigma_p = m^d \mod p , \quad \sigma_q \neq m^d \mod q
\]
and the resulting (faulty) signature $\sigma$ satisfies

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

Therefore given one such faulty signature $\sigma$ the attacker can factor $N$ by computing

$$\gcd(\sigma^e - m \mod N, N) = p.$$  \hfill (2)

Boneh, DeMillo and Lipton’s fault attack is easily extended to any signature scheme which padding is deterministic, such as the Full Domain Hash (FDH) [4] encoding where

$$\sigma = H(m)^d \mod N$$

and $H$ is some hash function. The attack can also be extended to probabilistic signature schemes where the randomness used to generate the signature is made part of the signature, as with the Probabilistic Full Domain Hash (PFDH) [12] encoding where the signature is $\sigma \parallel r$ with $\sigma = H(m \parallel r)^d \mod N$. In that case, given the faulty value of $\sigma$ and knowing $r$, the attacker can still retrieve a factor of $N$ as

$$\gcd(\sigma^e - H(m \parallel r) \mod N, N) = p.$$  \hfill (3)

1.2 Randomized encodings: the fault-attacker’s dilemma

However, if the randomness is not an explicit part of the signature, the attack does not work anymore. This occurs when the signature has the form $\sigma = \mu(m, r)^d \mod N$ where $r$ is the random nonce, as is the case in the PSS [4] encoding. Here the value of $r$ can be recovered by the verifier only after completing the verification process; however the randomness $r$ can only be recovered when verifying a correct signature. Given only a faulty signature, the attacker therefore cannot retrieve $r$ nor deduce $\mu(m, r)$ to reveal

$$\gcd(\sigma^e - \mu(m, r) \mod N, N) = p$$  \hfill (3)

and the Boneh-DeMillo-Lipton attack does not apply. So the attacker faces this apparent paradox: recovering the random bits $r$ involved in the faulty signature $\sigma$ seem to require that $\sigma$ be a correct and verifiable signature. Yet, obviously, a correct signature does not open the way to factoring $N$. This contradiction cannot be solved unless $r$ is short enough to be exhaustively searched, which happens only marginally in practical applications. Inducing faults on many signatures will not help in any way since a different random value is used for each signature. As a result, randomized RSA paddings $\mu(m, r)$ are usually considered inherently immune against fault-based attacks.

1.3 Our achievements

We overcome the above dilemma by showing how to extract the randomness involved in the generation of faulty RSA signatures. We put forward several attack techniques which extend the Boneh-DeMillo-Lipton attack to a large class of randomized RSA paddings. Our results assume that certain conditions on the unknown parts of the encoded message be fulfilled; these conditions may depend on the encoding function itself and the way inner hash functions are cascaded. To elaborate on this and illustrate our attacks, we have chosen to consider the ISO/IEC 9796-2 encoding scheme [17] which currently counts among the most widely spread RSA padding in secure embedded platforms. ISO/IEC 9796-2 is originally
a deterministic encoding scheme but is often used in combination with a randomization of the message. The encoded message has the form

\[ \mu(m) = 6A_{16} \parallel m[1] \parallel H(m) \parallel BC_{16} \]

where \( m = m[1] \parallel m[2] \) is split in two parts. We show that if the randomness introduced in \( m[1] \) is not too large (less than 160 bits for a 2048-bit RSA modulus), then a single faulty signature enables to factor \( N \) as in the original Boneh-DeMillo-Lipton attack\(^1\). Our work is based on a result by Herrmann and May [13] for finding small roots of linear equations modulo an unknown factor \( p \) of \( N \); this result is itself based on Coppersmith's technique for finding small roots of polynomial equations [7] using the LLL algorithm [21]. We also show how to extend our attack to multiple blocks of randomness and to the case when two or more faulty signatures can be obtained from the device.

We comment that our techniques readily apply to the case where parts of the encoded message are unknown instead of being random. The conditions for the attack to take place (i.e. the bounds we provide on the total size of the unknown parts) are then exactly the same as in the random case. Also, it is easily seen that other encoding functions such as PKCS#1 v1.5 are also broken by our attack even when parts of the signed message are random or unknown. We elaborate on this in more detail at the end of the paper.

1.4 The ISO/IEC 9796-2 standard

ISO/IEC 9796-2 is an encoding standard allowing partial or total message recovery [17, 18]. The encoding can be used with hash functions \( H(m) \) of diverse digest sizes \( k_h \). For the sake of simplicity we assume that \( k_h \), the size of \( m \) and the size of \( N \) (denoted \( k \)) are all multiples of 8. The ISO/IEC 9796-2 encoding of a message \( m = m[1] \parallel m[2] \) is

\[ \mu(m) = 6A_{16} \parallel m[1] \parallel H(m) \parallel BC_{16} \]

where \( m[1] \) consists of the \( k - k_h - 16 \) leftmost bits of \( m \) and \( m[2] \) represents the remaining bits of \( m \). Therefore the size of \( \mu(m) \) is always \( k - 1 \) bits. Note that the original version of the standard recommended \( 128 \leq k_h \leq 160 \) for partial message recovery (see [17], §5, note 4). In [11], Coron, Naccache and Stern introduced an attack against ISO/IEC 9796-2; the authors estimated that attacking \( k_h = 128 \) and \( k_h = 160 \) would require respectively \( 2^{54} \) and \( 2^{61} \) operations. After Coron et al.'s publication, the ISO/IEC 9796-2 standard was amended and the current official requirement (see [18]) is now \( k_h \geq 160 \).

For the sake of this work, we consider a message \( m = m[1] \parallel m[2] \) of the form

\[ m[1] = \alpha \parallel r \parallel \alpha' \], \quad m[2] = \text{data} \]

where \( r \) is a random unknown to the adversary, \( \alpha \) and \( \alpha' \) are strings known to the adversary and \( \text{data} \) is some known or unknown string\(^2\). The size of \( r \) is denoted \( k_r \) and the size of \( m[1] \) is \( k - k_h - 16 \) as required in ISO/IEC 9796-2. The encoded message is then

\[ \mu(m) = 6A_{16} \parallel \alpha \parallel r \parallel \alpha' \parallel H(\alpha \parallel r \parallel \alpha' \parallel \text{data}) \parallel BC_{16} \tag{4} \]

where the total number of unknown bits is \( k_r + k_h \).

\(^1\) With our techniques, it does not matter how large is the random part in \( m[2] \).

\(^2\) Our attack will work in both cases alike.
2 Our Attack on Randomized ISO/IEC 9796-2

In this section, we show how to extend the Boneh-DeMillo-Lipton fault attack to randomized signatures on messages encoded as previously discussed. We assume that after injecting a fault while the device is generating the signature, one has obtained a faulty signature $\sigma$ such that

$$\sigma^e = \mu(m) \mod p, \quad \sigma^e \neq \mu(m) \mod q.$$  

From (4) we can write

$$\mu(m) = C + r \cdot 2^k + H(m) \cdot 2^8$$

where $C$ is a known value. Note that both $r$ and $H(m)$ are unknown to the adversary. From (5) we obtain:

$$\sigma^e = C + r \cdot 2^k + H(m) \cdot 2^8 \mod p.$$ 

This shows that $(r, H(m))$ must be a solution of the equation

$$a + b \cdot x + c \cdot y = 0 \mod p$$

where $a := C - \sigma^e \mod N$, $b := 2^k r$ and $c := 2^8$ are known. Therefore we are left with solving Equation (6) which is linear in the two variables $x, y$ and admits a small root $(x_0, y_0) = (r, H(m))$. However the equation holds modulo an unknown divisor $p$ of $N$, not modulo $N$. Such equations were already considered by Herrmann and May [13] to factor an RSA modulus $N = pq$ when some bits of $p$ are revealed. Their method is based on Coppersmith’s technique for finding small roots of polynomial equations [7]. The idea consists in using LLL to obtain two polynomials $h_1(x, y)$ and $h_2(x, y)$ such that

$$h_1(x_0, y_0) = h_2(x_0, y_0) = 0$$

holds over the integers. Then one takes the resultant between $h_1$ and $h_2$ to recover the common root $(x_0, y_0)$. To do this, one must assume that $h_1$ and $h_2$ are algebraically independent. This ad hoc assumption is what makes the method heuristic; nonetheless it turns out to work quite well in practice. Then, given the root $(x_0, y_0)$ one finds the randomized encoded message $\mu(m)$ and factors $N$ by GCD.

**Theorem 1 (Herrmann-May [13]).** Let $N$ be a sufficiently large composite integer with a divisor $p \geq N^\beta$. Let $f(x, y) = a + b \cdot x + c \cdot y \in \mathbb{Z}[x, y]$ be a linear polynomial in two variables. Assume $f(x_0, y_0) = 0 \mod p$ for some $(x_0, y_0)$ such that $|x_0| \leq N^\gamma$ and $|y_0| \leq N^\delta$. Then for any $\varepsilon > 0$, under the condition

$$\gamma + \delta \leq 3\beta - 2 + 2(1 - \beta)^{3/2} - \varepsilon$$

one can find two polynomials $h_1(x, y), h_2(x, y) \in \mathbb{Z}[x, y]$ such that $h_1(x_0, y_0) = h_2(x_0, y_0) = 0$ over $\mathbb{Z}$, in time polynomial in $\log N$ and $\varepsilon^{-1}$.

In what follows, only a sketch of proof is given; we refer to [13] for full details. We can assume that $b = 1$ in the polynomial $f$, since otherwise we can multiply $f$ by $b^{-1} \mod N$. Therefore we consider the polynomial

$$f(x, y) = a + x + c \cdot y$$

and we must find $(x_0, y_0)$ such that $f(x_0, y_0) = 0 \mod p$. The basic idea consists in generating a family $\mathcal{G}$ of polynomials admitting $(x_0, y_0)$ has a root modulo some large enough bound $B$. Any linear combination of these polynomials will also be a polynomial which admits $(x_0, y_0)$ as a root modulo $B$.
and we will make use of LLL to find such polynomials with small coefficients. To do so, we view any polynomial \( h(x, y) = \sum h_{i,j}x^iy^j \) as the vector of coefficients \((h_{i,j}X^iY^j)_{i,j}\) and denote by \( \|h(X,Y)\| \) the euclidian norm of this vector. It is easily seen that performing linear combinations on polynomials is equivalent to performing linear operations on their vector representation, so that applying LLL to the lattice spanned by the vectors in \( G \) will provide short vectors representing polynomials with root \((x_0, y_0) \mod B\). We now define the family \( G \) of polynomials as

\[
g_{k,i}(x, y) := y^i \cdot f^k(x, y) \cdot N^{\max(t-k,0)}
\]

for \( 0 \leq k \leq m, \ 0 \leq i \leq m - k \) and integer parameters \( t \) and \( m \). For all values of indices \( k, i \), it holds that

\[
g_{k,i}(x_0, y_0) = 0 \mod p^t.
\]

We sort the polynomials \( g_{k,i} \) by increasing values of \( k \) first, and then by increasing values of \( i \). Noting \( X = N^\gamma \) and \( Y = N^\delta \), we write the coefficients of the polynomial \( g_{k,i}(x_0, y_0) \) in the basis \( x^{k'} \cdot y^{i'} \) for \( 0 \leq k' \leq m \) and \( 0 \leq i' \leq m - k' \). One then obtains a matrix of row vectors as illustrated on Figure 1.

| \( \begin{array}{cccccccc} 1 & y^m & x & xy^{m-1} & \ldots & x^iy^{m-1} & \ldots & x^{m-1}y & x^m \\
g_{0,0}(xX, yY) & N^t & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
g_{0,m}(xX, yY) & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
g_{1,0}(xX, yY) & XN^t & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
g_{1,m-1}(xX, yY) & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
g_{t,0}(xX, yY) & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
g_{t,m-1}(xX, yY) & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
g_{m-1,0}(xX, yY) & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
g_{m-1,1}(xX, yY) & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
g_{m,0}(xX, yY) & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\end{array} \) |

**Fig. 1.** Lattice of row vectors from the coefficients of the polynomials \( g_{k,i}(xX, yY) \). The matrix is lower triangular; we only represent the elements on the diagonal.

Let \( L \) be the corresponding lattice; its dimension is

\[
\omega = \dim(L) = \frac{m^2 + 3m + 2}{2} = \frac{(m + 1)(m + 2)}{2}
\]

and we have

\[
\det L = X^{s_x}Y^{s_y}N^{s_N}
\]

where

\[
s_x = s_y = \sum_{k=0}^{m} \sum_{i=0}^{m-k} i = \frac{m(m + 1)(m + 2)}{6}
\]
and

\[ s_N = \sum_{i=0}^{t} (m + 1 - i) \cdot (t - i). \]

We now apply LLL to the lattice \( L \) in order to find two polynomials \( h_1(x, y) \) and \( h_2(x, y) \) with short coefficients.

**Theorem 2 (LLL [21]).** Let \( L \) be a lattice spanned by \((u_1, \ldots, u_\omega)\). Given the vectors \((u_1, \ldots, u_\omega)\), the LLL algorithm finds in polynomial time two linearly independent vectors \( b_1, b_2 \) such that

\[ \|b_1\|, \|b_2\| \leq 2^{\omega/4} (\det L)^{1/(\omega-1)}. \]

Therefore using LLL we can get two polynomials \( h_1(x, y) \) and \( h_2(x, y) \) such that

\[ \|h_1(xX, yY)\|, \|h_2(xX, yY)\| \leq 2^{\omega/4} \cdot (\det L)^{1/(\omega-1)}. \] (7)

Now using Howgrave-Graham’s lemma, we can determine the required bound on the norm of \( h_1 \) and \( h_2 \) to ensure that \( (x_0, y_0) \) is a root of both \( h_1 \) and \( h_2 \) over the integers:

**Lemma 1 (Howgrave-Graham [14]).** Assume \( h(x, y) \in \mathbb{Z}[x, y] \) is a sum of at most \( \omega \) monomials and assume further that \( h(x_0, y_0) = 0 \mod B \) where \(|x_0| \leq X \) and \(|y_0| \leq Y \) and \( \|h(xX, yY)\| < B/\sqrt{\omega} \). Then \( h(x_0, y_0) = 0 \) holds over the integers.

**Proof.** We have

\[ |h(x_0, y_0)| = \left| \sum h_{ij} x_0^i y_0^j \right| = \left| \sum h_{ij} X^i Y^j \left( \frac{x_0}{X} \right)^i \left( \frac{y_0}{Y} \right)^j \right| \leq \sum \left| h_{ij} X^i Y^j \left( \frac{x_0}{X} \right)^i \left( \frac{y_0}{Y} \right)^j \right| \leq \sum \left| h_{ij} X^i Y^j \right| \leq \sqrt{\omega} \|h(xX, yY)\| < B \]

Since \( h(x_0, y_0) = 0 \mod B \), this implies that \( h(x_0, y_0) = 0 \) over the integers. \( \Box \)

We apply Howgrave-Graham’s lemma with \( B := p' \). Using (7) this gives the condition:

\[ 2^{\omega/4} \cdot (\det L)^{1/(\omega-1)} \leq \frac{N^{\beta t}}{\sqrt{\omega}}. \] (8)

It is shown in [13] that by letting \( t = \tau \cdot m \) with \( \tau = 1 - \sqrt{1 - \beta} \), one gets the condition:

\[ \gamma + \delta \leq 3\beta - 2 + 2(1 - \beta)^{3/2} - \frac{3\beta(1 + \sqrt{1 - \beta})}{m} \]

Therefore one must take parameter \( m \) such that

\[ m \geq \frac{3\beta(1 + \sqrt{1 - \beta})}{\varepsilon}. \]

Since LLL is polynomial time in the lattice dimension and the lattice coefficients, the running time is polynomial in \( \log N \) and \( 1/\varepsilon \).
2.1 Discussion

For a balanced RSA modulus, we have $\beta = 1/2$, which gives the condition

$$\gamma + \delta \leq \frac{\sqrt{2} - 1}{2} \approx 0.207 \tag{9}$$

This means that for a 1024-bit RSA modulus $N$, the total size of the unknowns $x_0$ and $y_0$ can be at most 212 bits. Applied to our context, this implies that for ISO/IEC 9796-2 with $k_h = 160$, the size of the unknown random $r$ can be as large as 52 bits. Section 3 reports the result of practical experiments which show that the attack works well in practice. In Appendix we provide a Python code that enables to compute the bound on the size $k_r + k_h$ of the unknown values for various modulus sizes.

In the following we discuss three types of extension: extension to several separate blocks of unknown randomness in the encoding function, extension to two faults, one modulo $p$ and one modulo $q$, and extension to two or more faults modulo the same prime factor.

2.2 Extension to several blocks of random bits

Assume that the randomness used in ISO/IEC 9796-2 is split into $n$ different blocks, namely

$$\mu(m) = 6A_{16} \parallel \alpha_1 \parallel r_1 \parallel \alpha_2 \parallel r_2 \parallel \cdots \parallel \alpha_n \parallel r_n \parallel \alpha_{n+1} \parallel H(m) \parallel BC_{16} \tag{10}$$

where the random nonces $r_1, \ldots, r_n$ are all part of the message $m$ and unknown to the attacker, and the $\alpha_i$ blocks are known. We then make use of the extended result from Herrmann and May [13], which enables (heuristically) to find the solutions $(y_1, \ldots, y_n)$ of a linear equation modulo a factor $p$ of $N$:

$$a_0 + \sum_{i=1}^{n} a_i \cdot x_i = 0 \mod p$$

with $p \geq N^\beta$ and $|y_i| \leq N^{\gamma_i}$, under the condition [13]

$$\sum_{i=1}^{n} \gamma_i \leq 1 - (1 - \beta)^{\frac{n+1}{n}} - (n + 1)(1 - \sqrt{1 - \beta})(1 - \beta) .$$

For $\beta = 1/2$ and for a large number of blocks $n$, one gets the bound

$$\sum_{i=1}^{n} \gamma_i \leq \frac{1 - \ln 2}{2} \approx 0.153$$

This shows that if the total number of random bits plus the size of the hash value is less that 15.3% of the size of $N$, then the randomness can still be fully recovered from the faulty signature and the Boneh-DeMillo-Lipton attack applies again. However the number of blocks cannot be too large because the running time of the attack increases exponentially in $n$. 
2.3 Extension to two faults modulo different factors

Assume that we can get two faulty signatures, one incorrect modulo $p$ and the other modulo $q$. This gives us the two equations

\[
\begin{align*}
    a + b \cdot x_0 + c \cdot y_0 &= 0 \pmod{p} \\
    a' + b' \cdot x_1 + c' \cdot y_1 &= 0 \pmod{q}
\end{align*}
\]

with small unknowns $x_0, y_0, x_1, y_1$. By multiplying the two equations, we get an equation in four variables modulo $N$:

\[
aa' + ab' \cdot x_1 + ba' \cdot x_0 + ac' \cdot y_0 + a'c \cdot y_1 + bb' \cdot x_0x_1 + bc' \cdot x_0y_1 + cb' \cdot y_0x_1 + cc' \cdot y_0y_1 = 0 \pmod{N}
\]

We can linearize the equation (by replacing products of unknowns with new variables) and obtain a new equation of the form:

\[
a_0 + \sum_{i=1}^{8} a_i \cdot z_i = 0 \pmod{N}
\]

where the coefficients $a_0, \ldots, a_8$ are known and the unknowns $z_1, \ldots, z_8$ are small. Using LLL again, we recover the $z_i$’s (and then $x_0, x_1, y_0, y_1$) as long as the cumulated size of the $z_i$’s is at most the size of $N$. This yields the condition

\[
6 \cdot (k_r + k_h) \leq k
\]

which, using the notation of Theorem 1, can be reformulated as

\[
\gamma + \delta \leq \frac{1}{6} \approx 0.167.
\]

This remains weaker than condition (9). However, the attack is significantly faster because it works over a lattice of constant dimension 9. Moreover, the previous bound could most likely be improved using Coppersmith’s technique instead of plain linearization.

2.4 Extension to several faults modulo the same factor

In order to exploit single faults, we have shown how to use lattice-based techniques to recover $p$ given the modulus $N$ and a bivariate linear equation $f(x, y)$ admitting a small root $(x_0, y_0)$ modulo $p$. In this context, we have used Theorem 1 which is based on approximate GCD techniques from [15]. In the present extension, we would like to generalize this in order to make use of $\ell$ different polynomials of the same form, each having a small root modulo $p$. More precisely, let $\ell$ be a fixed parameter and assume that as the result of $\ell$ successive fault attacks, we are given $\ell$ different polynomials

\[
f_u(x_u, y_u) = a_u + x_u + c_u y_u
\]

where each polynomial $f_u$ has a small root $(\xi_u, \nu_u)$ modulo $p$ with $|\xi_u| \leq X$ and $|\nu_u| \leq Y$. Note that, as in the basic case, we renormalized each polynomial $f_u$ to ensure that the coefficient of $x_u$ in $f_u$ is equal to 1. In order to avoid double subscripts, we now use the Greek letters $\xi$ and $\nu$ to represent the root values. We would like to use a lattice approach to construct new multivariate polynomials in the variables $(x_1, \ldots, x_\ell, y_1, \ldots, y_\ell)$ with the root $R = (\xi_1, \ldots, \xi_\ell, \nu_1, \ldots, \nu_\ell)$. For this purpose, we fix two parameters $m$ and $\ell$ and build a lattice on a family $\mathcal{G}$ of polynomials of degree at most $m$ with root $R$ modulo $B = p^\ell$. This family is composed of all polynomials of the form

\[
y_1^{\iota_1} y_2^{\iota_2} \cdots y_\ell^{\iota_\ell} f_1(x_1, y_1)^{\iota_1} f_2(x_2, y_2)^{\iota_2} \cdots f_\ell(x_\ell, y_\ell)^{\iota_\ell} N^{\max(t-j,0)}
\]
where each \(i_u, j_u\) is non-negative, \(i = \sum_{u=1}^{\ell} i_u, j = \sum_{u=1}^{\ell} j_u\) and \(0 \leq i + j \leq m\). Once again, let \(L\) be the corresponding lattice. Its dimension \(\omega\) is equal to the number of monomials of degree at most \(m\) in \(2\ell\) unknowns, i.e.

\[
\omega = \binom{m + 2\ell}{2\ell}.
\]

Since we have a common upper bound \(X\) for all values \(|\xi_u|\) and a common bound for all \(|\nu_u|\) we can compute the determinant of the lattice as

\[
\det(L) = X^{s_x} Y^{s_y} N^{s_N},
\]

where \(s_x\) is the sum of the exponents of all unknowns \(x_u\) in all occurring monomials, \(s_y\) the sum of the exponents of the \(y_u\) and \(s_N\) the sum of the exponents of \(N\) in all occurring polynomials. For obvious symmetry reasons, we have \(s_x = s_y\) and recalling that the number of polynomials of degree exactly \(d\) in \(\ell\) unknowns is \(\binom{d+\ell-1}{\ell-1}\) we find

\[
s_x = s_y = \sum_{d=0}^{m} d \binom{d + \ell - 1}{\ell - 1} \binom{m - d + \ell}{\ell}.
\]

Likewise, summing on polynomials with a non-zero exponent \(v\) for \(N\), where the sum of the \(j_u\) values is \(t - v\) we obtain

\[
s_N = \sum_{v=1}^{t} v \binom{t - v + \ell - 1}{\ell - 1} \binom{m - t + v + \ell}{\ell}.
\]

As usual, assuming that \(p = N^\beta\) we can find a polynomial with the correct root over the integer under the condition from Equation (8).

**Concrete bounds:** Using the same notation as in Theorem 1, we compute effective bounds on

\[
\gamma + \delta = \log(XY)/\log(N)
\]

from the logarithm of condition (8). When taking this logarithm, we ignore the terms \(\sqrt{\omega}\) and \(2^{\omega/4}\) with are negligible when \(N\) becomes large. For concrete values of \(N\), the bounds are slightly smaller. After dividing by \(\log(N)\), we find

\[
s_x \cdot (\gamma + \delta) + s_N \leq \beta t \omega.
\]

Thus, given \(k, t\) and \(m\), we can achieve at best

\[
\gamma + \delta = \frac{\beta t \omega - s_N}{s_x}.
\]

We have computed the achievable values of \(\gamma + \delta\) for \(\beta = 1/2\) and for various parameters with lattice dimension \(\omega\) up to approximately 1000. The results are presented in Table 4 in Appendix B.
Recovering the root: With $2\ell$ unknowns instead of 2, applying the usual heuristic and hoping that the lattice reduction directly outputs $2\ell$ algebraically independent polynomials with the prescribed root over the integer becomes a wild hope. Thankfully, a milder heuristic assumption is enough to make the attack work. The idea is to start with $K$ equations instead of $\ell$ and to run the lattice reduction attack several times for several subsets of $\ell$ equations among the $K$ available equations. Potentially, we can perform $\binom{K}{\ell}$ such lattice reductions. Clearly, since each equation involves a different subset of unknowns, they are all different. Note that this is not sufficient to guarantee algebraic independence; in particular, if we generate more than $K$ equations they cannot be algebraically independent. However, we only need to make sure that the root $R$ can be extracted from the available set of equations. This can be done, using Gröbner basis techniques, under the heuristic assumption that the set of equations spans a multivariate ideal of dimension 0 i.e. that the number of solutions is finite.

Note that we need to choose reasonably small values of $\ell$ and $K$ in order to be able to use this approach in practice. Indeed, the lattice we consider should not become too large and, in addition, it should be possible to solve the resulting system of equations using either resultants or Gröbner basis which means that neither the degree nor the number of unknowns should increase too much.

Asymptotic bounds: Despite the fact that we cannot hope to run the multi-polynomial variant of our attack when the parameters become too large, it is interesting from a theoretical point of view to determine the limit of the achievable value of $\gamma + \delta$ as the number of faults $\ell$ increases. For this purpose, we assume as previously that $\beta = 1/2$, we let $t = \tau m$ and replace $\omega$, $s_x$ and $s_N$ by the following approximations:

$$\omega \approx \frac{m^{2\ell}}{(2\ell)!}, \quad s_x \approx \sum_{d=0}^{m} \frac{d^\ell (m-d)^\ell}{(\ell-1)! \ell!}, \quad s_N \approx \sum_{v=1}^{t} \frac{(t-v)^{\ell-1}(m-t+v)^\ell}{(\ell-1)! \ell!}.$$

For small values of $\ell$, we provide in Table 1 the corresponding bounds on $\gamma + \delta$. Although we do not provide further details here due to lack of space, one can show that the bound $\gamma + \delta$ tends to 1/2 as the number of faults $\ell$ tends to infinity.

<table>
<thead>
<tr>
<th>$\ell$</th>
<th>$\gamma + \delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.207</td>
</tr>
<tr>
<td>2</td>
<td>0.293</td>
</tr>
<tr>
<td>3</td>
<td>0.332</td>
</tr>
<tr>
<td>4</td>
<td>0.356</td>
</tr>
<tr>
<td>5</td>
<td>0.371</td>
</tr>
<tr>
<td>6</td>
<td>0.383</td>
</tr>
<tr>
<td>7</td>
<td>0.391</td>
</tr>
<tr>
<td>8</td>
<td>0.399</td>
</tr>
<tr>
<td>9</td>
<td>0.405</td>
</tr>
<tr>
<td>10</td>
<td>0.410</td>
</tr>
</tbody>
</table>

Table 1. Bound for the relative size $\gamma + \delta$ of the unknowns as a function of the number of faults $\ell$.

3 Simulation Results

In the following we describe the result of a number of practical experiments we have carried out to benchmark our attack. It must be said that, given the importance of off-line computations in our approach, we did not perform a practical fault attack on an actual cryptographic device. So we assume that fault injection can easily be performed on any unprotected device (see e.g. [1] for more details). Therefore we have simulated fault injection i.e. in a first phase we generate a faulty signature (using the factors $p$ and $q$) and in a second phase we apply the mathematical part of the attack as discussed in the above sections to recover the factors of $N$. 


3.1 Running times given one faulty signature

We first consider the main attack with only one block of randomness introduced in the message and one faulty signature. We focused on the case when the hash function is SHA-1, hence $k_h = 160$. For LLL we have used the implementation provided by the SAGE library [22]. Computations were executed on a typical 2GHz Intel notebook.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$k_r$</th>
<th>$k_h$</th>
<th>$m$</th>
<th>$t$</th>
<th>$\omega$</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>6</td>
<td>160</td>
<td>10</td>
<td>3</td>
<td>66</td>
<td>4 min</td>
</tr>
<tr>
<td>1024</td>
<td>13</td>
<td>160</td>
<td>13</td>
<td>4</td>
<td>105</td>
<td>51 min</td>
</tr>
<tr>
<td>1536</td>
<td>70</td>
<td>160</td>
<td>8</td>
<td>2</td>
<td>45</td>
<td>39 s</td>
</tr>
<tr>
<td>1536</td>
<td>90</td>
<td>160</td>
<td>10</td>
<td>3</td>
<td>66</td>
<td>9 min</td>
</tr>
<tr>
<td>2048</td>
<td>158</td>
<td>160</td>
<td>8</td>
<td>2</td>
<td>45</td>
<td>55 s</td>
</tr>
</tbody>
</table>

Table 2. Mathematical part of our fault attack for one block of randomness and one faulty signature, with various values of the modulus size $k$, randomness size $k_r$, hash size $k_h$, parameters $m$ and $t$, lattice dimension $\omega$ and LLL running time.

The result of practical experiments are summarized in Table 2. We see that for 1024-bit RSA the random $r$ must be quite small and the attack is actually less efficient than exhaustive search\(^3\). However for larger modulus sizes, the attack becomes more efficient. As a typical example, when dealing with a 2048-bit RSA modulus, the factors of $N$ are recovered from a single faulty signature in less than a minute for a random $r$ of 158 bits.

3.2 Running times given several faulty signatures

To test the practicality of the extension presented in Section 2.4, we have set $(\ell, t, m) = (3, 1, 3)$. Hence we assume to be given three faulty signatures. This leads to a lattice of dimension 84 and a bound $\gamma + \delta \leq 0.204$. Our experiments were carried out with RSA moduli of 1024, 1536 and 2048 bits. This implementation also relies on the SAGE library [22] running on a single PC. Quite surprisingly, we observe a very large number of polynomials with the expected root over the integers. We ran the test for three random instances corresponding to the parameters in Table 3.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$k_r$</th>
<th>$k_h$</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>40</td>
<td>160</td>
<td>49 s</td>
</tr>
<tr>
<td>1536</td>
<td>150</td>
<td>160</td>
<td>74 s</td>
</tr>
<tr>
<td>2048</td>
<td>250</td>
<td>160</td>
<td>111s</td>
</tr>
</tbody>
</table>

Table 3. Mathematical part of our fault attack for one block of randomness and three faulty signatures, with various values of the modulus size $k$, randomness size $k_r$, hash size $k_h$ and total running time.

For each set of parameters, the first 71 vectors in the reduced lattice have the expected root. In fact, it is even possible to solve the system of equations without using a Gröbner basis algorithm. Instead,\(^3\) Exhaustive search on a 13-bit random took 0.13 seconds in our experiment.
we use a much simpler strategy. The first step is to consider the system modulo a prime \( p \) above \( 2^{160} \) (or above \( 2^{250} \) for the 2048-bit experiment). With this system, a very simple linearization approach suffices to obtain through echelonization the polynomials \( x_i - \xi_i \) and \( y_i - \nu_i \). Since \( p \) is larger than the bounds on the values, this yields the exact values of \( \xi_1, \xi_2, \xi_3, \nu_1, \nu_2 \) and \( \nu_3 \). Once this is done, the factors of \( N \) are easily recovered by computing the GCD of \( N \) with any of the values \( f_i(\xi_i, \nu_i) \).

We see that with three faults, the attack summarized in Table 3 is more efficient than the single fault attack in Table 2. In particular for a 1024-bit RSA modulus, the three-fault attack enables to recover a 40-bit random \( r \) in 49 seconds\(^4\), whereas the single-fault attack could only recover a 13-bit random in 51 minutes.

4 Conclusion

The paper introduced an effective approach to perform fault attacks against ISO/IEC 9796-2 with randomized messages and subsequently factor the modulus \( N \) given one single faulty signature. Although the attack is heuristic, it works well in practice and becomes more efficient for larger modulus sizes. Alternately, assuming several faulty signatures are given, we allow larger sizes for the randomness and hash value introduced in the encoded message.

We note that our techniques are more generally applicable to any context where the signed messages and partially unknown, in which case our approach gives explicit size conditions for the fault attack to apply. This has a direct impact on other encoding functions, such as the PKCS#1 v1.5 standard. In PKCS#1 v1.5, a message \( m \) is encoded as

\[
\mu(m) = 0001_{16} \parallel (FF_{16})^{k_1} \parallel 00_{16} \parallel T \parallel H(m)
\]

where \( T \) is a known sequence of bytes which encodes the identifier of the hash function \( H \) and \( k_1 \) is a size parameter which can be adjusted to make \( \mu(m) \) have the same number of bytes than the modulus. Assuming a 2048-bit modulus and \( H = \text{SHA-384} \), our results show that the modulus can be efficiently factored from one single faulty signature \( \sigma \) even when the signed message is totally unknown. This stems from the fact that \( \gamma = 0 \) and \( \delta = 384/2048 \), meaning that the required condition \( \gamma + \delta \leq 0.207 \) (condition (9)) is realized. Alternately, we see that \( N \) can also be factored when \( H = \text{SHA-512} \) given 5 faulty signatures on unknown messages. This enables fault attacks on intricate cryptographic scenarios where e.g. the device and a terminal exchange RSA signatures on encrypted messages.

References


\(^4\) We estimate that exhaustive search on a 40-bit random would take roughly one year on the same single PC.
22. SAGE, Mathematical Library available at www.sagemath.org
24. V. Shoup, Number Theory C++ Library (NTL) version 5.3.1. www.shoup.net/ntl.

A Estimating the Size of the Roots

from math import log,sqrt

def LatticeExpo(t,m):
    sy=sx=sn=w=0
    for k in range(m+1):
        for i in range(m-k+1):
            j=max(t-k,0)
            sy+=i; sx+=k; sn+=j; w+=1
    return (sx,sy,sn,w)

def bound(t,m,n):
    (sx,sy,sn,w)=LatticeExpo(t,m)
    nxy=(w*(n*t*.5-.25*w-log(sqrt(w),2))-n*sn)/sx
    return nxy,w
### B Achievable Bound on $\gamma + \delta$

We provide the achievable bound on $\gamma + \delta$, as a function of the number of faults $\ell$ and parameters $(t, m)$.

<table>
<thead>
<tr>
<th>Dimension $\omega$</th>
<th>Bound $\gamma + \delta$ $(\ell, t, m)$</th>
<th>Table 4. Achievable bound on $\gamma + \delta = \log(XY)/\log(N)$</th>
<th>Dimension $\omega$</th>
<th>Bound $\gamma + \delta$ $(\ell, t, m)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.100 (1, 1, 3)</td>
<td></td>
<td>21</td>
<td>0.129 (1, 1, 5)</td>
</tr>
<tr>
<td>28</td>
<td>0.143 (1, 2, 6)</td>
<td></td>
<td>45</td>
<td>0.158 (1, 2, 8)</td>
</tr>
<tr>
<td>55</td>
<td>0.161 (1, 3, 9)</td>
<td></td>
<td>78</td>
<td>0.171 (1, 3, 11)</td>
</tr>
<tr>
<td>91</td>
<td>0.172 (1, 3, 12)</td>
<td></td>
<td>120</td>
<td>0.179 (1, 4, 14)</td>
</tr>
<tr>
<td>120</td>
<td>0.179 (1, 4, 15)</td>
<td></td>
<td>171</td>
<td>0.183 (1, 5, 17)</td>
</tr>
<tr>
<td>253</td>
<td>0.187 (1, 6, 21)</td>
<td></td>
<td>300</td>
<td>0.190 (1, 7, 23)</td>
</tr>
<tr>
<td>325</td>
<td>0.190 (1, 7, 24)</td>
<td></td>
<td>378</td>
<td>0.190 (1, 8, 26)</td>
</tr>
<tr>
<td>406</td>
<td>0.192 (1, 8, 27)</td>
<td></td>
<td>465</td>
<td>0.192 (1, 9, 29)</td>
</tr>
<tr>
<td>496</td>
<td>0.193 (1, 9, 30)</td>
<td></td>
<td>561</td>
<td>0.194 (1, 9, 32)</td>
</tr>
<tr>
<td>595</td>
<td>0.194 (1, 10, 33)</td>
<td></td>
<td>666</td>
<td>0.195 (1, 10, 35)</td>
</tr>
<tr>
<td>703</td>
<td>0.195 (1, 10, 36)</td>
<td></td>
<td>780</td>
<td>0.196 (1, 11, 38)</td>
</tr>
<tr>
<td>820</td>
<td>0.196 (1, 11, 39)</td>
<td></td>
<td>903</td>
<td>0.197 (1, 12, 41)</td>
</tr>
<tr>
<td>946</td>
<td>0.197 (1, 12, 42)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0.125 (2, 1, 2)</td>
<td></td>
<td>70</td>
<td>0.179 (2, 1, 4)</td>
</tr>
<tr>
<td>126</td>
<td>0.214 (2, 2, 5)</td>
<td></td>
<td>330</td>
<td>0.229 (2, 3, 7)</td>
</tr>
<tr>
<td>495</td>
<td>0.240 (2, 3, 8)</td>
<td></td>
<td>1001</td>
<td>0.248 (2, 4, 10)</td>
</tr>
<tr>
<td>28</td>
<td>0.167 (3, 1, 2)</td>
<td></td>
<td>210</td>
<td>0.222 (3, 2, 4)</td>
</tr>
<tr>
<td>462</td>
<td>0.247 (3, 2, 5)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>0.188 (4, 1, 2)</td>
<td></td>
<td>495</td>
<td>0.244 (4, 2, 4)</td>
</tr>
<tr>
<td>66</td>
<td>0.200 (5, 1, 2)</td>
<td></td>
<td>1001</td>
<td>0.258 (5, 2, 4)</td>
</tr>
<tr>
<td>91</td>
<td>0.208 (6, 1, 2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>0.214 (7, 1, 2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>0.219 (8, 1, 2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>0.222 (9, 1, 2)</td>
<td></td>
<td>276</td>
<td>0.227 (11, 1, 2)</td>
</tr>
<tr>
<td>325</td>
<td>0.229 (12, 1, 2)</td>
<td></td>
<td>435</td>
<td>0.232 (14, 1, 2)</td>
</tr>
<tr>
<td>496</td>
<td>0.233 (15, 1, 2)</td>
<td></td>
<td>630</td>
<td>0.235 (17, 1, 2)</td>
</tr>
<tr>
<td>703</td>
<td>0.236 (18, 1, 2)</td>
<td></td>
<td>861</td>
<td>0.238 (20, 1, 2)</td>
</tr>
<tr>
<td>946</td>
<td>0.238 (21, 1, 2)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>