On the Security of RSA Padding

Jean-Sébastien Coron[1+2] David Naccache[2] Julien P. Stern[3+4]

1. École Normale Supérieure
45 rue d'Ulm
F-75005, Paris, France
coron@clipper.ens.fr

2. Gemplus Card International
34 rue Guynemer
Issy-les-Moulineaux, F-92447, France
{coron,naccache}@gemplus.com

3. UCL Cryptography Group
Bâtiment Maxwell, place du Levant 3
Louvain-la-Neuve, B-1348, Belgium
stern@dice.ucl.ac.be

4. Université de Paris-Sud
Laboratoire de Recherche en Informatique
Bâtiment 490, F-91405, Orsay, France
stern@lri.fr

Abstract. This paper presents a new signature forgery strategy.

The attack is a sophisticated variant of Desmedt-Odlyzko's method [11] where the attacker obtains the signatures of $m_1, \ldots, m_{\tau-1}$ and exhibits the signature of an m_τ which was never submitted to the signer; we assume that all messages are padded by a redundancy function μ before being signed.

Before interacting with the signer, the attacker selects τ smooth[1] $\mu(m_i)$-values and expresses $\mu(m_\tau)$ as a multiplicative combination of the padded strings $\mu(m_1), \ldots, \mu(m_{\tau-1})$. The signature of m_τ is then forged using the homomorphic property of RSA.

A padding format that differs from ISO 9796-1 by one single bit was broken experimentally (*we emphasize that we could not extend our attack to* ISO 9796-1); for ISO 9796-2 the attack is more demanding but still much more efficient than collision-search or factoring.

For DIN NI-17.4, PKCS #1 V2.0 and SSL-3.02, the attack is only theoretical since it only applies to specific moduli and happens to be less efficient than factoring; *therefore, the attack does not endanger any of these standards.*

1 Introduction

At a recent count (`http://www.rsa.com`), over 300 million RSA-enabled products had been shipped worldwide. This popularity, and the ongoing standardizations of signature and encryption formats [2,13,20,21,22,36] highlight the need to challenge claims that such standards eradicate RSA's multiplicative properties.

[1] an integer is ℓ-*smooth* if it has no bigger factors than ℓ.

Michael Wiener (Ed.): CRYPTO'99, LNCS 1666, pp. 1–18, 1999.

Exponentiation is homomorphic and RSA-based protocols are traditionally protected against chosen-plaintext forgeries [9,11,35] by using a *padding* (or *redundancy*) function μ to make sure that :

$$\mathrm{RSA}(\mu(x)) \times \mathrm{RSA}(\mu(y)) \neq \mathrm{RSA}(\mu(x \times y)) \bmod n$$

In general, $\mu(x)$ hashes x and concatenates its digest to pre-defined strings; in some cases, substitution and permutation are used as well.

While most padding schemes gain progressive recognition as time goes by, several specific results exist : a few functions were broken by *ad-hoc* analysis ([16,24] showed, for instance, that homomorphic dependencies can still appear in $\mu(m) = a \times m + b$) while at the other extreme, assuming that the underlying building-blocks are ideal, some functions [5,6] are provably secure in the random oracle model.

The contribution of this paper is that the complexity of forging chosen message-signature pairs is sometimes *much* lower than that of breaking $\mathrm{RSA} \circ \mu$ by frontal attacks (factoring and collision-search). The strategy introduced in this article does not challenge RSA's traditional security assumptions; instead, it seeks for multiplicative relations using the expected smoothness of moderate-size integers (the technique is similar in this respect to the quadratic sieve [33], the number field sieve [32] and the index-calculus method for computing discrete logarithm [1]).

As usual, our playground will be a setting in which the attacker $\mathcal{A}$ and the signer $\mathcal{S}$ interact as follows :

- $\mathcal{A}$ asks $\mathcal{S}$ to provide the signatures of $\tau - 1$ chosen messages (τ being polylogarithmically-bounded in n). $\mathcal{S}$ will, of course, correctly pad all the plaintexts before raising them to his secret power d.
- *After* the query phase and some post-processing, $\mathcal{A}$ must exhibit the signature of at least one message (m_τ) which has never been submitted to $\mathcal{S}$.

Previous work : Misarsky's PKC'98 invited survey [30] is probably the best documented reference on multiplicative RSA forgeries. Davida's observation [9] is the basis of most RSA forgery techniques. [16,24] forge signatures that are similar to PKCS #1 v2.0 but do not produce their necessary SHA/MD5 digests [31,34]. [15] analyzes the security of RSA signatures in an interactive context. Michels *et al.* [28] create relations between the exponents of de Jonge-Chaum and Boyd's schemes; their technique extends to blind-RSA but does not apply to any of the padding schemes attacked in this paper. Baudron and Stern [4] apply lattice reduction to analyze the security of $\mathrm{RSA} \circ \mu$ in a security-proof perspective.

A Desmedt-Odlyzko variant [11] applicable to padded RSA signatures is sketched in section 3.5 of [30]. It consists in factoring $\mu(m_\tau)$ into small primes and obtaining the e-th roots of these primes from multiplicative combinations of signatures of messages which $\mu(m_i)$-values are smooth. The signature of m_τ is forged by multiplying the e-th roots of the factors of $\mu(m_\tau)$. The complexity of

this attack depends on the size of μ and not on the size of n; the approach is thus inapplicable to padding formats having the modulus' size (*e.g.* ISO 9796-2). In this paper we extend this strategy to padding schemes for which a linear combination of n and the padded value is small; when applied to William's scheme our attack allows to factor n.

2 A General Outline

Let $\{n, e\}$ be an RSA public key and d be the corresponding secret key. Although in this paper μ will alternatively denote ISO 9796-2, PKCS #1 v2.0, ANSI X9.31, SSL-3.02 or an ISO 9796-1 variant denoted $\mathcal{F}$, we will start by describing our attack in a simpler scenario where μ is SHA-1 or MD5 (in other words, messages will *only* be hashed before being exponentiated); the attack will be later adapted to the different padding standards mentioned above.

The outline of our idea is the following : since $\mu(m)$ is rather short (128 or 160 bits), the probability that $\mu(m)$ is ℓ-smooth (for a reasonably small ℓ) is small but non-negligible; consequently, if $\mathcal{A}$ can obtain the signatures of chosen smooth $\mu(m_i)$-values, then he could look for a message m_τ such that $\mu(m_\tau)$ has no bigger factors than p_k (the k-th prime) and construct $\mu(m_\tau)^d \bmod n$ as a multiplicative combination of the signatures of the chosen plaintexts $m_1, \ldots, m_{\tau-1}$.

The difficulty of finding ℓ-smooth digests is a function of ℓ and the size of $\mu(m)$. Defining $\psi(x, y) = \#\{v < x, \text{ such that } v \text{ is } y\text{-smooth}\}$, it is known [12,14,19] that, for large x, the *ratio* $\psi(x, \sqrt[t]{x})/x$ is equivalent to Dickman's function defined by :

$$\rho(t) = \begin{cases} 1 & \text{if } \ 0 \leq t \leq 1 \\ \rho(n) - \displaystyle\int_n^t \frac{\rho(v-1)}{v} dv & \text{if } \ n \leq t \leq n+1 \end{cases}$$

$\rho(t)$ is thus an approximation of the probability that a u-bit number is $2^{u/t}$-smooth; since $\rho(t)$ is somewhat cumbersome to compute, we refer the reader to appendix A for a lookup table.

Before we proceed, let us illustrate the concerned orders of magnitude. Referring to appendix A, we see that the probability that SHA/MD5 digests are 2^{24}-smooth is rather high ($\cong 2^{-19}, 2^{-13}$); this means that finding smooth digests would be practically feasible. This was confirmed by extensive simulations as illustrated by :

$$\begin{aligned} &\text{MD5}(\texttt{message 30854339 successfully forged}) = \\ &\texttt{955dd317dd4715d26465081e4bfac000}_{16} \qquad\quad = \end{aligned}$$

$$2^{14} \times 3 \times 5^3 \times 13 \times 227 \times 1499 \times 1789 \times 2441 \times 4673 \times 4691 \times 9109 \times 8377619$$

Several heuristics can, of course, accelerate the search : in our experiments, we factored only digests beginning or ending by a few zeroes; the optimal number of zeroes being a function of the running times of the attacker's hashing and factorization algorithms (parallelization is also possible).

In any case, denoting by L the size of the digest and by $F(L)$ the factoring cost, the complexity of finding p_k-smooth digests is :

$$C_{L,k} = \mathcal{O}(\frac{F(L)}{\rho(L/\log_2(p_k))}) = \mathcal{O}(\frac{kL\log_2(p_k)}{\rho(L/\log_2(p_k))}) = \mathcal{O}(\frac{kL\log_2(k\ln k)}{\rho(L/\log_2(k\ln k))})$$

this is justified by the fact that p_k-smooth L-bit digests are expected only once per $1/\rho(L/\log_2(p_k))$ and that the most straightforward way to factor L is k trial divisions by the first primes (where each division costs $L\log_2(p_i)$ bit-operations).

These formulae should, however, be handled with extreme caution for the following reasons :

• Although in complexity terms L can be analyzed as a variable, one should constantly keep in mind that L is a fixed value because the output size of *specific* hash functions is not extensible.

• Trial division is definitely not the best candidate for $F(L)$. In practice, our program used the following strategy to detect the small factors of $\mu(m)$: since very small divisors are very common, it is worthwhile attempting trial and error division up to $p_i \cong 2048$ before applying a primality test to $\mu(m)$ (the candidate is of course rejected if the test fails). As a next step, trial and error division by primes smaller than $15,000$ is performed and the resulting number is handed-over to Pollard-Brent's algorithm [7] which is very good at finding small factors. Since it costs $\mathcal{O}(\sqrt{p_i})$ to pull-out p_i using Pollard-Brent's method we can further bound $F(L)$ by $L\sqrt{p_k}$ to obtain :

$$C_{L,k} = \mathcal{O}(\frac{L\sqrt{k\ln k}}{\rho(L/\log_2(k\ln k))})$$

3 The Attack

The attack applies to RSA and Williams' scheme [37]; we assume that the reader is familiar with RSA but briefly recall Williams' scheme, denoting by $\mathrm{J}(x)$, the Jacobi symbol of x with respect to n.

In Williams' scheme $\mu(m) = 6 \bmod 16$ and :

$$\begin{array}{ll} p = 3 \bmod 8 & e = 2 \\ q = 7 \bmod 8 & d = (n - p - q + 5)/8 \end{array}$$

Before signing, $\mathcal{S}$ must check that $\mathrm{J}(\mu(m)) = 1$. If $\mathrm{J}(\mu(m)) = -1$, $\mu(m)$ is replaced by $\mu(m)/2$ to guarantee that $\mathrm{J}(\mu(m)) = 1$ since $\mathrm{J}(2) = -1$.

A signature s is valid if $w = s^2 \bmod n$ is such that :

$$\mu(m) \stackrel{?}{=} \begin{cases} w & \text{if} \quad w = 6 \bmod 8 \\ 2w & \text{if} \quad w = 3 \bmod 8 \\ n - w & \text{if} \quad w = 7 \bmod 8 \\ 2(n - w) & \text{if} \quad w = 2 \bmod 8 \end{cases}$$

3.1 Finding Homomorphic Dependencies

The attack's details slightly differ between the RSA and Williams' scheme. For RSA, $\tau - 1$ chosen signatures will yield an additional $\mu(m_\tau)^d \bmod n$ while in Williams' case, τ chosen signatures will factor n. All chosen messages have the property that there exists a linear combination of $\mu(m_i)$ and n such that :

$$a_i \times n - b_i \times \mu(m_i) \quad \text{is } p_k\text{-smooth}$$

where b_i is p_k-smooth as well.

It follows that $\mu(m_i)$ is the modular product of small primes :

$$\mu(m_i) = \prod_{j=1}^{k} p_j^{v_{i,j}} \bmod n \quad \text{for } 1 \leq i \leq \tau$$

Let us associate to each $\mu(m_i)$ a k-dimensional vector $\boldsymbol{V}_i$ with coordinates $v_{i,j}$ taken modulo the public exponent e :

$$\mu(m_i) \longmapsto \boldsymbol{V}_i = \{v_{i,1} \bmod e, \ldots, v_{i,k} \bmod e\}$$

We can now express, by Gaussian elimination, one of these vectors (re-indexed as $\boldsymbol{V}_\tau$) as a linear combination of the others :

$$\boldsymbol{V}_\tau = \sum_{i=1}^{\tau-1} \beta_i \boldsymbol{V}_i \mod e, \quad \text{with} \quad \beta_i \in \mathbb{Z}_e \tag{1}$$

From equation (1) we get :

$$v_{\tau,j} = \sum_{i=1}^{\tau-1} \beta_i v_{i,j} - \gamma_j \times e \quad \text{for all} \quad 1 \leq j \leq k$$

.

and denoting $x = \prod_{j=1}^{k} p_j^{-\gamma_j}$:

$$\mu(m_\tau) = x^e \times \prod_{i=1}^{\tau-1} \mu(m_i)^{\beta_i} \bmod n$$

For RSA, the forger will submit the $\tau - 1$ first messages to $\mathcal{S}$ and forge the signature of m_τ by :

$$\mu(m_\tau)^d = x \times \prod_{i=1}^{\tau-1} \left(\mu(m_i)^d\right)^{\beta_i} \bmod n$$

In Williams' case, the signature of m_τ will be computed from the other signatures using equation (2) if $\mathrm{J}(x) = 1$, using the fact that :

$$u = x^{2d} \bmod n = \begin{cases} x & \text{if } x \text{ is a square modulo } n \\ -x & \text{if not.} \end{cases}$$

$$\mu(m_\tau)^d = \pm x \times \prod_{i=1}^{\tau-1} \left(\mu(m_i)^d\right)^{\beta_i} \bmod n \tag{2}$$

If $\mathrm{J}(x) = -1$, then $u^2 = x^2 \bmod n$ and $(u-x)(u+x) = 0 \bmod n$. Since $\mathrm{J}(x) = -\mathrm{J}(u)$ we have $x \neq \pm u \bmod n$ and $\mathrm{GCD}(u-x, n)$ will factor n. $\mathcal{A}$ can thus submit the τ messages to $\mathcal{S}$, recover u, factor n and sign any message.

3.2 Expected Complexity

It remains, however, to estimate τ as a function of k :

• In the most simple setting e is prime and the set of vectors with k coordinates over $\mathbb{Z}_e$ is a k-dimensional linear space; $\tau = k+1$ vectors are consequently sufficient to guarantee that (at least) one of the vectors can be expressed as a linear combination (easily found by Gaussian elimination) of the other vectors.

• When e is the r-th power of a prime p, $\tau = k+1$ vectors are again sufficient to ensure that (at least) one vector can be expressed as a linear combination of the others. Using the p-adic expansion of the vectors' coefficients and Gaussian elimination on $k+1$ vectors, we can write one of the vectors as a linear combination of the others.

• Finally, the previous argument can be extended to the most general case :

$$e = \prod_{i=1}^{\omega} p_i^{r_i}$$

where it appears that $\tau = 1 + \omega k = \mathcal{O}(k \log e)$ vectors are sufficient to guarantee that (at least) one vector is a linear combination of the others; modulo each of the $p_i^{r_i}$, the attacker can find a set T_i of $(\omega-1)k+1$ vectors, each of which can be expressed by Gaussian elimination as a linear combination of k other vectors. Intersecting the T_i and using Chinese remaindering, one gets that (at least) one vector must be a linear combination of the others modulo e.

The overall complexity of our attack can therefore be bounded by :

$$C'_{L,k} = \mathcal{O}(\tau C_{L,k}) = \mathcal{O}\left(\frac{Lk \log e \sqrt{k \ln k}}{\rho(L/\log_2(k \ln k))}\right)$$

and the attacker can optimize his resources by operating at a k where $C'_{L,k}$ is minimal.

Space complexity (dominated by the Gaussian elimination) is $\mathcal{O}(k^2 \log^3 e)$.

4 Analyzing Different Signature Formats

4.1 The Security of ISO/IEC-9796-1-like Signatures

ISO/IEC-9796-1 [21] was published in 1991 by ISO as the first international standard for digital signatures. It specifies padding formats applicable to algorithms providing message recovery (algorithms are not explicit but map r bits to r bits). ISO 9796-1 is not hashing-based and there are apparently no attacks [16,18] other than factoring on this scheme ([30] : "...ISO 9796-1 *remains beyond the reach of all multiplicative attacks known today...*"). The scheme is used to sign messages of limited length and works as follows when n and m are respectively $N = 2\gamma + 1$ and γ-bit numbers and $\gamma = 4\ell$ is a multiple of eight.

Define by $a \cdot b$ the concatenation of a and b, let ω_i be the i-th nibble of m and denote by $s(x)$ the hexadecimal substitution table[2] :

$x =$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$s(x) =$	E	3	5	8	9	4	2	F	0	D	B	6	7	A	C	1

Letting $\bar{s}(x)$ force the most significant bit in $s(x)$ to 1 and $\tilde{s}(x)$ complement the least significant bit of $s(x)$, ISO 9796-1 specifies :

$$\begin{aligned}\mu(m) = \bar{s}(\omega_{\ell-1}) \cdot \tilde{s}(\omega_{\ell-2}) \cdot \omega_{\ell-1} \cdot \omega_{\ell-2} \cdot \\ s(\omega_{\ell-3}) \cdot s(\omega_{\ell-4}) \cdot \omega_{\ell-3} \cdot \omega_{\ell-4} \cdot \\ \ldots \\ s(\omega_3) \cdot s(\omega_2) \cdot \omega_3 \cdot \omega_2 \cdot \\ s(\omega_1) \cdot s(\omega_0) \cdot \omega_0 \cdot 6_{16}\end{aligned}$$

The attack that we are about to describe applies to a slight variant of ISO 9796-1 where $\tilde{s}(x)$ is replaced by $s(x)$; this variant (denoted $\mathcal{F}$) differs from ISO 9796-1 by one single bit.

Let a_j denote nibbles and consider messages of the form :

$$\begin{aligned}m_i = a_6 \cdot a_5 \cdot a_4 \cdot a_3 \cdot a_2 \cdot a_1 \cdot 66_{16} \cdot \\ a_6 \cdot a_5 \cdot a_4 \cdot a_3 \cdot a_2 \cdot a_1 \cdot 66_{16} \cdot \\ \ldots \\ a_6 \cdot a_5 \cdot a_4 \cdot a_3 \cdot a_2 \cdot a_1 \cdot 66_{16}\end{aligned}$$

which $\mathcal{F}$-padding is :

$$\begin{aligned}\mu(m_i) = \bar{s}(a_6) \cdot s(a_5) \cdot a_6 \cdot a_5 \cdot s(a_4) \cdot s(a_3) \cdot a_4 \cdot a_3 \cdot \\ s(a_2) \cdot s(a_1) \cdot a_2 \cdot a_1 \cdot 2_{16} \cdot 2_{16} \cdot 6_{16} \cdot 6_{16} \cdot \\ \ldots \\ s(a_6) \cdot s(a_5) \cdot a_6 \cdot a_5 \cdot s(a_4) \cdot s(a_3) \cdot a_4 \cdot a_3 \cdot \\ s(a_2) \cdot s(a_1) \cdot a_2 \cdot a_1 \cdot 2_{16} \cdot 2_{16} \cdot 6_{16} \cdot 6_{16}\end{aligned}$$

[2] actually, the bits of $s(x)$ are respectively $\overline{x_3} \oplus x_1 \oplus x_0$, $\overline{x_3} \oplus x_2 \oplus x_0$, $\overline{x_3} \oplus x_2 \oplus x_1$ and $x_2 \oplus x_1 \oplus x_0$ but this has no importance in our analysis.

Restricting the choice of a_6 to the (eight) nibbles for which $s = \bar{s}$, we can generate 2^{23} numbers of the form $\mu(m_i) = x \times \Gamma_{23}$ where x is the 8-byte number $s(a_6) \cdot s(a_5) \cdot a_6 \cdot a_5 \cdot s(a_4) \cdot s(a_3) \cdot a_4 \cdot a_3 \cdot s(a_2) \cdot s(a_1) \cdot a_2 \cdot a_1 \cdot \mathtt{2266}_{16}$ and :

$$\Gamma_{23} = \sum_{i=0}^{\gamma/32-1} 2^{64i}$$

Section 3 could thus apply (treat Γ_{23} as an extra p_i) as soon as the expectation of p_k-smooth x-values reaches $k+1$:

$$k + 1 \sim 2^{23} \times \rho\left(\frac{64}{\log_2(k \ln k)}\right) \tag{3}$$

Using $k = 3000$ we forged thousands of 1024-bit $\mathcal{F}$-signatures in less than a day on a Pentium-PC (an example is given in appendix C). The attack is applicable to any $(64 \times c + 1)$-bit modulus and its complexity is independent of $c \in \mathsf{IN}$ (once computed, the *same* x-strings work with any such n).

k	# of p_k-smooth x-values (amongst 2^{23})	forgeries
345	346	1
500	799	298
1000	3203	2202
1500	6198	4697
2000	9344	7343
2500	12555	10054
3000	15830	12829

Table 1. Experimental $\mathcal{F}$-forgeries for 64-bit x-values, prime e.

The attack is equally applicable to 32, 48, 80, 96 or 112-bit x-strings (which yield 7, 15, 31, 39 and 47-bit plaintext spaces); a combined attack, mixing x-strings of different types is also possible (this has the drawback of adding the unknowns $\Gamma_7, \Gamma_{15}, \ldots$ but improves the probability of finding p_k-smooth x-strings). Long *plain-English* messages ending by the letter `f` can be forged using a more technical approach sketched in appendix B ($\mathtt{66}_{16}$ represents the ASCII character `f`). Note, as a mere curiosity, a slight ($\cong 11\%$) experimental deviation from formula (3) due to the non-uniform distribution of the x-strings (which most and least significant bits can never be long sequences of zeroes). Finally, since the powers of 2 and Γ_{23} are identical, one can use k chosen messages instead of $k+1$, packing $p_1 = 2$ and $p_{k+1} = \Gamma_{23}$ into the updated unknown $p_1 = 2\Gamma_{23}$.

Non-impact on ISO 9796-1 **:** *The authors could not extend the attack to* ISO 9796-1 *and it would be wrong to state that* ISO 9796-1 *is broken.*

Note : When we first looked into the standard, we did not notice $\tilde{s}$ and we are grateful to Peter Landrock and Jørgen Brandt for drawing our attention to that. It appears from our discussions with ISO/JTC1/SC27 that $\tilde{s}$ (the alteration that codes the message-border) has also been introduced to prevent arithmetic operations on $\mu(m)$; further information on ISO 9796-1 and our attack on $\mathcal{F}$ will be soon posted on `http://www.iso.ch/jtc1/sc27`.

4.2 The Security of ISO 9796-2 Signatures

ISO 9796-2 is a generic padding standard allowing total or partial message recovery. Hash-functions of different sizes are acceptable and parameter L (in the standard k_h) is consequently a variable. Section 5, note 4 of [22] recommends $64 \leq L \leq 80$ for total recovery (typically an ISO 10118-2 [23]) and $128 \leq L \leq 160$ for partial recovery.

Partial Message Recovery. For simplicity, assume that N, L and the size of m are all multiples of eight and that the hash function is known to both parties. The message $m = m[1] \cdot m[2]$ is separated into two parts where $m[1]$ consists of the $N - L - 16$ most significant bits of m and $m[2]$ of all the remaining bits of m. The padding function is :

$$\mu(m) = \mathtt{6A}_{\mathtt{16}} \cdot m[1] \cdot \mathrm{HASH}(m) \cdot \mathtt{BC}_{\mathtt{16}}$$

and $m[2]$ is transmitted in clear.

Dividing $(\mathtt{6A}_{\mathtt{16}} + 1) \times 2^N$ by n we obtain :

$$(\mathtt{6A}_{\mathtt{16}} + 1) \times 2^N = i \times n + r \quad \text{with} \quad r < n < 2^N$$

$$n' = i \times n = \mathtt{6A}_{\mathtt{16}} \times 2^N + (2^N - r) = \mathtt{6A}_{\mathtt{16}} \cdot n'[1] \cdot n'[0]$$

where n' is $N + 7$ bits long and $n'[1]$ is $N - L - 16$ bits long.

Setting $m[1] = n'[1]$ we get :

$$t = i \times n - \mu(m) \times 2^8 = n'[0] - \mathrm{HASH}(m) \cdot \mathtt{BC00}_{\mathtt{16}}$$

where the size of t is less than $L + 16$ bits.

The forger can thus modify $m[2]$ (and therefore HASH(m)) until he gets a set of messages which t-values are p_k-smooth and express one such $\mu(m_\tau)$ as a multiplicative combination of the others.

Note that the attack is again independent of the size of n (forging 1024-bit signatures is *not* harder than forging 512-bit ones) but, unlike our $\mathcal{F}$-attack, forged messages are specific to a given n and can not be recycled when attacking different moduli.

To optimize efforts, $\mathcal{A}$ must use the k minimizing $C'_{L+16,k}$.

Although the optimal time complexities for $L = 160$ and $L = 128$ are lower than the birthday complexities of SHA and MD5 we consider that $L = 160$ implementations are still reasonably secure.

$L = k_h$	**optimal** $\log_2 k$	$\log_2$ **time**	$\log_2$ **space**
128	18	54	36
160	20	61	40

Table 2. Attacks on ISO 9796-2, small public exponent.

Total Message Recovery. Assuming again that the hash function is known to both parties, that N and L are multiples of eight and that the size of m is $N - L - 16$, function μ is :

$$\mu(m) = \texttt{4A}_{\texttt{16}} \cdot m \cdot \text{HASH}(m) \cdot \texttt{BC}_{\texttt{16}}$$

Let us separate $m = m[1] \cdot m[0]$ into two parts where $m[0]$ consists of the ℓ least significant bits of m and $m[1]$ of all the remaining bits of m and compute, as in the previous case, an i such that :

$$n' = i \times n = \texttt{4A}_{\texttt{16}} \cdot n'[1] \cdot n'[0]$$

where $n'[0]$ is $(L + \ell + 16)$-bits long and $n'[1] \cdot n'[0]$ is N-bits long.

Setting $m[1] = n'[1]$ we get :

$$t = i \times n - \mu(m) \times 2^8 = n'[0] - m[0] \cdot \text{HASH}(m) \cdot \texttt{BC00}_{\texttt{16}}$$

where the size of t is less than $L + \ell + 16$ bits.

$\mathcal{A}$ will thus modify $m[0]$ (and therefore HASH(m)) as needed and conclude the attack as in the partial recovery case. ℓ must be tuned to expect just enough p_k-smooth t-values with a reasonably high probability *i.e.* :

$$k \sim 2^{\ell} \times \rho\left(\frac{L + \ell + 16}{\log_2(k \ln k)}\right)$$

The complexities summarized in the following table (a few PC-weeks for $k_h = 64$) seem to suggest a revision of this standard.

$L = k_h$	**optimal** $\log_2 k$	$\log_2$ **time**	$\log_2$ **space**	ℓ
64	15	47	30	32
80	17	51	34	34

Table 2 (continued). Attacks on ISO 9796-2, small public exponent.

Note that our attack would have applied as well to :

$$\mu(m) = \texttt{4A}_{\texttt{16}} \cdot \text{HASH}(m) \cdot m \cdot \texttt{BC}_{\texttt{16}}$$

In which case take $n' = i \times n$ such that $n' \bmod 256 = \texttt{BC}_{\texttt{16}}$ and use m to replicate the least significant bits of n'; subtraction will then yield a moderate size integer times of a power of two.

An elegant protection against our attack is described in [13] (its security is basically comparable to that of PKCS #1 V2.0, discussed later on in this paper); a second efficient solution, suggested by Jean-Jacques Quisquater in the rump session of CRYPTO'97 is :

$$\mu(m) = \texttt{4A}_{\texttt{16}} \cdot (m \oplus \text{HASH}(m)) \cdot \text{HASH}(m) \cdot \texttt{BC}_{\texttt{16}}$$

4.3 Analyzing PKCS #1 v2.0, SSL-3.02 and ANSI X9.31

This section describes theoretical attacks on PKCS #1 v2.0, SSL-3.02 and ANSI X9.31 which are better than the birthday-paradox. Since our observations are not general (for they apply to moduli of the form $n = 2^k \pm c$) and more demanding than factorization, they *do not endanger current implementations of these standards*. It appears that $n = 2^k \pm c$ offers regular 1024-bit RSA security as far as c is not much smaller than 2^{500}, and square-free c-values as small as 400 bits may even be used [25]. In general ($n > 2^{512}$) such moduli appear to offer regular security as long as $\log_2(c) \cong \log_2(n)/2$ and c is square-free [26].

Although particular, $n = 2^k \pm c$ has been advocated by a number of cryptographers for it allows trial and error divisions to be avoided. For instance, the informative annex of ISO 9796-1 recommends "*...some forms of the modulus ($n = 2^k \pm c$) [that] simplify the modulo reduction and need less table storage.*". Note however, that even in our worst scenario, ISO 9796-1's particular form is still secure : for 1024-bit moduli, ISO 9796-1 recommends a 767-bit c whereas our attack will require a 400-bit c. The reader is referred to section 14.3.4 of [27] for further references on $n = 2^k \pm c$.

Assume that we are given a 1024-bit $n = 2^k - c$, where $\ell = \log_2(c) \cong 400$ and c is square-free; we start by analyzing SSL-3.02 where :

$$\mu(m) = \texttt{0001}_{16} \cdot \texttt{FFFF}_{16} \ldots \texttt{FFFF}_{16} \cdot \texttt{00}_{16} \cdot \mathrm{SHA}(m) \cdot \mathrm{MD5}(m)$$

$n - 2^{15} \times \mu(m)$ is an ℓ-bit number on which we conduct an ISO 9796-2-like attack which expected complexity is $C'_{\ell,k}$.

The characteristics of the attack are summarized in table 3 which should be compared to the birthday paradox (2^{144} time, negligible space) and the hardness of factorization ({time, space} denote the base-two logarithms of the time and space complexities of the attacks) :

$\log_2 n$	ℓ	**optimal** $\log_2 k$	**our attack**	**factorization**
606	303	28	{84, 56}	{68, 41}
640	320	29	{87, 58}	{70, 42}
768	384	33	{97, 66}	{75, 45}
1024	400	34	{99, 68}	{86, 50}
1024	512	39	{115, 78}	{86, 50}

Table 3. Estimates for SSL 3.02, small public exponent.

The phenomenon also scales-down to PKCS #1 v2.0 where :

$$\mu(m) = \texttt{0001}_{16} \cdot \texttt{FFFF}_{16} \ldots \texttt{FFFF}_{16} \cdot \texttt{00}_{16} \cdot c_{\mathrm{SHA}} \cdot \mathrm{SHA}(m)$$
$$\mu(m) = \texttt{0001}_{16} \cdot \texttt{FFFF}_{16} \ldots \texttt{FFFF}_{16} \cdot \texttt{00}_{16} \cdot c_{\mathrm{MD5}} \cdot \mathrm{MD5}(m)$$

$$c_{\mathrm{SHA}} = \texttt{3021300906052B0E03021A05000414}_{16}$$
$$c_{\mathrm{MD5}} = \texttt{3020300C06082A864886F70D020505000410}_{16}$$

and :

$\log_2 n$	ℓ	**optimal** $\log_2 k$	**our attack**	**factorization**
512	256	23	{77, 46}	{64, 39}
548	274	27	{80, 54}	{66, 40}

Table 4. Estimates for PKCS #1 V2.0 and ANSI X9.31, small public exponent.

These figures appear roughly equivalent to a birthday-attack on SHA, even for rather small (550-bit) moduli. Note that the attack applies to $n = 2^k + c$ by computing $n - 2^{14} \times \mu(m)$.

Note : In a recent correspondence, Burt Kaliski informed us that Ron Rivest developed in 1991 a forgery strategy which is a simple case of the one described in this paper; the design of PKCS #1 V1.5 took this into account, but Ron's observation was never published. Further information on our attack will appear soon in an RSA bulletin `http://www.rsa.com/rsalabs/`.

A similar analysis where the prescribed moduli begin by $\texttt{6BBBBB}\ldots_{\texttt{16}}$ is applicable to ANSI X9.31 (yielding exactly the same complexities as for PKCS #1 V2.0) where :

$$\mu(m) = \texttt{6B}_{\texttt{16}} \cdot \texttt{BBBB}_{\texttt{16}} \ldots \texttt{BBBB}_{\texttt{16}} \cdot \texttt{BA}_{\texttt{16}} \cdot \mathrm{SHA}(m) \cdot \texttt{33CC}_{\texttt{16}}$$

ANSI X9.31 recommends to avoid $n = 2^k \pm c$. If one strictly follows the standard $n = \texttt{6BBBBB}\ldots_{\texttt{16}}$ can not occur (the standard requires a bit length which is a multiple of eight) but one could in theory work with $2\mu(m)$ instead of $\mu(m)$.

Finally, we will consider a theoretical setting in which an authority certifies moduli generated by users who wish to join a network; naturally, users never reveal their secret keys but using storage optimizations as a pretext, the authority implements an ID-based scheme where different *random looking* bits (registration ID, account numbers *etc*) are forced into the most significant bits of each n [26]. Users generate moduli having the prescribed patterns they receive.

If the authority can find two small constants $\{u, v\}$ such that :

$$\log_2(u \times n - v \times \mu(m)) \cong \eta \quad \text{for a moderate } \eta \tag{4}$$

then our attack would extend to moduli which are not necessarily of the form $2^k \pm c$. To do so, oversimplify the setting to $\mu(m) = (2^w - 1) \cdot f(m)$ and $n = n[1] \cdot n[0]$ where $n[0]$ has the size of $f(m)$ and substitute these definitions in equation (4) :

$$\log_2(u \times (n[1] \cdot n[0]) - v \times ((2^w - 1) \cdot f(m))) \cong \eta$$

since the authority has no control over $f(m)$, the best thing to do would be to request that $u \times n[1] = v \times (2^w - 1)$ which results in an $\eta \cong \log_2(f(m)) + \log_2(\max\{u, v\})$.

The authority can thus prescribe moduli which most significant bits are $v_i \times (2^w - 1)/u_i$ where u_i are moderate-size factors of $2^w - 1$. Such factors look random and should not raise the user's suspicion.

We can therefore conclude that although practically safe, the use of authority-specified moduli in fixed-pattern padding contexts might be an interesting theoretical playground.

5 Conclusion and Further Research

Although the analysis presented in this paper indicates a weakness in ISO 9796-2 when $k_h \cong 64$, products using this standard should not be systematically withdrawn; a few product analyzes reveal that system-level specifications (message contents, insufficient access to $\mathcal{S}$ *etc.*) frequently make real-life attacks harder than expected.

It seems reasonable (although we can not base our belief on formal grounds) that good message recovery padding schemes should be usable for encryption as well; we motivate this recommendation by the functional similarity between RSA encryption and message recovery.

Full-domain-hash offers the best possible protection against our attack and we advocate its systematic use whenever possible. If impossible, it seems appropriate to link L and N since for a fixed L there is necessarily a point (birthday) above which increasing N will slow-down the legitimate parties without improving security.

We also recommend four research directions :

• An integer is $\{a, p_k\}$-semismooth [3] if each of its prime factors is smaller than a and all but one are smaller than p_k. A well known-strategy (called the *large prime variant*) consists of searching, using the birthday paradox, $\{a, p_k\}$-semismooth $\{\mu(x), \mu(y)\}$ pairs having an identical large prime factor (*e.g.* 80-bits long); the *ratio* $\mu(x)/\mu(y) \bmod n$ can then be used as one p_k-smooth input in the Gaussian elimination.

• It might be interesting to find out if our $\mathcal{F}$-attack could handle $\tilde{s}$ by using a different Γ :

$$\Gamma = \Delta \cdot 000000000001_{16} \cdot 000000000001_{16} \cdots 000000000001_{16}$$

In which case x-values should end by the pattern 2266_{16}, be p_k-smooth and such that $x' = x/\Delta$ is a valid message header. Note that different Δ-values might be mixed in the same attack, using a large prime variant where the different Γ-values are eliminated by modular division.

• Although we have no specific instances for the moment, one could also try to combine our technique with [4] to speed-up forgery in specific situations.

• Finally, it appears that incomplete *ad-hoc* analyzes of hash-functions (building digests with u prescribed bits in less than 2^u operations) could be the source of new problems in badly designed padding schemes.

Acknowledgments

We are grateful to Arjen Lenstra, Pascal Paillier and Michael Tunstall for their helpful comments, the authors also thank Pascal Autissier, Christophe Clavier, Renato Menicocci and Phong N'guyen for their insights into several mathematical details. Last but not least, we express our recognition to Burt Kaliski, Bart Preneel and Jean-Jacques Quisquater for their help.

References

1. L. Adleman, *A subexponential algorithm for the discrete logarithm problem with applications to cryptography*, Proceedings of the IEEE 20-th Annual symposium on the foundations of computer science, pp. 55-60, 1979.
2. ANSI X9.31, *Digital signatures using reversible public-key cryptography for the financial services industry (rDSA)*, 1998.
3. E. Bach and R. Peralta, *Asymptotic semismoothness probabilities*, Mathematics of computation, vol. 65, no. 216, pp. 1701–1715, 1996.
4. O. Baudron and J. Stern, *To pad or not to pad : does formatting degrade security ?*, 1999 RSA Data Security Conference proceeding book, 1999.
5. M. Bellare and P. Rogaway, *Random oracles are practical : a paradigm for designing efficient protocols*, Proceedings of the first annual conference on computer and communication security, ACM, 1993.
6. M. Bellare and P. Rogaway, *The exact security of digital signatures : how to sign with RSA and Rabin*, Advances in cryptology EUROCRYPT'96, Springer-Verlag, Lectures notes in computer science 1070, pp. 399–416, 1996.
7. R. Brent, *An improved Monte Carlo factorization algorithm*, Nordisk Tidskrift för Informationsbehandling (BIT) vol. 20, pp. 176–184, 1980.
8. N. de Bruijn, *On the number of positive integers $\leq x$ and free of prime factors $\geq y$*, *Indagationes Mathematicae*, vol. 13, pp. 50–60, 1951. (*cf.* as well to part II, vol. 28, pp. 236–247, 1966.).
9. G. Davida, *Chosen signature cryptanalysis of the RSA (MIT) public-key cryptosystem*, TR-CS-82-2, Department of electrical engineering and computer science, University of Wisconsin, Milwaukee, 1982.
10. D. Denning, *Digital signatures with RSA and other public-key cryptosystems*, Communications of the ACM, vol. 27-4, pp. 388–392, 1984.
11. Y. Desmedt and A. Odlyzko. *A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes*, Advances in cryptology CRYPTO'85, Springer-Verlag, Lectures notes in computer science 218, pp. 516–522, 1986.
12. K. Dickman, *On the frequency of numbers containing prime factors of a certain relative magnitude*, Arkiv för matematik, astronomi och fysik, vol. 22A, no. 10, pp. 1–14, 1930.

13. DIN NI-17.4, *Specification of chipcard interface with digital signature application/function according to SigG and SigV*, version 1.0, 1998.

14. J. Dixon, *Asymptotically fast factorization of integers*, Mathematics of computation, vol. 36, no. 153, pp. 255–260, 1981.

15. J. Evertse and E. van Heyst, *Which new RSA-signatures can be computed from certain given RSA signatures ?*, Journal of cryptology vol. 5, no. 1, 41–52, 1992.

16. M. Girault, J.-F. Misarsky, *Selective forgery of RSA signatures using redundancy*, Advances in cryptology EUROCRYPT'97, Springer-Verlag, Lectures notes in computer science 1233, pp. 495–507, 1997.

17. J. Gordon, *How to forge RSA key certificates*, Electronic Letters, vol. 21, no. 9, April 25-th, 1985.

18. L. Guillou, J.-J. Quisquater, M. Walker, P. Landrock and C. Shaer, *Precautions taken against various attacks in* ISO/IEC DIS 9796, Advances in cryptology EUROCRYPT'90, Springer-Verlag, Lectures notes in computer science 473, pp. 465–473, 1991.

19. H. Halberstam, *On integers whose prime factors are small*, Proceedings of the London mathematical society, vol. 3, no. 21, pp. 102–107, 1970.

20. K. Hickman, *The SSL Protocol*, December 1995. Available electronically at : `http://www.netscape.com/newsref/std/ssl.html`

21. ISO/IEC 9796, *Information technology - Security techniques - Digital signature scheme giving message recovery, Part 1 : Mechanisms using redundancy*, 1999.

22. ISO/IEC 9796-2, *Information technology - Security techniques - Digital signature scheme giving message recovery, Part 2 : Mechanisms using a hash-function*, 1997.

23. ISO/IEC 10118-2, *Information technology - Security techniques - Hash-functions; Part 2 : Hash functions using an n-bit block-cipher algorithm*, 1994.

24. W. de Jonge and D. Chaum. *Attacks on some RSA signatures*, Advances in cryptology CRYPTO'85, Springer-Verlag, Lectures notes in computer science 218, pp. 18–27, 1986.

25. A. Lenstra, *Generating RSA moduli with a predetermined portion*, Advances in cryptology ASIACRYPT'98, Springer-Verlag, Lectures notes in computer science 1514, pp. 1–10, 1998.

26. A. Lenstra, *de auditu*, January 1999.

27. A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of applied cryptography*, CRC Press.

28. M. Michels, M. Stadler and H.-M. Sun, *On the security of some variants of the RSA signature scheme*, Computer security-ESORICS'98, Springer-Verlag, Lectures notes in computer science 1485, pp. 85–96, 1998.

29. J.-F. Misarsky, *A multiplicative attack using LLL algorithm on RSA signatures with redundancy*, Advances in cryptology CRYPTO'97, Springer-Verlag, Lectures notes in computer science 1294, pp. 221–234, 1997.

30. J.-F. Misarsky, *How (not) to design RSA signature schemes*, Public-key cryptography, Springer-Verlag, Lectures notes in computer science 1431, pp. 14–28, 1998.
31. National Institute of Standards and Technology, *Secure hash standard*, FIPS publication 180-1, April 1994.
32. J. Pollard, *Factoring with cubic integers*, The development of the number field sieve, Springer-Verlag, Lectures notes in computer science 1554, pp. 4–10, 1993.
33. C. Pomerance, *The quadratic sieve factoring algorithm*, Advances in cryptology EUROCRYPT'84, Springer-Verlag, Lectures notes in computer science 209, pp. 169–182, 1985.
34. R. Rivest, *RFC 1321 : The MD5 message-digest algorithm*, Internet activities board, April 1992.
35. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, vol. 21-2, pp. 120–126, 1978.
36. RSA Laboratories, PKCS #1 *: RSA cryptography specifications*, version 2.0, September 1998.
37. H. Williams, *A modification of the RSA public key encryption procedure*, IEEE TIT, vol. 26, pp. 726–729, 1980.

APPENDIX A

The following (redundant) look-up table lists ρ for the various smoothness and digest-size values concerned by this paper; $\rho(136/24)$, the probability that a 136-bit number has no prime factors larger than 2^{24} is $2^{-14.2}$:

$-\log_2 \rho \searrow$	**16**	**20**	**24**	**28**	**32**	**36**	**40**	**44**	**48**	**52**	**56**	**60**	**64**	**68**	**72**
32	1.7	0.9	0.5	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
48	4.4	2.7	1.7	1.1	0.8	0.5	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
64	7.7	5.0	3.4	2.4	1.7	1.2	0.9	0.7	0.5	0.3	0.2	0.0	0.0	0.0	0.0
80	11.5	7.7	5.4	3.9	2.9	2.2	1.7	1.3	1.0	0.8	0.6	0.5	0.4	0.3	0.2
96	15.6	10.7	7.7	5.7	4.4	3.4	2.7	2.1	1.7	1.4	1.1	0.9	0.8	0.6	0.5
112	20.1	13.9	10.2	7.7	5.9	4.7	3.8	3.1	2.5	2.1	1.7	1.4	1.2	1.0	0.8
128	24.9	17.4	12.8	9.8	7.7	6.1	5.0	4.1	3.4	2.8	2.4	2.0	1.7	1.4	1.2
136	27.4	19.2	14.2	10.9	8.6	6.9	5.6	4.6	3.9	3.2	2.8	2.3	2.0	1.7	1.5
144	29.9	21.1	15.6	12.0	9.5	7.7	6.3	5.2	4.4	3.7	3.1	2.7	2.3	2.0	1.7
152	32.4	22.9	17.1	13.2	10.5	8.5	7.0	5.8	4.9	4.1	3.5	3.0	2.6	2.3	2.0
160	35.1	24.9	18.6	14.4	11.5	9.3	7.7	6.4	5.4	4.6	3.9	3.4	2.9	2.6	2.2
168	37.9	26.9	20.1	15.6	12.5	10.2	8.4	7.0	5.9	5.1	4.4	3.8	3.3	2.9	2.5
176	40.6	28.9	21.7	16.9	13.5	11.0	9.1	7.7	6.5	5.6	4.8	4.2	3.6	3.2	2.8
400	129.	95.2	73.9	59.2	49.0	41.5	35.1	30.2	26.5	23.1	20.8	18.5	16.7	15.1	13.7
512	179.	133	104	84.0	69.8	59.0	50.8	44.0	38.8	34.1	30.6	27.2	24.9	22.5	20.6

The table uses the exact formula (section 2) for $t \leq 10$ and de Bruijn's approximation [8] for $t > 10$:

$$\rho(t) \cong (2\pi t)^{-1/2} \exp\left(\gamma - t\zeta + \int_0^{\zeta} \frac{e^s - 1}{s} ds\right)$$

where ζ is the positive solution of $e^{\zeta} - 1 = t\zeta$ and γ is Euler's constant.

APPENDIX B

The attack's time-consuming part is the exhaustive-search of k appropriate x-strings; therefore, when one wants the x-strings to be 256-bits long, the increase in k makes the attack impractical.

To overcome this problem, we suggest the following : as a first step, collect the signatures corresponding to moderate-size p_k-smooth x-strings (which are relatively easy to find) and extract from their appropriate multiplicative combinations the e-th roots of the k first primes. Then, exhaustive-search two plain-English 128-bit messages $\{m, m'\}$ ending by the letter f such that $\mu(m)/\Gamma$ and $\mu(m')/\Gamma$ are both p_k-smooth, with :

$$\Gamma = 2^{256(c-1)} + \ldots + 2^{256} + 1$$

for a $(256 \times c + 1)$-bit modulus. Since we only need two such numbers, the overall workload is very tolerable. Next, submit m to $\mathcal{S}$ and divide its signature by the e-th roots of its small prime factors to recover $\Gamma^d \bmod n$. Using $\Gamma^d \bmod n$ and the e-th roots of the k first primes we can now forge, by multiplication, the signature of m'.

APPENDIX C

This appendix contains an $\mathcal{F}$ forgery that works with any 1025-bit modulus; to fit into the appendix, the example was computed for $e = 3$ but forgeries for other public exponents are as easy to obtain.

step 1 : Select any 1025-bit RSA modulus, generate $d = 3^{-1} \bmod \phi(n)$, let $\mu = \mathcal{F}$ and form the 180 messages :

$$m_i = (256 \times \texttt{message}[i]_{16} + 102) \times \sum_{j=0}^{11} 2^{32j}$$

where `message[`i`]` denotes the elements of the following table :

00014E	008C87	00D1E8	01364B	0194D8	01C764	021864	03442F	0399FB	048D9E	073284	0863DE	09CCE8
0A132E	0A2143	0BD886	0C364A	0C368C	0C6BCF	0D3AC1	0D5C02	0EA131	0F3D68	0F9931	31826A	31BE81
31ED6B	31FCD0	320B25	32B659	332D04	3334D8	33EAFC	33EB1D	343B49	353D02	35454C	35A1A9	36189E
362C79	365174	3743AB	3765F6	37C1E2	3924AC	3998A8	3AF8A7	3B6900	3B9EEB	3BC1FF	3DE2DE	3E51BE
3E8191	3F49F3	3F69AC	4099D9	40BF29	41C36C	41D8C0	424EE8	435DB7	446DC1	4499CC	44AA20	44EE53
4510E8	459041	45A464	45AA03	460B80	4771E7	486B6A	499D40	4A5CF8	4AC449	4ADA0A	4B87A8	4C06A1
4C5C17	4D4685	4E39EA	4EB6B6	4F8464	716729	71C7D3	71FA22	722209	72DBF1	7619AB	765082	767C39
76885C	78F5F3	79E412	79FAD6	7CD0ED	7D0ABA	7DBA1D	7DE6A5	7E06A2	7EA5F2	7EC1ED	7EEC78	90BB4B
90DE38	9139D7	934C2C	9366C5	941809	941BFB	947EB4	94DB29	952D45	9745BD	978897	97A589	9827AF
984FAC	9A193D	9A83E2	9B74E3	9BEAE9	9C704F	9DBA98	9F9337	A00D15	A02E3D	A10370	A429A6	A4DADD
A4F689	A5485D	A6D728	A76B0F	A7B249	A87DF3	A95438	A96AA4	AB1A82	AD06A8	AEA0D0	AEB113	D076C5
D13F0E	D18262	D1B0A7	D35504	D3D9D4	D3DEE4	D4F71B	D91C0B	D96865	DA3F44	DB76A8	DE2528	DE31DD
DE46B8	DE687D	DEB8C8	DF24C3	DFDFCF	DFF19A	E12FAA	E1DD15	E27EC1	E39C56	E40007	E58CC8	E63CE0
E6596C	E7831E	E796FB	E7E80C	E85927	E89243	E912B4	E9BFFF	EA0DFC	EACF65	EB29FA		

step 2 : construct the message $m' = \texttt{EE7E8E66}_{16} \times \sum_{j=0}^{11} 2^{32j}$ and obtain from the signer the 180 signatures $s_i = \mu(m_i)^d \bmod n$.

step 3 : the signature of m' is :

$$\mu(m')^d = \prod_{i=0}^{345} p_i^{-\texttt{gamma}[i]} \prod_{i=1}^{180} s_i^{\texttt{beta}[i]} \bmod n$$

where p_i denotes the i-th prime (with $p_0 = \Gamma_{23}$) and `beta[`i`]` denotes the elements of the following table :

1	2	1	2	2	2	2	1	2	2	2	1	1	2	2	2	1	1	2	1	2	2	2	2	2	1	1	2	1	1	2	1	1	2	1	1
1	1	1	1	1	1	2	1	1	1	1	2	1	1	1	1	2	2	1	1	2	1	2	1	1	2	2	1	1	1	1	2	1	1	2	1
1	1	1	1	2	2	1	2	1	2	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	2
1	1	1	2	2	2	2	1	2	2	1	1	2	2	2	2	1	1	2	1	2	2	2	2	1	1	1	2	1	1	2	1	1	1	1	2
2	1	1	1	1	2	2	1	2	2	1	1	2	2	2	1	2	1	2	2	2	1	2	1	2	2	1	2	1	2	2	1	1	2	1	1

`gamma[`i`]` represents the hexadecimal values :

57	57	68	33	27	18	16	13	10	0F	0E	0B	09	09	0D	05	0B	07	04	08	07	07	07	09	0A	03	07
04	05	05	03	04	03	01	02	03	04	03	01	03	03	03	02	06	03	03	04	06	02	04	04	02	02	03
02	04	04	03	04	01	04	03	02	03	02	01	02	02	01	03	01	01	01	01	03	03	01	03	02	02	01
04	02	04	02	02	01	02	01	01	01	03	03	01	02	01	01	00	03	02	03	01	01	02	01	02	02	03
03	04	03	03	02	03	01	02	03	02	01	03	02	02	01	01	00	02	01	01	03	01	01	01	01	01	02
00	02	00	00	01	02	01	01	01	00	01	01	00	01	01	02	02	01	01	01	00	01	00	01	01	04	02
02	02	01	02	02	01	02	01	02	00	01	00	02	01	02	02	00	01	02	01	01	01	02	01	01	01	02
01	00	01	01	00	00	01	02	00	01	00	01	01	00	01	00	01	02	02	01	01	02	00	00	02	01	02
02	01	00	00	01	00	01	00	01	00	02	00	00	00	01	01	00	00	01	01	00	00	00	01	00	00	00
00	00	00	01	01	00	00	01	02	01	01	01	00	01	02	01	01	01	02	00	00	00	01	01	00	01	00
00	00	02	02	01	00	01	02	00	01	00	01	02	00	01	00	00	01	00	01	01	01	00	01	01	00	01
01	01	01	00	00	01	01	00	00	01	01	00	01	01	00	00	01	00	00	00	01	01	02	02	01	01	00
00	01	02	01	02	00	01	01	00	01	00	00	00	00	00	00	01	00	00	01	02	01					