ISO/IEC JTC 1/SC 27 N 2329

ISO/IEC JTC 1/SC 27/WG 2 N 429

REPLACES: N

ISO/IEC JTC 1/SC 27

Information technology - Security techniques

Secretariat: DIN, Germany

DOC TYPE:	Expert Contribution
TITLE:	An expert contribution to project 1.27.07 (9796): " A new signature forgery strategy"
SOURCE:	J. –S. Coron D. Naccache J. P. Stern
DATE:	1999-05-05
PROJECT:	1.27.07
STATUS:	This document is being circulated for information and discussion by SC 27/WG 2 experts at their next WG 2 meeting.
ACTION ID:	ACT
DUE DATE:	
DISTRIBUTION:	P, O, L Members L. Rajchel, JTC 1 Secretariat K. Brannon, ITTF W. Fumy, SC 27 Chairman M. De Soete, T. Humphreys, S. Knapskog, WG-Conveners
MEDIUM:	Server
NO. OF PAGES:	22

Secretariat ISO/IEC JTC 1/SC 27 - DIN Deutsches Institut für Normung e. V., Burggrafenstr. 6, 10772 Berlin, Germany
Telephone: + 49 30 2601-2652; Facsimile:+ 49 30 2601-1723; e-mail: passia@ni.din.de

ISO/IEC JTC1/SC27

ISO/IEC JTC 1/SC27/WG2 **N429**
April 19, 1999

ISO International Organisation for Standardisation
IEC International Electrotechnical Commission

JTC 1 Information Technology
SC27 Security Techniques

WG2 Security Techniques and Mechanisms

TITLE: A new signature forgery strategy

SOURCE: Expets contribution

PROJECT:

STATUS: For information and discussion by WG 2 experts at their next WG 2 meeting.

A NEW SIGNATURE FORGERY STRATEGY

APPLICABLE TO ISO-9796-1/2, ECASH™, PKCS#1 V2.0, ANSI X9.31, SSL-3.02

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.

When applied to ISO 9796-1, ISO 9796-2, PKCS #1 V2.0, ECASH™, ANSI X9.31 and SSL-3.02, our analysis reveals the existence of weaknesses which make forgery substantially easier than frontal attacks on hash-then-decrypt RSA (collisions and factoring).

It took less than a day to forge thousands of 1024-bit ISO 9796-1 RSA signatures on a powerful PC. ISO 9796-2 signatures with $64 \leq k_h \leq 80$ (k_h is the digest-size recommended in §5 note 4 of ISO 9796-2) could be forged in a few weeks in similar experimental conditions. In SSL-3.02, we show that specific bit-patterns in n (frequently motivated by computational efficiency or storage optimization) might degrade security if specified malevolently. Finally, ECASH™, PKCS #1 V2.0 and ANSI X9.31 could offer less resistance than expected in some specific situations.

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 [1, 24, 26, 27, 41] highlight the need to challenge claims that such standards eradicate RSA's multiplicative properties.

Exponentiation is homomorphic and RSA-based protocols are traditionally protected against chosen-plaintext forgeries [13, 15, 40] 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 ([20, 29] 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 [3, 4] are provably secure in the random oracle model.

Unbroken padding formats are therefore challenging targets in a rarely visited hunting-area. The contribution of this paper is that the complexity of forging message-signature couples is frequently *much* lower than that of breaking $\text{RSA} \circ \mu$ by frontal attacks (factoring and collision-search).

The strategy introduced in this article does not brave the *invia* of RSA's traditional security assumptions; instead, it seeks for Naccache-Stern-like shortcuts [36] using the expected smoothness of moderate-size integers.

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

Previous work : Misarsky's PKC'98 invited survey [35] is probably the most complete and the best documented reference on multiplicative RSA forgeries. Davida's observation [13] is the focal point of most RSA forgery techniques. [20, 29] forge signatures that are similar to PKCS #1 V2.0 but do not to produce their necessary SHA/MD5 digests [37, 39]. [18, 19] analyze the security of RSA signatures in an interactive context. Michels *et al.* [33] 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. Bleichenbacher [5] exhibited a powerful attack on PKCS #1 encryption; this attack is however completely different from the ones presented here. Finally, Baudron and Stern [2] apply lattice reduction to analyze the security of $\text{RSA} \circ \mu$ in a security-proof perspective.

2 The attack

Let $\{n, e\}$ be an RSA public key and d be the corresponding secret key. Although in this paper μ will alternatively denote ISO 9796-1, ISO 9796-2, ECASH™, PKCS #1 V2.0, ANSI X9.31 or SSL-3.02 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); once understood, the attack will be progressively 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 smooth $\mu(m_i)$-values, then he could look for a message m' such that $\mu(m')$ has no bigger factors than p_k (the k-th prime) and construct $\mu(m')^d \bmod n$ as a multiplicative combination of the signatures of the chosen plaintexts $m_1, \ldots, m_\tau$. 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 [16, 17, 23] 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 :

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

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

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 straightforward byproduct of $\rho(t)$ and the *ratio* between the running times of the attacker's hashing and factorization algorithms (parallelization is also a possible option).

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 :

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

this is motivated 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 $L\sqrt{p_k}$:

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

As illustrated in appendix B, the attacker can optimize his resources by operating at a k where $C_{L,k}$ is minimal.

3 Finding homomorphic dependencies

As explained in the previous sections, we start by submitting the chosen messages $m_1, \ldots, m_\tau$ to $\mathcal{S}$. All selected 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 :

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

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

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

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

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

From equation (1) we get :

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

and the signature of m_τ can be computed from the other signatures by :

$$\mu(m_\tau)^d = \big(\prod_{j=1}^{k} p_j^{\gamma_j}\big)^{-1} \times \prod_{i=1}^{\tau-1} \big(\mu(m_i)^d\big)^{\beta_i} \bmod n$$

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 of the others.

• When e is the r-th power of a prime p, the set of vectors with k coordinates modulo e is an additive group of order p^{kr}. Let Δ_ℓ be the sub-group of order p^{a_ℓ} spanned by the set $\{\mathbf{V}_1, \ldots, \mathbf{V}_\ell\}$. $(a_\ell)_{\ell \geq 1}$ is a non-decreasing sequence of integers with $1 \leq a_\ell \leq kr$ for $\ell \geq 1$. Thus there exists an integer ℓ with $1 \leq \ell \leq kr$ for which $a_\ell = a_{\ell+1}$ and $\mathbf{V}_{\ell+1}$ can be expressed as a linear combination of the vectors $\mathbf{V}_1, \ldots, \mathbf{V}_\ell$. We therefore see that in this case $\tau = kr + 1$ vectors are sufficient to ensure that (at least) one signature can be expressed as a multiplicative combination of the others.

• The previous argument can be extended, *mutatis mutandis*, to the most general case :

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

where it appears that :

$$\tau = 1 + k \sum_{i=1}^{\omega} r_i = \mathcal{O}(k \log e)$$

vectors are sufficient to guarantee that (at least) one vector is a linear combination of the others.

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

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

which presents again a local minimum depicted in Appendix C.

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

4 Attacking the different standards

4.1 Forging ISO/IEC-9796-1 signatures

ISO/IEC-9796-1 [26] 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 ℓ bits to ℓ bits). ISO 9796-1 is not hashing-based and there are apparently no attacks [20, 22] other than factoring on this scheme ([35] : "...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.

Define by $a \cdot b$ the concatenation of a and b, let m_i be the nibbles of m and denote by $s(x)$ a substitution table (*cf.* to appendix G) such that $s(\mathtt{6}_{16}) = \mathtt{2}_{16}$:

$$\begin{array}{rl} \mu(m) = & \bar{s}(m_{\ell-1}) \cdot s(m_{\ell-2}) \cdot m_{\ell-1} \cdot m_{\ell-2} \cdot \\ & s(m_{\ell-3}) \cdot s(m_{\ell-4}) \cdot m_{\ell-3} \cdot m_{\ell-4} \cdot \\ & \ldots \\ & s(m_3) \quad \cdot s(m_2) \quad \cdot m_3 \quad \cdot m_2 \quad \cdot \\ & s(m_1) \quad \cdot s(m_0) \quad \cdot m_0 \quad \cdot \mathtt{6}_{16} \end{array}$$

where $\bar{s}(x)$ forces the most significant bit in $s(x)$ to 1.

Let a_j denote nibbles and consider messages of the form :

$$\begin{array}{rl} m_i = & a_6 \cdot a_5 \cdot a_4 \cdot a_3 \cdot a_2 \cdot a_1 \cdot \mathtt{66}_{16} \cdot \\ & a_6 \cdot a_5 \cdot a_4 \cdot a_3 \cdot a_2 \cdot a_1 \cdot \mathtt{66}_{16} \cdot \\ & \ldots \\ & a_6 \cdot a_5 \cdot a_4 \cdot a_3 \cdot a_2 \cdot a_1 \cdot \mathtt{66}_{16} \end{array}$$

for which :

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

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 can now plainly apply (treat Γ_{23} as an extra p_i) and forge signatures provided that :

$$k + 1 \ll 2^{23} \times \rho(64/\log_2(p_k)) \tag{2}$$

Using $k = 3000$ we forged thousands of 1024-bit signatures in less than a day on a 200 MHz Pentium-PC (an example is given in Appendix D). The complexity of our attack (summarized in table 1) is independent of integer constants multiplying 1024 : it is *not* harder to forge 1024-bit signatures than it is to forge 2048-bit ones; nor does the attack depend on the precise value of n (once computed, the *same* messages work with $(1024 \times c + 1)$-bit n for any $c \in \mathbb{N}$).

k	**chosen messages**	**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 forgeries of 1024-bit ISO 9796-1 signatures.

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 I (66_{16} represents the ASCII character f). Note, as a mere curiosity, a slight ($\cong 11\%$) experimental deviation from formula 2 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 $p_1 = 2\Gamma_{23}$.

The above indicates that a modification of ISO 9796-1 is necessary.

4.2 The security of ISO 9796-2 signatures for $128 \leq L \leq 160$

ISO 9796-2 is a generic standard where hash-functions of different sizes are possible; parameter L (re-named k_h) is consequently a variable [27].

The standard specifies the following padding algorithm : 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 :

$$\mu(m) = 6A_{16} \cdot m[1] \cdot \mathrm{HASH}(m) \cdot BC_{16}$$

and $m[2]$ is transmitted in clear. Section 5, note 4 of the standard recommends $64 \leq L \leq 80$ when there is no $m[2]$ (*i.e.* m is entirely included in $\mu(m)$) and $128 \leq L \leq 160$ in the general case where a part of m is transmitted in clear.

To keep the analysis simple, we will first analyze a setting where $m[2]$ exists and $L = 160$ (SHA); once understood, the attack will be generalized to $128 \leq L \leq 160$; the case $64 \leq L \leq 80$ will be broken separately in the next section.

Note that the pattern $\mathtt{6A}_{16} = \mathtt{01101010}_{02}$ is sufficient to insure that $\mu(m) < n$ when the leading nibble of n is one of the digits : $\mathtt{7}_{16}$,...,$\mathtt{F}_{16}$. The attack assumes that the most significant byte of n is $\mathtt{D4}_{16}$ or $\mathtt{D5}_{16}$ (which happens with probability $1/64 \cong 1.5\%$); when the most significant byte of n is not $\mathtt{D4}_{16}$ or $\mathtt{D5}_{16}$, the attacker can simply look for the smallest i such that the leading bits of $n' = i \times n$ are $\mathtt{1101010}_{02}$ and apply the attack to n'; one can show that the expected size of i is $\cong 6.8$ bits.

Let us represent such moduli as binary strings $n = \mathtt{1101010}_{02} \cdot n[1] \cdot n[0]$ where $n[1]$ is an $(N - 176)$-bit pattern and $n[0]$ is 169-bit long.

Form the message $m = u \cdot n[1]$ and the string :

$$\sigma = \mathtt{1101010}_{02} \cdot n[1] \cdot \mathrm{SHA}(m) \cdot \mathtt{178}_{16}$$

since n and σ are nearly identical, we get :

$$t = n - \sigma = n[0] - \mathrm{SHA}(m) \cdot \mathtt{178}_{16} \cong 2^{168}$$

Referring to appendix A, t is 2^{16}-smooth with probability $\rho(168/16) \cong 2^{-40}$. The forger will thus modify u (and therefore $\mathrm{SHA}(m)$) until he hits a 2^{16}-smooth t and, using Gaussian elimination, compute $\sqrt[e]{t}$ by multiplication. This is done by submitting to $\mathcal{S}$ queries for which $n - 2\mu(m) \cong 2^{168}$ is 2^{16}-smooth, and adapting the formulae of section 3 accordingly; the attack was confirmed by simplified simulations using a computational short-cut.

The attack is again independent of the size of n (RSA's complexity barrier has thus been neutralized : forging 4096-bit signatures is *not* harder than forging 512-bit ones) but, unlike our ISO 9796-1 attack, forged messages are specific to a given n and can not be recycled when attacking different moduli (forgeries could however be recycled if one generates n as in section 2.2 of [30]).

To optimize efforts, $\mathcal{A}$ must use the k minimizing :

$$\frac{(L+8)k\sqrt{k \ln k}}{\rho((L+8)/\log_2(k \ln k))}$$

the optimal time and space complexities for $L = 160$ and $L = 128$ are much lower than the birthday complexities of SHA and MD5 and seem well-adapted to an Internet-distributed attack.

$L = k_h$	**optimal** $\log_2 k$	$\log_2$ **time**	$\log_2$ **space**
128	16	51	32
160	20	60	40

Table 2. Attacks on ISO 9796-2.

4.3 Forging ISO 9796-2 signatures for $64 \leq L \leq 80$

When the whole message is embedded in $\mu(m)$, we must sacrifice some smoothness and use a few message bits as a counter. In the typical setting where f is a 64-bit ISO 9797 DES-MAC [28] and :

$$\mu(m) = \texttt{01101010}_{\texttt{02}} \cdot m \cdot f(m) \cdot \texttt{BC}_{\texttt{16}}$$

one can forge signatures in a few PC-weeks as follows : form the 2^{32} messages $m_w = n[1] \cdot w$ (represent w as 32-bit string) and apply section 3 to the subset of messages for which $n - 2\mu(m_w) \cong 2^{L+40}$ is p_k-smooth (the 40 in the exponent is the sum of the 8 bits of $\texttt{BC}_{\texttt{16}}$ and the 32 bits of w).

The complexities summarized in the following table look alarming enough to suggest a modification of this standard.

$L = k_h$	**optimal** $\log_2 k$	$\log_2$ **time**	$\log_2$ **space**
64	14	45	28
80	16	49	32

Table 2 (continued). Attacks on ISO 9796-2.

4.4 The security of ECASH™ signatures

ECASH™ is a software-based payment system providing anonymity for the payer. ECASH™ can send PC-to-PC payments over the Internet and has already been adopted by several banks in Norway, Austria, Germany, Switzerland, Finland and Australia [42].

In ECASH™, m forms the less significant bits of $\mu(m)$; to some extent, ECASH™ is a reversed ISO 9796-2 where :

$$y_0 = m \quad \text{and} \quad y_{i+1} = \text{SHA}(y_i) \cdot y_i$$

and the padded message $\mu(m) = y_t$ is truncated so that $\mu(m) < n$.

In the available specifications (*op. cit.*), m is restricted to 160-bits. Although for this message-size μ *is very secure*, security degrades when m is $\cong N - 160$ bits long :

$$\mu(m) = \text{SHA}(m) \cdot m$$

To attack this instance, form the $(N - 160)$-bit messages :

$$m_w = w \cdot (n \bmod 2^{N-160-\ell})$$

where w is an ℓ-bit counter and :

$$\frac{n - \mu(m_w)}{2^{N-160-\ell}} \cong 2^{160+\ell}$$

the rest of the attack is similar to the one on ISO 9796-2.

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

This section describes attacks on PKCS #1 v2.0, SSL-3.02 and ANSI X9.31 which are better than the birthday-paradox. Since our attacks are not general (for they apply to moduli of the form $n = 2^k \pm c$), they *do not endanger current implementations of these standards* but have the sole (yet important) merit of showing that the concerned schemes open the door to attacks which would have been impossible otherwise. Note however, 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 [30]. In general ($n > 2^{512}$) such moduli appear to offer regular security as long as $c \cong \sqrt{n}$ and c is square-free [31].

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 [that] simplify the modulo reduction and need less table storage :*

$$\begin{array}{llll} n = 2^{64x} - c & \textit{of length} & k = 64x & \textit{bits} \\ n = 2^{64x} + c & \textit{of length} & k = 64x + 1 & \textit{bits} \end{array}$$

where $1 \leq y \leq 2x$ *and* $c \leq 2^{64x-8y} \leq 2c$."

The reader is referred to section 14.3.4 of [32] and [11, 43, 44, 45] for further references.

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

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

Subtracting $n - 2^{15} \times \mu(m)$ we get a 400-bit number t and conduct an ISO 9796-2-like attack which expected complexity is :

$$\frac{400k\sqrt{k \ln k}}{\rho(400/\log_2(k \ln k))}$$

The characteristics of the attack are summarized in table 3 which should be compared to the birthday paradox's {2^{144} time, 2^{144} space} and the hardness of factorization (Appendix E).

$\log_2 n$	$\log_2 c$	**optimal** $\log_2 k$	$\log_2$ **time**	$\log_2$ **space**
606	303	28	84	56
640	320	29	87	58
768	384	33	97	66
1024	400	34	99	68
1024	512	39	115	78

Table 3. Attacks on SSL 3.02.

The attack also scales-down to PKCS #1 V2.0 (see appendix F) where :

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

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

and :

$\log_2 n$	$\log_2 c$	**optimal** $\log_2 k$	$\log_2$ **time**	$\log_2$ **space**
512	256	23	77	46
548	274	27	80	54

Table 4. Attacks on PKCS #1 V2.0 and ANSI X9.31.

Although satisfactory, these figures do not appear much better than a birthday-attack on SHA, even for rather small (550-bit) moduli.

Note that the attack described in this section extends to $n = 2^k + c$ as well (multiply $\mu(m)$ by 2^{14} to cause a borrow chain-reaction during the subtraction, which yields again a small t).

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

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

Since ANSI X9.31 already recommends to avoid $n = 2^k \pm c$, it might be appropriate to recommend avoiding $n = \texttt{6BBBBB}\ldots_{16}$ as well.

We will now consider a theoretical setting in which an authority certifies PKCS #1 V2.0 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 [31, 43]. Users generate moduli having the prescribed patterns they receive.

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

$$u \times n - v \times \mu(m) \cong 2^{\eta} \quad \text{for a moderate } \eta. \tag{3}$$

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 $n = n[1] \cdot n[0]$ and $\mu(m) = (2^k - 1) \cdot f(m)$ and substitute these definitions in equation 3 :

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

since the authority has no control over $f(m)$, the best thing to do would be to request that $u \times n[1] = v \times (2^k - 1)$ 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 $(2^k - 1)/u_i$ where u_i are moderate-size factors of $2^k - 1$. Such factors [8] look innocuous

and should not raise suspicion as illustrated in the following example where $u = 19995773632843536676957 7$:

$$\frac{2^{253}-1}{u} = \texttt{C1781158CEFC1F6F33E8D8F07070A914443FAC95DF67}_{16}$$

We can therefore conclude that although tolerable, the use of authority-specified moduli in PKCS #1 V2.0, ANSI X9.31 and SSL-3.02 might be a questionable practice. This confirms Bellare and Rogaway's intuition [4] :

"...We draw attention to the fact that the security of hash-then-decrypt signature depends very much on how exactly one implements hash. In particular, it is important to recognize that the security of a signature scheme like PKCS *can't be justified given (only) that RSA is trapdoor one-way, even under the assumption that hash function H is ideal. (The reason is that the set of points* PKCS(m) : $\{m \in \{0,1\}^*\}$ *has size at most* 2^{128} *and hence is very sparse, and a very structured, subset of* $\mathbb{Z}_n^*$*). We consider this to be a disadvantage. We stress that we don't know of any attack on this scheme. But we prefer, for such important primitives, to have some proof of security rather than just an absence of known attacks..."*

5 Conclusion and further research

Although the analysis presented in this paper indicates that ISO 9796-1 and ISO 9796-2 (for $64 \leq L \leq 128$) should be modified, products using these standards should not be systematically withdrawn; six analyzes of ISO 9796-based products reveal that system-level specifications (message contents, insufficient access to $\mathcal{S}$ *etc.*) frequently make real-life attacks harder than expected. In two other cases our attack had severe product security consequences. ISO 9796 users wishing to have their system specifications reviewed may contact the second author.

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 three research directions : although we have no specific instances for the moment, one could try to combine our technique with [2] to speed-up forgery in specific situations. When e is even, $\mathcal{A}$ could try to construct all-zero relations and factor n; this has more devastating consequences than the attacks described in this paper as $\mathcal{A}$ could later forge signatures of *any* chosen messages. 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.

6 Acknowledgements

We are grateful to Arjen Lenstra (CITIBANK), Pascal Paillier (GEMPLUS-ÉNST) and Michael Tunstall (GEMPLUS) for their helpful comments, the auhtors also thank Pascal Autissier (ÉNS), Christophe Clavier (GEMPLUS) and Phong N'guyen (ÉNS) for their insights into several mathematical details.

References

1. ANSI X9.31, *Digital signatures using reversible public-key cryptography for the financial services industry (rDSA)*, 1998.
2. O. Baudron and J. Stern, *To pad or not to pad : does formatting degrade security ?*, 1999 RSA Data Security Conference proceeding book, 1999.
3. 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.
4. 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.
5. D. Bleichenbacher, *Chosen ciphertext attacks against protocols based on the RSA encryption standard* PKCS #1, Advances in cryptology CRYPTO'98, Springer-Verlag, Lectures notes in computer science 1462, pp. 1–12, 1998.
6. D. Bleichenbacher, M. Joye, J.-J. Quisquater, *A new and optimal chosen-message attack on RSA-type cryptosystems*, Proceedings of ICICS 1997, Springer-Verlag, Lectures notes in computer science 1334, pp. 302–313, 1997.
7. R. Brent, *An improved Monte Carlo factorization algorithm*, Nordisk Tidskrift för Informationsbehandling (BIT) vol. 20, pp. 176–184, 1980.
8. J. Brillhart, D. Lehmer, J. Selfridge, B. Tucherman and S. Wagstaff, *Factorisations of $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ up to high powers*, AMS, Providence R.I., 1983.
9. 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.
10. N. de Bruijn, *On the number of positive integers $\leq x$ and free of prime factors $\geq y$, II*, *Indagationes Mathematicae*, vol. 28, pp. 236–247, 1966.
11. D. Coppersmith, *Finding a small root of a bivariate integer equation; factoring with high bits known*, Advances in cryptology EUROCRYPT'96, Springer-Verlag, Lectures notes in computer science 1070, pp. 178–189, 1996.
12. D. Coppersmith, *Analysis of* ISO/CCITT *document* X.509 *annex* D, IBM T.J. Watson research center *memorandum*, Yorktown Heights, NY, 10598, USA, June 11, 1989.

13. 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.
14. D. Denning, *Digital signatures with RSA and other public-key cryptosystems*, Communications of the ACM, vol. 27-4, pp. 388-392, 1984.
15. 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.
16. 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.
17. J. Dixon, *Asymptotically fast factorization of integers*, Mathematics of computation, vol. 36, no. 153, pp. 255–260, 1981.
18. 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.
19. J. Evertse and E. van Heyst, *Which new RSA-signatures can be computed from RSA signatures, obtained in a specific interactive protocol ?*, Advances in cryptology EUROCRYPT'92, Springer-Verlag, Lectures notes in computer science 658, pp. 378–389, 1993.
20. 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.
21. J. Gordon, *How to forge RSA key certificates*, Electronic Letters, vol. 21, no. 9, April 25-th, 1985.
22. 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.
23. H. Halberstam, *On integers whose prime factors are small*, Proceedings of the London mathematical society, vol. 3, no. 21, pp. 102–107, 1970.
24. K. Hickman, *The SSL Protocol*, December 1995. Available electronically at : `http://www.netscape.com/newsref/std/ssl.html`
25. ISO/IEC 8825-1, *Information technology - ASN.1 encoding rules : specifications of basic encoding rules (BER), canonical encoding rules (CER) and distinguished encoding rules (DER)*, 1998.
26. ISO/IEC 9796, *Information technology - Security techniques - Digital signature scheme giving message recovery, Part 1 : Mechanisms using redundancy*, 1991.
27. ISO/IEC 9796-2, *Information technology - Security techniques - Digital signature scheme giving message recovery, Part 2 : Mechanisms using a hash-function*, 1997.
28. ISO/IEC 9797, *Information technology - Security techniques - Data integrity mechanism using a cryptographic check function employing a block cipher algorithm*, 1994.

29. 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.
30. 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.
31. A. Lenstra, *de auditu*, January 1999.
32. A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of applied cryptography*, CRC Press.
33. 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.
34. 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.
35. 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.
36. D. Naccache and J. Stern, *A new public-key cryptosystem*,Advances in cryptology EUROCRYPT'97, Springer-Verlag, Lectures notes in computer science 1233, pp. 27–36, 1997.
37. National Institute of Standards and Technology, *Secure hash standard*, FIPS publication 180-1, April 1994 (also ANSI X9.30:2-1997, *Public-key cryptography for the financial services industry : part II, The secure hash algorithm SHA-1*).
38. A. Odlyzko, *The future of integer factorization*, CRYPTOBYTES, vol. 1, no. 2, pp. 5–12, 1995.
39. R. Rivest, *RFC 1321 : The MD5 message-digest algorithm*, Internet activities board, April 1992.
40. 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.
41. RSA Laboratories, PKCS #1 : *RSA cryptography specifications*, version 2.0, September 1998.
42. B. Schoenmakers, *Basic security of the* ECASH™ *payment system*, Computer security and industrial cryptography, Springer-Verlag, Lectures notes in computer science 1528, pp. 00–00, 1997.
43. S. Vanstone and R. Zuccharato, *Short RSA keys and their generation*, Journal of cryptology vol. 8, no. 2, 101–114, 1995.
44. M. de Waleffe and J.-J. Quisquater, *CORSAIR, a smart-card for public-key cryptosystems*, Advances in cryptology CRYPTO'90, Springer-Verlag, Lectures notes in computer science 1462, pp. 502–513, 1991.
45. C. Walter, *Faster modular multiplication by operand scaling* Advances in cryptology CRYPTO'91, Springer-Verlag, Lectures notes in computer science 576, pp. 313–323, 1992.

APPENDIX A

The following (redundant) look-up table lists ρ for the various smoothness and digest-size values concerned by this paper; $\rho(288/24)$, the probability that a 288-bit number has no prime factors larger than 2^{24} is $2^{-46.2}$:

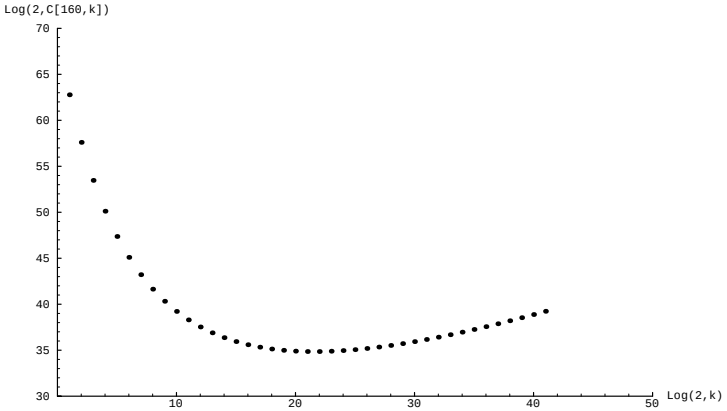
$-\log_2 \rho \searrow$	**16**	**20**	**24**	**28**	**32**	**36**	**40**	**44**	**48**	**52**	**56**	**60**	**64**	**68**	**72**
32	1.7	0.9	0.5	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
48	4.4	2.7	1.7	1.1	0.8	0.5	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
64	7.7	5.0	3.4	2.4	1.7	1.2	0.9	0.7	0.5	0.3	0.2	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
184	43.4	30.9	23.3	18.2	14.6	11.9	9.9	8.3	7.1	6.1	5.2	4.6	4.0	3.5	3.1
192	46.2	33.0	24.9	19.5	15.6	12.8	10.7	9.0	7.7	6.6	5.7	5.0	4.4	3.8	3.4
200	49.0	35.1	26.5	20.8	16.7	13.7	11.5	9.7	8.3	7.1	6.2	5.4	4.7	4.2	3.7
208	51.9	37.4	28.2	22.1	17.8	14.7	12.3	10.4	8.9	7.7	6.7	5.8	5.1	4.5	4.0
216	54.8	39.6	29.9	23.5	19.0	15.6	13.1	11.1	9.5	8.2	7.2	6.3	5.5	4.9	4.4
224	57.7	41.7	31.6	24.9	20.1	16.6	13.9	11.8	10.2	8.8	7.7	6.7	5.9	5.3	4.7
232	60.7	44.0	33.3	26.3	21.3	17.6	14.8	12.6	10.8	9.4	8.2	7.2	6.4	5.7	5.0
240	63.7	46.2	35.1	27.7	22.5	18.6	15.6	13.3	11.5	10.0	8.7	7.7	6.8	6.0	5.4
248	66.7	48.5	37.0	29.1	23.7	19.6	16.5	14.1	12.1	10.5	9.2	8.1	7.2	6.4	5.8
256	69.8	50.7	38.8	30.6	24.9	20.6	17.4	14.9	12.8	11.2	9.8	8.6	7.7	6.8	6.1
264	72.9	53.0	40.6	32.1	26.1	21.7	18.3	15.6	13.5	11.8	10.3	9.1	8.1	7.3	6.5
272	76.0	55.4	42.5	33.6	27.4	22.7	19.2	16.4	14.2	12.4	10.9	9.6	8.6	7.7	6.9
280	79.1	57.7	44.3	35.1	28.6	23.8	20.1	17.2	14.9	13.0	11.5	10.2	9.0	8.1	7.3
288	82.3	60.1	46.2	36.8	29.9	24.9	21.1	18.0	15.6	13.7	12.0	10.7	9.5	8.5	7.7
296	85.5	62.5	48.1	38.3	31.2	26.0	22.0	18.9	16.4	14.3	12.6	11.2	10.0	9.0	8.1
304	88.7	64.9	50.0	39.9	32.4	27.1	22.9	19.7	17.1	15.0	13.2	11.7	10.5	9.4	8.5
312	91.9	67.3	51.9	41.4	33.7	28.2	23.9	20.5	17.8	15.6	13.8	12.3	11.0	9.8	8.9
320	95.2	69.8	53.8	43.0	35.1	29.3	24.9	21.4	18.6	16.3	14.4	12.8	11.5	10.3	9.3
384	122.	90.0	68.8	56.0	46.2	38.2	33.0	28.7	24.9	22.0	19.5	17.4	15.6	14.0	12.8
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 [9, 10] for $t > 10$:

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

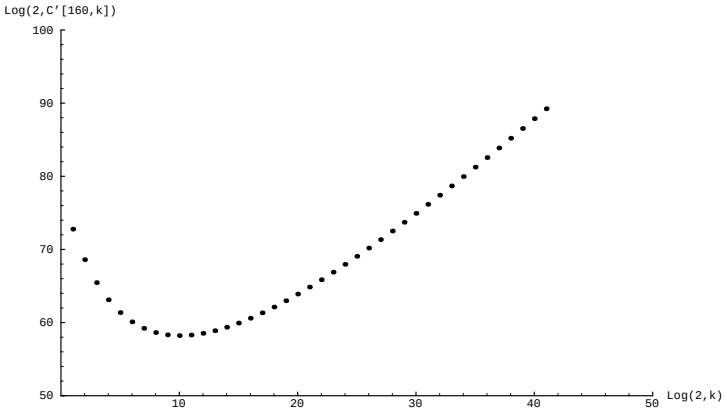
where ζ is the positive solution of $e^{\zeta} - 1 = t\zeta$ and γ is Euler's constant.

APPENDIX B



$$C_{160,k} = \frac{160\sqrt{k \ln k}}{\rho(160/\log_2(k \ln k))} \quad \text{for} \quad 2^{10} \leq k \leq 2^{40}$$

APPENDIX C



$$C'_{160,k} = k \times C_{160,k} \quad \text{for} \quad 2^{10} \leq k \leq 2^{40}$$

APPENDIX D

This appendix contains an ISO 9796-1 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 $\mu =$ ISO 9796-1 and form the 180 messages :

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

where `message` 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 the exception $p_0 = \Gamma_{23}$) and `beta` 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
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	1	1	1	2	2	1	2	1	2	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	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	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 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					

APPENDIX E

The next table, adapted from [38], converts MIPS-years (by convention a one MIPS machine is equivalent to the DEC VAX 11/780 in computing power) to powers of two expressing the estimated complexities of different factoring efforts. As usual, such empirical figures should only be used as *rough* estimates, subject to constant algorithmic improvements.

$\log_2 n$	$\log_2$ **time**	$\log_2$ **space**
512	64	39
640	70	42
768	75	45
1024	86	50
2048	116	66
4096	155	86

APPENDIX F

The constants c_{MD5} and c_{SHA} have no cryptographic importance and encode various format data (we include them in this appendix for completeness and reference [25] for further details).

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

APPENDIX G

Function s in ISO 9796-1 maps the hexadecimal nibble x to the hexadecimal $s(x)$-values listed in the following table :

$x =$	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

The four bits of $s(x)$ are respectively $\overline{x_4} \oplus x_2 \oplus x_1$, $\overline{x_4} \oplus x_3 \oplus x_1$, $\overline{x_4} \oplus x_3 \oplus x_2$ and $x_3 \oplus x_2 \oplus x_1$ where x_i denotes the i-th bit of x.

APPENDIX H

Note that independently of the attack of section 4.1, the use of moduli with prescribed patterns could also be a source of smoothness in ISO 9796-1; we see no better illustration than using the $n_{\text{ISO}} > 2^{512}$ given as example in Annex B section B.1.1. of ISO 9796-1 for which :

$$\texttt{FF77}_{16} \times n_{\text{ISO}} - \texttt{FFFF}_{16} \times \mu(m) \cong 2^{401}$$

where :

$$m = \texttt{7777777777777777FEDE17FC6EE81EDA}$$
$$\texttt{F3E370E4f504A4148C0C6E328052D412}_{16}$$

other examples are easy to produce.

APPENDIX I

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

To overcome this problem, we suggest the following : as a first step, collect a big number (*e.g.* $p_k \cong 2^{40}$) of moderate-size smooth x-strings (which are relatively easy to find) and, using Gaussian elimination, extract the e-th roots of the k first primes. Then, exhaustive-search two plain-English 256-bit messages $\{m, m'\}$ ending by the letter f such that $\mu(m)$ and $\mu(m')$ are both p_k-smooth. The probability that a 256-bit number is p_k-smooth is $\cong 2^{-15}$ and 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 appropriate p_i-roots to obtain $\Gamma^d = (2^{256} + 1)^d \bmod n$. Using Γ^d and the e-th roots of the k first primes we can now forge, by multiplication, the signature of m'.

This article was processed using the LaTeX macro package with LLNCS style