J. Cryptol. (2008) 21: 27–51
DOI: 10.1007/s00145-007-9007-5

Journal of
CRYPTOLOGY

Cryptanalysis of ISO/IEC 9796-1

D. Coppersmith
IBM T.J. Watson Research Center, Yorktown Heights, NY, USA

J.S. Coron
University of Luxembourg, Luxembourg, Luxembourg
jean-sebastien.coron@uni.lu

F. Grieu
Spirtech, 1 rue Danton, 75006 Paris, France
francois.grieu@spirtech.com

S. Halevi and C. Jutla
IBM T.J. Watson Research Center, Hawthorne, NY, USA
shaih@us.ibm.com; csjutla@us.ibm.com

D. Naccache
Ecole normale supérieure, Paris, France
david.naccache@ens.fr

J.P. Stern
Cryptolog International SAS, 16-18 rue Vulpian, 75013 Paris, France
julien@cryptolog.com

Communicated by Stefan Wolf

Received 2 February 2004 and revised 23 March 2007
Online publication 5 September 2007

Abstract. We describe two different attacks against the ISO/IEC 9796-1 signature standard for RSA and Rabin. Both attacks consist in an existential forgery under a chosen-message attack: the attacker asks for the signature of some messages of his choice, and is then able to produce the signature of a message that was never signed by the legitimate signer. The first attack is a variant of Desmedt and Odlyzko's attack and requires a few hundreds of signatures. The second attack is more powerful and requires only three signatures.

Key words. Cryptanalysis, ISO/IEC 9796-1 signature standard, RSA signatures, Rabin signatures, Encoding scheme

1. Introduction

A digital signature of a message is a bit string obtained from a secret known only to the signer, and the message being signed. Additionally, a digital signature must be verifiable by a third party without knowing the signer's secret. To accomplish this, a signature scheme is generally based on a public-key cryptosystem. A private and public key pair is generated by the user, who publishes the public-key while the private-key remains secret. The private key is used to generate a signature of a given message, and the public key is used to verify the signature of a message.

The first realization of digital signatures was based on the RSA cryptosystem, invented in 1977 by Rivest, Shamir and Adleman [13], which is by now the most widely used public-key cryptosystem. In this scheme, the public key is a composite integer N and a public exponent e, and the secret key is a private exponent d such that $ed = 1 \bmod \phi(N)$. To sign a message m, the signer first applies some encoding function μ that maps m into a number smaller than N, and then raises $\mu(m)$ to the private exponent d modulo N. The signature is then $s = \mu(m)^d \bmod N$. The signature can be verified by checking that $s^e = \mu(m) \bmod N$, where e is the public exponent.

A signature scheme is said to be secure if it is infeasible to produce a valid signature of a message without knowing the private key. This task should remain infeasible even if the attacker can obtain the signature of any message of his choice. This security notion was formalized by Goldwasser, Micali and Rivest in [5] and is called *existential unforgeability under an adaptive chosen message attack*. It is the strongest security notion for a signature scheme and it is now considered as the standard security notion for signature schemes.

The ISO/IEC 9796-1 standard [8] was published in 1991 by ISO as the first international standard for digital signatures. It specifies some encoding function μ (among other things). For many years, the standard was believed to be secure, as no attack better than factoring the modulus N was known; see [7] for the rationale behind the design of ISO/IEC 9796-1, and [12] for a survey on RSA-based digital signatures.

In this paper, we describe two different attacks against the ISO/IEC 9796-1 signature standard. Each of the two attacks constitutes existential forgery under a chosen-message attack: the attacker asks for the signature of some messages of his choice, and is then able to produce the signature of a message that was never signed by the owner of the private key. The first attack [1], designed by Coppersmith, Halevi and Jutla, appeared as a research contribution to P1363. It is a variant of an attack, published at Crypto '99 by Coron, Naccache and Stern [2], against a slightly modified variant of the ISO/IEC 9796-1 standard. These attacks are a variant of Desmedt and Odlyzko's attack against RSA and require a few hundred signatures. The second attack was published by Grieu at Eurocrypt 2000 [6] and uses a different technique; it is more powerful as it requires only three signatures. We describe both attacks in this paper because the first attack, albeit less powerful, is more algebraic and easier to understand. Note that after the publication of these attacks, the ISO/IEC 9796-1 standard was withdrawn.

2. RSA and Rabin Signature Schemes

2.1. *The RSA Signature Scheme*

In this section, we briefly recall the RSA signature scheme, based on the RSA cryptosystem. The user generates two random primes p and q of approximately the same size, and computes the modulus $N = p \cdot q$. He randomly picks an encryption exponent $e \in \mathbb{Z}^*_{\phi(N)}$

and computes the corresponding decryption exponent d such that $e \cdot d = 1 \bmod \phi(N)$. Alternatively, the user can select a small exponent e such as $e = 3$ or $e = 2^{16} + 1$. The public-key is then (N, e) and the private key is (N, d). The RSA signature scheme is specified by an encoding function μ, which takes as input a message m and returns an integer modulo N, denoted $\mu(m)$. Below we sometime call $\mu(m)$ "the redundant message" (since μ would typically add some redundancy). The signature of a message m is then

$$s = \mu(m)^d \bmod N.$$

The signature is verified by checking that

$$\mu(m) \stackrel{?}{=} s^e \bmod N.$$

2.2. *The Rabin Signature Scheme*

The Rabin–Williams signature scheme (see [11]) is similar to RSA, but it uses a public exponent $e = 2$; it is a variant of the Rabin signature scheme that enables deterministic signing. As for RSA, it uses an encoding function $\mu(m)$, but with the additional property that $\mu(m) = 6 \bmod 16$ for all m.

Key generation: on input 1^k, generate two $k/2$-bit primes p and q such that $p = 3 \bmod 8$ and $q = 7 \bmod 8$. The public key is $N = p \cdot q$ and the private key is $d = (N - p - q + 5)/8$.

Signature generation: compute the Jacobi symbol $J = (\frac{\mu(m)}{N})$. The signature of m is then $s = \min(\sigma, N - \sigma)$, where

$$\sigma = \begin{cases} \mu(m)^d \bmod N & \text{if } J = 1, \\ (\mu(m)/2)^d \bmod N & \text{otherwise.} \end{cases}$$

Signature verification: compute $\omega = s^2 \bmod N$ and check that

$$\mu(m) \stackrel{?}{=} \begin{cases} \omega & \text{if } \omega = 6 \bmod 8, \\ 2 \cdot \omega & \text{if } \omega = 3 \bmod 8, \\ N - \omega & \text{if } \omega = 7 \bmod 8, \\ 2 \cdot (N - \omega) & \text{if } \omega = 2 \bmod 8. \end{cases}$$

To prove the signature scheme's soundness, we first recall some known facts about Legendre and Jacobi symbols. The Legendre symbol relative to an odd prime p is defined by

$$\left(\frac{x}{p}\right) = \begin{cases} 1 & \text{if } x \neq 0 \bmod p \text{ and } x \text{ is a square modulo } p, \\ 0 & \text{if } x = 0 \bmod p, \\ -1 & \text{otherwise.} \end{cases}$$

Lemma 1. *Let $p \neq 2$ be a prime. For any integer x,*

$$\left(\frac{x}{p}\right) = x^{\frac{p-1}{2}} \bmod p.$$

The Jacobi symbol relative to an odd integer $n = \prod p_i^{e_i}$ is defined from Legendre symbols as follows:

$$\left(\frac{x}{n}\right) = \prod \left(\frac{x}{p_i}\right)^{e_i}.$$

The Jacobi symbol can be computed without knowing the factorization of n; we refer to [15] for a detailed study. The following lemma enables to show that signature verification of Rabin–Williams signature scheme works. In particular, the fact that $(\frac{2}{N}) = -1$ ensures that either $\mu(m)$ or $\mu(m)/2$ has Jacobi symbol equal to 1.

Lemma 2. *Let N be an RSA-modulus with $p = 3 \bmod 8$ and $q = 7 \bmod 8$. Then $(\frac{2}{N}) = -1$ and $(\frac{-1}{N}) = 1$. Let $d = (N - p - q + 5)/8$. Then for any integer x such that $(\frac{x}{N}) = 1$, we have that $x^{2d} = x \bmod N$ if x is a square modulo N, and $x^{2d} = -x \bmod N$ otherwise.*

3. Desmedt and Odlyzko's Attack

This attack [3] applies to the RSA and Rabin signature schemes and provides an existential forgery against a chosen-message attack.

1. Select a bound y and let $L = (p_1, \ldots, p_\ell)$ be the list of primes smaller than y.
2. Find at least $\ell + 1$ messages m_i such that each $\mu(m_i)$ is the product of primes in L.
3. Express one $\mu(m_j)$ as a multiplicative combination of the other $\mu(m_i)$, by solving a linear system given by the exponent vectors of the $\mu(m_i)$ with respect to the primes in L.
4. Ask for the signature of the m_i for $i \neq j$ and forge the signature of m_j.

The attack complexity depends on the length of L and on the difficulty of finding at step 2 enough $\mu(m_i)$ which are the product of primes in L. Generally, the attack applies only if $\mu(m)$ is small; otherwise, the probability that $\mu(m)$ is the product of small primes only is too small.

3.1. *The Desmedt and Odlyzko Attack for RSA with Prime e*

In the following, we describe the attack in more detail. First, we focus on RSA, that is we have $\gcd(e, \phi(N)) = 1$, and assume that e is a prime integer. We let τ be the number of messages m_i obtained at step 2. We say that an integer is B-smooth if all its prime factors are smaller than B. The integers $\mu(m_i)$ obtained at step 2 are therefore y-smooth and we can write for all messages m_i, $1 \le i \le \tau$:

$$\mu(m_i) = \prod_{j=1}^{\ell} p_j^{v_{i,j}}. \tag{1}$$

Step 3 works as follows. To each $\mu(m_i)$ we associate the ℓ-dimensional vector of the exponents modulo e:

$$\boldsymbol{V}_i = (v_{i,1} \bmod e, \ldots, v_{i,\ell} \bmod e).$$

Since e is assumed to be prime, the set of all ℓ-dimensional vectors modulo e form a linear space of dimension ℓ. Therefore, if $\tau \geq \ell + 1$, one can express one vector, say $\boldsymbol{V}_\tau$, as a linear combination of the others modulo e, using Gaussian elimination, which gives for all $1 \leq j \leq \ell$:

$$v_{\tau,j} = \gamma_j \cdot e + \sum_{i=1}^{\tau-1} \beta_i \cdot v_{i,j}$$

for some $\gamma_1, \ldots, \gamma_\ell \in \mathbb{Z}$. Then using (1), one obtains

$$\mu(m_\tau) = \prod_{j=1}^{\ell} p_j^{v_{\tau,j}} = \prod_{j=1}^{\ell} p_j^{\gamma_j \cdot e + \sum_{i=1}^{\tau-1} \beta_i \cdot v_{i,j}} = \left(\prod_{j=1}^{\ell} p_j^{\gamma_j}\right)^e \cdot \prod_{j=1}^{\ell} \prod_{i=1}^{\tau-1} p_j^{v_{i,j} \cdot \beta_i}, \tag{2}$$

$$\mu(m_\tau) = \left(\prod_{j=1}^{\ell} p_j^{\gamma_j}\right)^e \cdot \prod_{i=1}^{\tau-1} \left(\prod_{j=1}^{\ell} p_j^{v_{i,j}}\right)^{\beta_i} = \delta^e \cdot \prod_{i=1}^{\tau-1} \mu(m_i)^{\beta_i} \tag{3}$$

where we denote

$$\delta = \prod_{j=1}^{\ell} p_j^{\gamma_j}. \tag{4}$$

Therefore, we obtain that $\mu(m_\tau)$ can be written as a multiplicative combination of the other $\mu(m_i)$. Then, at step 4, the attacker will ask for the signature of the $\tau - 1$ first messages m_i and forge the signature of m_τ using

$$\mu(m_\tau)^d = \delta \cdot \prod_{i=1}^{\tau-1} (\mu(m_i)^d)^{\beta_i} \bmod N. \tag{5}$$

The attack's complexity depends on ℓ and on the probability that the integers $\mu(m_i)$ are y-smooth. We define $\psi(x, y) = \#\{v \leq x, \text{ such that } v \text{ is } y\text{-smooth}\}$. It is known [4] that, for large x, the ratio $\psi(x, \sqrt[t]{x})/x$ is equivalent to Dickman's function defined by

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

$\rho(t)$ is thus an approximation of the probability that a u-bit number is $2^{u/t}$-smooth. Table 1 gives the numerical value of $\rho(t)$ (on a logarithmic scale) for $1 \leq t \leq 10$.

In the following, we provide an asymptotic analysis of the algorithm's complexity, based on the assumption that the integers $\mu(m)$ are uniformly distributed between zero

Table 1. The value of Dickman's function.

t	1	2	3	4	5	6	7	8	9	10
$\log_2 \rho(t)$	0	−1.7	−4.4	−7.7	−11.5	−15.6	−20.1	−24.9	−29.9	−35.1

and some given bound x. Letting β be a constant and letting

$$y = L_x[\beta] = \exp(\beta \cdot \sqrt{\log x \log \log x})$$

one obtains [4] that, for large x, the probability that an integer uniformly distributed between one and x is $L_x[\beta]$-smooth is

$$\frac{\psi(x,y)}{x} = L_x\left[-\frac{1}{2\beta} + o(1)\right].$$

Therefore, we have to generate on average $L_x[1/(2\beta) + o(1)]$ integers $\mu(m)$ before we can find one which is y-smooth.

Using the ECM factorization algorithm [10], a prime factor p of an integer n can be found in time $L_p[\sqrt{2} + o(1)]$. A y-smooth integer can thus be factored in time $L_y[\sqrt{2} + o(1)] = L_x[o(1)]$. The complexity of finding a random integer in $[0, x]$ which is y-smooth using the ECM is thus $L_x[1/(2\beta) + o(1)]$. Moreover, the number τ of integers which are necessary to find a vector which is a linear combination of the others is $\ell + 1 \leq y$. Therefore, one must solve a system with $r = L_x[\beta + o(1)]$ equations in $r = L_x[\beta + o(1)]$ unknowns. Using Lanzos' iterative algorithm [9], the time required to solve such system is $\mathcal{O}(r^2)$ and the space required is roughly $\mathcal{O}(r)$.

To summarize, the time required to obtain the $L_x[\beta + o(1)]$ equations is asymptotically $L_x[\beta + 1/(2\beta) + o(1)]$ and the system is solved in time $L_x[2\beta + o(1)]$. The total complexity is minimal by taking $\beta = 1/\sqrt{2}$. We obtain time complexity

$$L_x[\sqrt{2} + o(1)]$$

and space complexity

$$L_x\left[\frac{\sqrt{2}}{2} + o(1)\right].$$

This complexity is sub-exponential in the size of the integers $\mu(m)$. Therefore, without any modification, the attack will be practical only if $\mu(m)$ is small. In particular, when $\mu(m)$ is about the same size as the modulus N, the complexity of the attack is no better than factoring N.

3.2. *Extension to any Exponent* ≥ 3

When e is prime, the set of ℓ-dimensional vectors modulo e is a ℓ-dimensional linear space; $\tau = \ell + 1$ vectors are consequently sufficient to guarantee that (at least) one of the vectors can be expressed as a linear combination of the others.

If we assume that e is the r-th power of a prime p, then $\tau = \ell + 1$ 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 vector coefficients and Gaussian elimination on $\ell + 1$ vectors, one can write one of the vectors as a linear combination of the others.

Finally, in the general case, writing $e = \prod_{i=1}^{\omega} p_i^{r_i}$, then $\tau = 1 + \omega \cdot \ell$ vectors are sufficient to guarantee that (at least) one vector is a linear combination of the others. Namely, for each of the $p_i^{r_i}$, using the previous argument one can find a set T_i of

$(\omega - 1)\ell + 1$ vectors, each of which can be expressed by Gaussian elimination as a linear combination of ℓ 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. We obtain the same asymptotic complexity as previously.

3.3. *Extension to Rabin–Williams Signatures*

Previously, we assumed that e is invertible modulo $\phi(n)$. This is no longer the case for Rabin–Williams signatures, where $e = 2$. We modify the attack as follows:

For each message m_i at step 2, we replace $\mu(m_i)$ by $\mu(m_i)/2$ if $(\frac{\mu(m_i)}{N}) = -1$. The attack continues without modification until (3), which gives

$$\mu(m_\tau)^d = \delta^{2d} \cdot \prod_{i=1}^{\tau-1} (\mu(m_i)^d)^{\beta_i} \bmod N. \tag{6}$$

We distinguish two cases: if the integer δ given by (4) is such that $(\frac{\delta}{N}) = 1$, then using Lemma 2 we obtain that $\delta^{2d} = \pm\delta \bmod N$, which gives

$$\mu(m_\tau)^d = \pm\delta \cdot \prod_{i=1}^{\tau-1} (\mu(m_i)^d)^{\beta_i} \bmod N$$

instead of (5). This shows that, as previously, one can forge the signature of m_τ using the signatures of $m_1, \ldots, m_{\tau-1}$.

Otherwise, if $(\frac{\delta}{N}) = -1$, then we see from (6) that we can compute from the signatures of the τ messages $m_1, \ldots, m_\tau$ the integer

$$u = \delta^{2d} \bmod N.$$

From Lemma 2 we have that $u^2 = \delta^2 \bmod N$, which gives $(u - \delta)(u + \delta) = 0 \bmod N$. Since u is a square, we have that $(\frac{u}{N}) = 1$; then since $(\frac{-1}{N}) = 1$, we cannot have $\delta = \pm u \bmod N$. Therefore, $\gcd(u \pm \delta, N)$ must disclose the factorization of N.

3.4. *Practical Experiments*

We have implemented the previous attack, using Shoup's NTL library [14]. Instead of computing $\mu(m_i)$ for some particular function μ, we have generated a sequence of random integers x_i uniformly distributed between zero and $x = 2^a$, for various integers a. Our goal was to express one x_i as a multiplicative combination of the others modulo some given RSA-modulus N, using the previous attack.

Let ℓ be, as before, the number of primes in the list L, and let p_ℓ be the ℓ-th prime. We have that $p_\ell \simeq \ell \log \ell$. Then, the probability that a random x_i is p_ℓ-smooth can be approximated by

$$\alpha = \rho\left(\frac{a \log 2}{\log(\ell \log \ell)}\right). \tag{7}$$

We have to generate on the average $1/\alpha$ integers x_i in order to find one that is p_ℓ-smooth, and we need $\ell + 1$ such p_ℓ-smooth integers. Therefore, we need to generate on the average ℓ/α integers x_i.

Table 2. Running time, observed (on a 733 MHz PC) and estimated, for various sizes of x_i, with the $\log_2$ total number of x_i to generate in order to find one that is a multiplicative combination of the others.

Size	# primes ℓ	Running time	$\log_2$ number of x_i	Estimated time	Estimated $\log_2$ number of x_i
48 bits	250	8 s	17	14 s	18
64 bits	700	9 min	21	15 min	22
80 bits	2000	5 hours	25	11 hours	25
96 bits	5000	—	—	14 days	29
128 bits	20000	—	—	22 years	35

Using the NTL library, we observed that the time required to perform brute-force division by the first ℓ primes on a given integer x_i is linear in $\ell \cdot a$; we obtained the following running time t_u per integer x_i, on a 733 MHz PC, in seconds units:

$$t_u(a, \ell) = 5 \cdot 10^{-9} \cdot \ell \cdot a$$

so that we can estimate the total running time as a function of a and ℓ, in seconds units:

$$t(a, \ell) = 5 \cdot 10^{-9} \cdot \frac{a \cdot \ell^2}{\rho\left(\frac{a \log 2}{\log(\ell \log \ell)}\right)}. \tag{8}$$

We chose the number of primes ℓ so as to minimize the total running time. We found that the matrix solving step took a negligible amount of time. The result of practical experiments, and theoretical estimates based on (8) are summarized in Table 2. They show that when the size of the x_i is less than approximately 80 bits, the attack is feasible, but for larger sizes (more than 128 bits) it quickly becomes impractical. Note however that the attack's first step (finding smooth integers) is fully parallelizable.

3.5. *An Improved Attack*

Let $\mathcal{M}$ be a message subset and let X be the set of corresponding encodings, that is $X = \{\mu(m) \mid m \in \mathcal{M}\}$. Assume now that X can be written as

$$X = \{u + v \mid u \in U,\ v \in V\}$$

for two sets U and V; this is trivially done for ISO/IEC 9796-1. Then one can derive a much faster attack, as follows:

Improved attack for $X = U + V$
Input: sets U, V and X; the set L of the ℓ first primes.
Output: a subset X' of X such that all elements of X' are p_ℓ-smooth.

1. Generate a table $T[x] \leftarrow \log x$ for all $x \in X$.
2. For each $p \in L$ do
   (a) Generate the following partition of V, with $0 \le i < p$:

   $$V_i = \{v \in V \mid v \bmod p = i\}.$$

   (b) For each $u \in U$ do

(i) Let $i = -u \bmod p$
(ii) For each $v \in V_i$ do
(A) Let $x = u + v$ (at this point, $x = 0 \bmod p$)
(B) Let $T[x] \leftarrow T[x] - \log p$.

3. Let θ be some constant threshold (for example, $\theta = 2$). Then for each $x \in X$ do:
   (a) If $T[x] \leq \theta$, check that x is p_ℓ-smooth; in this case, let $X' \leftarrow X' \cup \{x\}$.
4. Output X'.

We provide a heuristic analysis of the algorithm's complexity. Our analysis is heuristic because we assume that for each prime $p \in L$, the partition of V is balanced, that is

$$|V_i| \leq \eta \cdot \frac{|V|}{p}$$

for all $0 \leq i < p$, for some constant $\eta > 0$.

As previously, let denote by a the maximum bit-size of the integers in X. When generating the partition of V, each computation of $v \bmod p$ takes $\mathcal{O}(a \cdot \log \ell)$ time, so the complexity of step 2(a) for a given p is $\mathcal{O}(|V| \cdot a \cdot \log \ell)$. For all p, the total complexity is therefore $\mathcal{O}(\ell \cdot |V| \cdot a \cdot \log \ell)$.

The complexity of step 2(b)(ii) is $\mathcal{O}(a)$. Thanks to our balanced partition assumption, the complexity of step 2(b)(ii) for a given p is therefore $\mathcal{O}(a \cdot |V|/p)$. Using

$$\sum_{i=1}^{\ell} \frac{1}{p_\ell} \leq \sum_{i=1}^{\ell} \frac{1}{\ell} = \mathcal{O}(\log \ell)$$

we obtain that for all $p \in L$ and all $u \in U$, the total complexity of step 2(b)(ii) is $\mathcal{O}(|U| \cdot a \cdot |V| \cdot \log \ell)$. Similarly, the total complexity of step 2(b)(i) for all $u \in V$ and $p \in L$ is $\mathcal{O}(|U| \cdot \ell \cdot a \cdot \log \ell)$. Therefore, the algorithm's total complexity is

$$\mathcal{O}\big(a \cdot \log \ell \cdot (|X| + \ell \cdot (|U| + |V|))\big).$$

Taking $|U| = |V| = \sqrt{|X|}$ and assuming that $\ell = \mathcal{O}(\sqrt{|X|})$, we obtain a complexity of

$$\mathcal{O}(a \cdot |X| \cdot \log \ell).$$

As in the first attack, we need to generate on average ℓ/α integers x_i, so we must take $|X| = \ell/\alpha$, where α is given by (7). The attack's complexity is therefore

$$t'(a, \ell) = \frac{a \cdot \ell \cdot \log \ell}{\rho\left(\frac{a \log 2}{\log(\ell \log \ell)}\right)} \cdot \mathcal{O}(1).$$

Note that compared to the previous attack, the ℓ^2 factor has been replaced by $\ell \cdot \log \ell$; however the attack is memory bound as it requires $\mathcal{O}(|X|)$ memory (whereas the previous attack required only negligible memory).

As in the previous attack, we choose the number of primes ℓ so as to minimize the running time. In Table 3, we summarize the result of practical experiments; we find that the new attack provides a significant improvement: for 96 bits, it takes 8 minutes instead

Table 3. Running time observed (on a 2 GHz PC) for various sizes of x_i, with the $\log_2$ total number of x_i necessary; $|X|$ is the size of the sieving set.

Size	# primes ℓ	Running time	$\log_2 \lvert X \rvert$	$\log_2$ number of x_i
48 bits	400	0.3 s	17	17
64 bits	1500	4 s	21	21
80 bits	5000	45 s	25	25
96 bits	15000	8 min	28	28
128 bits	120000	81 hours	28	34

of an estimated 14 days; for 128 bits, it takes 81 hours instead of an estimated 22 years; note that for 128 bits the number of required x_i is 2^{34}; since we could not store an array of 2^{34} elements in memory, we performed repeated sieving with $|X| = 2^{28}$ only.

4. The ISO/IEC 9796-1 Signature Standard

The ISO/IEC 9796-1 standard [8] was published in 1991 by ISO as the first international standard for digital signatures. It specifies (among other things) an encoding function μ_{ISO} for messages that are shorter than half the modulus size. The encoding function μ_{ISO} embeds the message m itself in the integer $\mu(m)$ (with some additional redundancy). Thus it enjoys "message recovery", which means that the message is recovered when verifying the signature.

In the following, we restrict ourselves to moduli of size $k = 16 \cdot z + 1$ bits and to messages of size $8z$ bits, for some integer z. This allows for a simpler description of the ISO/IEC 9796-1 standard. We denote by m_i the i-th 4-bit nibble of m, for $0 \le i \le 2z-1$. In this case, the encoding function—denoted μ_{ISO}—is defined as follows:

$$\begin{aligned}\mu_{\text{ISO}}(m) = {} & \bar{s}(m_{2z-1})\ \tilde{s}(m_{2z-2})\ m_{2z-1}\ m_{2z-2} \\ & s(m_{2z-3})\ s(m_{2z-4})\ m_{2z-3}\ m_{2z-4} \\ & \quad\vdots \\ & s(m_3) \quad s(m_2) \quad m_3 \quad m_2 \\ & s(m_1) \quad s(m_0) \quad m_0 \quad 6\end{aligned}$$

The permutation $s(x)$ in defined as

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

$\tilde{s}(x)$ denotes the nibble $s(x)$ with the least significant bit flipped (i.e., $\tilde{s}(x) = s(x) \oplus 1$), and $\bar{s}(x)$ is the result of setting the most significant bit of $s(x)$ to '1', that is, $\bar{s}(x) = 1000 \text{ OR } s(x)$.

5. Attack against Modified ISO/IEC 9796-1

First, we describe an attack against a slight variant of ISO/IEC 9796-1, in which the encoding function is modified by one single bit. This attack was published at Crypto '99 by Coron, Naccache and Stern [2].

We consider a modified ISO/IEC 9796-1, in which the function $\tilde{s}(x)$ which appears in the definition of $\mu(m)$ is replaced by $s(x)$. We obtain the following modified encoding:

$$\begin{aligned}\mu'(m) = \bar{s}(m_{2z-1})\; s(m_{2z-2})\; m_{2z-1}\; m_{2z-2}\\ s(m_{2z-3})\; s(m_{2z-4})\; m_{2z-3}\; m_{2z-4}\\ \vdots \qquad\qquad\qquad\\ s(m_3) \quad s(m_2) \quad m_3 \quad m_2\\ s(m_1) \quad s(m_0) \quad m_0 \quad 6\end{aligned}$$

We assume that the modulus size k is such that $k = 1 \bmod 64$ and let $k = 64 \cdot u + 1$. We consider a message m of size $32 \cdot u = 8 \cdot z$ bits, consisting in u times the same 32-bit pattern:

$$\begin{aligned}m = a_6\; a_5\; a_4\; a_3\; a_2\; a_1\; \mathtt{66}_{16}\\ a_6\; a_5\; a_4\; a_3\; a_2\; a_1\; \mathtt{66}_{16}\\ \vdots \qquad\qquad\quad\\ a_6\; a_5\; a_4\; a_3\; a_2\; a_1\; \mathtt{66}_{16}\end{aligned}$$

where $a_1, \ldots, a_6$ are 4-bit nibbles. Its modified padding is given by

$$\begin{aligned}\mu'(m) = \bar{s}(a_6)\; s(a_5)\; a_6\; a_5\;\; s(a_4)\; s(a_3)\; a_4\;\; a_3\\ s(a_2)\; s(a_1)\; a_2\; a_1\;\; \mathtt{2}_{16}\;\; \mathtt{2}_{16}\;\; \mathtt{6}_{16}\; \mathtt{6}_{16}\\ \vdots \qquad\qquad\qquad\qquad\\ s(a_6)\; s(a_5)\; a_6\; a_5\;\; s(a_4)\; s(a_3)\; a_4\;\; a_3\\ s(a_2)\; s(a_1)\; a_2\; a_1\;\; \mathtt{2}_{16}\;\; \mathtt{2}_{16}\;\; \mathtt{6}_{16}\; \mathtt{6}_{16}\end{aligned}$$

We restrict the choice of a_6 to the eight nibbles for which $s = \bar{s}$, so that the structure of $\mu'(m_i)$ is fully periodic. This enables us to write $\mu'(m)$ as

$$\mu'(m) = \Gamma \cdot x \tag{9}$$

where x is a 64-bit integer, a concatenation of the following nibbles:

$$x = s(a_6)\; s(a_5)\; a_6\; a_5\; s(a_4)\; s(a_3)\; a_4\; a_3\; s(a_2)\; s(a_1)\; a_2\; a_1\; \mathtt{2266}_{16}$$

and the constant Γ is given by

$$\Gamma = \sum_{i=0}^{u-1} 2^{64 \cdot i}.$$

The factorization given by (9) writes $\mu'(m)$ as the product of a constant Γ by some small integer x. This enables us to apply Desmedt and Odlyzko's attack described in Sect. 3. The only modification consists in including the constant Γ in the list L of small primes, so as to write

$$\mu(m_i) = \Gamma \cdot \prod_{j=1}^{\ell} p_j^{v_{i,j}} \bmod N \quad \text{for } 1 \le i \le \tau.$$

Then, to each $\mu(m_i)$ we associate a $\ell+1$-dimensional vector $\boldsymbol{V}_i=(1,v_{i,1},\ldots,v_{i,\ell})$, instead of $(v_{i,1},\ldots,v_{i,\ell})$, and the attack carries out as described in Sect. 3.

We see in Table 2 that for 64-bit integers, the attack demands the generation of approximately 2^{22} integers, and takes only a few minutes on a single PC (running at 733 MHz). There are 2^{23} possible values for x, so the attack against modified ISO/IEC 9796-1 is likely to work in practice. This is confirmed by experiments performed in [2], in which an example of forgery is given using only 181 messages.

6. Attack against the Full ISO/IEC 9796-1

The actual encoding function that is used in the ISO/IEC 9796-1 standard is slightly different than the function μ' above. Namely, for these parameters, the difference between $\mu'(m)$ and $\mu_{\mathrm{ISO}}(m)$ is that the lowest bit in the second-most-significant nibble of $\mu_{\mathrm{ISO}}(m)$ is flipped.

One can see that we cannot simply represent the encoding $\mu_{\mathrm{ISO}}(m)$ as a product $\Gamma\cdot x$ with Γ, x as above. Hence the attack must be modified to apply to this encoding function. The extension of the previous attack to the full ISO/IEC 9796-1 was done by Coppersmith, Halevi and Jutla [1].

6.1. *Modifying the Attack*

The modified attack is similar to the attack described in the previous section, except that it uses a slightly different structure for Γ and x. In the previous attack, the constant Γ consisted of several ones that were separated by as many zeroes as there are bits in x. In the modified attack, we again have a constant Γ which consists of a few ones separated by many zeroes, but this time there are fewer separating zeroes.

We start with an example. Consider a 64-bit integer x, which is represented as four 16-bit words $x=abcd$ (so a is the most-significant word of x, b is the second-most-significant, *etc.*). Also, consider the 112-bit constant $\Gamma=1001001$, where again each digit represents a 16-bit word. Now consider what happens when we multiply $\Gamma\cdot x$. We have

$$\begin{array}{rccccccccc}
\Gamma\cdot x= & & & & & & a & b & c & d\\
 & & & & \cdot\,1 & 0 & 0 & 1 & 0 & 0\;1\\
\hline
 & & & & & & a & b & c & d\\
 & & & a & b & c & d & & & \\
 & a & b & c & d & & & & & \\
\hline
 & a & b & c & e & b & c & e & b & c\;d
\end{array}$$

where $e=a+d$ (assuming that no carry is generated in the addition $a+d$). Notice that the 16-bit d appears only as the least-significant word of the result, and the 16-bit a appears only as the most-significant word of the result. It is therefore possible to arrange things so that the form of the words a, d be different than the form of the words b, c and e, and this could match the different forms of the least- and most-significant words in the encoded message $\mu_{\mathrm{ISO}}(m)$.

More precisely, we consider three types of 16-bit words. For a 16-bit word x, we say that:

- x is a *valid low word* if it has the form $x = s(u)\ s(v)\ v\ 6$, for some two nibbles u, v.
- x is a *valid middle word* if it has the form $x = s(u)\ s(v)\ u\ v$, for some two nibbles u, v.
- x is a *valid high word* if it has the form $x = \bar{s}(u)\ \tilde{s}(v)\ u\ v$, for some two nibbles u, v.

We note that there are exactly 256 valid low words, 256 valid middle words, and 256 valid high words (since in each case we can arbitrarily choose the nibbles u, v).

In the example above, we needed a to be a valid high word, d to be a valid low word, b and c to be valid middle words, and we also needed $e = a + d$ to be a valid middle word. We note the following:

- There are 64 pairs x, y such that x is a valid high word, y is a valid low word, and $z = x + y$ is a valid middle word (this is what we needed for the example above). We call such a pair (x, y) a *high–low pair*. The 64 high–low pairs are listed in Appendix A.
- There are 84 pairs x, y such that x is a valid high word, y is a valid middle word, and $z = x + y$ is a valid middle word. We call such a pair (x, y) a *high–mid pair*.
- There are 150 pairs x, y such that x is a valid middle word, y is a valid low word, and $z = x + y$ is a valid middle word. We call such a pair (x, y) a *mid–low pair*.
- There are 468 pairs x, y such that x is a valid middle word, y is a valid middle word, and $z = x + y$ is also a valid middle word. We call such a pair (x, y) a *mid–mid pair*.

We are now ready to present the attack. For clarity of presentation we start by presenting the attack for the special cases where the modulus size is $1024 + 1$ bits and $2048 + 1$ bits. We later describe the general case.

6.2. *Moduli of Size* $1024 + 1$ *Bits*

When the modulus size is $k = 1025$ bits, we need to encode the messages as 1024-bit integers with the high bit set to one. The attack proceeds similarly to the above example: we consider 64-bit integers $x = abcd$, where a is a valid high-word, d is a valid low-word, and b, c and $e = a + d$ are valid middle words. There are 64 choices for the high–low pair (a, d) and 256 choices for each of b, c, so there are total of 2^{22} integers x of the right form. We then set

$$\Gamma_{1024} = \sum_{i=0}^{20} 2^{48i} = \underbrace{1\,001\,001\ \ldots\ 001_{\,2^{16}}}_{\text{1 followed by 20 repetitions of 001 (base } 2^{16})}.$$

This gives us

$$M = \Gamma_{1024} \cdot x = a\ \underbrace{bce\ bce\ \ldots\ bce}_{\text{20 repetitions}}\ bcd$$

which is a valid encoding of some message $M = \mu_{\mathrm{ISO}}(m)$, because of the way in which x was chosen. We can see that the attack applies more generally to moduli of size $48 \cdot t + 65$, for any integer t.

With a 64-bit integer x, the attack's complexity is the same as before. The only difference is that there are now 2^{22} possible values for x instead of 2^{23}. In Appendix B, we provide an example of a forgery using 273 messages.

6.3. *Moduli of Size* 2048 + 1 *Bits*

When the modulus size is $k = 2049$ bits, we need to encode messages as 2048-bit integers with the high bit set to one. Here we need to modify the attack a little bit, by changing the length of x and the amount of "overlap" that is used in the product $\Gamma \cdot x$. Specifically, we can work with 128-bit integers x, with $x = abcdefgh$, where a is a valid high-word, h is a valid low-word, and b, c, d, e, f, g and also $i = a + g$ and $j = b + h$ are valid middle-words, as exemplified:

$$
\begin{array}{rrrrrrrrrrrrrrrrrr}
\Gamma \cdot x = & & & & & & & & & & a & b & c & d & e & f & g & h \\
 & & & & & \cdot & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
 & & & & & & & & & & a & b & c & d & e & f & g & h \\
 & & & & & & a & b & c & d & e & f & g & h & & & & \\
 & a & b & c & d & e & f & g & h & & & & & & & & & \\
\hline
 & a & b & c & d & e & f & i & j & c & d & e & f & i & j & c & d & e & f & g & h
\end{array}
$$

This gives us 84 choices for the high–mid pair (a, g), 150 choices for the mid–low pair (b, h) and 256 choices for each of c, d, e, f, so we have total of more than 2^{45} choices for x. We set

$$
\Gamma_{2048} = \sum_{i=0}^{20} 2^{96i} = 1\ \underbrace{000001\ \ldots\ 000001}_{\text{20 repetitions}}\ {}_{2^{16}}
$$

and so we get

$$
M = \Gamma_{2048} \cdot x = ab\ \underbrace{cdefij\ \ldots\ cdefij}_{\text{20 repetitions}}\ cdefgh
$$

which is again a valid encoding.

We see in Table 2 that for a 128-bit integer x, we have to generate 2^{35} integers x (therefore the 2^{45} possible choices for x are more than enough) and the attack's estimated running time is 22 years. Using the improved attack in Table 3, the running time is only 81 hours.

6.4. *The General Case*

For a modulus whose size is $16z + 1$ bits (for an even z), we need to encode messages as $16z$-bit integers, which means that the encodings should have z 16-bit words. We write the integer z as $z = \alpha \cdot m + \beta$, where α, β, m are all integers with $\alpha, \beta \geq 1$ and $m \geq 2$. For reasons that will soon become clear, we try to get $\alpha + \beta$ as small as possible, while making sure that $\alpha - \beta$ is at least 2 or 3.

The attack then works with integers x of $\alpha + \beta$ 16-bit words (which is why we want to minimize $\alpha + \beta$), and use the "overlap" of β words in the product $\Gamma \cdot x$. If we denote

$\gamma = \alpha + \beta$, then we have $x = a_\gamma \dots a_1$, where a_γ is a valid high-word, a_1 is a valid low-word, and the other a_i's are valid middle words (and we also need some of the sums to be valid middle words). We then set

$$\Gamma_{16z} = \sum_{i=0}^{m-1} 2^{16\alpha i} = 1 \quad \underbrace{0\,..\,0\,1 \quad 0\,..\,0\,1 \quad \dots \quad 0\,..\,0\,1}_{m-1 \text{ repetitions of } 0..01\ (\alpha-1\ 0\text{'s followed by } 1)} \quad .$$

When we multiply $\Gamma_{16z} \cdot x$ we get

$$\begin{array}{rr}
\Gamma_{16z} \cdot x = & a_\gamma \;..\; a_{\alpha+1}\; a_\alpha \;..\; a_\beta \;..\; a_1 \\
& \dots 0 \quad 1\,0 \quad .. \quad 0 \quad 1\,0 \quad .. \quad 0\;\;1 \\ \hline
& a_\gamma \;..\; a_{\alpha+1}\; a_\alpha \;..\; a_\beta \;..\; a_1 \\
& a_\gamma \;..\; a_{\alpha+1}\; a_\alpha \;..\; a_\beta \;..\; a_1 \qquad\qquad\qquad\qquad \\
& \dots a_\beta \;..\; a_1 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \\ \hline
\end{array}$$

hence we also need the sums $(a_\gamma + a_\beta), \dots, (a_{\alpha+2} + a_2), (a_{\alpha+1} + a_1)$ to be valid middle words.

If $\beta = 1$ (as in the case of 1025-bit moduli above), we have 64 choices for the high–low pair (a_γ, a_1) and 256 choices for each of the other a_i's, so we get total of $64 \cdot 256^{\alpha-1}$ choices for x.

If $\beta \geq 2$ (as in the case of 2049-bit moduli above), we have 84 choices for the high–mid pair (a_γ, a_β), 150 choices for the mid–low pair $(a_{\alpha+1}, a_1)$, 468 choices for each of the mid–mid pairs $(a_{\gamma-1}, a_{\beta-1}) \dots (a_{\alpha+2}, a_2)$. Thus the total number of choices for x is $84 \cdot 150 \cdot 468^{\beta-2} \cdot 256^{\alpha-\beta}$. (This is the reason for which we want $\alpha - \beta$ to be at least 2 or 3.) For the attack to be successful, we should set the parameters α, β so that there are enough smooth x's to guarantee the "homomorphic dependencies" that we need.

As another example for the general case, consider $768 + 1$-bit moduli. We need to encode the messages as 768-bit integers, or $768/16 = 48$ words. We can write $48 = 5 \cdot 9 + 3$, so we have $\alpha = 5, \beta = 3$. Hence we work with x's of $5 + 3 = 8$ words (128 bits) and use an overlap of 3 words. For this case we have $84 \cdot 150 \cdot 468 \cdot 256^2 > 2^{38}$ choices for x. Using Table 2, we see that the attack has the same complexity as for the $(2048 + 1)$-bit moduli.

6.5. *Possible Extensions*

The attack that we described above was intended to work against moduli of size $16z + 1$ bits for an even integer z, but there are a few straightforward ways to extend the attack to handle other moduli sizes. For example, for a modulus of size $16z$-bits (with z even), we should encode messages as integers with $16z - 1$ bits, which we can view as z-word integers with the highest bit set to zero and the second-highest bit set to one. To handle these integers, we re-define a *valid high-word* as a 16-bit word of the form $x = \hat{s}(u)\tilde{s}(v)uv$, for some two nibbles u, v, where $\hat{s}(u)$ is the nibble $s(u)$ with the highest bit set to zero and the second-highest bit set to one. Although we did not check this, we suspect that the modified definition of a valid high-word will not significantly change the number of high–low and high–mid pairs, so the complexity of an attack against $16z$-bit moduli should be roughly the same as that of an attack against moduli of $16z + 1$ bits.

Another extension of the attack is to consider also the cases where there are some carry bits between the nibbles in the computation of $\Gamma \cdot x$. For example, for the case of $\beta \geq 2$ (see Sect. 6.4) we can have carry bits between the "overlap" words in the multiplication without affecting the attack. We estimate that considering these carry bits can increase the number of possible x's by about a factor of $2^{\beta-1}$ (since we can have x's that cause any pattern of carry bits inside a string of length β nibbles).

Yet another plausible extension is to handle the case where not only the first and last words of the encoding have different formats, but also one other word in the middle. This is the case, for example, when we encode a message m of length less than half the size of the modulus. In that case, the form of the highest word would be $x = \bar{s}(u)s(v)uv$, the form of the lowest word would be $x = \bar{s}(u)s(v)v6$, and there would be one other word somewhere in the middle of the form $x = s(u)\tilde{s}(v)uv$. In this case we may be able to modify Γ a little, so that the spacing of the ones is not equal throughout the number. For example, if we have $x = abcd$ and $\Gamma = 10010001$, we get

$$\begin{array}{rcccccccccc}
\Gamma \cdot x = & & & & & & & a & b & c & d \\
& & & \cdot & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\hline
& & & & & & & a & b & c & d \\
& & & & a & b & c & d & & & \\
& a & b & c & d & & & & & & \\
\hline
& a & b & c & e & b & c & d & a & b & c & d
\end{array}$$

Now notice that the word e only appears once in the middle, and so we can arrange it so that it would have a different form than the other words. This technique can potentially be used to find more forgeries, or to reduce the complexity of the attack against certain moduli-lengths.

7. Second Attack against ISO/IEC 9796-1

7.1. *Introduction*

At Eurocrypt 2000, Grieu [6] presented a more efficient attack against ISO/IEC 9796-1. The attack comprises of finding pairs of message (m, m') such that

$$\frac{\mu(m)}{\mu(m')} = \frac{a}{b}$$

for some given small integers a, b. One obtains two such pairs of messages, (m_1, m_1') and (m_2, m_2'), and then using

$$\mu(m_1) \cdot \mu(m_2') = \mu(m_1') \cdot \mu(m_2)$$

it is possible to express the signature of m_1 as a function of the signatures of the three other messages.

We restrict the attack and the description of ISO/IEC 9796-1 to moduli of size k where $k \bmod 16 \in \{0, \pm 1, \pm 2\}$, and to messages of size $z = \lfloor (k+2)/16 \rfloor$ bytes, the maximum allowed message size. (Note that the attacks described in Sects. 5 and 6 were restricted to the case $k \equiv 1 \bmod 16$.)

With these restrictions, the construction of the redundant message $\mu(m)$ amounts to the local transformation of each byte m_i of the message m by an injection F_i, yielding the redundant message

$$\mu(m) = F_z(m_z) \parallel F_{z-1}(m_{z-1}) \parallel .. \parallel F_2(m_2) \parallel F_1(m_1)$$

with the injections F_i transforming an individual byte m_i of two 4 bit digits $x \parallel y$ as defined by

$$\begin{aligned} F_1(x \parallel y) &= s(x) \parallel s(y) \parallel y \parallel [\mathtt{6}]_4, \\ F_i(x \parallel y) &= s(x) \parallel s(y) \parallel x \parallel y \quad \text{for } 1 < i < z, \\ F_z(x \parallel y) &= [\mathtt{1}]_1 \parallel [s(x)]_{k+2 \bmod 16} \parallel s(y) \oplus \mathtt{1} \parallel x \parallel y \end{aligned} \tag{10}$$

where $[w]_i$ denotes the least significant i bits of w (so $[w]_i \equiv w \bmod 2^i$), and $s(x)$ is the permutation defined in Sect. 4. As we said above, the attack consists of selecting two small positive integers a, b and search for message pairs A, B that yield redundant messages satisfying

$$\frac{\mu(A)}{\mu(B)} = \frac{a}{b}. \tag{11}$$

7.2. *Choosing the Ratio a/b*

The encoding function μ imposes some restrictions on the ratio a/b that can be used for this attack. First, we can restrict our choice of a, b to $a < b$, since the ratios a/b and b/a correspond to the same message pairs (in reverse order). Similarly, we can restrict ourselves to relatively prime a, b. Also, since $\mu(A)$ and $\mu(B)$ are strings of equal length with the most significant bit set to one, we must have $b < 2a$. Next, we observe that (11) can be written as

$$\mu(B) \cdot a = \mu(A) \cdot b,$$

and since the encoding μ dictates that $\mu(B) \bmod 16 = \mu(A) \bmod 16 = 6$, it follows that we must have $6a \equiv 6b \bmod 16$, or in other words $a \equiv b \bmod 8$. Finally, in the attack below it will be convenient to assume that $a \geq 9$. Thus, in the following we restrict our choice of the ratio a/b to co-prime integers a, b with $9 \leq a < b < 2a$ and $a \equiv b \bmod 8$. Some examples of ratios a/b satisfying these requirements are $9/17$, $11/19$, and $13/21$.

7.3. *Making the Search Manageable*

Consider a hypothetical message pair A, B satisfying (11). Since the fraction a/b is chosen to be irreducible, then denoting $W = \gcd(\mu(A), \mu(B))$ we have

$$\mu(A) = a \cdot W \quad \text{and} \quad \mu(B) = b \cdot W. \tag{12}$$

We break up A, B into z bytes. We notice that our choice $9 \leq a < b$, in conjunction with the restriction we put on $k \bmod 16$, implies $W < 2^{16z}$. Thus, we can similarly break up W into z 16-bit strings

$$\begin{aligned} A &= a_z \parallel a_{z-1} \parallel .. \parallel a_2 \parallel a_1 \quad (a_i < 2^8), \\ B &= b_z \parallel b_{z-1} \parallel .. \parallel b_2 \parallel b_1 \quad (b_i < 2^8), \\ W &= w_z \parallel w_{z-1} \parallel .. \parallel w_2 \parallel w_1 \quad (w_i < 2^{16}). \end{aligned}$$

We break up each of the two multiplications appearing in (12) into z multiply and add steps operating on each of the w_i, performed from right to left, with $z-1$ steps generating an overflow to the next step, and a last step producing the remaining left $(k + 2 \bmod 16) + 13$ bits. We define the *overflows*

$$\begin{aligned} &\bar{a}_0 = \bar{a}_z = 0, && \bar{b}_0 = \bar{b}_z = 0, \\ &\bar{a}_i = \lfloor (a w_i + \bar{a}_{i-1})/2^{16} \rfloor, && \bar{b}_i = \lfloor (b w_i + \bar{b}_{i-1})/2^{16} \rfloor \quad \text{for } 1 \le i < z. \end{aligned} \tag{13}$$

The notations above can be pictorially described as follows:

$$\begin{array}{|lccccc||ccccc|} \text{overflows:} & \bar{a}_{z-1} & \bar{a}_{z-2} & .. & \bar{a}_1 & 0 & \bar{b}_{z-1} & \bar{b}_{z-2} & .. & \bar{b}_1 & 0 \\ \hline & w_z & w_{z-1} & .. & w_2 & w_1 & w_z & w_{z-1} & .. & w_2 & w_1 \\ \times & & & & & a & \times & & & & b \\ \hline & = F_z(a_z) & F_{z-1}(a_{z-1}) & .. & F_2(a_2) & F_1(a_1) & = F_z(b_z) & F_{z-1}(b_{z-1}) & .. & F_2(b_2) & F_1(b_1) \\ \hline \end{array}$$

Using these notations, we can transform (12) into the equivalent

$$\begin{aligned} &F_i(a_i) = a w_i + \bar{a}_{i-1} \bmod 2^{16}, && F_i(b_i) b w_i + \bar{b}_{i-1} \bmod 2^{16} \quad \text{for } 1 \le i < z, \\ &F_i(a_z) = a w_z + \bar{a}_{z-1}, && F_z(b_z) b w_z + \bar{b}_{z-1}. \end{aligned} \tag{14}$$

The search for message pairs A, B satisfying (11) is equivalent to the search of w_i, a_i, b_i, $\bar{a}_i$, $\bar{b}_i$ satisfying (13) and (14). This is z smaller problems, linked together by the overflows $\bar{a}_i$, $\bar{b}_i$.

7.4. *Reducing Overflows $\bar{a}_i$, $\bar{b}_i$ to One Link l_i*

Definition (13) of the overflows $\bar{a}_i$, $\bar{b}_i$ implies, by induction

$$\bar{a}_i = \left\lfloor \frac{a[W]_{16i}}{2^{16i}} \right\rfloor \quad \text{and} \quad \bar{b}_i = \left\lfloor \frac{b[W]_{16i}}{2^{16i}} \right\rfloor \quad \text{for } 1 \le i < z. \tag{15}$$

Since $0 \le [W]_{16i} < 2^{16i}$ we have

$$0 \le \bar{a}_i < a \quad \text{and} \quad 0 \le \bar{b}_i < b. \tag{16}$$

We also observe that $\bar{a}_i / \bar{b}_i$ is roughly equal to the ratio a/b, more precisely (15) implies successively

$$a \frac{[W]_{16i}}{2^{16i}} - 1 < \bar{a}_i \le a \frac{[W]_{16i}}{2^{16i}} \quad \text{and} \quad b \frac{[W]_{16i}}{2^{16i}} - 1 < \bar{b}_i \le b \frac{[W]_{16i}}{2^{16i}},$$

$$\frac{\bar{a}_i}{a} \le \frac{[W]_{16i}}{2^{16i}} < \frac{\bar{a}_i+1}{a} \quad \text{and} \quad \frac{\bar{b}_i}{b} \le \frac{[W]_{16i}}{2^{16i}} < \frac{\bar{b}_i+1}{b},$$
$$a\frac{\bar{b}_i}{b} - 1 < \bar{a}_i < a\frac{\bar{b}_i+1}{b} \quad \text{and} \quad b\frac{\bar{a}_i}{a} - 1 < \bar{b}_i < b\frac{\bar{a}_i+1}{a},$$

so, as consequence of their definition, the $\bar{a}_i, \bar{b}_i$ must satisfy

$$-a < a\bar{b}_i - b\bar{a}_i < b. \tag{17}$$

For a given $\bar{b}_i$ with $0 \le \bar{b}_i < b$, one or two $\bar{a}_i$ are solutions of (17): $\lfloor a\bar{b}_i/b \rfloor$, and $\lfloor a\bar{b}_i/b \rfloor + 1$ if and only if $a\bar{b}_i \bmod b > b - a$.

It is handy to group $\bar{a}_i, \bar{b}_i$ into a single *link* defined as

$$l_i = \bar{a}_i + \bar{b}_i + 1 \quad \text{with } 1 \le l_i < a + b \tag{18}$$

so we can rearrange (17) into

$$\bar{a}_i = \left\lfloor \frac{al_i}{a+b} \right\rfloor \quad \text{and} \quad \bar{b}_i = \left\lfloor \frac{bl_i}{a+b} \right\rfloor. \tag{19}$$

7.5. *Turning the Problem into a Graph Traversal*

For $1 \le i \le z$, we define a set of triples T_i as

$$T_i = \{(l_i, w_i, l_{i-1}) \mid \exists(a_i, b_i, \bar{a}_i, \bar{b}_i, \bar{a}_{i-1}, \bar{b}_{i-1}) \text{ satisfying (13), (14), (16), (18), (19)}\}.$$

We consider a layered graph, where the vertices in the i-th layer are all the elements of T_i, and there is an edge between the two vertices $(l_i, w, l_{i-1}) \in T_i$ and $(l'_{i-1}, w', l'_{i-2}) \in T_{i-1}$ if and only if $l_{i-1} = l'_{i-1}$. Solving (11) is equivalent to finding a connected path from an element of T_1 to an element of T_z. If this can be achieved, a suitable W is obtained by concatenating the w_i in the path, and $\mu(A)$, $\mu(B)$ follow from (12).

7.6. *Building and Traversing the Graph*

The graph can be explored in either direction with about equal ease, we describe the right to left procedure. Initially we start with the only link $l_0 = 1$. At step $i = 1$ and growing, for each of the link at the previous step, we vary b_i in range $[0, \dots, 2^8 - 1]$ and directly compute

$$w_i = \left(F_i(b_i) - \left\lfloor \frac{bl_{i-1}}{a+b} \right\rfloor\right) b^{-1} \bmod 2^{16}. \tag{20}$$

Using an inverted table of F_i we can determine in one lookup if there exist an a_i such that

$$F_i(a_i) = aw_i + \left\lfloor \frac{al_{i-1}}{a+b} \right\rfloor \bmod 2^{16} \tag{21}$$

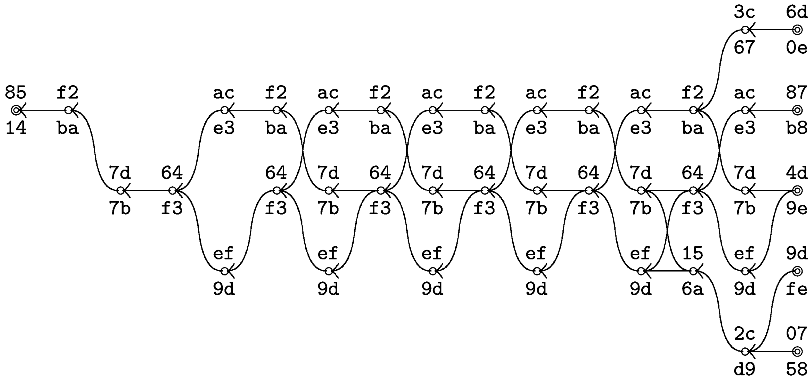

Fig. 1. Graph of solutions of (11) for $k = 256$ and $a/b = 11/19$.

and in that case we record the new triple (l_i, w_i, l_{i-1}) with the new link

$$l_i = \left\lfloor \frac{aw_i + \left\lfloor \frac{al_{i-1}}{a+b} \right\rfloor}{2^{16}} \right\rfloor + \left\lfloor \frac{bw_i + \left\lfloor \frac{bl_{i-1}}{a+b} \right\rfloor}{2^{16}} \right\rfloor + 1. \tag{22}$$

We repeat this process until a step has failed to produce any link, or we reach $i = z$ where we need to modify (20)–(22) by replacing the term 2^{16} by $2^{(k+2 \bmod 16)+13}$, and reject nodes where $l_z \neq 1$.

If we produce a link in the last step $i = z$, we can obtain a solution to (11) by backtracking any path followed, and the resulting graph covers all the solutions.

Exploration for the simplest ratio 9/17 stops on the first step, but 11/19 is more fruitful. For example, for modulus size $k = 256$, and restricting to nodes belonging to a solution, we can draw the graph in Fig. 1.

Using this graph to produce solutions to (11) is simple: message pairs are obtained by choosing a path between terminal nodes, and collecting the message bytes a_i (resp. b_i) shown above (resp. below) the nodes[1]. For example, if we follow the bottom link, the graph gives the messages:

$$A = \texttt{85f27d64ef64ef64ef64ef64ef152c07},$$

$$B = \texttt{14ba7bf39df39df39df39df39d6ad958}$$

and the redundant messages:

$$\mu(A) = \texttt{458515f2fa7d2964c1ef2964c1ef2964c1ef2964c1ef2964c1}$$
$$\texttt{ef3415572cef76},$$

$$\mu(B) = \texttt{78146bbaf67b18f3da9d18f3da9d18f3da9d18f3da9d18f3da9}$$
$$\texttt{d2b6aadd94086}$$

[1] For the sake of convenience we have shown the bytes a_i, b_i of messages A, B instead of the triples (l_i, w_i, l_{i-1}).

with indeed $\mu(A)/\mu(B) = 11/19$.

By following the upper link, we can compute another message pair C, D with the same ratio $\mu(C)/\mu(D)$, as:

$$C = \texttt{85f27d64acf27d64acf27d64acf23c6d},$$

$$D = \texttt{14ba7bf3e3ba7bf3e3ba7bf3e3ba670e}$$

which gives:

$$\mu(C) = \texttt{458515f2fA7d2964b7ac15f2fA7d2964b7ac15f2fA7d2964b7ac}$$
$$\texttt{15f2873c2ad6},$$
$$\mu(D) = \texttt{78146bbaf67b18f3c8e36bbaf67b18f3c8e36bbaf67b18f3c8e}$$
$$\texttt{36bba2f67ece6}.$$

7.7. *Existential Forgery from the Signature of Three Chosen Messages*

By selecting a ratio a/b and finding two messages pairs A, B and C, D solutions of (11), we can now construct four messages A, B, C, D as exemplified in the previous section such that

$$\mu(A) \cdot \mu(D) = \mu(B) \cdot \mu(C). \tag{23}$$

In the RSA case, this enables us to express the signature of A as a function of the other signatures:

$$\mu(A)^d = \frac{\mu(B)^d \cdot \mu(C)^d}{\mu(D)^d} \bmod N.$$

In Rabin's case, we must distinguish two cases. The first case is when we have:

$$\left(\frac{\mu(A)}{N}\right) = \left(\frac{\mu(D)}{N}\right) = -\left(\frac{\mu(B)}{N}\right) = -\left(\frac{\mu(C)}{N}\right).$$

We can assume without loss of generality that

$$\left(\frac{\mu(A)}{N}\right) = \left(\frac{\mu(D)}{N}\right) = 1.$$

Then we can write

$$\mu(A) \cdot \mu(D) = 2^2 \cdot \frac{\mu(B)}{2} \cdot \frac{\mu(C)}{2} \bmod N$$

and denoting by $\sigma_A, \sigma_B, \sigma_C, \sigma_D$ the signatures of messages A, B, C, D, we obtain

$$\sigma_A \cdot \sigma_D = 2^{2d} \cdot \sigma_B \cdot \sigma_C \bmod N.$$

Therefore, from the four signatures we obtain the value of $2^{2d} \bmod N$. As explained in Sect. 3.3, since $(\frac{2}{N}) = -1$, this allows to recover the factorization of N. Note that this can only happen if the ratio a/b is such that $(\frac{a}{N}) = -(\frac{b}{N})$.

Otherwise, one obtains the following relation between the four signatures:

$$\sigma_A \cdot \sigma_D = \sigma_B \cdot \sigma_C \bmod N$$

which enables to forge one signature knowing the three others.

7.8. *Reducing the Number of Required Signatures for Small e*

Assume that we can find two messages A, B, solution of

$$\frac{\mu(A)}{\mu(B)} = \frac{a^e}{b^e} \quad \text{with } a \neq b \tag{24}$$

for some known integers a, b. For the RSA case, we can then forge the signature of A given the signature of B:

$$\mu(A)^d = \frac{a}{b} \cdot \mu(B)^d \bmod N.$$

For the Rabin case, we can either forge the signature of A given the signature of B if $(\frac{a}{N}) = (\frac{b}{N})$, or factor N given the two signatures if $(\frac{a}{N}) = -(\frac{b}{N})$.

An example with $e = 2$ and $k = 512$ with the ratio $19^2/25^2$ is the following message pair:

$A =$ `ECE8F706C09CA276A3FC8F00803C821D90A3C03222C37DE26F5C3FD37A886FE4`,

$B =$ `CA969C94FA0B801DDEEA0C22932D80570F95A9C767D27FA8F06A56E7371B16DF`.

An example for $e = 3$ with $k = 510$ and ratio $49^3/57^3$ is:

$A =$ `C6C058A3239EE6D5ED2C4D17588B02B884A30D92B5D414DDB4B5A6DA58B6901B`,

$B =$ `20768B854644F693DB1508DE0124B4457CD7261DF699F422D9634D5E4D5781A4`.

8. Conclusion

We have shown two different attacks against the ISO/IEC 9796-1 signature standard. The first attack is based on Desmedt and Odlyzko's attack and produces a forgery with a few hundred messages. The second attack is based on a graph traversal and constructs two messages pairs whose expansion are in a common ratio; this allows to produce a forgery from only three messages. After the publication of those attacks, the ISO/IEC 9796-1 standard has been withdrawn.

$x =$	8f30	af60	8f80	bfa0	afd0	b211	d221	9241	c251	d291	92f1	a462
$y =$	0316	4316	4316	2266	1316	0d96	1ce6	1d96	0d96	2ce6	1ce6	3ba6
$z =$	9246	f276	d296	e206	c2e6	bfa7	ef07	afd7	cfe7	ff77	afd7	e008
$x =$	a4d2	94f2	d923	9943	8983	99f3	8834	a864	8884	b8a4	a8d4	8585
$y =$	4ba6	3ba6	2456	4456	2456	5316	1316	5316	5316	3266	2316	6086
$z =$	f078	d098	fd79	dd99	add9	ed09	9b4a	fb7a	db9a	eb0a	cbea	e60b
$x =$	95f5	d326	9346	8386	93f6	ae67	aed7	9ef7	8138	8138	9148	b1a8
$y =$	6086	2456	4456	2456	5316	3ba6	4ba6	3ba6	2ba6	6ad6	3ba6	4ad6
$z =$	f67b	f77c	d79c	a7dc	e70c	ea0d	fa7d	da9d	acde	ec0e	ccee	fc7e
$x =$	a1d8	cc59	8c89	ba1a	8a3a	9a4a	8a8a	caea	c75b	c7eb	97fb	b61c
$y =$	1ad6	2526	2526	4456	5456	2456	4456	2316	1ba6	0ba6	1ba6	1f76
$z =$	bcae	f17f	b1af	fe70	de90	bea0	cee0	ee00	e301	d391	b3a1	d592
$x =$	a66c	96fc	bb1d	8b3d	9b4d	8b8d	bbad	9bfd	cd5e	cdee	9dfe	b01f
$y =$	1f76	4e06	2ce6	1d96	2d96	6ce6	1ce6	2ce6	1ba6	0ba6	1ba6	4456
$z =$	c5e2	e502	e803	a8d3	c8e3	f873	d893	c8e3	e904	d994	b9a4	f475
$x =$	803f	904f	808f	c0ef								
$y =$	5456	2456	4456	2316								
$z =$	d495	b4a5	c4e5	e405								

Fig. A.1. High–low pairs (x, y) and their sum $z = x + y$.

v[1..273] =

113C2789	2103E5FE	213488FE	215041FE	21A1F6FE	23979965	23A9DF65	26013565	26182D65	261B3865
26235B65	26729D65	26EB1465	30157C81	3038C281	304D5B81	30CF6581	34045BF1	340AC4F1	34596BF1
34B660F1	34E1B0F1	34FF49F1	3814BA6A	38585D6A	3873976A	38A9396A	38E2F86A	38EEE56A	385192BD
3854A9BD	3882F7BD	389E88BD	38BB52BD	3A16E425	3A3C6125	3A797525	3A9B4E25	3AB30125	3ABFBC25
3AD30A25	3D12D3F9	3D6C4AF9	3D8AF3F9	3D91E4F9	3D9E3BF9	3DD521F9	3DE363F9	3DEDAFF9	3F09D025
3F198D25	3F3DFC25	3FCE9B25	410AB2F9	4122BDF9	412F08F9	413EDBF9	41C584F9	41EE50F9	41F296F9
4345DC55	43486155	4372C655	43793F55	4385E655	43EE7B55	4617F255	4627D755	463CF255	4665D455
468AA555	46DB9055	484B4E1A	488ED71A	48E4B91A	48EE6D1A	4A55A165	4A6F6565	4A77DA65	4A905D65
4AC74265	4AEE8465	4D069469	4D147369	4D31AB69	4D420C69	4D499369	4D532169	4D56A869	4D758769
4D84EE69	4DD22969	4F2BF565	4F2C2665	4F758F65	4FA5A565	4FD7BD65	51C43089	51DA7A89	51E7E789
590CC262	59733762	59F54062	5B07E9FA	5B9EFDFA	5BBC4BFA	5BDC93FA	5BFCCEFA	5E062FFA	5E157DFA
5E4550FA	5E7CB6FA	5E963AFA	5ED3F8FA	6015AF51	60326151	60372751	604F6B51	60708951	607F0B51
60931F51	60D7FF51	6297391A	6486D321	6496D721	64F0D121	6758901A	675ED11A	67F7F31A	6C3FB8F7
6C9916F7	6CAA47F7	6CD886F7	806BD551	806F2D51	80A83051	831D3465	833A6E65	837B2565	837F0865
83B16265	83DA9C65	840FAF21	84149621	84704721	84802A21	84A25A21	84F1E221	84FDA321	858D66B8
85EB0BB8	861A4765	8634B865	866AB865	868D6165	86AC2F65	891EF962	89220762	892C2662	893ABD62
8950EA62	89CFD062	89DA4562	8A049B55	8A27EF55	8A32DF55	8A489755	8A523055	8A7F9955	8AB3CA55
8AD3AD55	8AF88555	8DA35BBE	8DC6B0BE	8DDAC3BE	8F1F7855	8F5F5F55	8FC42755	8FEC2655	913BD36E
9158BF6E	9199DF6E	91B4856E	91D1546E	91E5696E	A0B92266	A0BA2B66	A4401E16	A4DFFF16	A4ED5A16
A4F64416	A8668A5D	AD0C6EFE	AD8124FE	ADB3D7FE	ADC5A6FE	ADDAF5FE	D00806F1	D07D68F1	D0D26DF1
D0DDC2F1	D20C395A	D25CE85A	D278785A	D2B6C25A	D2BF0D5A	D2E44D5A	D400B761	D41E1961	D4732D61
D494FC61	D4A85061	D79B1B5A	D79FAA5A	D801D7FD	D815D2FD	D868D1FD	D8F292FD	EA43E961	EA485761
EA4E1261	EB355C8A	EB37F78A	EB73DA8A	EED7308A	EEDBF58A	EEE9118A	EF784561	EF7CB861	EF8FDE61
F10F04FE	F146DAFE	F18C0CFE	F196ACFE	F1B831FE	F1CFA5FE	F1D371FE	F269861A	F26A251A	F28A8D1A
F32E2E21	F3369421	F3EB6821	F52952B8	F55C47B8	F5CC08B8	F6202521	F64ABA21	F6683921	F684CE21
F6DE0521	F6F67621	F7BDBD1A	F7D0F01A	F7D2411A	F7F60F1A	FB6E9AFA	FBA2B8FA	FBF809FA	FC8BA450
FCBC2050	FCD65150	FCEFE550	FD705E6E	FDBACE6E	FDE3756E	FE0395FA	FE0F38FA	FE0FABFA	FE2ECFFA
FE56C3FA	FE9C2EFA	FEEFA7FA							

Fig. B.1. A table of $v[i] = u_{i,1}u_{i,2}u_{i,3}u_{i,4}u_{i,5}u_{i,6}u_{i,7}u_{i,8}$.

Acknowledgements

The improved attack of Sect. 3.5 was suggested by one of the referees.

$\mathtt{b}[1..272] =$

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

$\mathtt{g}[1..272] =$

8B	89	4F	3D	20	25	1D	14	14	13	11	0F	10	0B	0D	0B	0A	0B	07	08
09	07	0B	08	0B	07	05	04	08	08	05	04	08	01	07	04	07	04	02	04
0A	05	07	07	06	05	05	04	03	05	03	04	05	04	03	04	05	05	03	04
02	03	03	02	02	02	02	02	03	02	02	02	02	01	01	02	04	05	02	02
06	04	02	01	01	04	01	02	02	01	04	03	02	02	01	02	01	02	03	02
00	02	02	02	03	02	01	01	02	03	04	03	02	02	02	02	02	01	01	02
02	05	00	00	01	01	03	01	02	02	00	01	01	02	01	00	02	03	02	01
02	02	01	01	02	02	01	02	01	03	01	00	01	01	02	01	01	02	00	02
02	00	02	00	02	01	02	01	03	01	01	01	01	03	02	00	01	01	02	02
00	01	02	01	00	01	01	01	01	01	01	01	02	01	01	01	02	01	03	02
02	01	01	01	03	03	01	00	00	01	01	02	01	01	01	01	02	02	02	01
02	01	00	01	01	00	01	02	01	02	00	01	01	02	00	04	02	01	01	01
00	02	00	01	00	00	01	00	01	00	01	01	00	00	01	00	03	00	01	00
02	03	02	01	01	01	01	01	00	02	01	02	00	00	02	02	00	01	00	01
02	02	02	01	00	01	01	02	00	02	01	02	00	01	00	00	02	01	01	01
01	01	00	01	00	01	01	02	00	01	02	00	01	03	02	00	00	02	00	01
01	00	02	00	00	00	01	00	01	01	00	01	00	01	01	00	02	01	01	00
02	00	00	00	01	01	01	02	01	01	00	00	00	00	01	01	01	00	01	01
02	02	01	01	01	01	01	00	00	01	00	00	00	01	01	01	01	00	01	00
00	01	00	00	00	02	02	00	01	00	00	00	01	01	00	00	00	02	02	00
00	00	00	01	00	00	01	00	00	00	01	01	01	00	01	02	00	01	00	00
01	01	01	01	00	01	01	01	00	00	01	01	00	00	01	00	01	00	01	01
01	00	01	00	01	00	02	00	01	00	01	00	02	01	00	00	01	00	00	00
00	00	02	01	00	00	00	01	00	00	00	00	00	00	03	00	00	01	00	00
00	01	00	00	01	02	00	00	01	00	02	00	00	00	00	02	00	01	00	00
00	00	00	00	01	01	01	00	00	01	02	00	00	00	00	01	00	00	01	00
00	00	00	00	01	01	00	01	00	00	00	01	00	01	00	00	00	01	00	00
01	01	00	00	00	00	00	01	00	01	01	00	00	01	00	01	00	00	00	00
01	01	02	00	00	00	00	01	00	00	00	01	00	01	01	00	00	00	01	00
01	00	00	01	00	02	00													

Fig. B.2. The exponents b[i] and g[i] from (B.1).

Appendix A. Useful Pairs for the Attack from Sect. 6

We provide in Fig. A.1 the list of high–low pairs (x, y) of 16-bit words, together with their sum $z = x + y$. Recall that a high–low pair (x, y) is such that x is a valid high word, y is a valid low word, and $z = x + y$ is a valid middle word. All the constants in the table are given in hexadecimal (base-16) representation.

Appendix B. A Concrete ISO/IEC 9796-1 Forgery Using the Attack from Sect. 6

The forgery is given for a 1025-bit modulus with $e = 3$. Let us denote the 112-bit constant $\Gamma = 1001001$, where each digit represents a 16-bit word.

Step 1: For $1 \leq i \leq 273$, we let $x_i = (a_i b_i c_i d_i)$ be an integer such that

$$a_i = \bar{s}(u_{i,1})\tilde{s}(u_{i,2})u_{i,1}u_{i,2},$$

$$b_i = s(u_{i,3})s(u_{i,4})u_{i,3}u_{i,4},$$

$$c_i = s(u_{i,5})s(u_{i,6})u_{i,5}u_{i,6},$$

$$d_i = s(u_{i,7})s(u_{i,8})u_{i,8}\,6$$

where $v[i] = u_{i,1}u_{i,2}u_{i,3}u_{i,4}u_{i,5}u_{i,6}u_{i,7}u_{i,8}$ is given in Fig. B.1. We obtain $M_i = \Gamma \cdot x_i$, which is a valid encoding for a message m_i, such that $M_i = \mu(m_i)$.

Step 2: Obtain the 272 signatures $s_i = \mu_{\mathrm{ISO}}(m_i)^d \bmod N$ for $1 \le i \le 272$.

Step 3: The signature of m_{273} is given by

$$\mu(m_{273})^d = \Gamma^{-139} \prod_{i=1}^{587} p_i^{-\mathtt{g}[i]} \prod_{i=1}^{272} s_i^{\mathtt{b}[i]} \bmod N, \tag{B.1}$$

where p_i is the i-th prime, and the `b[i]`'s and `g[i]`'s are given in Fig. B.2.

References

[1] D. Coppersmith, S. Halevi, C. Jutla, ISO 9796-1 and the new forgery strategy, Research contribution to P1363, 1999. Available at http://grouper.ieee.org/groups/1363/contrib.html.

[2] J.S. Coron, D. Naccache, J.P. Stern, On the security of RSA padding, in *Proceedings of Crypto '99*. Lecture Notes in Computer Science, vol. 1666 (Springer, Berlin, 1999), pp. 1–18.

[3] Y. Desmedt, A. Odlyzko, A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes, in *Proceedings of Crypto '85*. Lecture Notes in Computer Science, vol. 218 (Springer, Berlin, 1985), pp. 516–522.

[4] K. Dickman, On the frequency of numbers containing prime factors of a certain relative magnitude. *Ark. Mat. Astron. Fys.* **22A**(10), 1–14 (1930).

[5] S. Goldwasser, S. Micali, R. Rivest, A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.* **17**(2), 281–308 (1988).

[6] F. Grieu, A chosen message attack on the ISO/IEC 9796-1 signature scheme, in *Advances in Cryptology—Eurocrypt 2000*. Lecture Notes in Computer Science, vol. 1807 (Springer, Berlin, 2000), pp. 70–80.

[7] L. Guillou, J.-J. Quisquater, M. Walker, P. Landrock, C. Shaer, Precautions taken against various attacks in ISO/IEC DIS 9796, in *Proceedings of Eurocrypt' 90*. Lecture Notes in Computer Science, vol. 473 (Springer, Berlin, 1990), pp. 465–473.

[8] ISO/IEC 9796, Information Technology—Security Techniques—Digital Signature Scheme Giving Message Recovery, Part 1: Mechanisms Using Redundancy, 1991.

[9] C. Lanczos, An iterative method for the solution of the eigenvalue problem of linear differential and integral operator. *J. Res. Nat. Bur. Standards* **45**, 255–282 (1950).

[10] H.W. Lenstra Jr., Factoring integers with elliptic curves. *Ann. Math.* **126**(2), 649–673 (1987).

[11] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography* (CRC Press, Boca Raton, 1996).

[12] J.-F. Misarsky, How (not) to design RSA signature schemes, in *Public-Key Cryptography*. Lecture Notes in Computer Science, vol. 1431 (Springer, Berlin, 1998), pp. 14–28.

[13] R. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public key cryptosystems. *Commun. ACM* **21**, 120–126 (1978).

[14] V. Shoup, Number Theory C++ Library (NTL) version 5.3.1. Available at www.shoup.net.

[15] D. Stinson, *Cryptography: Theory and Practice* (CRC Press, Boca Raton, 1995).