Practical Cryptanalysis of ISO/IEC 9796-2 and EMV Signatures

Jean-Sébastien Coron[1], David Naccache[2],
Mehdi Tibouchi[2], and Ralf-Philipp Weinmann[1]

[1] Université du Luxembourg
6, rue Richard Coudenhove-Kalergi
L-1359 Luxembourg, Luxembourg
{jean-sebastien.coron,ralf-philipp.weinmann}@uni.lu
[2] École normale supérieure
Département d'informatique, Groupe de Cryptographie
45, rue d'Ulm, F-75230 Paris CEDEX 05, France
{david.naccache,mehdi.tibouchi}@ens.fr

Abstract. In 1999, Coron, Naccache and Stern discovered an existential signature forgery for two popular RSA signature standards, ISO/IEC 9796-1 and 2. Following this attack ISO/IEC 9796-1 was withdrawn. ISO/IEC 9796-2 was amended by increasing the message digest to at least 160 bits. Attacking this amended version required at least 2^{61} operations.

In this paper, we exhibit algorithmic refinements allowing to attack the *amended* (currently valid) version of ISO/IEC 9796-2 for all modulus sizes. A practical forgery was computed in only two days using 19 servers on the Amazon EC2 grid for a total cost of $\simeq$ US$800. The forgery was implemented for $e = 2$ but attacking odd exponents will not take longer. The forgery was computed for the RSA-2048 challenge modulus, whose factorization is still unknown.

The new attack blends several theoretical tools. These do not change the asymptotic complexity of Coron *et al.*'s technique but significantly accelerate it for parameter values previously considered beyond reach.

While less efficient (US$45,000), the acceleration also extends to EMV signatures. EMV is an ISO/IEC 9796-2-compliant format with extra redundancy. Luckily, this attack does not threaten any of the 730 million EMV payment cards in circulation for *operational* reasons.

Costs are per modulus: after a first forgery for a given modulus, obtaining more forgeries is virtually immediate.

Keywords: digital signatures, forgery, RSA, public-key cryptanalysis, ISO/IEC 9796-2, EMV.

1 Introduction

RSA [34] is certainly the most popular public-key cryptosystem. A chosen-ciphertext attack against RSA textbook encryption was described by Desmedt and

S. Halevi (Ed.): CRYPTO 2009, LNCS 5677, pp. 428–444, 2009.

Odlyzko in [17]. In RSA textbook encryption, a message m is simply encrypted as:

$$c = m^e \mod N$$

where N is the RSA modulus and e is the public exponent.

As noted in [31], Desmedt and Odlyzko's attack also applies to RSA signatures:

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

where $\mu(m)$ is an encoding function and d the private exponent. Desmedt and Odlyzko's attack only applies if the encoding function $\mu(m)$ produces integers much smaller than N. In which case, one obtains an existential forgery under a chosen-message attack. In this attack the opponent can ask for signatures of any messages of his choosing before computing, by his own means, the signature of a (possibly meaningless) message which was never signed by the legitimate owner of d.

As of today, two encoding function species co-exist:

1. *Ad-hoc encodings* are "handcrafted" to thwart certain classes of attacks. While still in use, *ad-hoc* encodings are currently being phased-out. PKCS#1 v1.5 [26], ISO/IEC 9796-1 [22] and ISO/IEC 9796-2 [23,24] are typical *ad-hoc* encoding examples.
2. *Provably secure encodings* are designed to make cryptanalysis equivalent to inverting RSA (possibly under additional assumptions such as the Random Oracle Model [2]). OAEP [3] (for encryption) and PSS [4] (for signature) are typical provably secure encoding examples.

For *ad-hoc* encodings, there is no guarantee that forging signatures is as hard as inverting RSA. And as a matter of fact, many such encodings were found to be weaker than the RSA problem. We refer the reader to [8,11,10,25,15,20] for a few characteristic examples. It is thus a practitioner's rule of thumb to use provably secure encodings whenever possible. Nonetheless, *ad-hoc* encodings continue to populate hundreds of millions of commercial products (*e.g.* EMV cards) for a variety of practical reasons. A periodic re-evaluation of such encodings is hence necessary.

ISO/IEC 9796-2 is a specific μ-function standardized by ISO [23]. In [16], Coron, Naccache and Stern discovered an attack against ISO/IEC 9796-2. Their attack is an adaptation of Desmedt and Odlyzko's cryptanalysis which could not be applied directly since in ISO/IEC 9796-2, the encoding $\mu(m)$ is almost as large as N. ISO/IEC 9796-2 can be used with hash-functions of diverse digest-sizes k_h. Coron *et al.* estimated that attacking $k_h = 128$ and $k_h = 160$ will require (respectively) 2^{54} and 2^{61} operations. After Coron *et al.*'s publication ISO/IEC 9796-2 was amended and the current official requirement (see [24]) is $k_h \geq 160$. It was shown in [13] that ISO/IEC 9796-2 can be proven secure (in the random oracle model) for $e = 2$ and if the digest size k_h is a least 2/3 the size of the modulus.

In this paper, we describe an improved attack against the currently valid (amended) version of ISO/IEC 9796-2, that is for $k_h = 160$. The new attack

applies to EMV signatures as well. EMV is an ISO/IEC 9796-2-compliant format with extra redundancy. The attack is a Coron *et al.* forgery with new refinements: better message choice, Bernstein's smoothness detection algorithm (instead of trial division), large prime variant and optimized exhaustive search.

Using these refinements, a forgery for ISO/IEC 9796-2 was computed in only two days, using a few dozens of servers on the Amazon EC2 grid, for a total cost of US$800. The forgery was implemented for $e = 2$ but attacking odd exponents will not take longer. We estimate that under similar conditions an EMV signature forgery would cost US$45,000. Note that all costs are per modulus. After computing a first forgery for a given N, additional forgeries come at a negligible cost.

2 The ISO/IEC 9796-2 Standard

ISO/IEC 9796-2 is an encoding standard allowing partial or total message recovery [23,24]. Here we consider only partial message recovery. As we have already mentioned, ISO/IEC 9796-2 can be used with hash-functions $\text{HASH}(m)$ of diverse digest-sizes k_h. For the sake of simplicity we assume that k_h, the size of m and the size of N (denoted k) are all multiples of 8; this is also the case in the EMV specifications.

The ISO/IEC 9796-2 encoding function is:

$$\mu(m) = \texttt{6A}_{16} \| m[1] \| \text{HASH}(m) \| \texttt{BC}_{16}$$

where the message $m = m[1]\|m[2]$ is split in two: $m[1]$ consists of the $k - k_h - 16$ leftmost bits of m and $m[2]$ represents all the remaining bits of m. The size of $\mu(m)$ is therefore always $k - 1$ bits.

The original version of the standard recommended $128 \leq k_h \leq 160$ for partial message recovery (see [23], §5, note 4). The new version of ISO/IEC 9796-2 [24] requires $k_h \geq 160$. The EMV specifications also use $k_h = 160$.

3 Desmedt-Odlyzko's Attack

In Desmedt and Odlyzko's attack [31] (existential forgery under a chosen-message attack), the forger asks for the signature of messages of his choice before computing, by his own means, the signature of a (possibly meaningless) message that was never signed by the legitimate owner of d. In the case of Rabin-Williams signatures (see the full version of the paper [12]), it may even happen that the attacker factors N; *i.e.* a total break.

The attack only applies if $\mu(m)$ is much smaller than N and works as follows:

1. Select a bound B and let $\mathfrak{P} = \{p_1, \ldots, p_\ell\}$ be the list of all primes smaller than B.
2. Find at least $\ell + 1$ messages m_i such that each $\mu(m_i)$ is a product of primes in $\mathfrak{P}$.

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 $\mathfrak{P}$.
4. Ask for the signatures of the m_i for $i \neq j$ and forge the signature of m_j.

In the following we assume that e is prime; this includes $e = 2$. 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 B-smooth and we can write for all messages m_i, $1 \leq i \leq \tau$:

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

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 prime, the set of all ℓ-dimensional vectors modulo e forms 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 $1 \leq j \leq \ell$:

$$\boldsymbol{V}_\tau = \boldsymbol{\Gamma} \cdot e + \sum_{i=1}^{\tau-1} \beta_i \boldsymbol{V}_i$$

for some $\boldsymbol{\Gamma} = (\gamma_1, \ldots, \gamma_\ell) \in \mathbb{Z}^\ell$. That is,

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

Then using (1), one obtains :

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

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

That is:

$$\mu(m_\tau) = \delta^e \cdot \prod_{i=1}^{\tau-1} \mu(m_i)^{\beta_i}, \quad \text{where we denote: } \delta = \prod_{j=1}^{\ell} p_j^{\gamma_j} \tag{2}$$

Therefore, we see that $\mu(m_\tau)$ can be written as a multiplicative combination of the other $\mu(m_i)$. For RSA signatures, the attacker will ask for the signatures of $m_1, \ldots, m_{\tau-1}$ and forge the signature of m_τ using the relation:

$$\sigma_\tau = \mu(m_\tau)^d = \delta \cdot \prod_{i=1}^{\tau-1} \left(\mu(m_i)^d\right)^{\beta_i} = \delta \cdot \prod_{i=1}^{\tau-1} \sigma_i^{\beta_i} \mod N$$

In the full version of the paper [12] we describe the corresponding forgery for Rabin-Williams signatures, where, in some cases, the attacker may even factor N.

The attack's complexity depends on ℓ and on the probability that the integers $\mu(m_i)$ are B-smooth. The reader is referred to the full version of the paper [12] for a complexity analysis (see also [14]). In practice, the attack is feasible only if the $\mu(m_i)$ are relatively small (*e.g.* less than 200 bits).

4 Coron-Naccache-Stern's Attack

In ISO/IEC 9796-2, the encoding function's output $\mu(m)$ is as long as N. This thwarts Desmedt and Odlyzko's attack. Coron-Naccache-Stern's workaround [16] consisted in generating messages m_i such that a *linear combination* t_i of $\mu(m_i)$ and N is much smaller than N. Then, the attack can be applied to the integers t_i instead of $\mu(m_i)$.

More precisely, Coron *et al.* observed that it is sufficient to find a constant a and messages m_i such that:

$$t_i = a \cdot \mu(m_i) \mod N$$

is small, instead of requiring that $\mu(m_i)$ is small. Namely, the factor a can be easily dealt with by regarding $a^{-1} \bmod N$ as an "additional factor" in $\mu(m_i)$; to that end we only need to add one more column in the matrix considered in Section 3. In their attack Coron *et al.* used $a = 2^8$.

Obtaining a small $a \cdot \mu(m) \bmod N$ is done in [16] as follows. From the definition of ISO/IEC 9796-2:

$$\begin{aligned} \mu(m) &= \mathtt{6A_{16}} \quad \| \; m[1] \quad \| \; \text{HASH}(m) \quad \| \; \mathtt{BC_{16}} \\ &= \mathtt{6A_{16}} \cdot 2^{k-8} + m[1] \cdot 2^{k_h+8} + \text{HASH}(m) \cdot 2^8 + \mathtt{BC_{16}} \end{aligned}$$

Euclidean division by N provides b and $0 \leq r < N < 2^k$ such that:

$$(\mathtt{6A_{16}} + 1) \cdot 2^k = b \cdot N + r$$

Denoting $N' = b \cdot N$ one can write:

$$\begin{aligned} N' &= \mathtt{6A_{16}} \cdot 2^k + \quad (2^k - r) \\ &= \mathtt{6A_{16}} \quad \| \; N'[1] \| N'[0] \end{aligned}$$

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

Consider the linear combination:

$$\begin{aligned} t &= b \cdot N - a \cdot \mu(m) \\ &= N' - 2^8 \cdot \mu(m) \end{aligned}$$

By setting $m[1] = N'[1]$ we get:

$$\begin{aligned} t = \quad & \mathtt{6A}_{16} \parallel N'[1] \parallel N'[0] \\ & - \mathtt{6A}_{16} \parallel m[1] \parallel \text{HASH}(m) \| \mathtt{BC}\mathtt{00}_{16} \\ = \quad & \cancel{\mathtt{6A}_{16}} \parallel \cancel{N'[1]} \parallel N'[0] \\ & - \cancel{\mathtt{6A}_{16}} \parallel \cancel{N'[1]} \parallel \text{HASH}(m) \| \mathtt{BC}\mathtt{00}_{16} \\ = \quad & N'[0] - (\text{HASH}(m) \| \mathtt{BC}\mathtt{00}_{16}) < 2^{k_h+16} \end{aligned}$$

For $k_h = 160$, the integer t is therefore at most 176-bits long.

The forger can thus modify $m[2]$ (and therefore HASH(m)), until he gets a set of messages whose t-values are B-smooth and express one such $\mu(m_\tau)$ as a multiplicative combination of the others. As per the analysis in [16], attacking the instances $k_h = 128$ and $k_h = 160$ requires (respectively) 2^{54} and 2^{61} operations.

5 The New Attack's Building-Blocks

We improve the above complexities by using four new ideas: we accelerate Desmedt-Odlyzko's process using Bernstein's smoothness detection algorithm [6], instead of trial division; we also use the large prime variant [1]; moreover, we modify Coron *et al.*'s attack by selecting better messages and by optimizing exhaustive search to equilibrate complexities. In this section we present these new building-blocks.

5.1 Bernstein's Smoothness Detection Algorithm

Bernstein [6] describes the following algorithm for finding smooth integers.

Algorithm: Given prime numbers $p_1, \ldots, p_\ell$ in increasing order and positive integers $t_1, \ldots, t_n$, output the p_ℓ-smooth part of each t_k:

1. Compute $z \leftarrow p_1 \times \cdots \times p_\ell$ using a product tree.
2. Compute $z_1 \leftarrow z \bmod t_1, \ldots, z_n \leftarrow z \bmod t_n$ using a remainder tree.
3. For each $k \in \{1, \ldots, n\}$: Compute $y_k \leftarrow (z_k)^{2^e} \bmod t_k$ by repeated squaring, where e is the smallest non-negative integer such that $2^{2^e} \geq t_k$.
4. For each $k \in \{1, \ldots, n\}$: output $\gcd(t_k, y_k)$.

We refer the reader to [5] for a description of the product and remainder trees.

Theorem 1 (Bernstein). *The algorithm computes the p_ℓ-smooth part of each integer t_k, in $\mathcal{O}(b \log^2 b \log\log b)$ time, where b is the number of input bits.*

In other words, given a list of n_t integers $t_i < 2^a$ and the list of the first ℓ primes, the algorithm will detect the B-smooth t_i's, where $B = p_\ell$, in complexity:

$$\mathcal{O}(b \cdot \log^2 b \cdot \log\log b)$$

where $b = n_t \cdot a + \ell \cdot \log_2 \ell$ is the total number of input bits.

When n_t is very large, it becomes more efficient to run the algorithm k times, on batches of $n'_t = n_t/k$ integers. We explain in the full version of the paper [12] how to select the optimal n'_t, and derive the corresponding running time.

Bernstein recommends a number of speed-up ideas of which we used a few. In our experiments we used the *scaled remainder tree* [7], which replaces most division steps in the remainder tree by multiplications. This algorithm is fastest when FFT multiplications are done modulo numbers of the form $2^\alpha - 1$: we used this *Mersenne* FFT *multiplication* as well, as implemented in Gaudry, Kruppa and Zimmermann's GMP patch [19]. Other optimizations included computing the product z only once, and treating the prime 2 separately.

Bernstein's algorithm was actually the main source of the attack's improvement. It proved $\simeq 1000$ faster than the trial division used in [16].

5.2 The Large Prime Variant

An integer is *semi-smooth* with respect to y and z if its greatest prime factor is $\leq y$ and all other factors are $\leq z$. Bach and Peralta [1] define the function $\sigma(u, v)$, which plays for semi-smoothness the role played by Dickman's ρ function for smoothness (see the full version of the paper [12]): $\sigma(u, v)$ is the asymptotic probability that an integer n is semi-smooth with respect to $n^{1/v}$ and $n^{1/u}$.

After an integer t_i has had all its factors smaller than B stripped-off, if the remaining factor ω is lesser than B^2 then ω must be prime. This is very easy to detect using Bernstein's algorithm. As Bernstein computes the B-smooth part z_i of each t_i, it only remains to check whether t_i/z_i is small enough. In most cases it isn't even necessary to perform the actual division since comparing the sizes of t_i and z_i suffices to rule out most non-semi-smooth numbers.

Hence, one can use a second bound B_2 such that $B < B_2 < B^2$ and keep the t_i's whose remaining factor ω is $\leq B_2$, hoping to find a second t_i with the same remaining factor ω to divide ω out. We refer the reader to the full version of the paper [12] for a detailed analysis of the large prime variant in our context.

5.3 Constructing Smaller $a \cdot \mu(m) - b \cdot N$ Candidates

In this paragraph we show how to construct smaller $t_i = a \cdot \mu(m_i) - b \cdot N$ values for ISO/IEC 9796-2. Smaller t_i-values increase smoothness probability and hence accelerate the forgery process.

We write:

$$\mu(x, h) = \mathtt{6A_{16}} \cdot 2^{k-8} + x \cdot 2^{k_h+8} + h \cdot 2^8 + \mathtt{BC_{16}}$$

where $x = m[1]$ and $h = \text{HASH}(m)$, with $0 < x < 2^{k-k_h-16}$.

We first determine $a, b > 0$ such that the following two conditions hold:

$$0 < b \cdot N - a \cdot \mu(0,0) < a \cdot 2^{k-8} \quad (3)$$

$$b \cdot N - a \cdot \mu(0,0) = 0 \mod 2^8 \quad (4)$$

and a is of minimal size. Then by Euclidean division we compute x and r such that:

$$b \cdot N - a \cdot \mu(0,0) = (a \cdot 2^{k_h+8}) \cdot x + r$$

where $0 \leq r < a \cdot 2^{k_h+8}$ and using (3) we have $0 \leq x < 2^{k-k_h-16}$ as required. This gives:

$$b \cdot N - a \cdot \mu(x,0) = b \cdot N - a \cdot \mu(0,0) - a \cdot x \cdot 2^{k_h+8} = r$$

Moreover as per (4) we must have $r = 0 \mod 2^8$; denoting $r' = r/2^8$ we obtain:

$$b \cdot N - a \cdot \mu(x,h) = r - a \cdot h \cdot 2^8 = 2^8 \cdot (r' - a \cdot h)$$

where $0 \leq r' < a \cdot 2^{k_h}$. We then look for smooth values of $r' - a \cdot h$, whose size is at most k_h plus the size of a.

If a and b are both 8-bit integers, this gives 16 bits of freedom to satisfy both conditions (3) and (4); heuristically each of the two conditions is satisfied with probability $\simeq 2^{-8}$; therefore, we can expect to find such an $\{a, b\}$ pair. For example, for the RSA-2048 challenge, we found $\{a, b\}$ to be $\{625, 332\}$; therefore, for RSA-2048 and $k_h = 160$, the integer to be smooth is 170-bits long (instead of 176-bits in Coron *et al.*'s original attack). This decreases further the attack's complexity.

6 Attacking ISO/IEC 9796-2

We combined all the building-blocks listed in the previous section to compute an actual forgery for ISO/IEC 9796-2, with the RSA-2048 challenge modulus. The implementation replaced Coron *et al.*'s trial division by Bernstein's algorithm, replaced Coron *et al.*'s $a \cdot \mu(m) - b \cdot N$ values by the shorter t_i's introduced in paragraph 5.3 and took advantage of the large prime variant. Additional speed-up was obtained by exhaustive searching for particular digest values. Code was written in C++ and run on 19 Linux-based machines on the Amazon EC2 grid. The final linear algebra step was performed on a single PC.

6.1 The Amazon Grid

Amazon.com Inc. offers virtualized computer instances for rent on a pay by the hour basis, which we found convenient to run our computations. Various models are available, of which the best-suited for CPU-intensive tasks, as we write these lines, features 8 Intel Xeon 64-bit cores clocked at 2.4GHz supporting the Core2 instruction set and offering 7GB RAM and 1.5TB disk space. Renting such a capacity costs US$0.80 per hour (plus tax). One can run up to 20 such instances

in parallel, and possibly more subject to approval by Amazon (20 were enough for our purpose so we didn't apply for more).

When an instance on the grid is launched, it starts up from a disk image containing a customizable UNIX operating system. In the experiment, we ran a first instance using the basic Fedora installation provided by default, installed necessary tools and libraries, compiled our own programs and made a disk image containing our code, to launch subsequent instances with. When an instance terminates, its disk space is freed, making it necessary to save results to some permanent storage means. We simply `rsync`'ed results to a machine of ours. Note that Amazon also charges for network bandwidth but data transmission costs were negligible in our case.

All in all, we used about 1,100 instance running hours (including setup and tweaks) during a little more than two days. While we found the service to be rather reliable, one instance failed halfway through the computation, and its intermediate results were lost.

6.2 The Experiment: Outline, Details and Results

The attack can be broken down into the following elementary steps, which we shall review in turn:

1. Determining the constants $a, b, x, \mu(x, 0)$ for the RSA-2048 challenge modulus N.
2. Computing the product of the first ℓ primes, for a suitable choice of ℓ.
3. Computing the integers $t_i = bN - a\mu(m_i)$, and hence the SHA-1 digests, for sufficiently many messages m_i.
4. Finding the smooth and semi-smooth integers amongst the t_i's.
5. Factoring the smooth integers, as well as the colliding pairs of semi-smooth integers, obtaining the sparse, singular matrix of exponents, with ℓ rows and more than ℓ columns.
6. Reducing this matrix modulo $e = 2$, with possible changes in the first row (corresponding to the prime 2) depending on the Jacobi symbols $(2|t_i)$ and $(2|a)$.
7. Finding nontrivial vectors in the kernel of this reduced matrix and inferring a forgery.

Steps 2–4 were executed on the Amazon EC2 grid, whereas all other steps were run on one offline PC. Steps 3–4, and to a much lesser extent step 7, were the only steps that claimed a significant amount of CPU time.

Determining the constants. The attack's complexity doesn't depend on the choice of N. Since N has to be congruent to 5 mod 8 for Rabin-Williams signatures, we used the RSA-2048 challenge. The resulting constants were computed in SAGE [35]. We found the smallest $\{a, b\}$ pair to be $\{625, 332\}$, and the $\mu(x, 0)$ value given in the full version of the paper [12]. The integers $t_i = bN - a\mu(x, h_i)$ are thus at most 170-bits long.

Product of the first primes. The optimal choice of ℓ for 170 bits is about 2^{21}. Since the Amazon instances are memory-constrained (less than 1GB of RAM per core), we preferred to use $\ell = 2^{20}$. This choice had the additional advantage of making the final linear algebra step faster, which is convenient since this step was run on a single off-line PC. Computing the product of primes itself was done once and for all in a matter of seconds using MPIR.

Hashing. Since the attack's smoothness detection part works on batches of t_i's (in our cases, we chose batches of 2^{19} integers), we had to compute digests of messages m_i in batches as well. The messages themselves are 2048-bit long, *i.e.* as long as N, and comply with the structure indicated in the full version of the paper [12]: a constant 246-byte prefix followed by a 10-byte seed. The first two bytes identify a family of messages examined on a single core of one Amazon instance, and the remaining eight bytes are explored by increments of 1 starting from 0.

Messages were hashed using OpenSSL's SHA-1 implementation. For each message, we only need to compute one SHA-1 block, since the first three 64-byte blocks are fixed. This computation is relatively fast compared to Bernstein's algorithm, so we have a bit of leeway for exhaustive search. We can compute a large number of digests, keeping the ones likely to give rise to a smooth t_i. We did this by selecting digests for which the resulting t_i would have many zeroes as leading and trailing bits.

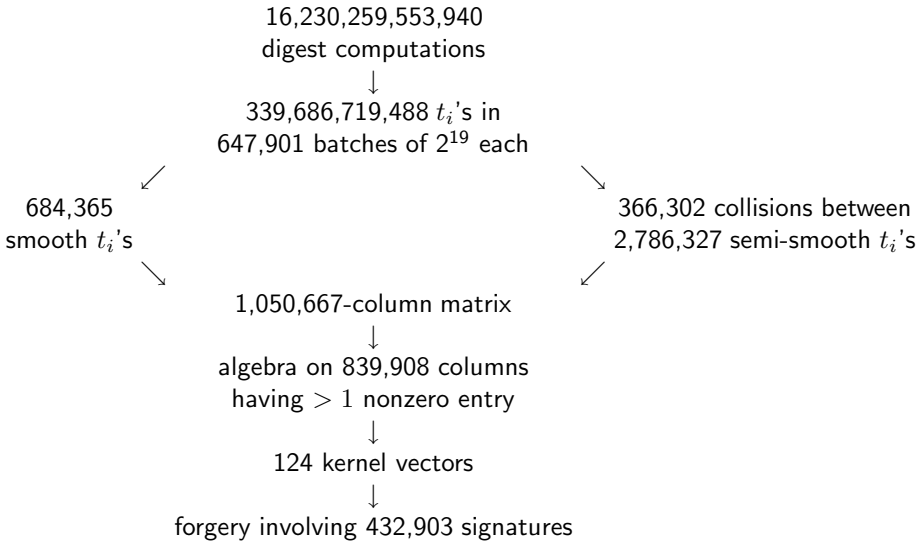
More precisely, we looked for a particular bit pattern at the beginning and at the end of each digest h_i, such that finding n matching bits results in n null bits at the beginning and at the end of t_i. The probability of finding n matching bits when we add the number of matches at the beginning and at the end is $(1 + n/2) \cdot 2^{-n}$, so we expect to compute $2^n/(1 + n/2)$ digests per selected message. We found $n = 8$ to be optimal: on average, we need *circa* 50 digests to find a match, and the resulting t_i is at most $170 - 8 = 162$ bit long (once powers of 2 are factored out).

Note that faster (*e.g.* hardware-enhanced) ways to obtain digests can significantly reduce the attack's complexity (*cf.* the full version of the paper [12]). We considered for example an FPGA-based solution called COPACOBANA [32], which could in principle perform a larger amount of exhaustive search, and accelerate the attack dramatically. It turned out that our attack was fast enough, hence pursuing the hardware-assisted search idea further proved unnecessary, but a practical attack on EMV (*cf.* section 8) could certainly benefit from hardware acceleration.

Finding smooth and semi-smooth integers. Once a batch of 2^{19} appropriate t_i's is generated, we factor out powers of 2, and feed the resulting odd numbers into our C++ implementation of Bernstein's algorithm. This implementation uses the MPIR multi-precision arithmetic library [21], which we chose over vanilla GMP because of a number of speed improvements, including J.W. Martin's patch for the Core2 architecture. We further applied Gaudry, Kruppa and Zimmermann's FFT patch, mainly for their implementation of *Mersenne* FFT multiplication, which is useful in the scaled remainder tree [7].

We looked for B-smooth as well as for (B, B_2)-semi-smooth t_i's, where $B =$ 16,290,047 is the 2^{20}-th prime, and $B_2 = 2^{27}$. Each batch took $\simeq 40$ seconds to generate and to process, and consumed about 500MB of memory. We ran 8 such processes in parallel on each instance to take advantage of the 8 cores, and 19 instances simultaneously.

The entire experiment can be summarized as follows:



Finding the 1,050,667 columns (slightly in excess of the $\ell = 2^{20} = 1{,}048{,}576$ required) took a little over 2 days.

Factoring and finding collisions. The output of the previous stage is a large set of text files containing the smooth and semi-smooth t_i's together with the corresponding message numbers. Turning this data into a matrix suitable for the linear algebra stage mostly involved text manipulation in Perl to convert it to commands that could be piped into PARI/GP [33]. The resulting $1{,}048{,}576 \times 1{,}050{,}667$ matrix had 14,215,602 non-zero entries (13.5 per column on average, or 10^{-5} sparsity; the columns derived from the large prime variant tend to have twice as many non-zero entries, of course).

Linear algebra. We found non-zero kernel elements of the final sparse matrix over GF(2) using Coppersmith's block Wiedemann algorithm [9] implemented in WLSS2 [27,30], with parameters $m = n = 4$ and $\kappa = 2$. The whole computation took 16 hours on one 2.7GHz personal computer, with the first (and longest) part of the computation using 2 cores, and the final part using 4 cores.

The program discovered 124 kernel vectors with Hamming weights ranging from 337,458 to 339,641. Since columns obtained from pairs of semi-smooth numbers account for two signatures each, the number of signature queries required to produce the 124 corresponding forgeries is slightly larger, and ranges between 432,903 and 435,859.

Being written with the quadratic sieve in mind, the block Wiedemann algorithm in WLSS2 works over GF(2). There exist, however, other implementations for different finite fields.

Evidencing forgery. An interesting question is that of exhibiting a *compact evidence of forgery*. In the full version of the paper [12] we exhibit statistical evidence that a multiplicative relation between ISO/IEC 9796-2 signatures, (*i.e.* a forgery) was indeed constructed.

Fewer signature queries. In the full version of the paper [12] we address the question of reducing the number of signature queries in the attack.

7 Cost Estimates

The experiment described in the previous section can be used as a benchmark to estimate the attack's cost as a function of the size of the t_i's, denoted a; this will be useful for analyzing the security of the EMV specifications, where a is bigger (204 bits instead of 170 bits).

Table 1. Bernstein+Large prime variant. Estimated parameter trade-offs, running times and costs, for various t_i sizes.

$a = \log_2 t_i$	$\log_2 \ell$	Estimated TotalTime	$\log_2 \tau$	EC2 cost (US$)
64	11	15 seconds	20	negligible
128	19	4 days	33	10
160	21	6 months	38	470
170	22	1.8 years	40	1,620
176	23	3.8 years	41	3,300
204	25	95 years	45	84,000
232	27	19 centuries	49	1,700,000
256	30	320 centuries	52	20,000,000

Results are summarized in Table 1. We assume that the t_i's are uniformly distributed a-bit integers and express costs as a function of a. Cost figures do not include the linear algebra step whose computational requirements are very low compared to the smoothness detection step. Another difference with our experiment is that here we do not assume any exhaustive search on the t_i's; this is why the cost estimate for $a = 170$ in Table 1 is about the double of the cost of our experimental ISO/IEC 9796-2 forgery.

Running times are given for a single 2.4GHz PC. Costs correspond to the Amazon EC2 grid, as in the previous section. Estimates show that the attack is feasible up to $\simeq 200$ bits, but becomes infeasible for larger values of a. We also estimate $\log_2 \tau$, where τ is the number of messages in the forgery.

8 Application to EMV Signatures

EMV is a collection of industry specifications for the inter-operation of payment cards, POS terminals and ATMs. The EMV specifications [18] rely on ISO/IEC 9796-2 signatures to certify public-keys and to authenticate data. For instance, when an Issuer provides application data to a Card, this data must be signed using the Issuer's private key S_{I}. The corresponding public-key P_{I} must be signed by a Certification Authority (CA) whose public-key is denoted P_{CA}. The signature algorithm is RSA with $e = 3$ or $e = 2^{16} + 1$. The bit length of all moduli is always a multiple of 8.

EMV uses special message formats; 7 different formats are used, depending on the message type. We first describe one of these formats: the *Static Data Authentication, Issuer Public-key Data* (SDA-IPKD), and adapt our attack to it. We then consider the other six formats.

8.1 EMV Static Data Authentication, Issuer Public-key Data (SDA-IPKD)

We refer the reader to §5.1, Table 2, page 41 in EMV [18]. SDA-IPKD is used by the CA to sign the issuer's public-key P_{I}. The message to be signed is as follows:

$$m = \texttt{02}_{16} \| X \| Y \| N_{\text{I}} \| \texttt{03}_{16}$$

where X represents 6 bytes that can be controlled by the adversary and Y represents 7 bytes that cannot be controlled. N_{I} is the Issuer's modulus to be certified. More precisely, $X = \text{ID} \| \text{DATE}$ where ID is the issuer identifier (4 bytes) and DATE is the *Certificate Expiration Date* (2 bytes); we assume that both can be controlled by the adversary. $Y = \text{CSN} \| C$ where CSN is the 3-bytes *Certificate Serial Number* assigned by the CA and C is a constant. Finally, the modulus to be certified N_{I} can also be controlled by the adversary.

With ISO/IEC 9796-2 encoding, this gives:

$$\mu(m) = \texttt{6A02}_{16} \| X \| Y \| N_{\text{I},1} \| \text{HASH}(m) \| \texttt{BC}_{16}$$

where $N_{\text{I}} = N_{\text{I},1} \| N_{\text{I},2}$ and the size of $N_{\text{I},1}$ is $k - k_h - 128$ bits. k denotes the modulus size and $k_h = 160$ as in ISO/IEC 9796-2.

8.2 Attacking SDA-IPKD

To attack SDA-IPKD write:

$$\mu(X, N_{\text{I},1}, h) = \texttt{6A02}_{16} \cdot 2^{k_1} + X \cdot 2^{k_2} + Y \cdot 2^{k_3} + N_{\text{I},1} \cdot 2^{k_4} + h$$

where Y is constant and $h = \text{HASH}(m) \| \texttt{BC}_{16}$. We have:

$$\begin{cases} k_1 = k \ - 16 \\ k_2 = k_1 - 48 = k - 64 \\ k_3 = k_2 - 56 = k - 120 \\ k_4 = k_h + \ 8 \ = 168 \end{cases}$$

Generate a random k_a-bit integer a, where $36 \leq k_a \leq 72$, and consider the equation:

$$b \cdot N - a \cdot \mu(X, 0, 0) = b \cdot N - a \cdot X \cdot 2^{k_2} - a \cdot (\texttt{6A02}_{16} \cdot 2^{k_1} + Y \cdot 2^{k_3})$$

If we can find integers X and b such that $0 \leq X < 2^{48}$ and:

$$0 \leq b \cdot N - a \cdot \mu(X, 0, 0) < a \cdot 2^{k_3} \tag{5}$$

then as previously we can compute $N_{\text{I},1}$ by Euclidean division:

$$b \cdot N - a \cdot \mu(X, 0, 0) = (a \cdot 2^{k_4}) \cdot N_{\text{I},1} + r \tag{6}$$

where $0 \leq N_{\text{I},1} < 2^{k_3 - k_4}$ as required, and the resulting $b \cdot N - a \cdot \mu(X, N_{\text{I},1}, h)$ value will be small for all values of h.

In the above we assumed Y to be a constant. Actually the first 3 bytes of Y encode the CSN assigned by the CA, and may be different for each new certificate (see the full version of the paper [12]). However if the attacker can predict the CSN, then he can compute a different a for every Y and adapt the attack by factoring a into a product of small primes.

Finding small X and b so as to minimize the value of

$$|b \cdot N - a \cdot X \cdot 2^{k_2} - a \cdot (\texttt{6A02}_{16} \cdot 2^{k_1} + Y \cdot 2^{k_3})|$$

is a Closest Vector Problem (CVP) in a bi-dimensional lattice; a problem that can be easily solved using the LLL algorithm [28]. We first determine heuristically the minimal size that can be expected; we describe the LLL attack in the full version of the paper [12].

Since $a \cdot \texttt{6A02}_{16} \cdot 2^{k_1}$ is an $(k + k_a)$-bit integer, with $X \simeq 2^{48}$ and $b \simeq 2^{k_a}$, odds are heuristically reasonable to find X and b such that:

$$0 \leq b \cdot N - a \cdot \mu(X, 0, 0) < 2^{(k+k_a)-48-k_a} = 2^{k-48} \simeq a \cdot 2^{k-48-k_a} = a \cdot 2^{k_3+72-k_a}$$

which is $(72 - k_a)$-bit too long compared to condition (5). Therefore, by exhaustive search we will need to examine roughly 2^{72-k_a} different integers a to find a pair (b, X) that satisfies (5); since a is k_a-bits long, this can be done only if $72 - k_a \leq k_a$, which gives $k_a \geq 36$. For $k_a = 36$ we have to exhaust the 2^{36} possible values of a.

Once this is done we obtain from (6):

$$t = b \cdot N - a \cdot \mu(X, N_{\text{I},1}, h) = r - a \cdot h$$

with $0 \leq r < a \cdot 2^{k_4}$. This implies that the final size of t values is $168 + k_a$ bits. For $k_a = 36$ this gives 204 bits (instead of 170 bits for plain ISO/IEC 9796-2). The attack's complexity will hence be higher than for plain ISO/IEC 9796-2.

In the full version of the paper [12] we exhibit concrete (a, b, X) values for $k_a = 52$ and for the RSA-2048 challenge; this required $\simeq 2^{23}$ trials (109 minutes on a single PC). We estimate that for $k_a = 36$ this computation will take roughly 13 years on a single PC, or equivalently US$11,000 using the EC2 grid.

Table 1 shows that attacking 204-bit t_i's would cost $\simeq$ US\$84,000. As for the ISO/IEC 9796-2 attack, we can decrease this cost by first doing exhaustive search on the bits of HASH(m) to obtain a smaller t-value. We found that with 8 bits of exhaustive search cost drops to $\simeq$ US\$45,000 (without the matrix step, but in our attack algebra takes a relatively small amount of time).

8.3 Summary

In the full version of the paper [12] we provide an analysis of the other formats in the EMV specifications, with corresponding attacks when such attacks exist. We summarize results in Table 2 where an X represents a string that can be controlled by the adversary, while Y cannot be controlled. The size of the final t-value to be smooth is given in bits. Note that cost estimates are cheaper than Table 1 because we first perform exhaustive search on 8 bits of HASH(m) = SHA-1(m); however here we do take into account the cost of computing these SHA-1(m) values.

Table 2. Various EMV message formats. X denotes a data field controllable by the adversary. Y is not controllable. Data sizes for X, Y and t are expressed in bits.

EMV mode	Format	$\|X\|$	$\|Y\|$	$\|t\|$	EC2 cost (US\$)
SDA-IPKD	$02_{16}\|X\|Y\|N_{\text{I}}\|03_{16}$	48	56	204	45,000
SDA-SAD	Y	-	$k-176$	-	-
ODDA-IPKD	$02_{16}\|X\|Y\|N_{\text{I}}\|03_{16}$	48	56	204	45,000
ODDA-ICC-PKD	$04_{16}\|X\|Y\|N_{\text{ICC}}\|03_{16}\|$data	96	56	204	45,000
ODDA-DAD1	Y	-	$k-176$	-	-
ODDA-DAD2	Y	-	$k-176$	-	-
ICC-PIN	$04_{16}\|X\|Y\|N_{\text{ICC}}\|03_{16}$	96	56	204	45,000

Table 2 shows that only four of the EMV formats can be attacked, with the same complexity as the SDA-IPKD format. The other formats seem out of reach because the non-controllable part Y is too large.

Note that these attacks do not threaten any of the 730 million EMV payment cards in use worldwide for *operational* reasons: the Issuer and the CA will never accept to sign the chosen messages necessary for conducting the attack.

9 Conclusion

This paper exhibited a *practically exploitable* flaw in the *currently valid* ISO/IEC 9796-2 standard and a conceptual flaw in EMV signatures. The authors recommend the definite withdrawal of the *ad-hoc* encoding mode in ISO/IEC 9796-2 and its replacement by a provably secure encoding function such as PSS.

References

1. Bach, E., Peralta, R.: Asymptotic semismoothness probabilities. Mathematics of Computation 65(216), 1701–1715 (1996)
2. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of CCS 1993, pp. 62–73. ACM, New York (1993)
3. Bellare, M., Rogaway, P.: Optimal Asymmetric Encryption: How to encrypt with RSA. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 92–111. Springer, Heidelberg (1995)
4. Bellare, M., Rogaway, P.: The Exact security of digital signatures: How to sign with RSA and Rabin. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (1996)
5. Bernstein, D.J.: Fast Multiplications and its applications. Algorithmic Number Theory 44 (2008)
6. Bernstein, D.J.: How to find smooth parts of integers (2004/05/10), `http://cr.yp.to/papers.html#smoothparts`
7. Bernstein, D.J.: Scaled remainder trees (2004/08/20), `http://cr.yp.to/papers.html#scaledmod`
8. Bleichenbacher, D.: Chosen ciphertext attacks against protocols based on the RSA encryption standard. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 1–12. Springer, Heidelberg (1998)
9. Coppersmith, D.: Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm. Mathematics of Computation 62(205), 333–350 (1994)
10. Coppersmith, D., Coron, J.-S., Grieu, F., Halevi, S., Jutla, C.S., Naccache, D., Stern, J.P.: Cryptanalysis of ISO/IEC 9796-1. Journal of Cryptology 21, 27–51 (2008)
11. Coppersmith, D., Halevi, S., Jutla, C.: ISO 9796-1 and the new, forgery strategy, Research contribution to P.1363 (1999), `grouper.ieee.org/groups/1363/Research`
12. Coron, J.S., Naccache, D., Tibouchi, M., Weinmann, R.P.: Practical Cryptanalysis of ISO / IEC 9796-2 and EMV Signatures, Cryptology ePrint Archive, Report 2009/203, `http://eprint.iacr.org/`
13. Coron, J.-S.: Security proofs for partial domain hash signature schemes. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 613–626. Springer, Heidelberg (2002)
14. Coron, J.-S., Desmedt, Y., Naccache, D., Odlyzko, A., Stern, J.P.: Index calculation attacks on RSA signature and encryption. Index calculation attacks on RSA signature and encryption Designs, Codes and Cryptography 38(1), 41–53 (2006)
15. Coron, J.-S., Naccache, D., Joye, M., Paillier, P.: New attacks on PKCS#1 v1.5 encryption. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 369–381. Springer, Heidelberg (2000)
16. Coron, J.-S., Naccache, D., Stern, J.P.: On the security of RSA padding. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 1–18. Springer, Heidelberg (1999)
17. Desmedt, Y., Odlyzko, A.: A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 516–522. Springer, Heidelberg (1986)
18. EMV, Integrated circuit card specifications for payment systems, Book 2. Security and Key Management. Version 4.2 (June 2008), `http://www.emvco.com`
19. Gaudry, P., Kruppa, A., Zimmermann, P.: A gmp-based implementation of Schőnhage-Strassen's large integer multiplication algorithm. In: Proceedings of ISSAC 2007, Waterloo, Ontario, Canada, pp. 167–174. ACM Press, New York (2007)

20. Grieu, F.: A chosen messages attack on the ISO/IEC 9796-1 signature scheme. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 70–80. Springer, Heidelberg (2000)
21. Hart, W.B., et al.: Multiple Precision Integers and Rationals, http://www.mpir.org
22. ISO / IEC 9796, Information technology – Security techniques – Digital signature scheme giving message recovery, Part 1: Mechanisms using redundancy (1999)
23. ISO / IEC 9796-2, Information technology – Security techniques – Digital signature scheme giving message recovery, Part 2: Mechanisms using a hash-function (1997)
24. ISO / IEC 9796-2:2002, Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: Integer factorization based mechanisms (2002)
25. Joux, A., Naccache, D., Thomé, E.: When e-th roots become easier than factoring. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 13–28. Springer, Heidelberg (2007)
26. Kaliski, B.: pkcs#1: RSA Encryption Standard, Version 1.5, RSA Laboratories (November 1993)
27. Kaltofen, E., Lobo, A.: Distributed matrix-free solution of large sparse linear systems over finite fields. Algorithmica 24, 331–348 (1999)
28. Lenstra, A.K., Lenstra Jr., H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen 261, 513–534 (1982)
29. Lenstra Jr., H.: Factoring integers with elliptic curves. Annals of Mathematics 126(2), 649–673 (1987)
30. Lobo, A.: WLSS2: an implementation of the homogeneous block Wiedemann algorithm, www4.ncsu.edu/~kaltofen/software/wiliss
31. Misarsky, J.-F.: How (not) to design RSA signature schemes. In: Imai, H., Zheng, Y. (eds.) PKC 1998. LNCS, vol. 1431, pp. 14–28. Springer, Heidelberg (1998)
32. Paar, C., Schimmer, M.: COPACOBANA: A Codebreaker for DES and other ciphers, www.copacobana.org
33. The PARI Group, PARI/GP, version 2.3.4, Bordeaux (2008), http://pari.math.u-bordeaux.fr
34. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public key cryptosystems. Communications of the ACM 21, 120–126 (1978)
35. The sage development team, sage mathematics software, Version 3.3 (2009), http://www.sagemath.org