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

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

[1]Université du Luxembourg

[2]École normale supérieure

CRYPTO 2009

Our Results in a Nutshell

- Improve upon a previous attack [CNS99] against ISO 9796-2 signatures by a large factor.
- Conduct the new attack in practice, demonstrating an actual vulnerability in the ISO 9796-2:2002 standard.
- Show how the attack applies to certain EMV signatures.

Our Results in a Nutshell

- Improve upon a previous attack [CNS99] against ISO 9796-2 signatures by a large factor.
- Conduct the new attack in practice, demonstrating an actual vulnerability in the ISO 9796-2:2002 standard.
- Show how the attack applies to certain EMV signatures.

Our Results in a Nutshell

- Improve upon a previous attack [CNS99] against ISO 9796-2 signatures by a large factor.
- Conduct the new attack in practice, demonstrating an actual vulnerability in the ISO 9796-2:2002 standard.
- Show how the attack applies to certain EMV signatures.

Outline

Outline

RSA Signatures

- Signing using textbook RSA:

$$\sigma = m^{1/e} \bmod N$$

is a bad idea (e.g. homomorphic properties).

- Therefore, encapsulate m using an encoding function μ:

$$\sigma = \mu(m)^{1/e} \bmod N$$

RSA Signatures

- Signing using textbook RSA:

$$\sigma = m^{1/e} \bmod N$$

is a bad idea (e.g. homomorphic properties).

- Therefore, encapsulate m using an encoding function μ:

$$\sigma = \mu(m)^{1/e} \bmod N$$

Encoding functions

- Two kinds of encoding functions:
  1. Ad-hoc encodings: PKCS#1 v1.5, ISO 9796-1, ISO 9796-2, etc. Designed to prevent specific attacks. Often exhibit other weaknesses.
  2. Provably secure encodings: RSA-FDH, RSA-PSS, Cramer-Shoup, etc. Proven to be secure under well-defined assumptions.
- Although potentially less secure, ad-hoc encodings remain in widespread use in real-world applications (including credit cards, e-passports, etc.). Re-evaluating them periodically is thus necessary.

Encoding functions

- Two kinds of encoding functions:
  1. Ad-hoc encodings: PKCS#1 v1.5, ISO 9796-1, ISO 9796-2, etc. Designed to prevent specific attacks. Often exhibit other weaknesses.
  2. Provably secure encodings: RSA-FDH, RSA-PSS, Cramer-Shoup, etc. Proven to be secure under well-defined assumptions.
- Although potentially less secure, ad-hoc encodings remain in widespread use in real-world applications (including credit cards, e-passports, etc.). Re-evaluating them periodically is thus necessary.

Encoding functions

- Two kinds of encoding functions:
  1. Ad-hoc encodings: PKCS#1 v1.5, ISO 9796-1, ISO 9796-2, etc. Designed to prevent specific attacks. Often exhibit other weaknesses.
  2. Provably secure encodings: RSA-FDH, RSA-PSS, Cramer-Shoup, etc. Proven to be secure under well-defined assumptions.
- Although potentially less secure, ad-hoc encodings remain in widespread use in real-world applications (including credit cards, e-passports, etc.). Re-evaluating them periodically is thus necessary.

Encoding functions

- Two kinds of encoding functions:
  1. Ad-hoc encodings: PKCS#1 v1.5, ISO 9796-1, ISO 9796-2, etc. Designed to prevent specific attacks. Often exhibit other weaknesses.
  2. Provably secure encodings: RSA-FDH, RSA-PSS, Cramer-Shoup, etc. Proven to be secure under well-defined assumptions.
- Although potentially less secure, ad-hoc encodings remain in widespread use in real-world applications (including credit cards, e-passports, etc.). Re-evaluating them periodically is thus necessary.

ISO 9796-2

- The ISO 9796-2 standard defines an ad-hoc encoding with partial or total message recovery. We only consider partial message recovery.
- Let k be the size of N. The encoding function has the following form:

$$\mu(m) = 6\mathtt{A}_{16} \| m[1] \| \textsc{hash}(m) \| \mathtt{BC}_{16}$$

with 2 fixed bytes, a digest of k_h bits and the first $k - k_h - 16$ bits of m.

- The size of $\mu(m)$ is thus always $k - 1$ bits.
- ISO 9796-2:1997 recommended $128 \leq k_h \leq 160$.
ISO 9796-2:2002 now recommends $k_h \geq 160$, and EMV uses $k_h = 160$.

ISO 9796-2

- The ISO 9796-2 standard defines an ad-hoc encoding with partial or total message recovery. We only consider partial message recovery.
- Let k be the size of N. The encoding function has the following form:

$$\mu(m) = \mathtt{6A}_{16} \| m[1] \| \textsc{hash}(m) \| \mathtt{BC}_{16}$$

with 2 fixed bytes, a digest of k_h bits and the first $k - k_h - 16$ bits of m.

- The size of $\mu(m)$ is thus always $k - 1$ bits.
- ISO 9796-2:1997 recommended $128 \leq k_h \leq 160$.
ISO 9796-2:2002 now recommends $k_h \geq 160$, and EMV uses $k_h = 160$.

ISO 9796-2

- The ISO 9796-2 standard defines an ad-hoc encoding with partial or total message recovery. We only consider partial message recovery.
- Let k be the size of N. The encoding function has the following form:

$$\mu(m) = \mathtt{6A}_{16} \| m[1] \| \textsc{hash}(m) \| \mathtt{BC}_{16}$$

with 2 fixed bytes, a digest of k_h bits and the first $k - k_h - 16$ bits of m.

- The size of $\mu(m)$ is thus always $k - 1$ bits.
- ISO 9796-2:1997 recommended $128 \leq k_h \leq 160$.
ISO 9796-2:2002 now recommends $k_h \geq 160$, and EMV uses $k_h = 160$.

ISO 9796-2

- The ISO 9796-2 standard defines an ad-hoc encoding with partial or total message recovery. We only consider partial message recovery.
- Let k be the size of N. The encoding function has the following form:

$$\mu(m) = \mathtt{6A}_{16} \| m[1] \| \textsc{hash}(m) \| \mathtt{BC}_{16}$$

with 2 fixed bytes, a digest of k_h bits and the first $k - k_h - 16$ bits of m.

- The size of $\mu(m)$ is thus always $k - 1$ bits.
- ISO 9796-2:1997 recommended $128 \leq k_h \leq 160$.
ISO 9796-2:2002 now recommends $k_h \geq 160$, and EMV uses $k_h = 160$.

ISO 9796-2

- The ISO 9796-2 standard defines an ad-hoc encoding with partial or total message recovery. We only consider partial message recovery.
- Let k be the size of N. The encoding function has the following form:

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

  with 2 fixed bytes, a digest of k_h bits and the first $k - k_h - 16$ bits of m.
- The size of $\mu(m)$ is thus always $k - 1$ bits.
- ISO 9796-2:1997 recommended $128 \leq k_h \leq 160$.
  ISO 9796-2:2002 now recommends $k_h \geq 160$, and EMV uses $k_h = 160$.

ISO 9796-2

- The ISO 9796-2 standard defines an ad-hoc encoding with partial or total message recovery. We only consider partial message recovery.
- Let k be the size of N. The encoding function has the following form:

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

with 2 fixed bytes, a digest of k_h bits and the first $k - k_h - 16$ bits of m.
- The size of $\mu(m)$ is thus always $k - 1$ bits.
- ISO 9796-2:1997 recommended $128 \leq k_h \leq 160$. ISO 9796-2:2002 now recommends $k_h \geq 160$, and EMV uses $k_h = 160$.

ISO 9796-2

- The ISO 9796-2 standard defines an ad-hoc encoding with partial or total message recovery. We only consider partial message recovery.
- Let k be the size of N. The encoding function has the following form:

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

  with 2 fixed bytes, a digest of k_h bits and the first $k - k_h - 16$ bits of m.
- The size of $\mu(m)$ is thus always $k - 1$ bits.
- ISO 9796-2:1997 recommended $128 \leq k_h \leq 160$.
  ISO 9796-2:2002 now recommends $k_h \geq 160$, and EMV uses $k_h = 160$.

Outline

The Desmedt-Odlyzko Attack

Suppose the encoded messages $\mu(m)$ are relatively short. In [DO85], Desmedt and Odlyzko proposed the following attack.

1. Choose a bound B and let $p_1, \dots, p_\ell$ be the primes smaller than B.
2. Find $\ell + 1$ messages m_i such that the $\mu(m_i)$ are B-smooth:

$$\mu(m_i) = p_1^{v_{i,1}} \cdots p_\ell^{v_{i,\ell}}$$

3. Obtain a linear dependence relation between the exponent vectors $v_i = (v_{i,1} \bmod e, \dots, v_{i,\ell} \bmod e)$ and deduce the expression of one $\mu(m_j)$ as a multiplicative combination of the $\mu(m_i)$, $i \neq j$.
4. Ask for the signatures of the m_i and forge the signature of m_j.

The Desmedt-Odlyzko Attack

Suppose the encoded messages $\mu(m)$ are relatively short. In [DO85], Desmedt and Odlyzko proposed the following attack.

1. Choose a bound B and let $p_1, \dots, p_\ell$ be the primes smaller than B.
2. Find $\ell + 1$ messages m_i such that the $\mu(m_i)$ are B-smooth:

$$\mu(m_i) = p_1^{v_{i,1}} \cdots p_\ell^{v_{i,\ell}}$$

3. Obtain a linear dependence relation between the exponent vectors $v_i = (v_{i,1} \bmod e, \dots, v_{i,\ell} \bmod e)$ and deduce the expression of one $\mu(m_j)$ as a multiplicative combination of the $\mu(m_i)$, $i \neq j$.
4. Ask for the signatures of the m_i and forge the signature of m_j.

The Desmedt-Odlyzko Attack

Suppose the encoded messages $\mu(m)$ are relatively short. In [DO85], Desmedt and Odlyzko proposed the following attack.

1. Choose a bound B and let $p_1, \ldots, p_\ell$ be the primes smaller than B.
2. Find $\ell + 1$ messages m_i such that the $\mu(m_i)$ are B-smooth:

$$\mu(m_i) = p_1^{v_{i,1}} \cdots p_\ell^{v_{i,\ell}}$$

3. Obtain a linear dependence relation between the exponent vectors $v_i = (v_{i,1} \bmod e, \ldots, v_{i,\ell} \bmod e)$ and deduce the expression of one $\mu(m_j)$ as a multiplicative combination of the $\mu(m_i)$, $i \neq j$.
4. Ask for the signatures of the m_i and forge the signature of m_j.

The Desmedt-Odlyzko Attack

Suppose the encoded messages $\mu(m)$ are relatively short. In [DO85], Desmedt and Odlyzko proposed the following attack.

1. Choose a bound B and let $p_1, \ldots, p_\ell$ be the primes smaller than B.
2. Find $\ell + 1$ messages m_i such that the $\mu(m_i)$ are B-smooth:

$$\mu(m_i) = p_1^{v_{i,1}} \cdots p_\ell^{v_{i,\ell}}$$

3. Obtain a linear dependence relation between the exponent vectors $v_i = (v_{i,1} \bmod e, \ldots, v_{i,\ell} \bmod e)$ and deduce the expression of one $\mu(m_j)$ as a multiplicative combination of the $\mu(m_i)$, $i \neq j$.
4. Ask for the signatures of the m_i and forge the signature of m_j.

The Coron-Naccache-Stern Attack

- The ISO 9796-2 encoding $\mu(m)$ has full size, so the [DO85] attack doesn't apply.
- However, Coron et al. noticed that the attack generalizes to the case where, for some fixed a, the $t_i = a \cdot \mu(m_i) \bmod N$ are small.
- Moreover, they show that for $a = 2^8$, one can choose the message prefix $m[1]$ such that all the corresponding $a \cdot \mu(m) \bmod N$ are of size $\leq k_h + 16$ bits.
- Attacking the instances $k_h = 128$ and $k_h = 160$ requires 2^{54} and 2^{61} operations respectively.

The Coron-Naccache-Stern Attack

- The ISO 9796-2 encoding $\mu(m)$ has full size, so the [DO85] attack doesn't apply.
- However, Coron et al. noticed that the attack generalizes to the case where, for some fixed a, the $t_i = a \cdot \mu(m_i) \bmod N$ are small.
- Moreover, they show that for $a = 2^8$, one can choose the message prefix $m[1]$ such that all the corresponding $a \cdot \mu(m) \bmod N$ are of size $\leq k_h + 16$ bits.
- Attacking the instances $k_h = 128$ and $k_h = 160$ requires 2^{54} and 2^{61} operations respectively.

The Coron-Naccache-Stern Attack

- The ISO 9796-2 encoding $\mu(m)$ has full size, so the [DO85] attack doesn't apply.
- However, Coron et al. noticed that the attack generalizes to the case where, for some fixed a, the $t_i = a \cdot \mu(m_i) \bmod N$ are small.
- Moreover, they show that for $a = 2^8$, one can choose the message prefix $m[1]$ such that all the corresponding $a \cdot \mu(m) \bmod N$ are of size $\leq k_h + 16$ bits.
- Attacking the instances $k_h = 128$ and $k_h = 160$ requires 2^{54} and 2^{61} operations respectively.

The Coron-Naccache-Stern Attack

- The ISO 9796-2 encoding $\mu(m)$ has full size, so the [DO85] attack doesn't apply.
- However, Coron et al. noticed that the attack generalizes to the case where, for some fixed a, the $t_i = a \cdot \mu(m_i) \bmod N$ are small.
- Moreover, they show that for $a = 2^8$, one can choose the message prefix $m[1]$ such that all the corresponding $a \cdot \mu(m) \bmod N$ are of size $\leq k_h + 16$ bits.
- Attacking the instances $k_h = 128$ and $k_h = 160$ requires 2^{54} and 2^{61} operations respectively.

Outline

Building Blocks of Our Attack

We improve upon [CNS99] using the following techniques.

1. Bernstein's batch smoothness detection algorithm: we use the technique of [B04] to find smooth numbers in a large collection of integers much faster than trial division (speed-up factor ≈ 1000).
2. The large prime variant: we looked for semi-smooth numbers in addition to smooth numbers to obtain additional relations (speed-up factor ≈ 1.4).
3. Smaller t_i values: in [CNS99], $t_i = a \cdot \mu(m_i) \bmod N$ with $a = 2^8$; we show that a careful choice of a depending on N yields smaller t_i values (speed-up factor ≈ 2).
4. Exhaustive search: we reduce the size of t_i further by selecting messages whose hash values match a certain bit pattern (speed-up factor ≈ 2).

Building Blocks of Our Attack

We improve upon [CNS99] using the following techniques.

1. Bernstein's batch smoothness detection algorithm: we use the technique of [B04] to find smooth numbers in a large collection of integers much faster than trial division (speed-up factor ≈ 1000).
2. The large prime variant: we looked for semi-smooth numbers in addition to smooth numbers to obtain additional relations (speed-up factor ≈ 1.4).
3. Smaller t_i values: in [CNS99], $t_i = a \cdot \mu(m_i) \mod N$ with $a = 2^8$; we show that a careful choice of a depending on N yields smaller t_i values (speed-up factor ≈ 2).
4. Exhaustive search: we reduce the size of t_i further by selecting messages whose hash values match a certain bit pattern (speed-up factor ≈ 2).

Building Blocks of Our Attack

We improve upon [CNS99] using the following techniques.

1. Bernstein's batch smoothness detection algorithm: we use the technique of [B04] to find smooth numbers in a large collection of integers much faster than trial division (speed-up factor ≈ 1000).
2. The large prime variant: we looked for semi-smooth numbers in addition to smooth numbers to obtain additional relations (speed-up factor ≈ 1.4).
3. Smaller t_i values: in [CNS99], $t_i = a \cdot \mu(m_i) \mod N$ with $a = 2^8$; we show that a careful choice of a depending on N yields smaller t_i values (speed-up factor ≈ 2).
4. Exhaustive search: we reduce the size of t_i further by selecting messages whose hash values match a certain bit pattern (speed-up factor ≈ 2).

Building Blocks of Our Attack

We improve upon [CNS99] using the following techniques.

1. Bernstein's batch smoothness detection algorithm: we use the technique of [B04] to find smooth numbers in a large collection of integers much faster than trial division (speed-up factor ≈ 1000).
2. The large prime variant: we looked for semi-smooth numbers in addition to smooth numbers to obtain additional relations (speed-up factor ≈ 1.4).
3. Smaller t_i values: in [CNS99], $t_i = a \cdot \mu(m_i) \mod N$ with $a = 2^8$; we show that a careful choice of a depending on N yields smaller t_i values (speed-up factor ≈ 2).
4. Exhaustive search: we reduce the size of t_i further by selecting messages whose hash values match a certain bit pattern (speed-up factor ≈ 2).

Building Blocks of Our Attack

We improve upon [CNS99] using the following techniques.

1. Bernstein's batch smoothness detection algorithm: we use the technique of [B04] to find smooth numbers in a large collection of integers much faster than trial division (speed-up factor ≈ 1000).
2. The large prime variant: we looked for semi-smooth numbers in addition to smooth numbers to obtain additional relations (speed-up factor ≈ 1.4).
3. Smaller t_i values: in [CNS99], $t_i = a \cdot \mu(m_i) \mod N$ with $a = 2^8$; we show that a careful choice of a depending on N yields smaller t_i values (speed-up factor ≈ 2).
4. Exhaustive search: we reduce the size of t_i further by selecting messages whose hash values match a certain bit pattern (speed-up factor ≈ 2).

Outline

Overview of the Experiment

We implemented the attack for $N = \text{RSA-2048}$, $e = 2$ and $\text{HASH} = \text{SHA-1}$. The attack step by step:

1. Determine the constants a, $m[1]$, etc.
2. Compute the product of the first ℓ primes ($\ell = 2^{20}$).
3. Compute $v_i = a \cdot \mu(m_i) \bmod N$ and hence SHA-1(m_i), for many messages m_i.
4. Find the smooth and semi-smooth v_i's.
5. Factor the smooth numbers and colliding pairs of semi-smooth integers, obtaining the sparse matrix of exponents.
6. Reduce modulo e.
7. Find nontrivial vectors in the kernel of this reduced matrix.

Overview of the Experiment

We implemented the attack for $N = \text{RSA-2048}$, $e = 2$ and $\text{HASH} = \text{SHA-1}$. The attack step by step:

1. Determine the constants a, $m[1]$, etc.
2. Compute the product of the first ℓ primes ($\ell = 2^{20}$).
3. Compute $t_i = a \cdot \mu(m_i) \bmod N$, and hence $\text{SHA-1}(m_i)$, for many messages m_i.
4. Find the smooth and semi-smooth t_i's.
5. Factor the smooth integers and colliding pairs of semi-smooth integers, obtaining the sparse matrix of exponents.
6. Reduce modulo e.
7. Find nontrivial vectors in the kernel of the reduced matrix.

Overview of the Experiment

We implemented the attack for $N = \text{RSA-2048}$, $e = 2$ and $\text{HASH} = \text{SHA-1}$. The attack step by step:

1. Determine the constants a, $m[1]$, etc.
2. Compute the product of the first ℓ primes ($\ell = 2^{20}$).
3. Compute $t_i = a \cdot \mu(m_i) \bmod N$, and hence $\text{SHA-1}(m_i)$, for many messages m_i.
4. Find the smooth and semi-smooth t_i's.
5. Factor the smooth integers and colliding pairs of semi-smooth integers, obtaining the sparse matrix of exponents.
6. Reduce modulo e.
7. Find nontrivial vectors in the kernel of the reduced matrix.

Overview of the Experiment

We implemented the attack for $N = \textsc{rsa}\text{-}2048$, $e = 2$ and $\textsc{hash} = \textsc{sha}\text{-}1$. The attack step by step:

1. Determine the constants a, $m[1]$, etc.
2. Compute the product of the first ℓ primes ($\ell = 2^{20}$).
3. Compute $t_i = a \cdot \mu(m_i) \bmod N$, and hence $\textsc{sha}\text{-}1(m_i)$, for many messages m_i.
4. Find the smooth and semi-smooth t_i's.
5. Factor the smooth integers and colliding pairs of semi-smooth integers, obtaining the sparse matrix of exponents.
6. Reduce modulo e.
7. Find nontrivial vectors in the kernel of the reduced matrix.

Overview of the Experiment

We implemented the attack for $N = \textsc{rsa}\text{-}2048$, $e = 2$ and $\textsc{hash} = \textsc{sha}\text{-}1$. The attack step by step:

1. Determine the constants a, $m[1]$, etc.
2. Compute the product of the first ℓ primes ($\ell = 2^{20}$).
3. Compute $t_i = a \cdot \mu(m_i) \bmod N$, and hence $\textsc{sha}\text{-}1(m_i)$, for many messages m_i.
4. Find the smooth and semi-smooth t_i's.
5. Factor the smooth integers and colliding pairs of semi-smooth integers, obtaining the sparse matrix of exponents.
6. Reduce modulo e.
7. Find nontrivial vectors in the kernel of the reduced matrix.

Overview of the Experiment

We implemented the attack for $N = \textsc{rsa}\text{-}2048$, $e = 2$ and $\textsc{hash} = \textsc{sha}\text{-}1$. The attack step by step:

1. Determine the constants a, $m[1]$, etc.
2. Compute the product of the first ℓ primes ($\ell = 2^{20}$).
3. Compute $t_i = a \cdot \mu(m_i) \bmod N$, and hence $\textsc{sha}\text{-}1(m_i)$, for many messages m_i.
4. Find the smooth and semi-smooth t_i's.
5. Factor the smooth integers and colliding pairs of semi-smooth integers, obtaining the sparse matrix of exponents.
6. Reduce modulo e.
7. Find nontrivial vectors in the kernel of the reduced matrix.

Overview of the Experiment

We implemented the attack for $N = \textsc{rsa}\text{-}2048$, $e = 2$ and $\textsc{hash} = \textsc{sha}\text{-}1$. The attack step by step:

1. Determine the constants a, $m[1]$, etc.
2. Compute the product of the first ℓ primes ($\ell = 2^{20}$).
3. Compute $t_i = a \cdot \mu(m_i) \bmod N$, and hence $\textsc{sha}\text{-}1(m_i)$, for many messages m_i.
4. Find the smooth and semi-smooth t_i's.
5. Factor the smooth integers and colliding pairs of semi-smooth integers, obtaining the sparse matrix of exponents.
6. Reduce modulo e.
7. Find nontrivial vectors in the kernel of the reduced matrix.

Overview of the Experiment

We implemented the attack for $N = \textsc{rsa}\text{-}2048$, $e = 2$ and $\textsc{hash} = \textsc{sha}\text{-}1$. The attack step by step:

1. Determine the constants a, $m[1]$, etc.
2. Compute the product of the first ℓ primes ($\ell = 2^{20}$).
3. Compute $t_i = a \cdot \mu(m_i) \bmod N$, and hence $\textsc{sha}\text{-}1(m_i)$, for many messages m_i.
4. Find the smooth and semi-smooth t_i's.
5. Factor the smooth integers and colliding pairs of semi-smooth integers, obtaining the sparse matrix of exponents.
6. Reduce modulo e.
7. Find nontrivial vectors in the kernel of the reduced matrix.

Overview of the Experiment

We implemented the attack for $N = \textsc{rsa}$-2048, $e = 2$ and $\textsc{hash} = \textsc{sha}$-1. The attack step by step:

1. Determine the constants a, $m[1]$, etc.
2. Compute the product of the first ℓ primes ($\ell = 2^{20}$).
3. Compute $t_i = a \cdot \mu(m_i) \bmod N$, and hence $\textsc{sha}\text{-}1(m_i)$, for many messages m_i.
4. Find the smooth and semi-smooth t_i's.
5. Factor the smooth integers and colliding pairs of semi-smooth integers, obtaining the sparse matrix of exponents.
6. Reduce modulo e.
7. Find nontrivial vectors in the kernel of the reduced matrix.

Setup stage: on a single PC, negligible time.

Overview of the Experiment

We implemented the attack for $N = \text{RSA-2048}$, $e = 2$ and $\text{HASH} = \text{SHA-1}$. The attack step by step:

1. Determine the constants a, $m[1]$, etc.
2. Compute the product of the first ℓ primes ($\ell = 2^{20}$).
3. Compute $t_i = a \cdot \mu(m_i) \bmod N$, and hence $\text{SHA-1}(m_i)$, for many messages m_i.
4. Find the smooth and semi-smooth t_i's.
5. Factor the smooth integers and colliding pairs of semi-smooth integers, obtaining the sparse matrix of exponents.
6. Reduce modulo e.
7. Find nontrivial vectors in the kernel of the reduced matrix.

Sieving stage: on Amazon EC2, 1100 CPU hours, 2 days.

Overview of the Experiment

We implemented the attack for $N = \textsc{rsa}\text{-}2048$, $e = 2$ and $\textsc{hash} = \textsc{sha}\text{-}1$. The attack step by step:

1. Determine the constants a, $m[1]$, etc.
2. Compute the product of the first ℓ primes ($\ell = 2^{20}$).
3. Compute $t_i = a \cdot \mu(m_i) \bmod N$, and hence $\textsc{sha}\text{-}1(m_i)$, for many messages m_i.
4. Find the smooth and semi-smooth t_i's.
5. Factor the smooth integers and colliding pairs of semi-smooth integers, obtaining the sparse matrix of exponents.
6. Reduce modulo e.
7. Find nontrivial vectors in the kernel of the reduced matrix.

Linear algebra stage: on a PC, a few hours.

Results of the Experiment

1. 16,230,259,553,940 ($\approx 2^{44}$) digest computations.
2. 739,686,719,488 ($\approx 2^{39}$) t_i's in 647,901 batches of 2^{19} each.
3. 684,365 smooth t_i's and 366,302 collisions between 2,786,327 semi-smooth t_i's.
4. 1,050,667-column matrix ($2^{20} + 1 = 1{,}048{,}577$ needed).
5. Algebra on 839,908 columns having > 1 nonzero entries.
6. 124 kernel vectors.
7. Forgery involving 432,903 signatures.

Results of the Experiment

1. 16,230,259,553,940 ($\approx 2^{44}$) digest computations.
2. 739,686,719,488 ($\approx 2^{39}$) t_i's in 647,901 batches of 2^{19} each.
3. 684,365 smooth t_i's and 366,302 collisions between 2,786,327 semi-smooth t_i's.
4. 1,050,667-column matrix ($2^{20} + 1 = 1{,}048{,}577$ needed).
5. Algebra on 839,908 columns having > 1 nonzero entries.
6. 124 kernel vectors.
7. Forgery involving 432,903 signatures.

Results of the Experiment

1. 16,230,259,553,940 ($\approx 2^{44}$) digest computations.
2. 739,686,719,488 ($\approx 2^{39}$) t_i's in 647,901 batches of 2^{19} each.
3. 684,365 smooth t_i's and 366,302 collisions between 2,786,327 semi-smooth t_i's.
4. 1,050,667-column matrix ($2^{20} + 1 = 1{,}048{,}577$ needed).
5. Algebra on 839,908 columns having > 1 nonzero entries.
6. 124 kernel vectors.
7. Forgery involving 432,903 signatures.

Results of the Experiment

1. 16,230,259,553,940 ($\approx 2^{44}$) digest computations.
2. 739,686,719,488 ($\approx 2^{39}$) t_i's in 647,901 batches of 2^{19} each.
3. 684,365 smooth t_i's and 366,302 collisions between 2,786,327 semi-smooth t_i's.
4. 1,050,667-column matrix ($2^{20} + 1 = 1{,}048{,}577$ needed).
5. Algebra on 839,908 columns having > 1 nonzero entries.
6. 124 kernel vectors.
7. Forgery involving 432,903 signatures.

Results of the Experiment

1. 16,230,259,553,940 ($\approx 2^{44}$) digest computations.
2. 739,686,719,488 ($\approx 2^{39}$) t_i's in 647,901 batches of 2^{19} each.
3. 684,365 smooth t_i's and 366,302 collisions between 2,786,327 semi-smooth t_i's.
4. 1,050,667-column matrix ($2^{20} + 1 = 1{,}048{,}577$ needed).
5. Algebra on 839,908 columns having > 1 nonzero entries.
6. 124 kernel vectors.
7. Forgery involving 432,903 signatures.

Results of the Experiment

1. 16,230,259,553,940 ($\approx 2^{44}$) digest computations.
2. 739,686,719,488 ($\approx 2^{39}$) t_i's in 647,901 batches of 2^{19} each.
3. 684,365 smooth t_i's and 366,302 collisions between 2,786,327 semi-smooth t_i's.
4. 1,050,667-column matrix ($2^{20} + 1 = 1{,}048{,}577$ needed).
5. Algebra on 839,908 columns having > 1 nonzero entries.
6. 124 kernel vectors.
7. Forgery involving 432,903 signatures.

Results of the Experiment

1. 16,230,259,553,940 ($\approx 2^{44}$) digest computations.
2. 739,686,719,488 ($\approx 2^{39}$) t_i's in 647,901 batches of 2^{19} each.
3. 684,365 smooth t_i's and 366,302 collisions between 2,786,327 semi-smooth t_i's.
4. 1,050,667-column matrix ($2^{20} + 1 =$ 1,048,577 needed).
5. Algebra on 839,908 columns having > 1 nonzero entries.
6. 124 kernel vectors.
7. Forgery involving 432,903 signatures.

Cost Estimates

Not counting speed-ups by exhaustive search, the CPU time and equivalent "Amazon cost" of our attack for various sizes of t_i should be as follows.

$a = \log_2 t_i$	$\log_2 \ell$	Estimated Time		$\log_2 \tau$	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

Outline

The EMV Data Formats

- The EMV specifications define several message formats for signing data related to payment cards with ISO 9796-2.
- For example, SDA-IPKD consists of messages of the following form:

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

including 2 fixed bytes, [illegible] by the adversary, and other [illegible]

- Other formats are similar, but not all of them are vulnerable.

The EMV Data Formats

- The EMV specifications define several message formats for signing data related to payment cards with ISO 9796-2.
- For example, SDA-IPKD consists of messages of the following form:

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

including 2 fixed bytes, 7 bytes Y that cannot be controlled by the adversary, and other bits controlled by the adversary.

- Other formats are similar, but not all of them are vulnerable.

The EMV Data Formats

- The EMV specifications define several message formats for signing data related to payment cards with ISO 9796-2.
- For example, SDA-IPKD consists of messages of the following form:

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

including 2 fixed bytes, 7 bytes Y that cannot be controlled by the adversary, and other bits controlled by the adversary.

- Other formats are similar, but not all of them are vulnerable.

The EMV Data Formats

- The EMV specifications define several message formats for signing data related to payment cards with ISO 9796-2.
- For example, SDA-IPKD consists of messages of the following form:

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

including 2 fixed bytes, 7 bytes Y that cannot be controlled by the adversary, and other bits controlled by the adversary.

- Other formats are similar, but not all of them are vulnerable.

The EMV Data Formats

- The EMV specifications define several message formats for signing data related to payment cards with ISO 9796-2.
- For example, SDA-IPKD consists of messages of the following form:

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

including 2 fixed bytes, 7 bytes Y that cannot be controlled by the adversary, and other bits controlled by the adversary.

- Other formats are similar, but not all of them are vulnerable.

The EMV Data Formats

- The EMV specifications define several message formats for signing data related to payment cards with ISO 9796-2.
- For example, SDA-IPKD consists of messages of the following form:

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

  including 2 fixed bytes, 7 bytes Y that cannot be controlled by the adversary, and other bits controlled by the adversary.
- Other formats are similar, but not all of them are vulnerable.

Attacking EMV

- With ISO 9796-2 encoding, SDA-IPKD gives:

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

- Since the adversary cannot completely choose m, adapt the attack by finding a and X such that $t_i = a \cdot \mu(m_i) \bmod N$ is small. Possible to find such an $a < 2^{36}$.
- The size of t_i is then 204 bits, corresponding to a \$84,000 attack on Amazon (\$45,000 with 8-bit exhaustive search). The search for a costs an additional \$11,000. Within reach!
- However, the CA for payment cards will not sign thousands of chosen messages: not an immediate threat to EMV cards.

Attacking EMV

- With ISO 9796-2 encoding, SDA-IPKD gives:

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

- Since the adversary cannot completely choose m, adapt the attack by finding a and X such that $t_i = a \cdot \mu(m_i) \bmod N$ is small. Possible to find such an $a < 2^{36}$.
- The size of t_i is then 204 bits, corresponding to a \$84,000 attack on Amazon (\$45,000 with 8-bit exhaustive search). The search for a costs an additional \$11,000. Within reach!
- However, the CA for payment cards will not sign thousands of chosen messages: not an immediate threat to EMV cards.

Attacking EMV

- With ISO 9796-2 encoding, SDA-IPKD gives:

$$\mu(m) = \mathtt{6A02}_{16} \| X \| Y \| N_{\mathrm{I},1} \| \textsc{hash}(m) \| \mathtt{BC}_{16}$$

- Since the adversary cannot completely choose m, adapt the attack by finding a and X such that $t_i = a \cdot \mu(m_i) \bmod N$ is small. Possible to find such an $a < 2^{36}$.
- The size of t_i is then 204 bits, corresponding to a \$84,000 attack on Amazon (\$45,000 with 8-bit exhaustive search). The search for a costs an additional \$11,000. Within reach!
- However, the CA for payment cards will not sign thousands of chosen messages: not an immediate threat to EMV cards.

Attacking EMV

- With ISO 9796-2 encoding, SDA-IPKD gives:

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

- Since the adversary cannot completely choose m, adapt the attack by finding a and X such that $t_i = a \cdot \mu(m_i) \bmod N$ is small. Possible to find such an $a < 2^{36}$.
- The size of t_i is then 204 bits, corresponding to a \$84,000 attack on Amazon (\$45,000 with 8-bit exhaustive search). The search for a costs an additional \$11,000. Within reach!
- However, the CA for payment cards will not sign thousands of chosen messages: not an immediate threat to EMV cards.

Conclusion

- Forging ISO 9796-2 signatures using a 160-bit hash function is now easily feasible.
- Therefore, ISO 9796-2:2002 should be phased out.
- Signature encodings based on this standard, such as EMV, are potentially vulnerable.

- Outlook
  - Implement further speed-ups (faster hashing, more large primes)?
  - Defeat ratification counters?
  - Predict forgery size?

Conclusion

- Forging ISO 9796-2 signatures using a 160-bit hash function is now easily feasible.
- Therefore, ISO 9796-2:2002 should be phased out.
- Signature encodings based on this standard, such as EMV, are potentially vulnerable.

- Outlook
  - Implement further speed-ups (faster hashing, more large primes)?
  - Defeat ratification counters?
  - Predict forgery size?

Conclusion

- Forging ISO 9796-2 signatures using a 160-bit hash function is now easily feasible.
- Therefore, ISO 9796-2:2002 should be phased out.
- Signature encodings based on this standard, such as EMV, are potentially vulnerable.

- Outlook
  - Implement further speed-ups (faster hashing, more large primes)?
  - Defeat ratification counters?
  - Predict forgery size?

Conclusion

- Forging ISO 9796-2 signatures using a 160-bit hash function is now easily feasible.
- Therefore, ISO 9796-2:2002 should be phased out.
- Signature encodings based on this standard, such as EMV, are potentially vulnerable.

- Outlook
  - Implement further speed-ups (faster hashing, more large primes)?
  - Defeat ratification counters?
  - Predict forgery size?

Conclusion

- Forging ISO 9796-2 signatures using a 160-bit hash function is now easily feasible.
- Therefore, ISO 9796-2:2002 should be phased out.
- Signature encodings based on this standard, such as EMV, are potentially vulnerable.

- Outlook
  - Implement further speed-ups (faster hashing, more large primes)?
  - Defeat ratification counters?
  - Predict forgery size?

Conclusion

- Forging ISO 9796-2 signatures using a 160-bit hash function is now easily feasible.
- Therefore, ISO 9796-2:2002 should be phased out.
- Signature encodings based on this standard, such as EMV, are potentially vulnerable.

- Outlook
  - Implement further speed-ups (faster hashing, more large primes)?
  - Defeat ratification counters?
  - Predict forgery size?

Conclusion

- Forging ISO 9796-2 signatures using a 160-bit hash function is now easily feasible.
- Therefore, ISO 9796-2:2002 should be phased out.
- Signature encodings based on this standard, such as EMV, are potentially vulnerable.

- Outlook
  - Implement further speed-ups (faster hashing, more large primes)?
  - Defeat ratification counters?
  - Predict forgery size?

Thank you!