A Tale of Three Signatures: Practical Attack of ECDSA with wNAF

Gabrielle De Micheli

Joint work with Rémi Piau and Cécile Pierrot

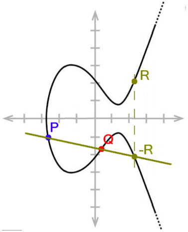
Université de Lorraine, Inria Nancy, France

December 17, 2019

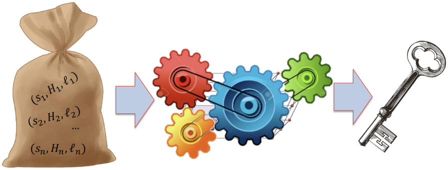
17^{th} IMA International Conference on Cryptography and Coding

How to attack ECDSA

1. Focus on the primitive: DLP on elliptic curves



2. OR get extra informations from an implementation: side channel attacks.



Our work

- Improve the processing step of already known side-channel ECDSA attacks, using the Extended Hidden Number Problem and lattice techniques.
- Optimize the attack to maximize the success probability and minimize the overall time.
- $\longrightarrow$ Perform an attack with the minimum number of signatures needed to recover the secret key: only 3 signatures!

Our work

- Improve the processing step of already known side-channel ECDSA attacks, using the Extended Hidden Number Problem and lattice techniques.
- Optimize the attack to maximize the success probability and minimize the overall time.
- $\longrightarrow$ Perform an attack with the minimum number of signatures needed to recover the secret key: only 3 signatures!

Our work

- Improve the processing step of already known side-channel ECDSA attacks, using the Extended Hidden Number Problem and lattice techniques.
- Optimize the attack to maximize the success probability and minimize the overall time.
- $\longrightarrow$ Perform an attack with the minimum number of signatures needed to recover the secret key: only 3 signatures!

Our target: ECDSA

Elliptic Curve Digital Signature Algorithm is a variant of the Digital Signature Algorithm, DSA, which uses elliptic curves instead of finite fields.

Public Parameters

- An elliptic curve E over a prime field.
- A generator G of prime order q on E.
- A hash function H to $\mathbb{Z}_q$.

Secret Key

- An integer $\alpha \in [1, q-1]$.

Public Key

- $p_k = [\alpha]G$: scalar multiplication of G by α.

Our target: ECDSA

Elliptic Curve Digital Signature Algorithm is a variant of the Digital Signature Algorithm, DSA, which uses elliptic curves instead of finite fields.

Public Parameters

- An elliptic curve E over a prime field.
- A generator G of prime order q on E.
- A hash function H to $\mathbb{Z}_q$.

Secret Key

- An integer $\alpha \in [1, q-1]$.

Public Key

- $p_k = [\alpha]G$: scalar multiplication of G by α.

Our target: ECDSA

Elliptic Curve Digital Signature Algorithm is a variant of the Digital Signature Algorithm, DSA, which uses elliptic curves instead of finite fields.

Public Parameters

- An elliptic curve E over a prime field.
- A generator G of prime order q on E.
- A hash function H to $\mathbb{Z}_q$.

Secret Key

- An integer $\alpha \in [1, q-1]$.

Public Key

- $p_k = [\alpha]G$: scalar multiplication of G by α.

Our target: ECDSA

Elliptic Curve Digital Signature Algorithm is a variant of the Digital Signature Algorithm, DSA, which uses elliptic curves instead of finite fields.

Public Parameters

- An elliptic curve E over a prime field.
- A generator G of prime order q on E.
- A hash function H to $\mathbb{Z}_q$.

Secret Key

- An integer $\alpha \in [1, q-1]$.

Public Key

- $p_k = [\alpha]G$: scalar multiplication of G by α.

Signing algorithm

To sign a message m:

Step 1: Randomly select nonce $k \leftarrow_R \mathbb{Z}_q$

Step 2: Compute the point $(r, y) = [k]G$.

Step 3: Compute $s = k^{-1}(H(m) + \alpha r) \bmod q$.

Step 4: Output the signature (r, s).

Signing algorithm

To sign a message m:

Step 1: Randomly select nonce $k \leftarrow_R \mathbb{Z}_q$

Step 2: Compute the point $(r, y) = [k]G$.

Step 3: Compute $s = k^{-1}(H(m) + \alpha r) \bmod q$.

Step 4: Output the signature (r, s).

Signing algorithm

To sign a message m:

Step 1: Randomly select nonce $k \leftarrow_R \mathbb{Z}_q$

Step 2: Compute the point $(r, y) = [k]G$.

Step 3: Compute $s = k^{-1}(H(m) + \alpha r) \bmod q$.

Step 4: Output the signature (r, s).

Signing algorithm

To sign a message m:

Step 1: Randomly select nonce $k \leftarrow_R \mathbb{Z}_q$

Step 2: Compute the point $(r, y) = [k]G$.

Step 3: Compute $s = k^{-1}(H(m) + \alpha r) \bmod q$.

Step 4: Output the signature (r, s).

Signing algorithm

To sign a message m:

Step 1: Randomly select nonce $k \leftarrow_R \mathbb{Z}_q$

Step 2: Compute the point $(r, y) = [k]G$.

Step 3: Compute $s = k^{-1}(H(m) + \alpha r) \bmod q$.

Step 4: Output the signature (r, s).

Scalar multiplication

$\boxed{\textbf{Step 2: Compute the point } (r, y) = [k]G}$

$\longrightarrow$ **Scalar multiplication**

- Requires a fast algorithm
- Ideally that doesn't leak any information on k!

Scalar multiplication

Step 2: Compute the point $(r, y) = [k]G$

$\longrightarrow$ **Scalar multiplication**

- Requires a fast algorithm
- Ideally that doesn't leak any information on k!

Scalar multiplication

$\boxed{\textbf{Step 2: Compute the point } (r, y) = [k]G}$

$\longrightarrow$ **Scalar multiplication**

- Requires a fast algorithm
- Ideally that doesn't leak any information on k!

Scalar multiplication

$\boxed{\textbf{Step 2: Compute the point } (r, y) = [k]G}$

$\longrightarrow$ **Scalar multiplication**

- Requires a fast algorithm
- Ideally that doesn't leak any information on k!

Double-and-add algorithm

Goal: compute fast point multiplication on elliptic curves

- Input: integer k and point G.
- Output: $Q = [k]G$

Step 1 : Convert k to binary: $k = k_0 + 2k_1 + 2^2 k_2 + \cdots + 2^t k_t$

Step 2 : Initialize $Q = \mathcal{O}$

Step 3 : For $j = t, \cdots, 0$, do:

- $Q \leftarrow 2Q$ double
- if $k_j = 1$: add $Q \leftarrow Q + G$

Step 4 : Return Q.

- Faster than repeated additions.
- Time of execution depends on number of 1s.
- Reduce Hamming weight of scalar k $\longrightarrow$ (w)NAF representation.

Double-and-add algorithm

Goal: compute fast point multiplication on elliptic curves

- Input: integer k and point G.
- Output: $Q = [k]G$

Step 1 : Convert k to binary:
$$k = k_0 + 2k_1 + 2^2 k_2 + \cdots + 2^t k_t$$

Step 2 : Initialize $Q = \mathcal{O}$

Step 3 : For $j = t, \cdots, 0$, do:

- $Q \leftarrow 2Q$ double
- if $k_j = 1$: add $Q \leftarrow Q + G$

Step 4 : Return Q.

- Faster than repeated additions.
- Time of execution depends on number of 1s.
- Reduce Hamming weight of scalar k $\longrightarrow$ (w)NAF representation.

Double-and-add algorithm

Goal: compute fast point multiplication on elliptic curves

- Input: integer k and point G.
- Output: $Q = [k]G$

Step 1 : Convert k to binary:
$$k = k_0 + 2k_1 + 2^2 k_2 + \cdots + 2^t k_t$$

Step 2 : Initialize $Q = \mathcal{O}$

Step 3 : For $j = t, \cdots, 0$, do:
- $Q \leftarrow 2Q$ double
- if $k_j = 1$: add $Q \leftarrow Q + G$

Step 4 : Return Q.

- Faster than repeated additions.
- Time of execution depends on number of 1s.
- Reduce Hamming weight of scalar k $\longrightarrow$ (w)NAF representation.

Double-and-add algorithm

Goal: compute fast point multiplication on elliptic curves

- Input: integer k and point G.
- Output: $Q = [k]G$

Step 1 : Convert k to binary:
$k = k_0 + 2k_1 + 2^2 k_2 + \cdots + 2^t k_t$

Step 2 : Initialize $Q = \mathcal{O}$

Step 3 : For $j = t, \cdots, 0$, do:

- $Q \leftarrow 2Q$ double
- if $k_j = 1$: add $Q \leftarrow Q + G$

Step 4 : Return Q.

- Faster than repeated additions.
- Time of execution depends on number of 1s.
- Reduce Hamming weight of scalar k $\longrightarrow$ (w)NAF representation.

Double-and-add algorithm

Goal: compute fast point multiplication on elliptic curves

- Input: integer k and point G.
- Output: $Q = [k]G$

Step 1 : Convert k to binary:
$$k = k_0 + 2k_1 + 2^2 k_2 + \cdots + 2^t k_t$$

Step 2 : Initialize $Q = \mathcal{O}$

Step 3 : For $j = t, \cdots, 0$, do:

- $Q \leftarrow 2Q$ double
- if $k_j = 1$: add $Q \leftarrow Q + G$

Step 4 : Return Q.

- Faster than repeated additions.
- Time of execution depends on number of 1s.
- Reduce Hamming weight of scalar k $\longrightarrow$ (w)NAF representation.

Double-and-add algorithm

Goal: compute fast point multiplication on elliptic curves

- Input: integer k and point G.
- Output: $Q = [k]G$

Step 1 : Convert k to binary:
$$k = k_0 + 2k_1 + 2^2 k_2 + \cdots + 2^t k_t$$

Step 2 : Initialize $Q = \mathcal{O}$

Step 3 : For $j = t, \cdots, 0$, do:

- $Q \leftarrow 2Q$ double
- if $k_j = 1$: add $Q \leftarrow Q + G$

Step 4 : Return Q.

- Faster than repeated additions.
- Time of execution depends on number of 1s.
- Reduce Hamming weight of scalar k $\longrightarrow$ (w)NAF representation.

Double-and-add algorithm

Goal: compute fast point multiplication on elliptic curves

- Input: integer k and point G.
- Output: $Q = [k]G$

Step 1 : Convert k to binary:
$$k = k_0 + 2k_1 + 2^2 k_2 + \cdots + 2^t k_t$$

Step 2 : Initialize $Q = \mathcal{O}$

Step 3 : For $j = t, \cdots, 0$, do:

- $Q \leftarrow 2Q$ double
- if $k_j = 1$: add $Q \leftarrow Q + G$

Step 4 : Return Q.

- Faster than repeated additions.
- Time of execution depends on number of 1s.
- Reduce Hamming weight of scalar k $\longrightarrow$ (w)NAF representation.

Double-and-add algorithm

Goal: compute fast point multiplication on elliptic curves

- Input: integer k and point G.
- Output: $Q = [k]G$

Step 1 : Convert k to binary:
$$k = k_0 + 2k_1 + 2^2 k_2 + \cdots + 2^t k_t$$

Step 2 : Initialize $Q = \mathcal{O}$

Step 3 : For $j = t, \cdots, 0$, do:

- $Q \leftarrow 2Q$ double
- if $k_j = 1$: add $Q \leftarrow Q + G$

Step 4 : Return Q.

- Faster than repeated additions.
- Time of execution depends on number of 1s.
- Reduce Hamming weight of scalar k $\longrightarrow$ (w)NAF representation.

Double-and-add algorithm

Goal: compute fast point multiplication on elliptic curves

- Input: integer k and point G.
- Output: $Q = [k]G$

Step 1 : Convert k to binary:
$$k = k_0 + 2k_1 + 2^2 k_2 + \cdots + 2^t k_t$$

Step 2 : Initialize $Q = \mathcal{O}$

Step 3 : For $j = t, \cdots, 0$, do:

- $Q \leftarrow 2Q$ double
- if $k_j = 1$: add $Q \leftarrow Q + G$

Step 4 : Return Q.

- Faster than repeated additions.
- Time of execution depends on number of 1s.
- Reduce Hamming weight of scalar k $\longrightarrow$ (w)NAF representation.

Double-and-add algorithm

Goal: compute fast point multiplication on elliptic curves

- Input: integer k and point G.
- Output: $Q = [k]G$

Step 1 : Convert k to binary:
$$k = k_0 + 2k_1 + 2^2 k_2 + \cdots + 2^t k_t$$

Step 2 : Initialize $Q = \mathcal{O}$

Step 3 : For $j = t, \cdots, 0$, do:
- $Q \leftarrow 2Q$ double
- if $k_j = 1$: add $Q \leftarrow Q + G$

Step 4 : Return Q.

- Faster than repeated additions.
- Time of execution depends on number of 1s.
- Reduce Hamming weight of scalar k $\longrightarrow$ (w)NAF representation.

Non-adjacent form (NAF) and windowed-NAF (wNAF)

NAF:

- Impossible to have two consecutive non-zero digits,
- signed digits -1, 0, 1

wNAF:

- Impossible to have two consecutive non-zero digits,
- signed digits are in a larger window: $\in [-2^w + 1, 2^w - 1]$.

Example, 3 representations of 23:

- binary: $23 = 2^4 + 2^2 + 2^1 + 2^0 = (1, 0, 1, 1, 1)$
- NAF: $23 = 2^5 - 2^3 - 2^0 = (1, 0, -1, 0, 0, -1)$
- wNAF (for w=3): $23 = 2^4 + 7 \times 2^0 = (1, 0, 0, 0, 7)$

Non-adjacent form (NAF) and windowed-NAF (wNAF)

NAF:

- Impossible to have two consecutive non-zero digits,
- signed digits -1, 0, 1

wNAF:

- Impossible to have two consecutive non-zero digits,
- signed digits are in a larger window: $\in [-2^w + 1, 2^w - 1]$.

Example, 3 representations of 23:

- binary: $23 = 2^4 + 2^2 + 2^1 + 2^0 = (1, 0, 1, 1, 1)$
- NAF: $23 = 2^5 - 2^3 - 2^0 = (1, 0, -1, 0, 0, -1)$
- wNAF (for w=3): $23 = 2^4 + 7 \times 2^0 = (1, 0, 0, 0, 7)$

Non-adjacent form (NAF) and windowed-NAF (wNAF)

NAF:

- Impossible to have two consecutive non-zero digits,
- signed digits -1, 0, 1

wNAF:

- Impossible to have two consecutive non-zero digits,
- signed digits are in a larger window: $\in [-2^w + 1, 2^w - 1]$.

Example, 3 representations of 23:

- binary: $23 = 2^4 + 2^2 + 2^1 + 2^0 = (1, 0, 1, 1, 1)$
- NAF: $23 = 2^5 - 2^3 - 2^0 = (1, 0, -1, 0, 0, -1)$
- wNAF (for w=3): $23 = 2^4 + 7 \times 2^0 = (1, 0, 0, 0, 7)$

wNAF in the wild

ECSDA with wNAF representation is used in:

- Bitcoin, as the signing algorithm for the transactions
- Some common libraries:
  - OpenSSL up to May 2019
  - Cryptlib
  - BouncyCastle
  - Apple's CommonCrypto

Oh no! Information is being leaked!

The power of side-channel attacks:

Double and add is **not** constant time (depends on the number of non-zero coeff).
$\longrightarrow$ (Cache) timing attacks **identify** (most) of **the positions of the non-zero coefficients in the wNAF representation of the nonce** k**.**

Real k (wNAF) representation (unknown from an attacker):
1 0 0 0 7 0 0 0 0 0 0 -7 0 0 0 0 0 0 3 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0
Information obtained by side channels:
$\star$ 0 0 0 $\star$ 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 0

Oh no! Information is being leaked!

The power of side-channel attacks:

Double and add is **not** constant time (depends on the number of non-zero coeff).

$\longrightarrow$ (Cache) timing attacks **identify** (most) of **the positions of the non-zero coefficients in the wNAF representation of the nonce** k**.**

Real k (wNAF) representation (unknown from an attacker):

1 0 0 0 7 0 0 0 0 0 0 -7 0 0 0 0 0 0 3 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0

Information obtained by side channels:

$\star$ 0 0 0 $\star$ 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 0

Oh no! Information is being leaked!

The power of side-channel attacks:

Double and add is **not** constant time (depends on the number of non-zero coeff).
$\longrightarrow$ (Cache) timing attacks **identify** (most) of **the positions of the non-zero coefficients in the wNAF representation of the nonce k.**

Real k (wNAF) representation (unknown from an attacker):
1 0 0 0 7 0 0 0 0 0 0 -7 0 0 0 0 0 0 3 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0
Information obtained by side channels:
$\star$ 0 0 0 $\star$ 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 0

Oh no! Information is being leaked!

The power of side-channel attacks:

Double and add is **not** constant time (depends on the number of non-zero coeff).
$\longrightarrow$ (Cache) timing attacks **identify** (most) of **the positions of the non-zero coefficients in the wNAF representation of the nonce k.**

Real k (wNAF) representation (unknown from an attacker):

1 0 0 0 7 0 0 0 0 0 0 -7 0 0 0 0 0 0 3 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0

Information obtained by side channels:

$\star$ 0 0 0 $\star$ 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 0

Oh no! Information is being leaked!

The power of side-channel attacks:

Double and add is **not** constant time (depends on the number of non-zero coeff).
$\longrightarrow$ (Cache) timing attacks **identify** (most) of **the positions of the non-zero coefficients in the wNAF representation of the nonce k.**

Real k (wNAF) representation (unknown from an attacker):

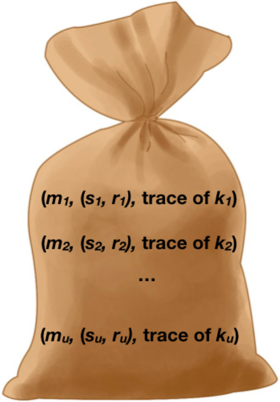
1 0 0 0 7 0 0 0 0 0 0 -7 0 0 0 0 0 0 3 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0

Information obtained by side channels:

$\star$ 0 0 0 $\star$ 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 0

Information collected

What we have:
Many messages m_i with their signatures (s_i, r_i), signed by a unique secret key α.



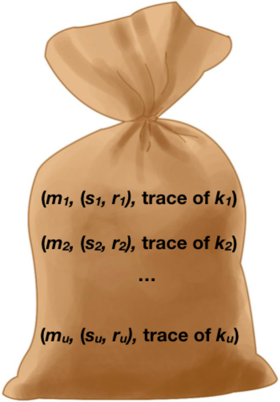
Side channels give the trace of k_i:
$\star$ 0 0 0 $\star$ 0 0 0 0 $\star$ 0 0 0 $\star$ 0 0 0 $\star$ 0 0 0

The important information is:

- number of non-zero coefficients, ℓ_i
- position of non-zero coefficients, $\lambda_1, \cdots, \lambda_{\ell_i}$

Information collected

What we have:
Many messages m_i with their signatures (s_i, r_i), signed by a unique secret key α.



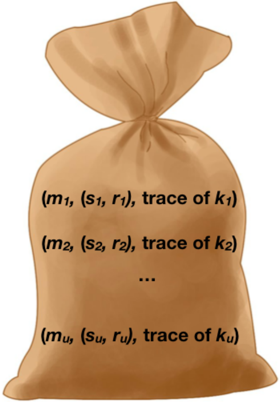
Side channels give the trace of k_i:
$\star$ 0 0 0 $\star$ 0 0 0 0 $\star$ 0 0 0 $\star$ 0 0 0 $\star$ 0 0 0

The important information is:

- number of non-zero coefficients, ℓ_i
- position of non-zero coefficients, $\lambda_1, \cdots, \lambda_{\ell_i}$

Information collected

What we have:
Many messages m_i with their signatures (s_i, r_i), signed by a unique secret key α.



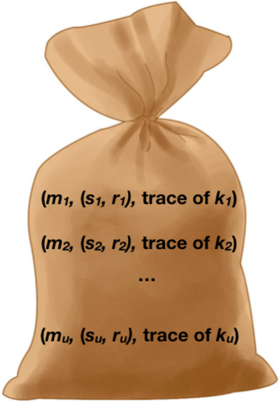
Side channels give the trace of k_i:
$\star$ 0 0 0 $\star$ 0 0 0 0 $\star$ 0 0 0 $\star$ 0 0 0 $\star$ 0 0 0

The important information is:

- number of non-zero coefficients, ℓ_i
- position of non-zero coefficients, $\lambda_1, \cdots, \lambda_{\ell_i}$

Information collected

What we have:
Many messages m_i with their signatures (s_i, r_i), signed by a unique secret key α.



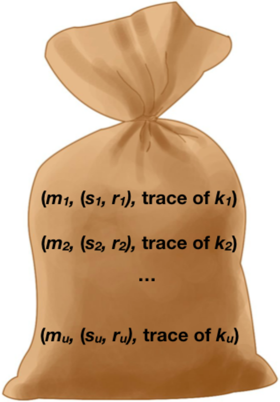
Side channels give the trace of k_i:
$\star$ 0 0 0 $\star$ 0 0 0 0 $\star$ 0 0 0 $\star$ 0 0 0 $\star$ 0 0 0

The important information is:

- number of non-zero coefficients, ℓ_i
- position of non-zero coefficients, $\lambda_1, \cdots, \lambda_{\ell_i}$

Information collected

What we have:
Many messages m_i with their signatures (s_i, r_i), signed by a unique secret key α.



Side channels give the trace of k_i:
$\star$ 0 0 0 $\star$ 0 0 0 0 $\star$ 0 0 0 $\star$ 0 0 0 $\star$ 0 0 0

The important information is:

- number of non-zero coefficients, ℓ_i
- position of non-zero coefficients, $\lambda_1, \cdots, \lambda_{\ell_i}$

The Extended Hidden Number Problem

Hlavác, Rosa (SAC 2007), *Extended hidden number problem and its cryptanalytic applications.*

Consider u congruences of the form

$$a_i\alpha + \sum_{j=1}^{\ell_i} b_{i,j}k_{i,j} \equiv c_i \pmod{q},$$

- Unknowns: the secret α and $0 \leqslant k_{i,j} \leqslant 2^{\eta_{ij}}$,
- known values: modulus q, η_{ij}, a_i, $b_{i,j}$, c_i, ℓ_i for $1 \leqslant i \leqslant u$,

Recover α in polynomial time.

The Extended Hidden Number Problem

Hlavác, Rosa (SAC 2007), *Extended hidden number problem and its cryptanalytic applications.*

Consider u congruences of the form

$$a_i\alpha + \sum_{j=1}^{\ell_i} b_{i,j}k_{i,j} \equiv c_i \pmod{q},$$

- Unknowns: the secret α and $0 \leqslant k_{i,j} \leqslant 2^{\eta_{ij}}$,
- known values: modulus q, η_{ij}, a_i, $b_{i,j}$, c_i, ℓ_i for $1 \leqslant i \leqslant u$,

$\boxed{\textbf{Recover } \alpha \textbf{ in polynomial time.}}$

The Extended Hidden Number Problem

Hlavác, Rosa (SAC 2007), *Extended hidden number problem and its cryptanalytic applications.*

Consider u congruences of the form

$$a_i\alpha + \sum_{j=1}^{\ell_i} b_{i,j}k_{i,j} \equiv c_i \pmod{q},$$

- Unknowns: the secret α and $0 \leqslant k_{i,j} \leqslant 2^{\eta_{ij}}$,
- known values: modulus q, η_{ij}, a_i, $b_{i,j}$, c_i, ℓ_i for $1 \leqslant i \leqslant u$,

Recover α in polynomial time.

The Extended Hidden Number Problem

Hlavác, Rosa (SAC 2007), *Extended hidden number problem and its cryptanalytic applications.*

Consider u congruences of the form

$$a_i\alpha + \sum_{j=1}^{\ell_i} b_{i,j}k_{i,j} \equiv c_i \pmod{q},$$

- Unknowns: the secret α and $0 \leqslant k_{i,j} \leqslant 2^{\eta_{ij}}$,
- known values: modulus q, η_{ij}, a_i, $b_{i,j}$, c_i, ℓ_i for $1 \leqslant i \leqslant u$,

Recover α in polynomial time.

The Extended Hidden Number Problem

Hlavác, Rosa (SAC 2007), *Extended hidden number problem and its cryptanalytic applications.*

Consider u congruences of the form

$$a_i\alpha + \sum_{j=1}^{\ell_i} b_{i,j}k_{i,j} \equiv c_i \pmod{q},$$

- Unknowns: the secret α and $0 \leqslant k_{i,j} \leqslant 2^{\eta_{ij}}$,
- known values: modulus q, η_{ij}, a_i, $b_{i,j}$, c_i, ℓ_i for $1 \leqslant i \leqslant u$,

Recover α in polynomial time.

Using EHNP to attack ECDSA

Goal: Transform ECDSA into an EHNP setup.

- ECDSA equation:

$$\alpha r = sk - H(m) \pmod{q}.$$

- Known information on the nonce k :

$$k = \sum_{j=1}^{\ell} k_j 2^{\lambda_j} = \bar{k} + \sum_{j=1}^{\ell} d_j 2^{\lambda_j + 1},$$

- By substitution:

$$\boxed{\alpha r_i - \sum_{j=1}^{\ell_i} 2^{\lambda_{i,j}+1} s_i d_{i,j} - (s_i \bar{k}_i - H(m_i)) \equiv 0 \pmod{q}}$$

Using EHNP to attack ECDSA

Goal: Transform ECDSA into an EHNP setup.

- ECDSA equation:

$$\alpha r = sk - H(m) \pmod{q}.$$

- Known information on the nonce k :

$$k = \sum_{j=1}^{\ell} k_j 2^{\lambda_j} = \bar{k} + \sum_{j=1}^{\ell} d_j 2^{\lambda_j + 1},$$

- By substitution:

$$\boxed{\alpha r_i - \sum_{j=1}^{\ell_i} 2^{\lambda_{i,j}+1} s_i d_{i,j} - (s_i \bar{k}_i - H(m_i)) \equiv 0 \pmod{q}}$$

Using EHNP to attack ECDSA

Goal: Transform ECDSA into an EHNP setup.

- ECDSA equation:

$$\alpha r = sk - H(m) \pmod{q}.$$

- Known information on the nonce k :

$$k = \sum_{j=1}^{\ell} k_j 2^{\lambda_j} = \bar{k} + \sum_{j=1}^{\ell} d_j 2^{\lambda_j + 1},$$

- By substitution:

$$\boxed{\alpha r_i - \sum_{j=1}^{\ell_i} 2^{\lambda_{i,j}+1} s_i d_{i,j} - (s_i \bar{k}_i - H(m_i)) \equiv 0 \pmod{q}}$$

Using EHNP to attack ECDSA

Goal: Transform ECDSA into an EHNP setup.

- ECDSA equation:

$$\alpha r = sk - H(m) \pmod{q}.$$

- Known information on the nonce k :

$$k = \sum_{j=1}^{\ell} k_j 2^{\lambda_j} = \bar{k} + \sum_{j=1}^{\ell} d_j 2^{\lambda_j + 1},$$

- By substitution:

$$\boxed{\alpha r_i - \sum_{j=1}^{\ell_i} 2^{\lambda_{i,j}+1} s_i d_{i,j} - (s_i \bar{k}_i - H(m_i)) \equiv 0 \pmod{q}}$$

The Extended Hidden Number Problem

We now have u congruences of the form

$$a_i\alpha + \sum_{j=1}^{\ell_i} b_{i,j}k_{i,j} \equiv c_i \pmod{q},$$

given by

$$E_i : \boxed{\alpha r_i - \sum_{j=1}^{\ell_i} 2^{\lambda_{i,j}+1} s_i d_{i,j} - (s_i \bar{k}_i - H(m_i)) \equiv 0 \pmod{q}}$$

- Unknowns: the secret key α and $0 \leqslant d_{i,j} \leqslant 2^{\mu_{i,j}}$,
- known values: modulus q, r_i, $\lambda_{i,j}$, s_i, $\bar{k}_i$, ℓ_i, $H(m_i)$, $\mu_{i,j}$ for $1 \leqslant i \leqslant u$,

Recover α in polynomial time.

HOW? $\longrightarrow$ with lattices

The Extended Hidden Number Problem

We now have u congruences of the form

$$a_i\alpha + \sum_{j=1}^{\ell_i} b_{i,j}k_{i,j} \equiv c_i \pmod{q},$$

given by

$$E_i : \boxed{\alpha r_i - \sum_{j=1}^{\ell_i} 2^{\lambda_{i,j}+1} s_i d_{i,j} - (s_i \bar{k}_i - H(m_i)) \equiv 0 \pmod{q}}$$

- Unknowns: the secret key α and $0 \leqslant d_{i,j} \leqslant 2^{\mu_{i,j}}$,
- known values: modulus $q, r_i, \lambda_{i,j}, s_i, \bar{k}_i, \ell_i, H(m_i), \mu_{i,j}$ for $1 \leqslant i \leqslant u$,

Recover α in polynomial time.

HOW? $\longrightarrow$ with lattices

The Extended Hidden Number Problem

We now have u congruences of the form

$$a_i \alpha + \sum_{j=1}^{\ell_i} b_{i,j} k_{i,j} \equiv c_i \pmod{q},$$

given by

$$E_i : \boxed{\alpha r_i - \sum_{j=1}^{\ell_i} 2^{\lambda_{i,j}+1} s_i d_{i,j} - (s_i \bar{k}_i - H(m_i)) \equiv 0 \pmod{q}}$$

- Unknowns: the secret key α and $0 \leqslant d_{i,j} \leqslant 2^{\mu_{i,j}}$,
- known values: modulus $q, r_i, \lambda_{i,j}, s_i, \bar{k}_i, \ell_i, H(m_i), \mu_{i,j}$ for $1 \leqslant i \leqslant u$,

Recover α in polynomial time.

HOW? $\longrightarrow$ with lattices

The Extended Hidden Number Problem

We now have u congruences of the form

$$a_i\alpha + \sum_{j=1}^{\ell_i} b_{i,j}k_{i,j} \equiv c_i \pmod{q},$$

given by

$$E_i : \boxed{\alpha r_i - \sum_{j=1}^{\ell_i} 2^{\lambda_{i,j}+1} s_i d_{i,j} - (s_i \bar{k}_i - H(m_i)) \equiv 0 \pmod{q}}$$

- Unknowns: the secret key α and $0 \leqslant d_{i,j} \leqslant 2^{\mu_{i,j}}$,
- known values: modulus $q, r_i, \lambda_{i,j}, s_i, \bar{k}_i, \ell_i, H(m_i), \mu_{i,j}$ for $1 \leqslant i \leqslant u$,

Recover α in polynomial time.

HOW? $\longrightarrow$ with lattices

The Extended Hidden Number Problem

We now have u congruences of the form

$$a_i\alpha + \sum_{j=1}^{\ell_i} b_{i,j}k_{i,j} \equiv c_i \pmod{q},$$

given by

$$E_i : \boxed{\alpha r_i - \sum_{j=1}^{\ell_i} 2^{\lambda_{i,j}+1}s_i d_{i,j} - (s_i\bar{k}_i - H(m_i)) \equiv 0 \pmod{q}}$$

- Unknowns: the secret key α and $0 \leqslant d_{i,j} \leqslant 2^{\mu_{i,j}}$,
- known values: modulus q, r_i, $\lambda_{i,j}$, s_i, $\bar{k}_i$, ℓ_i, $H(m_i)$, $\mu_{i,j}$ for $1 \leqslant i \leqslant u$,

Recover α in polynomial time.

HOW? $\longrightarrow$ with lattices

The Extended Hidden Number Problem

We now have u congruences of the form

$$a_i\alpha + \sum_{j=1}^{\ell_i} b_{i,j}k_{i,j} \equiv c_i \pmod{q},$$

given by

$$E_i : \boxed{\alpha r_i - \sum_{j=1}^{\ell_i} 2^{\lambda_{i,j}+1} s_i d_{i,j} - (s_i\bar{k}_i - H(m_i)) \equiv 0 \pmod{q}}$$

- Unknowns: the secret key α and $0 \leqslant d_{i,j} \leqslant 2^{\mu_{i,j}}$,
- known values: modulus q, r_i, $\lambda_{i,j}$, s_i, $\bar{k}_i$, ℓ_i, $H(m_i)$, $\mu_{i,j}$ for $1 \leqslant i \leqslant u$,

$\boxed{\textbf{Recover } \alpha \textbf{ in polynomial time.}}$

HOW? $\rightarrow$ with lattices

The Extended Hidden Number Problem

We now have u congruences of the form

$$a_i\alpha + \sum_{j=1}^{\ell_i} b_{i,j}k_{i,j} \equiv c_i \pmod{q},$$

given by

$$E_i : \boxed{\alpha r_i - \sum_{j=1}^{\ell_i} 2^{\lambda_{i,j}+1} s_i d_{i,j} - (s_i\bar{k}_i - H(m_i)) \equiv 0 \pmod{q}}$$

- Unknowns: the secret key α and $0 \leqslant d_{i,j} \leqslant 2^{\mu_{i,j}}$,
- known values: modulus $q, r_i, \lambda_{i,j}, s_i, \bar{k}_i, \ell_i, H(m_i), \mu_{i,j}$ for $1 \leqslant i \leqslant u$,

Recover α in polynomial time.

HOW? $\longrightarrow$ with lattices

Reducing the size of the system

- We start with our system of modular equations E_i.
- Basic trick: Reduce the size of the system by eliminating α from the equations: $r_1 E_i - r_i E_1$
  - Remember that

$$\alpha = r_1^{-1}\left(\sum_{i=1}^{\ell_1} 2^{\lambda_{1,j}+1} s_1 d_{1,j} + (s_1 \bar{k}_1 - z_1)\right) \pmod{q}.$$

- New Goal: recover the $d_{i,j}$, with a new system of equations:

$$E_i' : \quad \sum_{j=1}^{\ell_1} \underbrace{(2^{\lambda_{1,j}+1} s_1 r_i)}_{:=\tau_{j,i}} d_{1,j} + \sum_{j=1}^{\ell_i} \underbrace{(-2^{\lambda_{i,j}+1} s_i r_1)}_{:=\sigma_{i,j}} d_{i,j}$$
$$- \underbrace{r_1(s_i \bar{k}_i - H(m_i)) + r_i(s_1 \bar{k}_1 - H(m_1))}_{:=\gamma_i} \equiv 0 \pmod{q}.$$

Reducing the size of the system

- We start with our system of modular equations E_i.
- Basic trick: Reduce the size of the system by eliminating α from the equations: $r_1 E_i - r_i E_1$
  - Remember that

$$\alpha = r_1^{-1}\left(\sum_{i=1}^{\ell_1} 2^{\lambda_{1,j}+1} s_1 d_{1,j} + (s_1 \bar{k}_1 - z_1)\right) \pmod{q}.$$

- New Goal: recover the $d_{i,j}$, with a new system of equations:

$$\begin{aligned} E'_i: \quad & \textstyle\sum_{j=1}^{\ell_1} \underbrace{(2^{\lambda_{1,j}+1} s_1 r_i)}_{:=\tau_{j,i}} d_{1,j} + \sum_{j=1}^{\ell_i} \underbrace{(-2^{\lambda_{i,j}+1} s_i r_1)}_{:=\sigma_{i,j}} d_{i,j} \\ & - \underbrace{r_1(s_i \bar{k}_i - H(m_i)) + r_i(s_1 \bar{k}_1 - H(m_1))}_{:=\gamma_i} \equiv 0 \pmod{q}. \end{aligned}$$

Reducing the size of the system

- We start with our system of modular equations E_i.
- Basic trick: Reduce the size of the system by eliminating α from the equations: $r_1 E_i - r_i E_1$
  - Remember that

$$\alpha = r_1^{-1}\left(\sum_{i=1}^{\ell_1} 2^{\lambda_{1,j}+1} s_1 d_{1,j} + (s_1 \bar{k}_1 - z_1)\right) \pmod{q}.$$

- New Goal: recover the $d_{i,j}$, with a new system of equations:

$$E_i' : \quad \sum_{j=1}^{\ell_1} \underbrace{(2^{\lambda_{1,j}+1} s_1 r_i)}_{:=\tau_{j,i}} d_{1,j} + \sum_{j=1}^{\ell_i} \underbrace{(-2^{\lambda_{i,j}+1} s_i r_1)}_{:=\sigma_{i,j}} d_{i,j}$$
$$- \underbrace{r_1(s_i \bar{k}_i - H(m_i)) + r_i(s_1 \bar{k}_1 - H(m_1))}_{:=\gamma_i} \equiv 0 \pmod{q}.$$

Reducing the size of the system

- We start with our system of modular equations E_i.
- Basic trick: Reduce the size of the system by eliminating α from the equations: $r_1 E_i - r_i E_1$
  - Remember that

$$\alpha = r_1^{-1} \left(\sum_{i=1}^{\ell_1} 2^{\lambda_{1,j}+1} s_1 d_{1,j} + (s_1 \bar{k}_1 - z_1) \right) \pmod{q}.$$

- New Goal: recover the $d_{i,j}$, with a new system of equations:

$$\begin{aligned} E_i' : \quad & \textstyle\sum_{j=1}^{\ell_1} \underbrace{(2^{\lambda_{1,j}+1} s_1 r_i)}_{:=\tau_{j,i}} d_{1,j} + \sum_{j=1}^{\ell_i} \underbrace{(-2^{\lambda_{i,j}+1} s_i r_1)}_{:=\sigma_{i,j}} d_{i,j} \\ & - \underbrace{r_1(s_i \bar{k}_i - H(m_i)) + r_i(s_1 \bar{k}_1 - H(m_1))}_{:=\gamma_i} \equiv 0 \pmod{q}. \end{aligned}$$

Lattice: Definition, bad and good bases

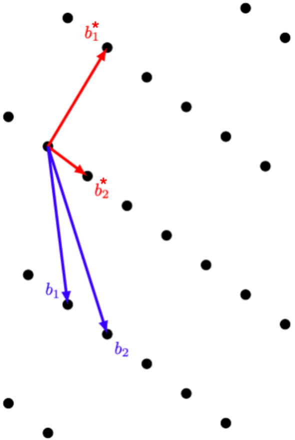
Definition

A lattice is a discrete additive subgroup of $\mathbb{R}^n$, usually identified by a basis $\{b_1, \cdots, b_n\}$.

Reduction algorithms: BKZ or LLL

Given an arbitrary basis $\{b_1, \cdots, b_n\}$, find a "better" basis $\{b_1^*, \cdots, b_n^*\}$.

Better → the first vectors are shorter (and more orthogonal) in the reduced basis.



Lattice: Definition, bad and good bases

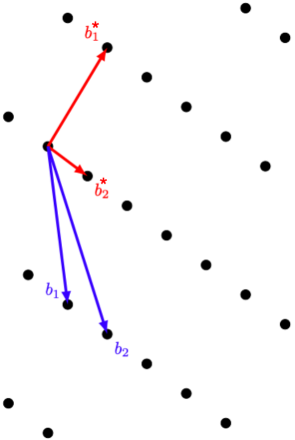
Definition

A lattice is a discrete additive subgroup of $\mathbb{R}^n$, usually identified by a basis $\{b_1, \cdots, b_n\}$.

Reduction algorithms: BKZ or LLL

Given an arbitrary basis $\{b_1, \cdots, b_n\}$, find a "better" basis $\{b_1^*, \cdots, b_n^*\}$.

Better $\rightarrow$ the first vectors are shorter (and more orthogonal) in the reduced basis.



Our lattice construction

We construct a lattice such that there exists a linear combination v of the lines containing the $d_{i,j}$:

$$v = (t_2, \cdots, t_u, d_{1,1}, \cdots, d_{u,\ell_u}, -1) \times \left(\begin{array}{cccc|cccc} q & & & & & & & \\ & \ddots & & & & & & \\ & & \ddots & & & & & \\ & & & q & & & & \\ \hline E'_2 & E'_3 & \dots & E'_u & 2^{m-\mu_{1,1}} & & & \\ \vdots & \vdots & \vdots & \vdots & & \ddots & & \\ \vdots & \vdots & \vdots & \vdots & & & 2^{m-\mu_{u,\ell_u}} & \\ & & & & 2^m & \dots & & 2^m \end{array} \right)$$

$$\boxed{v = (0, \dots, 0, d_{1,1}2^{m-\mu_{1,1}} - 2^{m-1}, \dots, d_{u,\ell_u}2^{m-\mu_{u,\ell_u}} - 2^{m-1}, -2^{m-1}).}$$

How to find v?

Goal: Find v.

- Good point: v has a particular shape
- ⚠ It has no reason to appear in the basis
- $\longrightarrow$
  1. Make it short (by ugly manipulations of the lattice)
  2. Run BKZ on the basis[1]
  3. Pray to find a good shaped vector in the reduced basis
  4. Try to reconstruct α with the plausible $d_{i,j}$ you get.

[1] In practice $80 \leqslant \dim(\text{lattice}) \leqslant 215$.

How to find v?

Goal: Find v.

- Good point: v has a particular shape
- ⚠ It has no reason to appear in the basis
- $\longrightarrow$
  1. Make it short (by ugly manipulations of the lattice)
  2. Run BKZ on the basis[1]
  3. Pray to find a good shaped vector in the reduced basis
  4. Try to reconstruct α with the plausible $d_{i,j}$ you get.

[1] In practice $80 \leqslant \dim(\text{lattice}) \leqslant 215$.

How to find v?

Goal: Find v.

- Good point: v has a particular shape
- ⚠ It has no reason to appear in the basis
- $\longrightarrow$
  1. Make it short (by ugly manipulations of the lattice)
  2. Run BKZ on the basis[1]
  3. Pray to find a good shaped vector in the reduced basis
  4. Try to reconstruct α with the plausible $d_{i,j}$ you get.

[1] In practice $80 \leqslant \dim(\text{lattice}) \leqslant 215$.

How to find v?

Goal: Find v.

- Good point: v has a particular shape
- ⚠ It has no reason to appear in the basis
- $\longrightarrow$
  1. Make it short (by ugly manipulations of the lattice)
  2. Run BKZ on the basis[1]
  3. Pray to find a good shaped vector in the reduced basis
  4. Try to reconstruct α with the plausible $d_{i,j}$ you get.

[1] In practice $80 \leqslant \dim(\text{lattice}) \leqslant 215$.

How to find v?

Goal: Find v.

- Good point: v has a particular shape
- ⚠ It has no reason to appear in the basis
- $\longrightarrow$
  1. Make it short (by ugly manipulations of the lattice)
  2. Run BKZ on the basis[1]
  3. Pray to find a good shaped vector in the reduced basis
  4. Try to reconstruct α with the plausible $d_{i,j}$ you get.

[1] In practice $80 \leqslant \dim(\text{lattice}) \leqslant 215$.

How to find v?

Goal: Find v.

- Good point: v has a particular shape
- ⚠ It has no reason to appear in the basis
- $\longrightarrow$
  1. Make it short (by ugly manipulations of the lattice)
  2. Run BKZ on the basis[1]
  3. Pray to find a good shaped vector in the reduced basis
  4. Try to reconstruct α with the plausible $d_{i,j}$ you get.

[1] In practice $80 \leqslant \dim(\text{lattice}) \leqslant 215$.

A new pre-processing method to speed-up the reduction

The slowest part of the attack: lattice reduction.

BKZ reduction time $\searrow$ if dimension $\searrow$ OR coefficients size $\searrow$.

Goal: Speed up the reduction time by $\searrow$ the size of the coefficients.

- Each trace t comes with a notion of "weight" $\mu(t)$.
- Each coefficient of the basis is multiplied by $m = \max \mu(t)$ to get integer coefficients.
- The size of the coefficients depends on m.

Idea: $\longrightarrow$ **pre-select traces with small weight**

$$\boxed{S_a = \{t \in \mathcal{T} | \mu(t) \leqslant a\}}$$

Numerical experiment: 5000 traces from OpenSSL: $a \in [11, 67]$.

A new pre-processing method to speed-up the reduction

The slowest part of the attack: lattice reduction.
BKZ reduction time $\searrow$ if dimension $\searrow$ OR coefficients size $\searrow$.

Goal: Speed up the reduction time by $\searrow$ the size of the coefficients.

- Each trace t comes with a notion of "weight" $\mu(t)$.
- Each coefficient of the basis is multiplied by $m = \max \mu(t)$ to get integer coefficients.
- The size of the coefficients depends on m.

Idea: $\longrightarrow$ **pre-select traces with small weight**

$$\boxed{S_a = \{t \in \mathcal{T} | \mu(t) \leqslant a\}}$$

Numerical experiment: 5000 traces from OpenSSL: $a \in [11, 67]$.

A new pre-processing method to speed-up the reduction

The slowest part of the attack: lattice reduction.
BKZ reduction time $\searrow$ if dimension $\searrow$ OR coefficients size $\searrow$.

Goal: Speed up the reduction time by $\searrow$ the size of the coefficients.

- Each trace t comes with a notion of "weight" $\mu(t)$.
- Each coefficient of the basis is multiplied by $m = \max \mu(t)$ to get integer coefficients.
- The size of the coefficients depends on m.

Idea: $\longrightarrow$ **pre-select traces with small weight**

$$\boxed{S_a = \{t \in \mathcal{T} | \mu(t) \leqslant a\}}$$

Numerical experiment: 5000 traces from OpenSSL: $a \in [11, 67]$.

A new pre-processing method to speed-up the reduction

The slowest part of the attack: lattice reduction.
BKZ reduction time $\searrow$ if dimension $\searrow$ OR coefficients size $\searrow$.

Goal: Speed up the reduction time by $\searrow$ the size of the coefficients.

- Each trace t comes with a notion of "weight" $\mu(t)$.
- Each coefficient of the basis is multiplied by $m = \max \mu(t)$ to get integer coefficients.
- The size of the coefficients depends on m.

Idea: $\longrightarrow$ **pre-select traces with small weight**

$$\boxed{S_a = \{t \in \mathcal{T} | \mu(t) \leqslant a\}}$$

Numerical experiment: 5000 traces from OpenSSL: $a \in [11, 67]$.

A new pre-processing method to speed-up the reduction

The slowest part of the attack: lattice reduction.
BKZ reduction time $\searrow$ if dimension $\searrow$ OR coefficients size $\searrow$.

Goal: Speed up the reduction time by $\searrow$ the size of the coefficients.

- Each trace t comes with a notion of "weight" $\mu(t)$.
- Each coefficient of the basis is multiplied by $m = \max \mu(t)$ to get integer coefficients.
- The size of the coefficients depends on m.

Idea: $\longrightarrow$ **pre-select traces with small weight**

$$\boxed{S_a = \{t \in \mathcal{T} | \mu(t) \leqslant a\}}$$

Numerical experiment: 5000 traces from OpenSSL: $a \in [11, 67]$.

A new pre-processing method to speed-up the reduction

The slowest part of the attack: lattice reduction.
BKZ reduction time $\searrow$ if dimension $\searrow$ OR coefficients size $\searrow$.

Goal: Speed up the reduction time by $\searrow$ the size of the coefficients.

- Each trace t comes with a notion of "weight" $\mu(t)$.
- Each coefficient of the basis is multiplied by $m = \max \mu(t)$ to get integer coefficients.
- The size of the coefficients depends on m.

Idea: $\longrightarrow$ **pre-select traces with small weight**

$$\boxed{S_a = \{t \in \mathcal{T} | \mu(t) \leqslant a\}}$$

Numerical experiment: 5000 traces from OpenSSL: $a \in [11, 67]$.

A new pre-processing method to speed-up the reduction

The slowest part of the attack: lattice reduction.
BKZ reduction time $\searrow$ if dimension $\searrow$ OR coefficients size $\searrow$.

Goal: Speed up the reduction time by $\searrow$ the size of the coefficients.

- Each trace t comes with a notion of "weight" $\mu(t)$.
- Each coefficient of the basis is multiplied by $m = \max \mu(t)$ to get integer coefficients.
- The size of the coefficients depends on m.

Idea: $\longrightarrow$ **pre-select traces with small weight**

$$\boxed{S_a = \{t \in \mathcal{T} | \mu(t) \leqslant a\}}$$

Numerical experiment: 5000 traces from OpenSSL: $a \in [11, 67]$.

A new pre-processing method to speed-up the reduction

The slowest part of the attack: lattice reduction.
BKZ reduction time $\searrow$ if dimension $\searrow$ OR coefficients size $\searrow$.

Goal: Speed up the reduction time by $\searrow$ the size of the coefficients.

- Each trace t comes with a notion of "weight" $\mu(t)$.
- Each coefficient of the basis is multiplied by $m = \max \mu(t)$ to get integer coefficients.
- The size of the coefficients depends on m.

Idea: $\longrightarrow$ **pre-select traces with small weight**

$$\boxed{S_a = \{t \in \mathcal{T} | \mu(t) \leqslant a\}}$$

Numerical experiment: 5000 traces from OpenSSL: $a \in [11, 67]$.

A new pre-processing method to speed-up the reduction

The slowest part of the attack: lattice reduction.
BKZ reduction time $\searrow$ if dimension $\searrow$ OR coefficients size $\searrow$.

Goal: Speed up the reduction time by $\searrow$ the size of the coefficients.

- Each trace t comes with a notion of "weight" $\mu(t)$.
- Each coefficient of the basis is multiplied by $m = \max \mu(t)$ to get integer coefficients.
- The size of the coefficients depends on m.

Idea: $\longrightarrow$ **pre-select traces with small weight**

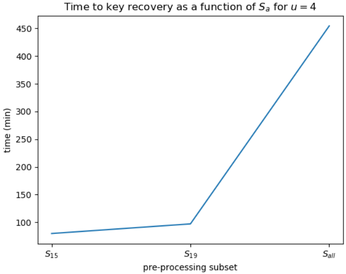
$$\boxed{S_a = \{t \in \mathcal{T} | \mu(t) \leqslant a\}}$$

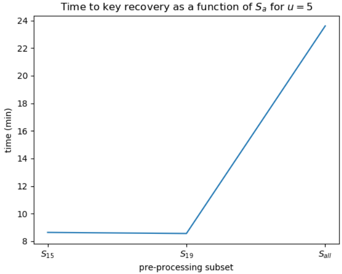
Numerical experiment: 5000 traces from OpenSSL: $a \in [11, 67]$.

The effect of pre-processing

Key recovery time $=$ time of 1 trial $\times$ nbr of trials to find the key.

- Considering 4 and 5 traces with BKZ-25.



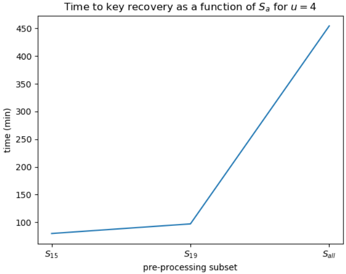


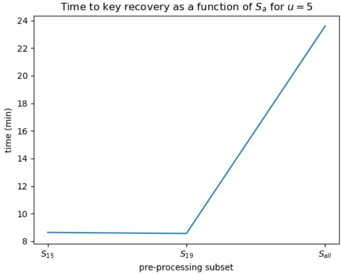
- S_{19}: already 44% of the traces
- 3 traces: from 12 days (S_{all}) to 39 h (S_{11}) on a single core.

The effect of pre-processing

Key recovery time $=$ time of 1 trial $\times$ nbr of trials to find the key.

- Considering 4 and 5 traces with BKZ-25.



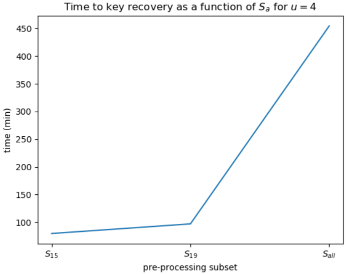


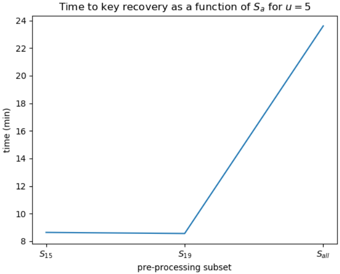
- S_{19}: already 44% of the traces
- 3 traces: from 12 days (S_{all}) to 39 h (S_{11}) on a single core.

The effect of pre-processing

Key recovery time = time of 1 trial × nbr of trials to find the key.

- Considering 4 and 5 traces with BKZ-25.



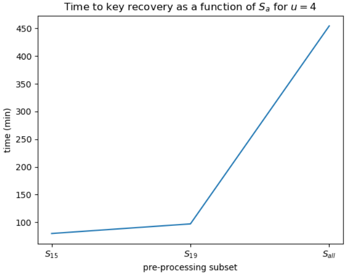


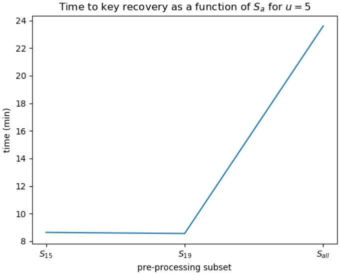
- S_{19}: already 44% of the traces
- 3 traces: from 12 days (S_{all}) to 39 h (S_{11}) on a single core.

The effect of pre-processing

Key recovery time = time of 1 trial × nbr of trials to find the key.

- Considering 4 and 5 traces with BKZ-25.





- S_{19}: already 44% of the traces
- 3 traces: from 12 days (S_{all}) to 39 h (S_{11}) on a single core.

3 ways to evaluate the attack

Several parameters need to be balanced to mount an attack:

- the preprocessing subset of traces S_a, if any
- BKZ block size β: varies between 20 and 35
    - $\beta \nearrow \Rightarrow$ probability of success of 1 trial $\nearrow$
    - but $\beta \nearrow \Rightarrow$ reduction time $\nearrow$
- a multiplying coeff. in the lattice

What is the minimal amount of signatures an attacker can use?

What are the parameters that lead to

- the fastest attack?
- the best probability of success?

3 ways to evaluate the attack

Several parameters need to be balanced to mount an attack:

- the preprocessing subset of traces S_a, if any
- BKZ block size β: varies between 20 and 35
  - $\beta \nearrow \;\Rightarrow$ probability of success of 1 trial $\nearrow$
  - but $\beta \nearrow \;\Rightarrow$ reduction time $\nearrow$
- a multiplying coeff. in the lattice

What is the minimal amount of signatures an attacker can use?

What are the parameters that lead to

- the fastest attack?
- the best probability of success?

3 ways to evaluate the attack

Several parameters need to be balanced to mount an attack:

- the preprocessing subset of traces S_a, if any
- BKZ block size β: varies between 20 and 35
  - $\beta \nearrow \Rightarrow$ probability of success of 1 trial $\nearrow$ ☺
  - but $\beta \nearrow \Rightarrow$ reduction time $\nearrow$ ☹
- a multiplying coeff. in the lattice

What is the minimal amount of signatures an attacker can use?

What are the parameters that lead to

- the fastest attack?
- the best probability of success?

3 ways to evaluate the attack

Several parameters need to be balanced to mount an attack:

- the preprocessing subset of traces S_a, if any
- BKZ block size β: varies between 20 and 35
  - $\beta \nearrow \Rightarrow$ probability of success of 1 trial $\nearrow$
  - but $\beta \nearrow \Rightarrow$ reduction time $\nearrow$
- a multiplying coeff. in the lattice

What is the minimal amount of signatures an attacker can use?

What are the parameters that lead to

- the fastest attack?
- the best probability of success?

Our Main Results

- 3 signatures: 39 hours, small probability of success, S_{11}, BKZ-35.

- Our fastest attack:
  - 4 signatures: 1 hour 17 minutes, BKZ-25, S_{15}
  - 8 signatures: 2 minutes 25 seconds, BKZ-20, S_{all}

- Our most successful attack:
  - 4 signatures: 4% of success per trial, BKZ-35, S_{all}
  - 8 signatures: 45% of success per trial, BKZ-35, S_{all}

Our Main Results

- 3 signatures: 39 hours, small probability of success, S_{11}, BKZ-35.
- Our fastest attack:
  - 4 signatures: 1 hour 17 minutes, BKZ-25, S_{15}
  - 8 signatures: 2 minutes 25 seconds, BKZ-20, S_{all}
- Our most successful attack:
  - 4 signatures: 4% of success per trial, BKZ-35, S_{all}
  - 8 signatures: 45% of success per trial, BKZ-35, S_{all}

Our Main Results

- 3 signatures: 39 hours, small probability of success, S_{11}, BKZ-35.
- Our fastest attack:
  - 4 signatures: 1 hour 17 minutes, BKZ-25, S_{15}
  - 8 signatures: 2 minutes 25 seconds, BKZ-20, S_{all}
- Our most successful attack:
  - 4 signatures: 4% of success per trial, BKZ-35, S_{all}
  - 8 signatures: 45% of success per trial, BKZ-35, S_{all}

Previous attacks on ECDSA with wNAF

- **Comparing with another variant of EHNP**
  Fan, Wang, Cheng (CCS 2016), *Attacking OpenSSL implementation of ECDSA with a few signatures*

Attack	# signatures	Probability of success	Overall time
[FWC2016]	5	4%	15 hours/18 minutes
	6	35%	1 hour 21 minutes/18 minutes
	7	68%	2 hours 23 minutes/34.5 minutes
Our attack	3	0.2%	39 hours
	4	4%	1 hour 17 minutes
	5	20%	8 minutes 20 seconds
	6	40%	5 minutes
	7	45%	3 minutes
	8	45%	2 minutes

- **Comparing with the Hidden Number Problem**
  Van de Pol, Smart, Yarom (CT-RSA 2015) *Just a Little Bit More.*
  13 signatures, 54% probability of success and 21 seconds total time to key recovery.

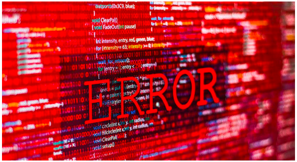
Previous attacks on ECDSA with wNAF

- **Comparing with another variant of EHNP**
  Fan, Wang, Cheng (CCS 2016), *Attacking OpenSSL implementation of ECDSA with a few signatures*

Attack	# signatures	Probability of success	Overall time
[FWC2016]	5	4%	15 hours/18 minutes
	6	35%	1 hour 21 minutes/18 minutes
	7	68%	2 hours 23 minutes/34.5 minutes
Our attack	3	0.2%	39 hours
	4	4%	1 hour 17 minutes
	5	20%	8 minutes 20 seconds
	6	40%	5 minutes
	7	45%	3 minutes
	8	45%	2 minutes

- **Comparing with the Hidden Number Problem**
  Van de Pol, Smart, Yarom (CT-RSA 2015) *Just a Little Bit More.*
  13 signatures, 54% probability of success and 21 seconds total time to key recovery.

Errors can occur, and they often do!



Side-channel analyzis is not perfect.

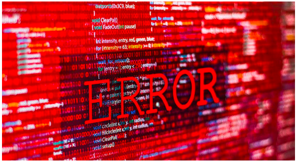
Real k (wNAF) representation (unknown from an attacker):

1 0 0 0 7 0 0 0 0 0 0 -7 0 0 0 0 0 0 3 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0

Information obtained by side channels:

$\star$ 0 0 0 $\star$ 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 0

Errors can occur, and they often do!



Side-channel analyzis is not perfect.

Real k (wNAF) representation (unknown from an attacker):

1 0 0 0 7 0 0 0 0 0 0 -7 0 0 0 0 0 0 3 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0

Information obtained by side channels:

$\star$ 0 0 0 $\star$ 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 $\star$ 0 0 0 0 0 0 0

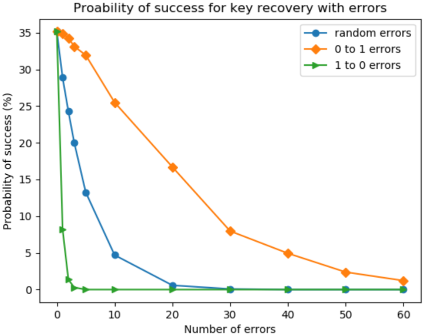
Probability of success with various types of error

Error type 1:

A 0 coefficient misread as $\star$: adds a new variable to the system, the nbr of non-zero digits is overestimated.

Error type 2:

A non-zero coefficient misread as 0: lose information necessary for key recovery.



Error 2 affects the probability of success of key recovery much more.

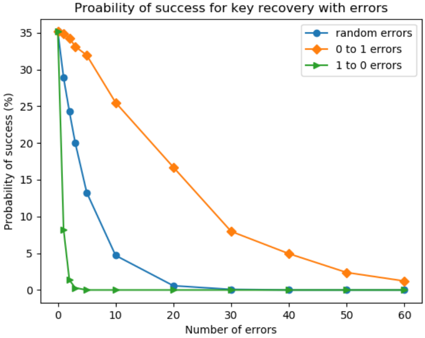
Probability of success with various types of error

Error type 1:

A 0 coefficient misread as $\star$: adds a new variable to the system, the nbr of non-zero digits is overestimated.

Error type 2:

A non-zero coefficient misread as 0: lose information necessary for key recovery.



Error 2 affects the probability of success of key recovery much more.

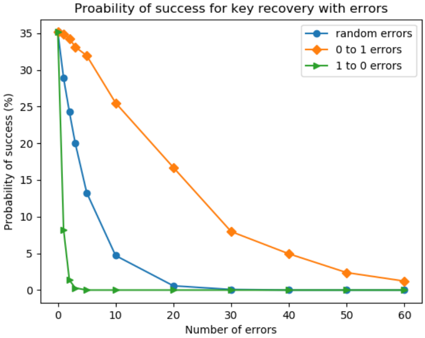
Probability of success with various types of error

Error type 1:

A 0 coefficient misread as $\star$: adds a new variable to the system, the nbr of non-zero digits is overestimated.

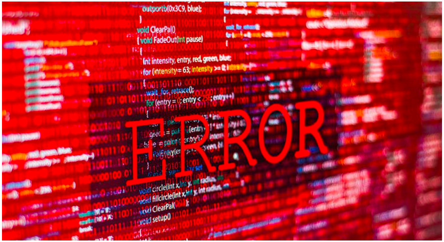
Error type 2:

A non-zero coefficient misread as 0: lose information necessary for key recovery.



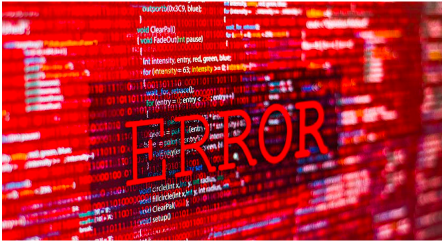
Error 2 affects the probability of success of key recovery much more.

Resilience up to 2% of errors



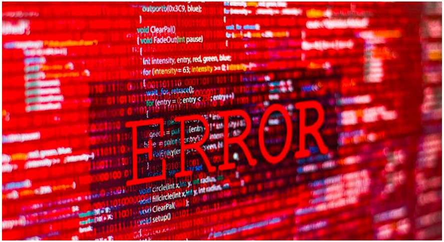
- Morality: Resilience to errors up to 2% of misread digits.
- Resilience increase to 4% if we avoid certain types of errors.
- Strategy: in the side channel part, if you are not confident about your reading, choose to put a $\star$ instead of a 0.

Resilience up to 2% of errors



- Morality: Resilience to errors up to 2% of misread digits.
- Resilience increase to 4% if we avoid certain types of errors.
- Strategy: in the side channel part, if you are not confident about your reading, choose to put a $\star$ instead of a 0.

Resilience up to 2% of errors



- Morality: Resilience to errors up to 2% of misread digits.
- Resilience increase to 4% if we avoid certain types of errors.
- Strategy: in the side channel part, if you are not confident about your reading, choose to put a $\star$ instead of a 0.

Thank you!

A Tale of Three Signatures: practical attack of ECDSA with wNAF
Gabrielle De Micheli, Cécile Pierrot, Rémi Piau
https://eprint.iacr.org/2019/861

Fastest attack

Number of signatures	Total time	Parameters			Probability of success (%)
		BKZ	Preprocessing	Δ	
3	39 hours	35	S_{11}	$\approx 2^3$	0.2
4	1 hour 17	25	S_{15}	$\approx 2^3$	0.5
5	8 min 20	25	S_{19}	$\approx 2^3$	6.5
6	3 min 55	20	S_{all}	$\approx 2^3$	7
7	2 min 43	20	S_{all}	$\approx 2^3$	17.5
8	2 min 25	20	S_{all}	$\approx 2^3$	29

Total time key recovery $=$ time of single trial $\times$ number of trials to find the key.

Highest probability of success of a single trial

Number of signatures	Probability of success (%)	Parameters			Total time
		BKZ	Preprocessing	Δ	
3	0.2	35	S_{11}	$\approx 2^3$	39 hours
4	4	35	S_{all}	$\approx 2^3$	25 hours 28
5	20	35	S_{all}	$\approx 2^3$	2 hours 42
6	40	35	S_{all}	$\approx 2^3$	1 hour 04
7	45	35	S_{all}	$\approx 2^3$	2 hours 36
8	45	35	S_{all}	$\approx 2^3$	5 hours 02

Comparing times with Fan et al, CCS 2016

Number of signatures	Our attack		Fan et al	
	Time	Success (%)	Time	Success (%)
3	39 hours	0.2%	–	–
4	1 hour 17 minutes	0.5%	41 minutes	1.5%
5	8 minutes 20 seconds	6.5%	18 minutes	1%
6	$\approx$ 5 minutes	25%	18 minutes	22%
7	$\approx$ 3 minutes	17.5%	34 minutes	24%
8	$\approx$ 2 minutes	29%	–	–

Comparing success probabilities with Fan et al, CCS 2016

Number of	Our attack		Fan et al	
signatures	Success (%)	Time	Success (%)	Time
3	0.2%	39 hours	–	–
4	4%	25 hours 28 minutes	1.5%	41 minutes
5	20%	2 hours 42 minutes	4%	36 minutes
6	40%	1 hour 4 minutes	35%	1 hour 43 minutes
7	45%	2 hours 36 minutes	68%	3 hours 58 minutes
8	45%	5 hours 2 minutes	–	–

Error analysis using BKZ-25, $\Delta \approx 2^3$ and S_{all}.

Number of signatures	Probability of success (%)				
	0 errors	5 errors	10 errors	20 errors	30 errors
4	0.28	$\ll 1$	0	0	0
5	4.58	0.86	0.18	$\ll 1$	0
6	19.52	5.26	1.26	0.14	$\ll 1$
7	33.54	10.82	3.42	0.32	$\ll 1$
8	35.14	13.26	4.70	0.58	$\ll 1$

- Corresponds to a resilience of 2% of errors.
- Total time: 1 out of 5000 experiments, 46 sec per experiment, 65 hours on a single core