Fault Attacks on a Cloud-Assisted ECDSA White-Box Based on the Residue Number System

Christophe Giraud and Agathe Houzelot

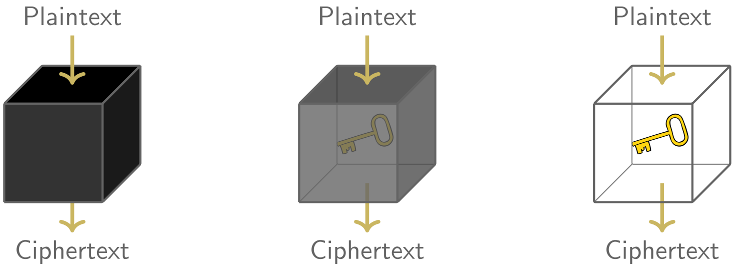
September 10, 2023

Outline

Preliminaries

1

Black-Box, Grey-Box, White-Box



Cryptanalysis	Side-channels/Faults	Read/modify binary

Fault Attacks in the White-Box Model

Goal

Disturb the execution and exploit the resulting faulty output

- Grey-box: laser, glitches
- White-box: modification of the binary, debugging tools
  - → Very precise faults, easy to reproduce
  - → New possibilities for the attacker (e.g. re-injection of intermediate values in later executions)

The Elliptic Curve Digital Signature Algorithm

- G : point of order n on an elliptic curve E
- d : 256-bit secret key
- e : hash of the plaintext

Algorithm 1: ECDSA signature

$k \xleftarrow{\$} [\![1, n-1]\!]$
$R \leftarrow [k]G$
$r \leftarrow x_R \bmod n$
$s \leftarrow k^{-1}(e + rd) \bmod n$
Return (r,s)

The Residue Number System (RNS)

- $\beta = \{p_1, \ldots, p_t\}$: set of pairwise coprime integers
- $x = (x_1, \ldots, x_t)$ with $x_i = x \bmod p_i$ if $0 \leq x < P = \prod_{i=1}^{t} p_i$
- For $\odot \in \{+, -, \times\}$, $z = x \odot y \Leftrightarrow z_i = x_i \odot y_i$ and $z = CRT(z_i)$ if $0 \leq z < P$

An RNS-based ECDSA White-Box

2

Zhou et al.'s Implementation

2020: Very first public ECDSA white-box scheme (Zhou et al.)

Zhou et al.'s Implementation

2020: Very first public ECDSA white-box scheme (Zhou et al.)

Initialization :
During a secured initialization phase a TTP generates:

Zhou et al.'s Implementation

2020: Very first public ECDSA white-box scheme (Zhou et al.)

Initialization :

During a secured initialization phase a TTP generates:

1. a basis $\beta = \{p_i\}_{1 \le i \le t}$ with $\prod_{i=1}^{t} p_i > n^2 + n$

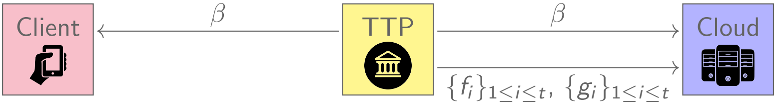
Zhou et al.'s Implementation

2020: Very first public ECDSA white-box scheme (Zhou et al.)

Initialization :

During a secured initialization phase a TTP generates:

1. a basis $\beta = \{p_i\}_{1 \le i \le t}$ with $\prod_{i=1}^{t} p_i > n^2 + n$
2. random permutations f_i and g_i on $\mathbb{Z}_{p_i}$



Zhou et al.'s Implementation

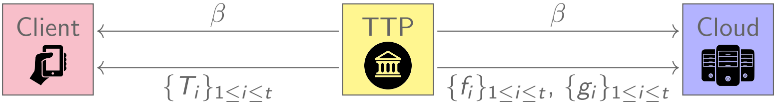
2020: Very first public ECDSA white-box scheme (Zhou et al.)

Initialization :

During a secured initialization phase a TTP generates:

1. a basis $\beta = \{p_i\}_{1 \le i \le t}$ with $\prod_{i=1}^{t} p_i > n^2 + n$
2. random permutations f_i and g_i on $\mathbb{Z}_{p_i}$
3. look-up tables

$$T_i(j, k) = f_i^{-1}(j) + g_i^{-1}(k) d_i \bmod p_i$$



Signature Algorithm

Client

1. $e \leftarrow H(m)$
2. $k_1 \xleftarrow{\$} [\![1, n-1]\!]$
3. $R_1 \leftarrow [k_1]G$

$\xrightarrow{e,\ R_1}$

Cloud

Signature Algorithm

Client

1. $e \leftarrow H(m)$
2. $k_1 \xleftarrow{\$} [\![1, n-1]\!]$
3. $R_1 \leftarrow [k_1]G$

$\xrightarrow{e,\, R_1}$

Cloud

4. $k_2 \xleftarrow{\$} [\![1, n-1]\!]$
5. $R \leftarrow [k_2]R_1 = [k_1 k_2]G$
6. $r \leftarrow x_R \bmod n$

Signature Algorithm

Client

1. $e \leftarrow H(m)$
2. $k_1 \xleftarrow{\$} [\![1, n-1]\!]$
3. $R_1 \leftarrow [k_1]G$

$\xrightarrow{e,\ R_1}$

Cloud

4. $k_2 \xleftarrow{\$} [\![1, n-1]\!]$
5. $R \leftarrow [k_2]R_1 = [k_1 k_2]G$
6. $r \leftarrow x_R \bmod n$
7. $u = (u_1, \ldots, u_t) \leftarrow k_2^{-1} e \bmod n$
8. $v = (v_1, \ldots, v_t) \leftarrow k_2^{-1} r \bmod n$

Signature Algorithm

Client

1. $e \leftarrow H(m)$
2. $k_1 \xleftarrow{\$} [\![1, n-1]\!]$
3. $R_1 \leftarrow [k_1]G$

$\xrightarrow{e,\ R_1}$

Cloud

4. $k_2 \xleftarrow{\$} [\![1, n-1]\!]$
5. $R \leftarrow [k_2]R_1 = [k_1 k_2]G$
6. $r \leftarrow x_R \bmod n$
7. $u = (u_1, \ldots, u_t) \leftarrow k_2^{-1} e \bmod n$
8. $v = (v_1, \ldots, v_t) \leftarrow k_2^{-1} r \bmod n$
9. **For** i in $[\![1, t]\!]$:
10. $\quad \overline{u_i} \leftarrow f_i(u_i)$
11. $\quad \overline{v_i} \leftarrow g_i(v_i)$

$\xleftarrow[\{\overline{v_i}\}_{1 \le i \le t}]{r,\ \{\overline{u_i}\}_{1 \le i \le t}}$

Signature Algorithm

Client

1. $e \leftarrow H(m)$
2. $k_1 \stackrel{\$}{\leftarrow} [\![1, n-1]\!]$
3. $R_1 \leftarrow [k_1]G$

$$\xrightarrow{e,\ R_1}$$

Cloud

4. $k_2 \stackrel{\$}{\leftarrow} [\![1, n-1]\!]$
5. $R \leftarrow [k_2]R_1 = [k_1 k_2]G$
6. $r \leftarrow x_R \bmod n$
7. $u = (u_1, \ldots, u_t) \leftarrow k_2^{-1} e \bmod n$
8. $v = (v_1, \ldots, v_t) \leftarrow k_2^{-1} r \bmod n$
9. **For** i in $[\![1, t]\!]$:
10. $\quad \overline{u_i} \leftarrow f_i(u_i)$
11. $\quad \overline{v_i} \leftarrow g_i(v_i)$

$$\xleftarrow[\{\overline{v_i}\}_{1 \leq i \leq t}]{r,\ \{\overline{u_i}\}_{1 \leq i \leq t}}$$

Client

12. **For** i in $[\![1, t]\!]$:
13. $\quad w_i \leftarrow T_i(\overline{u_i}, \overline{v_i})$
14. $w \leftarrow CRT(w_i) = u + vd \bmod P$
15. $s \leftarrow k_1^{-1} w \bmod n$
16. Return (r, s)

Breaking the White-Box with Faults

3

The Greatest Common Divisor Attack

Client

1. $e \leftarrow H(m)$
2. $k_1 \xleftarrow{\$} [\![1, n-1]\!]$
3. $R_1 \leftarrow [k_1]G$

$\xrightarrow{e,\ R_1}$

Cloud

4. $k_2 \xleftarrow{\$} [\![1, n-1]\!]$
5. $R \leftarrow [k_2]R_1 = [k_1 k_2]G$
6. $r \leftarrow x_R \bmod n$
7. $u = (u_1, \ldots, u_t) \leftarrow k_2^{-1} e \bmod n$
8. $v = (v_1, \ldots, v_t) \leftarrow k_2^{-1} r \bmod n$
9. **For** i in $[\![1, t]\!]$:
10. $\quad \overline{u_i} \leftarrow f_i(u_i)$
11. $\quad \overline{v_i} \leftarrow g_i(v_i)$

$\xleftarrow[\{\overline{v_i}\}_{1 \leq i \leq t}]{r,\ \{\overline{u_i}\}_{1 \leq i \leq t}}$

Client

12. **For** i in $[\![1, t]\!]$:
13. $\quad w_i \leftarrow T_i(\overline{u_i}, \overline{v_i})$
14. $w \leftarrow CRT(w_i) = u + vd \bmod P$
15. $s \leftarrow k_1^{-1} w \bmod n$
16. Return (r, s)

The Greatest Common Divisor Attack

Client

1. $e \leftarrow H(m)$
2. $k_1 \xleftarrow{\$} [\![1, n-1]\!]$
3. $R_1 \leftarrow [k_1]G$

$\xrightarrow{\quad e,\ R_1 \quad}$

Cloud

4. $k_2 \xleftarrow{\$} [\![1, n-1]\!]$
5. $R \leftarrow [k_2]R_1 = [k_1 k_2]G$
6. $r \leftarrow x_R \bmod n$
7. $u = (u_1, \ldots, u_t) \leftarrow k_2^{-1} e \bmod n$
8. $v = (v_1, \ldots, v_t) \leftarrow k_2^{-1} r \bmod n$
9. **For** i in $[\![1, t]\!]$:
10. $\quad \overline{u_i} \leftarrow f_i(u_i)$
11. $\quad \overline{v_i} \leftarrow g_i(v_i)$

$\xleftarrow[\{\overline{v_i}\}_{1 \le i \le t}]{r,\ \{\overline{u_i}\}_{1 \le i \le t}}$

Client

12. **For** i in $[\![1, t]\!]$:
13. $\quad w_i \leftarrow T_i(\overline{u_i}, \overline{v_i})$
14. $w \leftarrow CRT(w_i) = u + vd \bmod P$
15. $s \leftarrow k_1^{-1} w \bmod n$
16. Return (r, s)

The Greatest Common Divisor Attack

Client

1. $e \leftarrow H(m)$
2. $k_1 \xleftarrow{\$} [\![1, n-1]\!]$
3. $R_1 \leftarrow [k_1]G$

$$\xrightarrow{e,\, R_1}$$

Cloud

4. $k_2 \xleftarrow{\$} [\![1, n-1]\!]$
5. $R \leftarrow [k_2]R_1 = [k_1 k_2]G$
6. $r \leftarrow x_R \bmod n$
7. $u = (u_1, \ldots, u_t) \leftarrow k_2^{-1} e \bmod n$
8. $v = (v_1, \ldots, v_t) \leftarrow k_2^{-1} r \bmod n$
9. **For** i in $[\![1, t]\!]$:
10. $\quad \overline{u_i} \leftarrow f_i(u_i)$
11. $\quad \overline{v_i} \leftarrow g_i(v_i)$

$$\xleftarrow[\{\overline{v_i}\}_{1 \le i \le t}]{r,\, \{\overline{u_i}\}_{1 \le i \le t}}$$

12. **For** i in $[\![1, t]\!]$:
13. $\quad w_i \leftarrow T_i(\overline{u_i}, \overline{v_i})$
14. $w \leftarrow CRT(w_i) = u + vd \bmod P$
15. $s \leftarrow k_1^{-1} w \bmod n$
16. Return (r, s)

The Greatest Common Divisor Attack

❶ 1^{st} execution: store the values $w^{(1)} = u^{(1)} + v^{(1)} d \bmod P$ and $\overline{u_i^{(1)}}$

The Greatest Common Divisor Attack

1. 1^{st} execution: store the values $w^{(1)} = u^{(1)} + v^{(1)}d \bmod P$ and $\overline{u_i^{(1)}}$
2. 2^{nd} execution: change $\overline{u_i^{(2)}}$ into $\overline{u_i^{(1)}}$ in order to obtain $w^{(2)} = u^{(1)} + v^{(2)}d \bmod P$

The Greatest Common Divisor Attack

1. 1^{st} execution: store the values $w^{(1)} = u^{(1)} + v^{(1)} d \bmod P$ and $\overline{u_i^{(1)}}$
2. 2^{nd} execution: change $\overline{u_i^{(2)}}$ into $\overline{u_i^{(1)}}$ in order to obtain $w^{(2)} = u^{(1)} + v^{(2)} d \bmod P$
3. 3^{rd} execution: change $\overline{u_i^{(3)}}$ into $\overline{u_i^{(1)}}$ in order to obtain $w^{(3)} = u^{(1)} + v^{(3)} d \bmod P$

The Greatest Common Divisor Attack

1. 1^{st} execution: store the values $w^{(1)} = u^{(1)} + v^{(1)}d \bmod P$ and $\overline{u_i^{(1)}}$
2. 2^{nd} execution: change $\overline{u_i^{(2)}}$ into $\overline{u_i^{(1)}}$ in order to obtain $w^{(2)} = u^{(1)} + v^{(2)}d \bmod P$
3. 3^{rd} execution: change $\overline{u_i^{(3)}}$ into $\overline{u_i^{(1)}}$ in order to obtain $w^{(3)} = u^{(1)} + v^{(3)}d \bmod P$
4. Compute $\begin{cases} a = w^{(1)} - w^{(2)} = (v^{(1)} - v^{(2)})d \bmod P \\ b = w^{(1)} - w^{(3)} = (v^{(1)} - v^{(3)})d \bmod P \end{cases}$

The Greatest Common Divisor Attack

1. 1^{st} execution: store the values $w^{(1)} = u^{(1)} + v^{(1)} d \bmod P$ and $\overline{u_i^{(1)}}$
2. 2^{nd} execution: change $\overline{u_i^{(2)}}$ into $\overline{u_i^{(1)}}$ in order to obtain $w^{(2)} = u^{(1)} + v^{(2)} d \bmod P$
3. 3^{rd} execution: change $\overline{u_i^{(3)}}$ into $\overline{u_i^{(1)}}$ in order to obtain $w^{(3)} = u^{(1)} + v^{(3)} d \bmod P$
4. Compute $\begin{cases} a = w^{(1)} - w^{(2)} = (v^{(1)} - v^{(2)}) d \bmod P \\ b = w^{(1)} - w^{(3)} = (v^{(1)} - v^{(3)}) d \bmod P \end{cases}$
5. Compute $gcd(a, b) = \alpha \times d$ with $\alpha = gcd(v^{(1)} - v^{(2)}, v^{(1)} - v^{(3)})$

The Greatest Common Divisor Attack

❶ 1^{st} execution: store the values $w^{(1)} = u^{(1)} + v^{(1)} d \bmod P$ and $\overline{u_i^{(1)}}$

❷ 2^{nd} execution: change $\overline{u_i^{(2)}}$ into $\overline{u_i^{(1)}}$ in order to obtain $w^{(2)} = u^{(1)} + v^{(2)} d \bmod P$

❸ 3^{rd} execution: change $\overline{u_i^{(3)}}$ into $\overline{u_i^{(1)}}$ in order to obtain $w^{(3)} = u^{(1)} + v^{(3)} d \bmod P$

❹ Compute $\begin{cases} a = w^{(1)} - w^{(2)} = (v^{(1)} - v^{(2)}) d \bmod P \\ b = w^{(1)} - w^{(3)} = (v^{(1)} - v^{(3)}) d \bmod P \end{cases}$

❺ Compute $gcd(a, b) = \alpha \times d$ with $\alpha = gcd(v^{(1)} - v^{(2)}, v^{(1)} - v^{(3)})$

❻ Brute force the value of α and recover d

The Greatest Common Divisor Attack

Remark 1

We can compute the gcd of a and b because there is no reduction modulo P in the computations due to the fact that $P > n^2 + n$.

The Greatest Common Divisor Attack

Remark 1

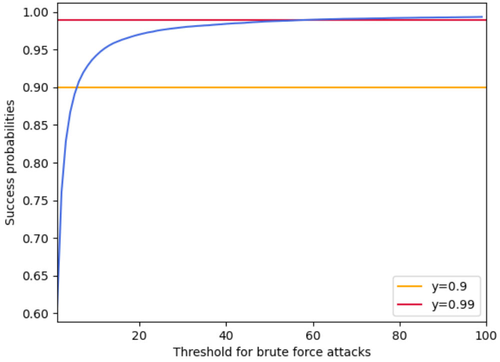
We can compute the gcd of a and b because there is no reduction modulo P in the computations due to the fact that $P > n^2 + n$.

Remark 2

Step 6 is possible because most of the time, α is very small. Theoretically, in 99% of cases, we have $\alpha \leq 62$.

Experiments

- We attacked 50 000 different white-box instances
- Limit for the brute force set at 58 $\Rightarrow$ success rate of 0.99 (coherent)



An Efficient Countermeasure

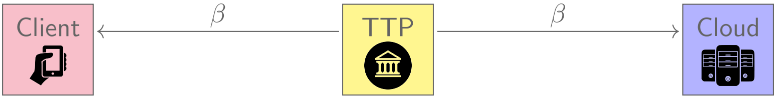
4

An Efficient Countermeasure

Key Idea

Both our attacks require to fault $\overline{u_i}$ without impacting $\overline{v_i}$.
A countermeasure is thus to bind these values together.

Initialization :



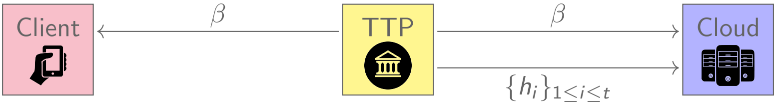
An Efficient Countermeasure

Key Idea

Both our attacks require to fault $\overline{u_i}$ without impacting $\overline{v_i}$.
A countermeasure is thus to bind these values together.

Initialization :

- A unique set of bigger permutations $\{h_i\}$ replaces $\{f_i\}$ and $\{g_i\}$



An Efficient Countermeasure

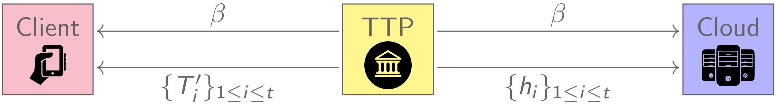
Key Idea

Both our attacks require to fault $\overline{u_i}$ without impacting $\overline{v_i}$.
A countermeasure is thus to bind these values together.

Initialization :

- A unique set of bigger permutations $\{h_i\}$ replaces $\{f_i\}$ and $\{g_i\}$
- The tables T_i' are constructed as

$$T_i'(j) = LSB(h_i^{-1}(j)) + MSB(h_i^{-1}(j))d_i \bmod p_i$$



New Signature Algorithm

Client

1. $e \leftarrow H(m)$
2. $k_1 \xleftarrow{\$} [\![1, n-1]\!]$
3. $R_1 \leftarrow [k_1]G$

$\xrightarrow{e,\, R_1}$

Cloud

4. $k_2 \xleftarrow{\$} [\![1, n-1]\!]$
5. $R \leftarrow [k_2]R_1$
6. $r \leftarrow x_R \bmod n$
7. $u = (u_1, \ldots, u_t) \leftarrow k_2^{-1} e \bmod n$
8. $v = (v_1, \ldots, v_t) \leftarrow k_2^{-1} r \bmod n$
9. **For** i in $[\![1, t]\!]$:
10. $\quad \overline{x_i} \leftarrow h_i(u_i \| v_i)$

$\xleftarrow{r,\, \{\overline{x_i}\}_{1 \le i \le t}}$

11. **For** i in $[\![1, t]\!]$:
12. $\quad w_i \leftarrow T_i'(\overline{x_i})$
13. $w \leftarrow CRT(w_i) = u + vd \bmod P$
14. $s \leftarrow k_1^{-1} w \bmod n$
15. Return (r, s)

Impact on Performances

On the cloud. One has to store and apply bigger permutations

$\rightarrow$ Memory overhead

On the client. The size of the tables remains unchanged

$\rightarrow$ No overhead

Conclusion

5

Conclusion

- We broke the firstly published ECDSA White-Box Scheme
- Two fault attacks based on re-injections were presented
- An efficient countermeasure has been suggested
- Protecting ECDSA in the white-box context is a very difficult task as shown by the WhibOx Contest 2021

Thank you for your attention

IDEMIA

Join us on

www.idemia.com