ECDSA White-Box Implementations

Attacks and Designs from WhibOx 2021 Contest

G. Barbu, W. Beullens, E. Dottax, C. Giraud, A. Houzelot,
C. Li, M. Mahzoun, A. Ranea and J. Xie

September 20, 2022

COSIC

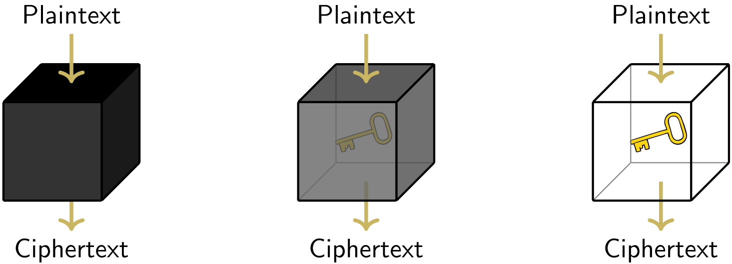
Technische Universiteit Eindhoven University of Technology

IBM

IDEMIA

LaBRI

Black-Box, Grey-Box, White-Box



Cryptanalysis	Side-channels/Faults	Read/modify binary

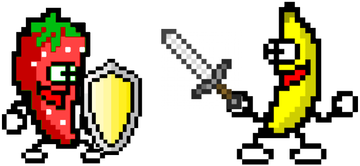
CHES 2021 Challenge - the WhibOx Contest

Designers

- Post C codes computing ECDSA
- Challenges gain strawberries (depending on performances and time until break)

Attackers

- Try to extract the secret key
- Receive bananas (number of strawberries of the challenge)



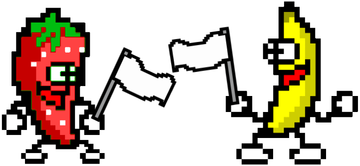
Our Contributions

zerokey

- Posted the 2 winning challenges
- Described the implementations

TheRealIdefix

- Broke the most challenges
- Described attacks, showing which ones succeeded for each candidate



ECDSA

- Let G be a point of order n on an elliptic curve E
- Let d be a 256-bit key
- Let m be a message and $e = H(m)$ its hash value

Algorithm 1: ECDSA signature

1 $k \xleftarrow{\$} [\![1, n-1]\!]$
2 $R \leftarrow kG$
3 $r \leftarrow x_R \bmod n$
4 $s \leftarrow k^{-1}(e + rd) \bmod n$
5 **if** $r == 0$ *or* $s == 0$ **then**
6   Go to step 1
7 **end**
8 Return (r,s)

ECDSA Sensitive Values

- Let G be a point of order n on an elliptic curve E
- Let d be a 256-bit key
- Let m be a message and $e = H(m)$ its hash value

Algorithm 1: ECDSA signature

1 $k \xleftarrow{\$} [\![1, n-1]\!]$
2 $R \leftarrow kG$
3 $r \leftarrow x_R \bmod n$
4 $s \leftarrow k^{-1}(e + rd) \bmod n$
5 **if** $r == 0$ *or* $s == 0$ **then**
6   Go to step 1
7 **end**
8 Return (r,s)

ECDSA Sensitive Values

- Let G be a point of order n on an elliptic curve E
- Let d be a 256-bit key
- Let m be a message and $e = H(m)$ its hash value

Algorithm 1: ECDSA signature

1 $k \xleftarrow{\$} [\![1, n-1]\!]$
2 $R \leftarrow kG$
3 $r \leftarrow x_R \bmod n$
4 $s \leftarrow k^{-1}(e + rd) \bmod n$
5 **if** $r == 0$ *or* $s == 0$ **then**
6   Go to step 1
7 **end**
8 Return (r,s)

Deterministic ECDSA

- Let G be a point of order n on an elliptic curve E
- Let d be a 256-bit key
- Let m be a message and $e = H(m)$ its hash value

Algorithm 1: ECDSA signature

1 $k \xleftarrow{\$} [\![1, n-1]\!]$ WB model $\Rightarrow$ No reliable source of randomness!
2 $R \leftarrow kG$
3 $r \leftarrow x_R \bmod n$
4 $s \leftarrow k^{-1}(e + rd) \bmod n$
5 **if** $r == 0$ *or* $s == 0$ **then**
6 Go to step 1
7 **end**
8 Return (r,s)

Deterministic ECDSA

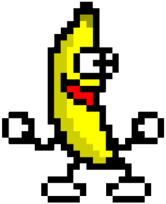
- Let G be a point of order n on an elliptic curve E
- Let d be a 256-bit key
- Let m be a message and $e = H(m)$ its hash value

Algorithm 1: ECDSA signature

```
1 k ← f(e)            WB model ⇒ No reliable source of randomness!
2 R ← kG
3 r ← x_R mod n
4 s ← k^{-1}(e + rd) mod n
5 if r == 0 or s == 0 then
6 |   Go to step 1
7 end
8 Return (r,s)
```

Section 1

Breaking the Candidates



Hooking Shared Libraries

Idea

Find some secret values that could be manipulated in the clear

- Use of the GMP library suggested by the contest rules
- Hook the calls to GMP functions (LD_PRELOAD)
- Update a log of the given parameters
- Search for d, k or related values in the log

Biased Nonce

First possibility

Find collisions: signatures generated with the same nonce

- Find (r_1, s_1) and (r_2, s_2) such that $r_1 = r_2$ (so $k_1 = k_2$)
- Solve the following system in k_1, d:

$$\begin{aligned} s_1 &= k_1^{-1}(e_1 + r_1 d) \\ s_2 &= k_1^{-1}(e_2 + r_1 d) \end{aligned}$$

Biased Nonce

First possibility

Find collisions: signatures generated with the same nonce

- Find (r_1, s_1) and (r_2, s_2) such that $r_1 = r_2$ (so $k_1 = k_2$)
- Solve the following system in k_1, d:

$$\begin{aligned} s_1 &= k_1^{-1}(e_1 + r_1 d) \\ s_2 &= k_1^{-1}(e_2 + r_1 d) \end{aligned}$$

Second possibility

Exploit biases in the nonce generation

- Use lattice-based attacks
- Allows to recover d from a few bits of k for several signatures.

Grey-box Attacks in the White-Box Model

- Side-channel attacks
  - ≻ Difficult to apply (huge size of the traces)
- Fault injections
  - ≻ Modify the binary, use debugging tools
  - ≻ Many fault attacks on deterministic ECDSA, for example:

Grey-box Attacks in the White-Box Model

- Side-channel attacks
  - ≻ Difficult to apply (huge size of the traces)
- Fault injections
  - ≻ Modify the binary, use debugging tools
  - ≻ Many fault attacks on deterministic ECDSA, for example:

Valid signature

$$r = x_R \bmod n$$
$$s = k^{-1}(e + rd) \bmod n$$

Grey-box Attacks in the White-Box Model

- Side-channel attacks
  - ≻ Difficult to apply (huge size of the traces)
- Fault injections
  - ≻ Modify the binary, use debugging tools
  - ≻ Many fault attacks on deterministic ECDSA, for example:

Valid signature	Faulty signature
$r = x_R \bmod n$	$r' = x_{R^{\lightning}} \bmod n$
$s = k^{-1}(e + rd) \bmod n$	$s' = k^{-1}(e + r'd) \bmod n$

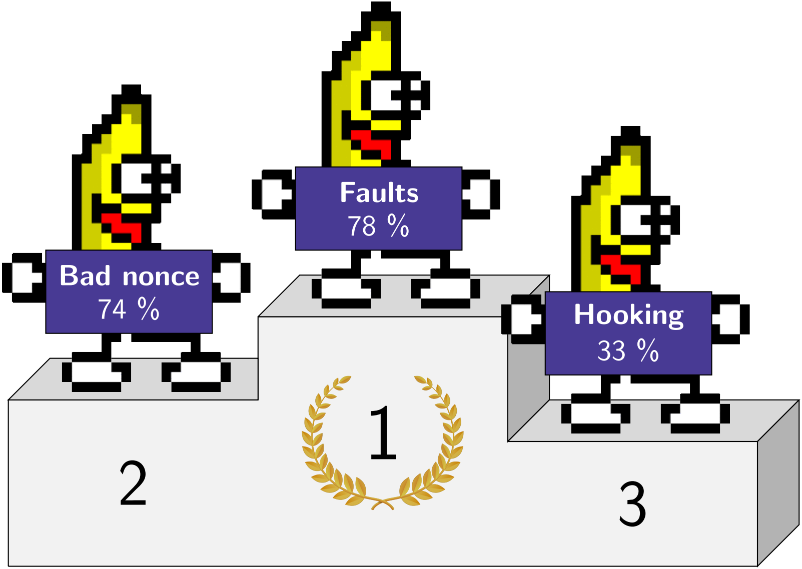
Grey-box Attacks in the White-Box Model

- Side-channel attacks
  - Difficult to apply (huge size of the traces)
- Fault injections
  - Modify the binary, use debugging tools
  - Many fault attacks on deterministic ECDSA, for example:

Valid signature	Faulty signature
$r = x_R \bmod n$	$r' = x_{R^{\lightning}} \bmod n$
$s = k^{-1}(e + rd) \bmod n$	$s' = k^{-1}(e + r'd) \bmod n$

$$d = (s(r - r')(s - s')^{-1} - r)^{-1} e \bmod n$$

Percentage of Challenges Broken by Each Attack



Section 2

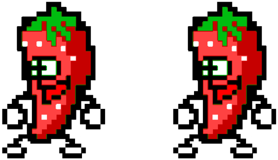
Design of the Winning Challenges

How to Win the Strawberries

- The implicit framework for white-box implementation
  - A novel encoding method

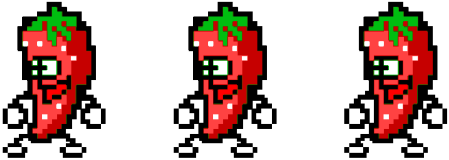
How to Win the Strawberries

- The implicit framework for white-box implementation
  - A novel encoding method
- Techniques from multivariate public key crypto
  - Additional countermeasures



How to Win the Strawberries

- The implicit framework for white-box implementation
  - A novel encoding method
- Techniques from multivariate public key crypto
  - Additional countermeasures
- C obfuscator
  - Use Tigress to obfuscate the source codes



Implicit Framework

Implicit Function and Implicit Evaluation

$$F(\boldsymbol{x}) = \boldsymbol{y} \iff T(\boldsymbol{x}, \boldsymbol{y}) = 0$$

Evaluate $F(\boldsymbol{a})$ by substituting $\boldsymbol{x} = \boldsymbol{a}$ and solving $T(\boldsymbol{a}, \boldsymbol{y}) = 0$

Implicit Framework

Implicit Function and Implicit Evaluation

$$F(\boldsymbol{x}) = \boldsymbol{y} \iff T(\boldsymbol{x}, \boldsymbol{y}) = 0$$

Evaluate $F(\boldsymbol{a})$ by substituting $\boldsymbol{x} = \boldsymbol{a}$ and solving $T(\boldsymbol{a}, \boldsymbol{y}) = 0$

Quasilinear Implicit Function (QIF)

$$\forall \boldsymbol{x},\ \text{function } \boldsymbol{y} \mapsto T(\boldsymbol{x}, \boldsymbol{y}) \text{ is affine}$$

This enables fast solving of $\boldsymbol{y}$

Implicit Implementation

$$F = F^{(t)} \circ F^{(t-1)} \circ \cdots \circ F^{(1)}$$

- Encoded implementation

$$\overline{F} = \overline{F^{(t)}} \circ \cdots \circ \overline{F^{(1)}} = (B^{(t)} \circ F^{(t)} \circ A^{(t)}) \circ \cdots \circ (B^{(1)} \circ F^{(1)} \circ A^{(1)})$$

- T is a QIF of $F^{(i)} \implies T' = M \circ T \circ (A, B^{-1})$ is a QIF of $F^{(i)'} = B \circ F^{(i)} \circ A$, for any linear permutation M

Implicit Implementation

$$F = F^{(t)} \circ F^{(t-1)} \circ \cdots \circ F^{(1)}$$

- Encoded implementation

$$\overline{F} = \overline{F^{(t)}} \circ \cdots \circ \overline{F^{(1)}} = (B^{(t)} \circ F^{(t)} \circ A^{(t)}) \circ \cdots \circ (B^{(1)} \circ F^{(1)} \circ A^{(1)})$$

- T is a QIF of $F^{(i)} \implies T' = M \circ T \circ (A, B^{-1})$ is a QIF of $F^{(i)'} = B \circ F^{(i)} \circ A$, for any linear permutation M

Implicit implementation = Encoded implementation + QIF

White-box implementation of ECDSA

Algorithm 2: White-box ECDSA for winning challenges

$e \leftarrow e \bmod p$
$(v_1, v_2, v_3) \leftarrow \overline{E^{(1)}}(e)$ `// implicit evaluation of` kG
for $\mathbf{o}$ *in* $\mathcal{L}$ **do**
  $(u_1, u_2, u_3) \leftarrow (v_1, v_2, v_3) + p \cdot \mathbf{o}$
  $(r, s) \leftarrow \overline{E^{(2)}}(u_1, u_2, u_3)$ `// implicit evaluation of` (r, s)
  if $\mathtt{VerifySignature}(r, s, e) = \text{valid}$ **then**
    return (r, s)
  end
end

- A round-based implementation of scalar multiplication
- Precompute $\mathcal{L}$ to deal with overflows when converting $\bmod p$ to $\bmod n$

Additional Countermeasures

Tricks from the multivariate public-key crypto

Obfuscation by multiplying random polynomials

- Implicit evaluation can be preserved w.h.p.

$$R(\boldsymbol{a})T(\boldsymbol{a},\boldsymbol{y}) = 0 \implies T(\boldsymbol{a},\boldsymbol{y}) = 0$$

- Set an initial value to prevent failures

Masking of the nonce

Avoid bias of the most significant part of the nonce k

keen_ptolemy (Challenge 227)

Implicit implementation + Multiplying random polynomials

	$\overline{T^{(1)}}$	$\{\overline{T^{(2)}}, \ldots, \overline{T^{(t-1)}}\}$	$\overline{T^{(t)}}$	$\overline{T^{(t+1)}}$
input variables	2+4	5+4	5+3	3+2
number of components	4	4	3	2
degree	3	3	4	2
number of coefficients	27×4	130×4	255×3	18×2

- Compiled binary: 4.42 MB
- Average running time: 0.04s
- Average RAM consumed: 6.14 MB

- Code obfuscation did not impact the running time but increased the binary size by 8% and the average RAM by 3%

clever_kare (Challenge 226)

Implicit implementation + Multiplying random polynomials + Masking

	$\overline{T^{(1)}}$	$\{\overline{T^{(2)}}, \ldots, \overline{T^{(t-1)}}\}$	$\overline{T^{(t)}}$	$\overline{T^{(t+1)}}$
input variables	2+6	7+6	7+5	5+2
number of components	6	6	5	2
degree	3	3	4	5
number of coefficients	37×6	322×6	854×5	504×2

- Compiled binary: 15.44 MB
- Average running time: 0.15s
- Average RAM consumed: 17.27 MB
- Impact of code obfuscation $< 1\%$

Security Analysis

Automated attacks do not work

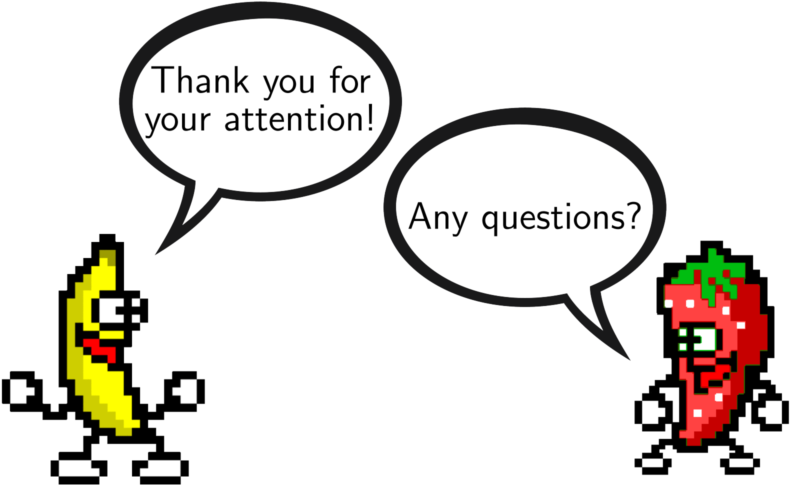
- ✗ Hooking shared libraries
- ✗ Biased nonces

✓ Fault attacks [BDG+22]

- remove only one line of code for each challenge
- induce uncontrolled fault in r
- defeat the verification steps

Summary

- We present automated attacks breaking a large number of challenges
- Fault attacks are the most efficient and effective
- We apply implicit implementation framework to ECDSA
- The best implementations were broken by fault attacks
- Securing white-box ECDSA is still an open problem


Thank you for your attention!
Any questions?