pyecsca

[pietska]

Reverse-engineering black-box elliptic curve cryptography via side-channel analysis

Jan Jancar, Vojtech Suchanek, Petr Svenda,
Vladimir Sedlacek, **Łukasz Chmielewski**

Who we are

Jan Jancar
- PhD candidate

Łukasz Chmielewski
- Assistant professor

Outline

- Status check
- Why? reverse-engineer ECC
  - Elliptic Curve Cryptography
  - Why Is Hardware Security of ECC Implementations Important?
  - Side-Channel Attacks: Assumptions & Reality
- What? are we reverse-engineering
  - 👉 ECC implementations
- How? to reverse-engineer ECC
  - 👉 RPA-RE

Status check

https://github.com/J08nY/pyecsca-tutorial-croatia2024

- Setup done?
  - Installed packages and running Jupyter Lab, or
  - Using launch binder
- Quickstart done?
  - `notebooks/start.ipynb`
- Knowledge of Python?
- Knowledge of ECC?
- Knowledge of SCA?

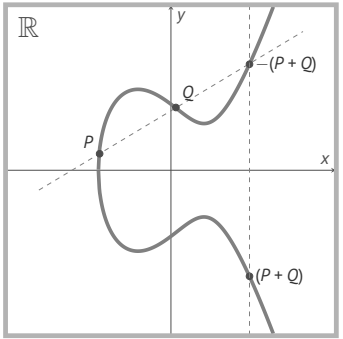
`setup.md`

Why?

Why?

Elliptic Curve Cryptography

- **Elliptic Curve**: $y^2 \equiv x^3 + ax + b$
  - Points $(x, y) \in E(\mathbb{K})$ form an abelian group
  - Scalar multiplication
    $[n] : E(\mathbb{K}) \to E(\mathbb{K})$
    $P \mapsto [n]P = \underbrace{P + P + \ldots + P}_{n \text{ times}}$
  - Discrete logarithm is hard when $\mathbb{K} = \mathbb{F}_p$
    ECDLP: Find x given $[x]G$ and $G \in E(\mathbb{F}_p)$

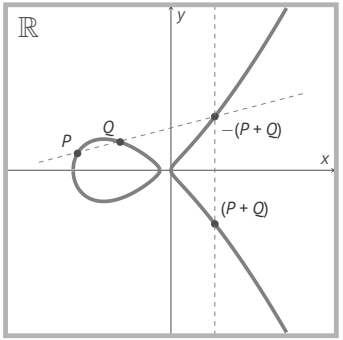


Short Weierstrass

Why?

Elliptic Curve Cryptography

- **Elliptic Curve**: $by^2 \equiv x^3 + ax^2 + x$
  - Points $(x, y) \in E(\mathbb{K})$ form an abelian group
  - Scalar multiplication
    $[n] : E(\mathbb{K}) \to E(\mathbb{K})$
    $P \mapsto [n]P = \underbrace{P + P + \ldots + P}_{n \text{ times}}$
  - Discrete logarithm is hard when $\mathbb{K} = \mathbb{F}_p$
    ECDLP: Find x given $[x]G$ and $G \in E(\mathbb{F}_p)$

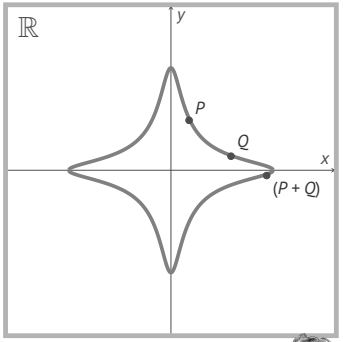


Montgomery

Why?

Elliptic Curve Cryptography

- **Elliptic Curve**: $x^2 + y^2 \equiv c^2(1 + dx^2y^2)$
  - Points $(x, y) \in E(\mathbb{K})$ form an abelian group
  - Scalar multiplication
    $[n] : E(\mathbb{K}) \to E(\mathbb{K})$
    $P \mapsto [n]P = \underbrace{P + P + \ldots + P}_{n \text{ times}}$
  - Discrete logarithm is hard when $\mathbb{K} = \mathbb{F}_p$
    ECDLP: Find x given $[x]G$ and $G \in E(\mathbb{F}_p)$

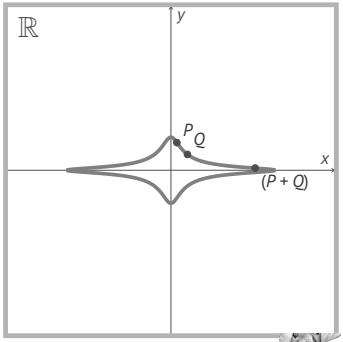


Edwards

Why?

Elliptic Curve Cryptography

- **Elliptic Curve**: $ax^2 + y^2 \equiv 1 + dx^2y^2$
  - Points $(x, y) \in E(\mathbb{K})$ form an abelian group
  - Scalar multiplication
    $[n] : E(\mathbb{K}) \to E(\mathbb{K})$
    $P \mapsto [n]P = \underbrace{P + P + \ldots + P}_{n \text{ times}}$
  - Discrete logarithm is hard when $\mathbb{K} = \mathbb{F}_p$
    ECDLP: Find x given $[x]G$ and $G \in E(\mathbb{F}_p)$



Twisted Edwards

Why?

Elliptic Curve Cryptography

- **Elliptic Curve**:
  - Points $(x, y) \in E(\mathbb{K})$ form an abelian group
  - Scalar multiplication
    $[n] : E(\mathbb{K}) \to E(\mathbb{K})$
    $P \mapsto [n]P = \underbrace{P + P + \ldots + P}_{n \text{ times}}$
  - Discrete logarithm is hard when $\mathbb{K} = \mathbb{F}_p$
    ECDLP: Find x given $[x]G$ and $G \in E(\mathbb{F}_p)$
- **ECDH**: Diffie-Hellman on $E(\mathbb{F}_p)$
  - Scalar multiplication + hash -> Shared secret

Why?

Elliptic Curve Cryptography

- **Elliptic Curve**:
  - Points $(x, y) \in E(\mathbb{K})$ form an abelian group
  - Scalar multiplication
    $[n] : E(\mathbb{K}) \to E(\mathbb{K})$
    $P \mapsto [n]P = \underbrace{P + P + \ldots + P}_{n \text{ times}}$
  - Discrete logarithm is hard when $\mathbb{K} = \mathbb{F}_p$
    ECDLP: Find x given $[x]G$ and $G \in E(\mathbb{F}_p)$
- **ECDH**: Diffie-Hellman on $E(\mathbb{F}_p)$
  - Scalar multiplication + hash -> Shared secret
- **ECDSA**: Digital Signature Algorithm on $E(\mathbb{F}_p)$
  - Random sample + scalar multiplication + hash -> Signature

Why?

Elliptic Curve Cryptography

- **Elliptic Curve**:
  - Points $(x, y) \in E(\mathbb{K})$ form an abelian group
  - Scalar multiplication
    $[n] : E(\mathbb{K}) \to E(\mathbb{K})$
    $P \mapsto [n]P = \underbrace{P + P + \ldots + P}_{n \text{ times}}$
  - Discrete logarithm is hard when $\mathbb{K} = \mathbb{F}_p$
    ECDLP: Find x given $[x]G$ and $G \in E(\mathbb{F}_p)$
- **ECDH**: Diffie-Hellman on $E(\mathbb{F}_p)$
  - Scalar multiplication + hash -> Shared secret
- **ECDSA**: Digital Signature Algorithm on $E(\mathbb{F}_p)$
  - Random sample + scalar multiplication + hash -> Signature
- **XDH**, **EdDSA**, **...**

Why?

Elliptic Curve Cryptography

- Many implementation possibilities

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
    - $y^2 \equiv x^3 + ax + b$

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
    - $y^2 \equiv x^3 + ax + b$
    - $x^2 + y^2 \equiv c^2(1 + dx^2y^2)$

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
    - $y^2 \equiv x^3 + ax + b$
    - $x^2 + y^2 \equiv c^2(1 + dx^2y^2)$
    - $ax^2 + y^2 \equiv 1 + dx^2y^2$

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
    - $y^2 \equiv x^3 + ax + b$
    - $x^2 + y^2 \equiv c^2(1 + dx^2y^2)$
    - $ax^2 + y^2 \equiv 1 + dx^2y^2$
    - $by^2 \equiv x^3 + ax^2 + x$

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates
    - (X, Y)

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates
    - (X, Y)
    - (X, Y, Z)

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates
    - (X, Y)
    - (X, Y, Z)
    - (X, Y, Z, ZZ) …

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates
  - Addition formulas

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates
  - Addition formulas

```
Y1Z2 = Y1*Z2
X1Z2 = X1*Z2
Z1Z2 = Z1*Z2
u = Y2*Z1-Y1Z2
uu = u2
v = X2*Z1-X1Z2
vv = v2
vvv = v*vv
R = vv*X1Z2
A = uu*Z1Z2-vvv-2*R
X3 = v*A
Y3 = u*(R-A)-vvv*Y1Z2
Z3 = vvv*Z1Z2
```

Why?

Elliptic Curve Cryptography

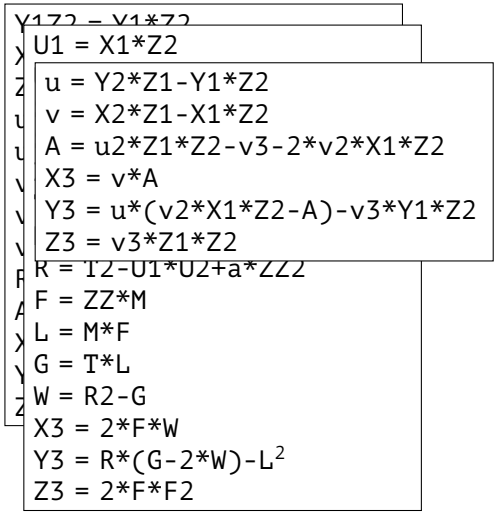
- Many implementation possibilities
  - Curve model
  - Coordinates
  - Addition formulas

```
U1 = X1*Z2
U2 = X2*Z1
S1 = Y1*Z2
S2 = Y2*Z1
ZZ = Z1*Z2
T = U1+U2
M = S1+S2
R = T2-U1*U2+a*ZZ2
F = ZZ*M
L = M*F
G = T*L
W = R2-G
X3 = 2*F*W
Y3 = R*(G-2*W)-L^2
Z3 = 2*F*F2
```

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates
  - Addition formulas



Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates
  - Addition formulas
  - Scalar multiplication

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates
  - Addition formulas
  - Scalar multiplication

Algorithm Left-to-right double-and-add

1: **function** LTR($\boldsymbol{G}, k = (k_l, \ldots, k_0)_2$)
2: $\quad \boldsymbol{R} \leftarrow \infty$
3: $\quad$ **for** $i \leftarrow l$ **downto** 0 **do**
4: $\quad\quad \boldsymbol{R} \leftarrow 2\boldsymbol{R}$
5: $\quad\quad$ **if** $k_i = 1$ **then**
6: $\quad\quad\quad \boldsymbol{R} \leftarrow \boldsymbol{R} + \boldsymbol{G}$
$\quad$ **return** $\boldsymbol{R}$

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates
  - Addition formulas
  - Scalar multiplication

Algorithm Right-to-left double-and-add

1: **function** RTL($\boldsymbol{G}, k = (k_l, \ldots, k_0)_2$)
2: $\quad \boldsymbol{R}_0 \leftarrow \boldsymbol{G}$
3: $\quad \boldsymbol{Q} \leftarrow \infty$
4: $\quad$ **for** $i \leftarrow 0$ **to** l **do**
5: $\quad\quad$ **if** $k_i = 1$ **then**
6: $\quad\quad\quad \boldsymbol{Q} \leftarrow \boldsymbol{Q} + \boldsymbol{R}_0$
7: $\quad\quad \boldsymbol{R}_0 \leftarrow 2\boldsymbol{R}_0$
$\quad$ **return** $\boldsymbol{Q}$

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates
  - Addition formulas
  - Scalar multiplication

Algorithm Fixed-window scalar multiplier

1: **function** Window($\boldsymbol{G}, k = (k_l, \ldots, k_0)_2$)
2: $\quad$ *PrecomuptedTable* = $[0 * \boldsymbol{G}, 1 * \boldsymbol{G}, \ldots, 2^w - 1 * \boldsymbol{G}]$
3: $\quad$ $\hat{k}$ = recode k to w-bit windows
4: $\quad$ $\boldsymbol{T} \leftarrow \infty$
5: $\quad$ **for** $i \leftarrow 0$ **to** $|\hat{k}|$ **do**
6: $\quad\quad$ $\boldsymbol{T} \leftarrow 2^w \boldsymbol{T}$
7: $\quad\quad$ $\boldsymbol{T} \leftarrow \boldsymbol{T}$ + *PrecomputedTable*$[\hat{k}_i]$
$\quad$ **return** $\boldsymbol{T}$

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates
  - Addition formulas
  - Scalar multiplication
  - Finite field operations

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates
  - Addition formulas
  - Scalar multiplication
  - Finite field operations
    - Multiplication: *Toom-Cook, Karatsuba, …*

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates
  - Addition formulas
  - Scalar multiplication
  - Finite field operations
    - Multiplication: *Toom-Cook, Karatsuba, …*
    - Squaring: *Toom-Cook, Karatsuba, …*

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates
  - Addition formulas
  - Scalar multiplication
  - Finite field operations
    - Multiplication: *Toom-Cook, Karatsuba, …*
    - Squaring: *Toom-Cook, Karatsuba, …*
    - Reduction: *Barret, Montgomery, …*

Why?

Elliptic Curve Cryptography

- Many implementation possibilities
  - Curve model
  - Coordinates
  - Addition formulas
  - Scalar multiplication
  - Finite field operations
    - Multiplication: *Toom-Cook, Karatsuba, …*
    - Squaring: *Toom-Cook, Karatsuba, …*
    - Reduction: *Barret, Montgomery, …*
    - Inversion: *GCD, Euler*

Why?

Real-world open-source ECC libraries

- Analyzed 18 open-source ECC libraries (📚)
- *BearSSL, BoringSSL, Botan, BouncyCastle, fastecdsa, Go crypto, Intel IPP cryptography, libgcrypt, LibreSSL, libsecp256k1, libtomcrypt, mbedTLS, micro-ecc, Nettle, NSS, OpenSSL, SunEC,* and *SymCrypt*
- Source-code analysis of ECDH, ECDSA, X25519, and Ed25519
- Curve model, Scalar multiplier, Coordinate system, Addition formulas

Why?

Real-world open-source ECC libraries

- **Specific implementations**
  - Curve or architecture-based (10 📄)
  - e.g. $a = -3$ or special prime arithmetic
- **Curve models**
  - Usually outside = inside
  - Montgomery outside, Twisted-Edwards inside (4 📄)
- **Scalar multipliers**
  - *fixed-base* + *variable-base* + *multi-scalar*
  - Comb, fixed-window, wNAF, GLV, …
  - 4 to 7 bit widths
- **Coordinate systems**
  - Usually Jacobian, also homogenous or xz
- **Addition formulas**
  - 112 formula implementations
  - 50 *"standard"*
  - 23 out-of-scope
  - 39 *"non-standard"*

What are the applications of ECC?

- Signatures: ECDSA, EdDSA
- Key Exchange: ECDH, XDH
- Zero-knowledge proofs, …

What are the applications of ECC?

- Signatures: ECDSA, EdDSA
- Key Exchange: ECDH, XDH
- Zero-knowledge proofs, …

- TLS, payments, cryptocurrencies ₿
- Threat: Side-Channel Analysis (SCA)

Passive SCA and Active SCA (Fault Injection)

Passive: analyze device behavior

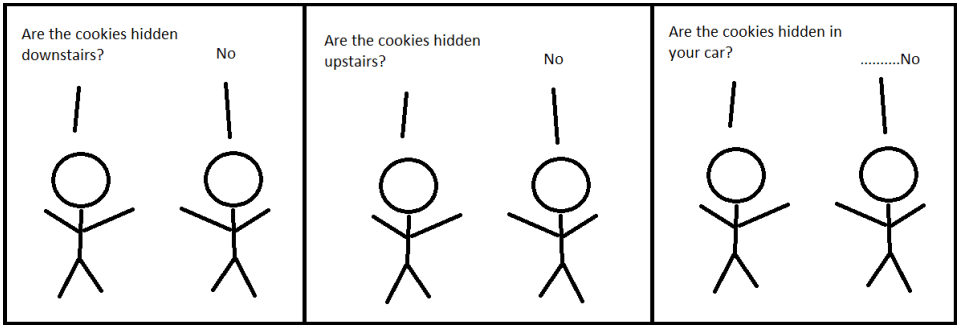
Active: change device behavior

Snow Example: what do you think it is?

Snow Example: what do you think it is? answer.

https://www.independent.co.uk/news/world/europe/melting-snow-being-used-by-police-to-find-cannabis-farms-in-the-netherlands-10036057.html

Cookies Example



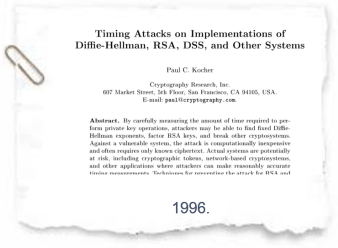
https://www.simplethread.com/great-scott-timing-attack-demo/

Example #1: SPA

```
ScalarMult(P){
 A = ∞
 for ( i = n-1;  i0; i)
        A = DOUBLE(A)
        if (k_i = =1)
               A = ADD(A,P)
        end if
 end for
 Return A = [k]P
}
```

D

A



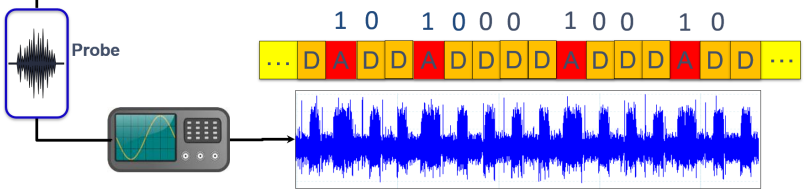
Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems

Paul C. Kocher

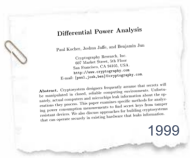
Cryptography Research, Inc.
607 Market Street, 5th Floor, San Francisco, CA 94105, USA.
E-mail: paul@cryptography.com.

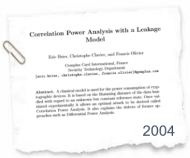
Abstract. By carefully measuring the amount of time required to perform private key operations, attackers may be able to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems. Against a vulnerable system, the attack is computationally inexpensive and often requires only known ciphertext. Actual systems are potentially at risk, including cryptographic tokens, network-based cryptosystems, and other applications where attackers can make reasonably accurate timing measurements. Techniques for preventing the attack for RSA and

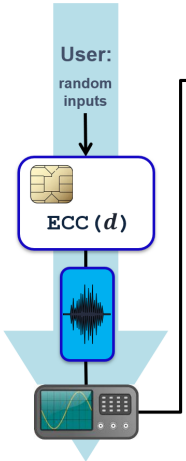
1996.

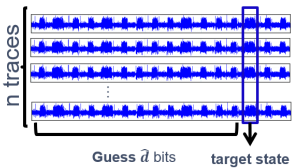


Example #2: DPA/CPA









Correlation Power Analysis on ECC

f_i = **Selection Function**(random inputs, $\hat{d}$, target state)

- HW of a state in a register
- HD between current and previous register state
- ID model: value of a register
- Most significant bit:

$$f_i = \begin{cases} 0 & if\ HW \leq 16 \\ 1 & if\ HW > 16 \end{cases} \qquad f_i = HW(reg\_state)$$

DPA = Difference of Means

CPA = Pearson correlation

Real-World Attacks on ECC Implementations

October 3, 2019

Researchers Discover ECDSA Key Recovery Method

October 3, 2019 – Add Comment – by Emma Davis

Real-World Attacks on ECC Implementations

November 13, 2019

October 3, 2019

Researchers Discover ECDSA Key Recovery Method

Real-World Attacks on ECC Implementations

November 13, 2019

May 28, 2020

LadderLeak: Side-channel security flaws exploited to break ECDSA cryptography

October 3, 2019

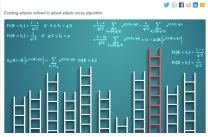
Researchers Discover ECDSA Key Recovery Method

Real-World Attacks on ECC Implementations

November 13, 2019

May 28, 2020

LadderLeak: Side-channel security flaws exploited to break ECDSA cryptography



SCA Titan: January 7, 2021

October 3, 2019

Researchers Discover ECDSA Key Recovery Method

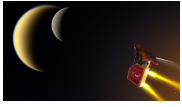
Real-World Attacks on ECC Implementations

November 13, 2019

May 28, 2020

LadderLeak: Side-channel security flaws exploited to break ECDSA cryptography

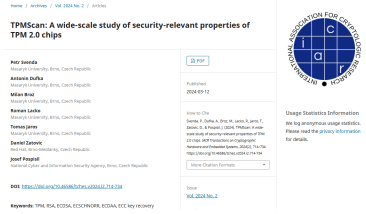
SCA Titan: January 7, 2021



October 3, 2019

Researchers Discover ECDSA Key Recovery Method

March 12, 2024



Why?

Side-Channel Attacks

- Simple Power Analysis
- Differential Power Analysis
- Correlation Power Analysis
- Mutual Information Analysis
- Refined Power Analysis, Zero-value Point Attack, Exceptional Procedure Attack
- Template attacks
- Leakage assessment
- Doubling attack, Collision attacks, Online Template Attack
- …

Why?

Side-Channel Attacks

- Simple Power Analysis
- Differential Power Analysis
- Correlation Power Analysis
- Mutual Information Analysis
- **Refined Power Analysis, Zero-value Point Attack, Exceptional Procedure Attack**
- Template attacks
- Leakage assessment
- Doubling attack, Collision attacks, Online Template Attack
- …

Why?

Assumptions

Why?

Assumptions

$\mathbb{F}_p$ with $p \neq \{2, 3\}$. The algorithm used for the hardware modular multiplication is assumed to be known to the attacker. Moreover, to simplify the attack

Why?

Assumptions

is assumed to be known

Why?

Assumptions

is assumed to be known

Why?

Assumptions

input to s (“fix class”). (This assumes a white-box evaluator that has access to implementation internals.)

is assumed to be known

Why?

Assumptions

assumes a white-box evaluator

is assumed to be known

Why?

Assumptions

assumes a white-box evaluator

e 6.1 abstractly depicts a side-channel measurement of such an ex-
tion. For the sake of simplicity, I assume it is a binary exponen-

is assumed to be known

Why?

Assumptions

assumes a white-box evaluator

assume it is a binary exp

is assumed to be known

Why?

Assumptions

assumes a white-box evaluator

assume it is a binary exp

is assumed to be known

values may be manipulated when working with points P and $2P$. However this idea only works when using the downward routine.

Why?

Assumptions

assumes a white-box evaluator

assume it is a binary exp

is assumed to be known

only works when using the downward routine.

Why?

Assumptions

assumes a white-box evaluator

assume it is a binary exp

is assumed to be known

only works when using the downward routine.

Why?

Assumptions

assumes a white-box evaluator

assume it is a binary exp

a doubling operation from an addition one. This technique, which allows to eventually recover the secret scalar, is applied to three different atomic formulae on elliptic curves,

is assumed to be known

only works when using the downward routine.

Why?

Assumptions

assumes a white-box evaluator

assume it is a binary exp

is applied to three different atomic formulae on elliptic curves,

is assumed to be known

only works when using the downward routine.

Why?

Assumptions

focus on 224-bit scalar multiplication over elliptic curves. Specifically, we
uss an attack on the Montgomery-López-Dahab ladder algorithm [19] pro-
ed by scalar randomization [4].

assume it is a binary exp

assumes a white-box evaluator

is applied to three different atomic formulae on elliptic curves,

is assumed to be known

only works when using the downward routine.

Why?

Assumptions

attack on the Montgomery-López-Dahab ladder algorithm

assumes a white-box evaluator

assume it is a binary exp

is applied to three different atomic formulae on elliptic curves,

is assumed to be known

only works when using the downward routine.

Why?

Assumptions

attack on the Montgomery-López-Dahab ladder algorithm

assumes a white-box evaluator

assume it is a binary exp

is applied to three different atomic formulae on elliptic curves,

is assumed to be known

full knowledge of all algorithms,

only works when using the downward routine.

Why?

Assumptions

attack on the Montgomery-López-Dahab ladder algorithm

assumes a white-box evaluator

assume it is a binary exp

knowledge of the ECSM and the elliptic curve formulae,

is applied to three different atomic formulae on elliptic curves,

is assumed to be known

full knowledge of all algorithms,

only works when using the downward routine.

Why?

Assumptions

attack on the Montgomery-López-Dahab ladder algorithm

assumes a white-box evaluator

assume it is a binary exp

knowledge of the ECSM and the elliptic curve formulae,

is applied to three different atomic formulae on elliptic curves,

is assumed to be known

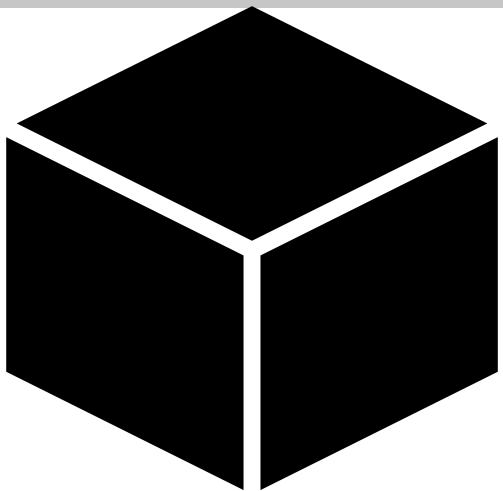
full knowledge of all algorithms,

only works when using the downward routine.

Lots of assumptions you've got there!

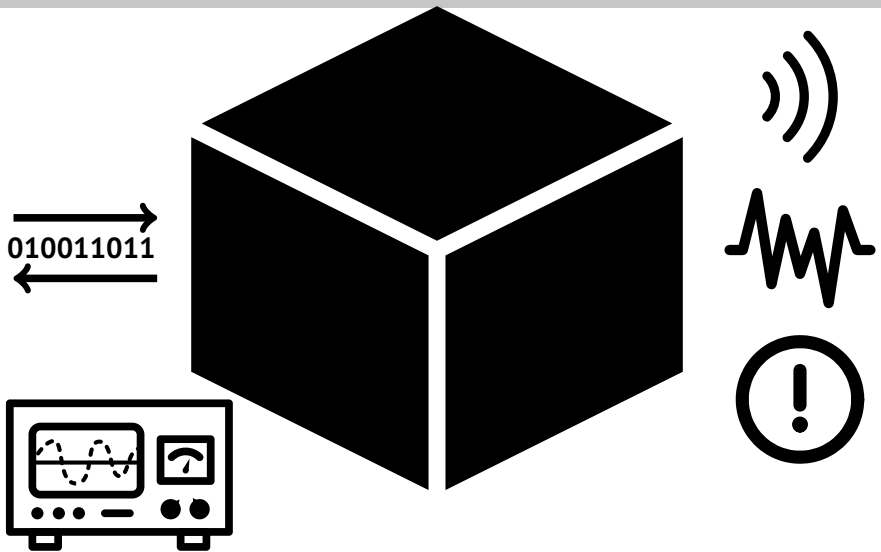
Why?

Reality



Why?

Reality



Why?

- There are a lot of ways in which ECC can be implemented.

Why?

- There are a lot of ways in which ECC can be implemented.
- Most of the known attacks need to know the details of the implementation (including the curve and the scalar multiplication algorithm) to succeed.

Why?

- There are a lot of ways in which ECC can be implemented.
- Most of the known attacks need to know the details of the implementation (including the curve and the scalar multiplication algorithm) to succeed.
- However, most of the real-world embedded implementations are black-box, and the attacker does not know the implementation.

Why?

- There are a lot of ways in which ECC can be implemented.
- Most of the known attacks need to know the details of the implementation (including the curve and the scalar multiplication algorithm) to succeed.
- However, most of the real-world embedded implementations are black-box, and the attacker does not know the implementation.
- How to learn the implementation details?
- Can side channels help?

What?

What?

Implementations

- The object of our study
- Scalar multiplication algorithms
- Real-world open-source ECC
- Countermeasures

Scalar multiplication algorithms

Naive Algorithm #1

Algorithm Left-to-right double-and-add algorithm

1: **Input:** $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $\boldsymbol{Q} = [k]\boldsymbol{P}$
3: $\boldsymbol{R}_0 \leftarrow \boldsymbol{O}$
4: **for** $i \leftarrow n - 1$ down to 0 **do**
5: $\quad \boldsymbol{R}_0 \leftarrow 2\boldsymbol{R}_0$
6: $\quad$ **if** $k_i = 1$ **then**
7: $\quad\quad \boldsymbol{R}_0 \leftarrow \boldsymbol{R}_0 + \boldsymbol{P}$
8: **return** $\boldsymbol{R}_0$

Scalar multiplication algorithms

Naive Algorithm #1

Algorithm Left-to-right double-and-add algorithm

1: **Input:** $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $\boldsymbol{Q} = [k]\boldsymbol{P}$
3: $\boldsymbol{R}_0 \leftarrow \boldsymbol{O}$
4: **for** $i \leftarrow n - 1$ down to 0 **do**
5: $\quad \boldsymbol{R}_0 \leftarrow 2\boldsymbol{R}_0$
6: $\quad$ **if** $k_i = 1$ **then**
7: $\quad\quad \boldsymbol{R}_0 \leftarrow \boldsymbol{R}_0 + \boldsymbol{P}$
8: **return** $\boldsymbol{R}_0$

- What is good about this algorithm?

Scalar multiplication algorithms

Naive Algorithm #1

Algorithm Left-to-right double-and-add algorithm

1: **Input:** $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $\boldsymbol{Q} = [k]\boldsymbol{P}$
3: $\boldsymbol{R}_0 \leftarrow \boldsymbol{O}$
4: **for** $i \leftarrow n - 1$ down to 0 **do**
5: $\quad \boldsymbol{R}_0 \leftarrow 2\boldsymbol{R}_0$
6: $\quad$ **if** $k_i = 1$ **then**
7: $\quad\quad \boldsymbol{R}_0 \leftarrow \boldsymbol{R}_0 + \boldsymbol{P}$
8: **return** $\boldsymbol{R}_0$

- What is good about this algorithm?
- What is wrong?

Scalar multiplication algorithms

Naive Algorithm #2

Algorithm Right-to-left double-and-add algorithm

1: **Input:** $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $\boldsymbol{Q} = [k]\boldsymbol{P}$
3: $\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$
4: $\boldsymbol{Q} \leftarrow \boldsymbol{O}$
5: **for** $i \leftarrow 0$ **to** $n - 1$ **do**
6: **if** $k_i = 1$ **then**
7: $\boldsymbol{Q} \leftarrow \boldsymbol{Q} + \boldsymbol{R}_0$
8: $\boldsymbol{R}_0 \leftarrow 2\boldsymbol{R}_0$
9: **return** $\boldsymbol{Q}$

Scalar multiplication algorithms

Naive Algorithm #2

Algorithm Right-to-left double-and-add algorithm

1: **Input:** $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $\boldsymbol{Q} = [k]\boldsymbol{P}$
3: $\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$
4: $\boldsymbol{Q} \leftarrow \boldsymbol{O}$
5: **for** $i \leftarrow 0$ **to** $n - 1$ **do**
6: **if** $k_i = 1$ **then**
7: $\boldsymbol{Q} \leftarrow \boldsymbol{Q} + \boldsymbol{R}_0$
8: $\boldsymbol{R}_0 \leftarrow 2\boldsymbol{R}_0$
9: **return** $\boldsymbol{Q}$

- What is good about this algorithm?

Scalar multiplication algorithms

Naive Algorithm #2

Algorithm Right-to-left double-and-add algorithm

1: **Input:** $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $\boldsymbol{Q} = [k]\boldsymbol{P}$
3: $\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$
4: $\boldsymbol{Q} \leftarrow \boldsymbol{O}$
5: **for** $i \leftarrow 0$ **to** $n - 1$ **do**
6: **if** $k_i = 1$ **then**
7: $\boldsymbol{Q} \leftarrow \boldsymbol{Q} + \boldsymbol{R}_0$
8: $\boldsymbol{R}_0 \leftarrow 2\boldsymbol{R}_0$
9: **return** $\boldsymbol{Q}$

- What is good about this algorithm?
- What is wrong?

Scalar multiplication algorithms

Let's improve a bit

Algorithm Left-to-right double-and-always-add algorithm

1: **Input:** $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $\boldsymbol{Q} = [k]\boldsymbol{P}$
3: $\boldsymbol{R}_0 \leftarrow \boldsymbol{O}$, $\boldsymbol{R}_1 \leftarrow \boldsymbol{O}$
4: **for** $i \leftarrow n - 1$ **down to** 0 **do**
5: $\quad \boldsymbol{R}_0 \leftarrow 2\boldsymbol{R}_0$
6: $\quad \boldsymbol{R}_{1-k_i} \leftarrow \boldsymbol{R}_{1-k_i} + \boldsymbol{P}$
7: **return** $\boldsymbol{R}_0$

Scalar multiplication algorithms

Let's improve a bit

Algorithm Left-to-right double-and-always-add algorithm

1: **Input:** $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $\boldsymbol{Q} = [k]\boldsymbol{P}$
3: $\boldsymbol{R}_0 \leftarrow \boldsymbol{O}$, $\boldsymbol{R}_1 \leftarrow \boldsymbol{O}$
4: **for** $i \leftarrow n - 1$ **down to** 0 **do**
5: $\quad \boldsymbol{R}_0 \leftarrow 2\boldsymbol{R}_0$
6: $\quad \boldsymbol{R}_{1-k_i} \leftarrow \boldsymbol{R}_{1-k_i} + \boldsymbol{P}$
7: **return** $\boldsymbol{R}_0$

- What is good about this algorithm? Is it more SCA secure?

Scalar multiplication algorithms

Let's improve a bit

Algorithm Left-to-right double-and-always-add algorithm

1: **Input:** $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $\boldsymbol{Q} = [k]\boldsymbol{P}$
3: $\boldsymbol{R}_0 \leftarrow \boldsymbol{O}$, $\boldsymbol{R}_1 \leftarrow \boldsymbol{O}$
4: **for** $i \leftarrow n - 1$ **down to** 0 **do**
5: $\quad \boldsymbol{R}_0 \leftarrow 2\boldsymbol{R}_0$
6: $\quad \boldsymbol{R}_{1-k_i} \leftarrow \boldsymbol{R}_{1-k_i} + \boldsymbol{P}$
7: **return** $\boldsymbol{R}_0$

- What is good about this algorithm? Is it more SCA secure?
- What is wrong? What about fault attacks?

Scalar multiplication algorithms

Let's improve a bit... more

Algorithm Left-to-right double-and-add-always algorithm [Coron99]

1: **Input:** $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $Q = [k]\boldsymbol{P}$
3: $\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$
4: **for** $i \leftarrow n - 2$ **down to** 0 **do**
5: $\quad \boldsymbol{R}_0 \leftarrow 2\boldsymbol{R}_0$
6: $\quad \boldsymbol{R}_1 \leftarrow \boldsymbol{R}_0 + \boldsymbol{P}$
7: $\quad \boldsymbol{R}_0 \leftarrow \boldsymbol{R}_{k_i}$
8: **return** $\boldsymbol{R}_0$

Scalar multiplication algorithms

Let's improve a bit... more

Algorithm Left-to-right double-and-add-always algorithm [Coron99]

1: **Input:** $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $\boldsymbol{Q} = [k]\boldsymbol{P}$
3: $\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$
4: **for** $i \leftarrow n - 2$ **down to** 0 **do**
5: $\quad \boldsymbol{R}_0 \leftarrow 2\boldsymbol{R}_0$
6: $\quad \boldsymbol{R}_1 \leftarrow \boldsymbol{R}_0 + \boldsymbol{P}$
7: $\quad \boldsymbol{R}_0 \leftarrow \boldsymbol{R}_{k_i}$
8: **return** $\boldsymbol{R}_0$

- What is good about this algorithm?

Scalar multiplication algorithms

Let's improve a bit... more

Algorithm Left-to-right double-and-add-always algorithm [Coron99]

1: **Input:** P, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $Q = [k]P$
3: $R_0 \leftarrow P$
4: **for** $i \leftarrow n - 2$ **down to** 0 **do**
5: $\quad R_0 \leftarrow 2R_0$
6: $\quad R_1 \leftarrow R_0 + P$
7: $\quad R_0 \leftarrow R_{k_i}$
8: **return** R_0

- What is good about this algorithm? Elimination of if-statements, even dummy.

Scalar multiplication algorithms

Let's improve a bit... more

Algorithm Left-to-right double-and-add-always algorithm [Coron99]

1: **Input:** P, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $Q = [k]P$
3: $R_0 \leftarrow P$
4: **for** $i \leftarrow n - 2$ **down to** 0 **do**
5: $\quad R_0 \leftarrow 2R_0$
6: $\quad R_1 \leftarrow R_0 + P$
7: $\quad R_0 \leftarrow R_{k_i}$
8: **return** R_0

- What is good about this algorithm? Elimination of if-statements, even dummy.
- What is different?

Scalar multiplication algorithms

Let's improve a bit... more

Algorithm Left-to-right double-and-add-always algorithm [Coron99]

1: **Input:** P, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $Q = [k]P$
3: $R_0 \leftarrow P$
4: **for** $i \leftarrow n - 2$ **down to** 0 **do**
5: $\quad R_0 \leftarrow 2R_0$
6: $\quad R_1 \leftarrow R_0 + P$
7: $\quad R_0 \leftarrow R_{k_i}$
8: **return** R_0

- What is good about this algorithm? Elimination of if-statements, even dummy.
- What is different? Initialization, one vs. two secret dependent accesses.
- Are safe-error attacks possible?

Scalar multiplication algorithms

What is the most common algorithm [High-Level]?

Algorithm The Montgomery Ladder

1: **Input:** $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $\boldsymbol{Q} = [k]\boldsymbol{P}$
3: $\boldsymbol{R}_0 \leftarrow \mathcal{O}$
4: $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$
5: **for** $i \leftarrow n - 1$ **down to** 0 **do**
6: $\quad b \leftarrow 1 - k_i$
7: $\quad \boldsymbol{R}_b \leftarrow \boldsymbol{R}_0 + \boldsymbol{R}_1$
8: $\quad \boldsymbol{R}_{k_i} \leftarrow 2 \cdot \boldsymbol{R}_{k_i}$
9: **return** $\boldsymbol{R}_0$

Scalar multiplication algorithms

What is the most common algorithm [High-Level]?

Algorithm The Montgomery Ladder

1: **Input:** $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $\boldsymbol{Q} = [k]\boldsymbol{P}$
3: $\boldsymbol{R}_0 \leftarrow \mathcal{O}$
4: $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$
5: **for** $i \leftarrow n - 1$ **down to** 0 **do**
6: $\quad b \leftarrow 1 - k_i$
7: $\quad \boldsymbol{R}_b \leftarrow \boldsymbol{R}_0 + \boldsymbol{R}_1$
8: $\quad \boldsymbol{R}_{k_i} \leftarrow 2 \cdot \boldsymbol{R}_{k_i}$
9: **return** $\boldsymbol{R}_0$

- What is good about this algorithm? Similarly to the previous one: Regularity...

Scalar multiplication algorithms

What is the most common algorithm [High-Level]?

Algorithm The Montgomery Ladder

1: **Input:** $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $\boldsymbol{Q} = [k]\boldsymbol{P}$
3: $\boldsymbol{R}_0 \leftarrow \mathcal{O}$
4: $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$
5: **for** $i \leftarrow n - 1$ **down to** 0 **do**
6: $\quad b \leftarrow 1 - k_i$
7: $\quad \boldsymbol{R}_b \leftarrow \boldsymbol{R}_0 + \boldsymbol{R}_1$
8: $\quad \boldsymbol{R}_{k_i} \leftarrow 2 \cdot \boldsymbol{R}_{k_i}$
9: **return** $\boldsymbol{R}_0$

- What is good about this algorithm? Similarly to the previous one: Regularity...
- Is it constant-time?

Scalar multiplication algorithms

What is the most common algorithm [High-Level]?

Algorithm The Montgomery Ladder

1: **Input:** $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
2: **Output:** $\boldsymbol{Q} = [k]\boldsymbol{P}$
3: $\boldsymbol{R}_0 \leftarrow \mathcal{O}$
4: $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$
5: **for** $i \leftarrow n - 1$ **down to** 0 **do**
6: $\quad b \leftarrow 1 - k_i$
7: $\quad \boldsymbol{R}_b \leftarrow \boldsymbol{R}_0 + \boldsymbol{R}_1$
8: $\quad \boldsymbol{R}_{k_i} \leftarrow 2 \cdot \boldsymbol{R}_{k_i}$
9: **return** $\boldsymbol{R}_0$

- What is good about this algorithm? Similarly to the previous one: Regularity...
- Is it constant-time? Caching

Scalar multiplication algorithms

What is the most common algorithm [Low-Level]? Is this really constant time?

Algorithm The Montgomery ladder for x-coordinate-based X25519 scalar multiplication on $E : y^2 = x^3 + 486662x^2 + x$

Input: $k \in \{0, \ldots, 2^{255} - 1\}$ and the x-coordinate x_P of a point P
Output: $x_{[k]P}$, the x-coordinate of $[k]P$
$X_1 \leftarrow 1, Z_1 \leftarrow 0, X_2 \leftarrow x_P, Z_2 \leftarrow 1$
$p \leftarrow 0$
for $i \leftarrow 254$ **down to** 0 **do**
    $c \leftarrow k_i \oplus p, p \leftarrow k_i$
    $(X_1, X_2) \leftarrow \text{cswap}(X_1, X_2, c), (Z_1, Z_2) \leftarrow \text{cswap}(Z_1, Z_2, c)$
    $(X_1, Z_1, X_2, Z_2) \leftarrow \text{ladderstep}(x_P, X_1, Z_1, X_2, Z_2)$
$(X_1, X_2) \leftarrow \text{cswap}(X_1, X_2, p), (Z_1, Z_2) \leftarrow \text{cswap}(Z_1, Z_2, p)$
return (X_1, Z_1)

Scalar multiplication algorithms

What is the most common algorithm [Low-Level]? Is this really constant time?

Algorithm The Montgomery ladder for x-coordinate-based X25519 scalar multiplication on $E : y^2 = x^3 + 486662x^2 + x$

Input: $k \in \{0, \ldots, 2^{255} - 1\}$ and the x-coordinate x_P of a point P
Output: $x_{[k]P}$, the x-coordinate of $[k]P$
$X_1 \leftarrow 1, Z_1 \leftarrow 0, X_2 \leftarrow x_P, Z_2 \leftarrow 1$
$p \leftarrow 0$
for $i \leftarrow 254$ **down to** 0 **do**
    $c \leftarrow k_i \oplus p, p \leftarrow k_i$
    $(X_1, X_2) \leftarrow \text{cswap}(X_1, X_2, c), (Z_1, Z_2) \leftarrow \text{cswap}(Z_1, Z_2, c)$
    $(X_1, Z_1, X_2, Z_2) \leftarrow \text{ladderstep}(x_P, X_1, Z_1, X_2, Z_2)$
$(X_1, X_2) \leftarrow \text{cswap}(X_1, X_2, p), (Z_1, Z_2) \leftarrow \text{cswap}(Z_1, Z_2, p)$
return (X_1, Z_1)

- There are several optimizations here: cswap, p, and ladderstep - let me explain.

Scalar multiplication algorithms

What is the most common algorithm [Low-Level]? Is this really constant time?

Algorithm The Montgomery ladder for x-coordinate-based X25519 scalar multiplication on $E : y^2 = x^3 + 486662x^2 + x$

Input: $k \in \{0, \ldots, 2^{255} - 1\}$ and the x-coordinate x_P of a point P
Output: $x_{[k]P}$, the x-coordinate of $[k]P$
$X_1 \leftarrow 1, Z_1 \leftarrow 0, X_2 \leftarrow x_P, Z_2 \leftarrow 1$
$p \leftarrow 0$
for $i \leftarrow 254$ **down to** 0 **do**
    $c \leftarrow k_i \oplus p, p \leftarrow k_i$
    $(X_1, X_2) \leftarrow \text{cswap}(X_1, X_2, c), (Z_1, Z_2) \leftarrow \text{cswap}(Z_1, Z_2, c)$
    $(X_1, Z_1, X_2, Z_2) \leftarrow \text{ladderstep}(x_P, X_1, Z_1, X_2, Z_2)$
$(X_1, X_2) \leftarrow \text{cswap}(X_1, X_2, p), (Z_1, Z_2) \leftarrow \text{cswap}(Z_1, Z_2, p)$
return (X_1, Z_1)

- There are several optimizations here: cswap, p, and ladderstep - let me explain. But is it constant time?

Countermeasures

Besides what we saw above, like regularity, what other countermeasures are possible?

- Algorithmic vs. Generic ones

Countermeasures

Besides what we saw above, like regularity, what other countermeasures are possible?

- Algorithmic vs. Generic ones
- Scalar randomization (next slide)

Countermeasures

Besides what we saw above, like regularity, what other countermeasures are possible?

- Algorithmic vs. Generic ones
- Scalar randomization (next slide)
- Coordinate (re-)randomization, point blinding
  - Projective coordinates: for $r \in_R \mathbb{F}_p$ compute: $(X, Y, Z) \mapsto (r \cdot X, r \cdot Y, r \cdot Z)$;
  - Jacobian representation: for $r \in_R \mathbb{F}_p$ compute: $(X, Y, Z) \mapsto (r^2 \cdot X, r^3 \cdot Y, r \cdot Z)$;

Countermeasures

Besides what we saw above, like regularity, what other countermeasures are possible?

- Algorithmic vs. Generic ones
- Scalar randomization (next slide)
- Coordinate (re-)randomization, point blinding
  - Projective coordinates: for $r \in_R \mathbb{F}_p$ compute: $(X, Y, Z) \mapsto (r \cdot X, r \cdot Y, r \cdot Z)$;
  - Jacobian representation: for $r \in_R \mathbb{F}_p$ compute: $(X, Y, Z) \mapsto (r^2 \cdot X, r^3 \cdot Y, r \cdot Z)$;
- The whole ECDSA and ECDH need protection
  - In ECDSA: final modular multiplication, among others.
  - In ECDH: coordinate conversations; we will look at traces from: `https://github.com/sca-secure-library-sca25519/sca25519`

Countermeasures

Besides what we saw above, like regularity, what other countermeasures are possible?

- Algorithmic vs. Generic ones
- Scalar randomization (next slide)
- Coordinate (re-)randomization, point blinding
  - Projective coordinates: for $r \in_R \mathbb{F}_p$ compute: $(X, Y, Z) \mapsto (r \cdot X, r \cdot Y, r \cdot Z)$;
  - Jacobian representation: for $r \in_R \mathbb{F}_p$ compute: $(X, Y, Z) \mapsto (r^2 \cdot X, r^3 \cdot Y, r \cdot Z)$;
- The whole ECDSA and ECDH need protection
  - In ECDSA: final modular multiplication, among others.
  - In ECDH: coordinate conversations; we will look at traces from: `https://github.com/sca-secure-library-sca25519/sca25519`
- Generic fault injection countermeasures:
  - a flow-counter and fault counter
  - random output in case of error
  - protecting against invalid curve attacks: duplication/detection

Countermeasures

Besides what we saw above, like regularity, what other countermeasures are possible?

- Algorithmic vs. Generic ones
- Scalar randomization (next slide)
- Coordinate (re-)randomization, point blinding
  - Projective coordinates: for $r \in_R \mathbb{F}_p$ compute: $(X, Y, Z) \mapsto (r \cdot X, r \cdot Y, r \cdot Z)$;
  - Jacobian representation: for $r \in_R \mathbb{F}_p$ compute: $(X, Y, Z) \mapsto (r^2 \cdot X, r^3 \cdot Y, r \cdot Z)$;
- The whole ECDSA and ECDH need protection
  - In ECDSA: final modular multiplication, among others.
  - In ECDH: coordinate conversations; we will look at traces from: `https://github.com/sca-secure-library-sca25519/sca25519`
- Generic fault injection countermeasures:
  - a flow-counter and fault counter
  - random output in case of error
  - protecting against invalid curve attacks: duplication/detection
- Specialized countermeasures: against template attacks, address-bit attacks...

Countermeasures

Scalar randomization techniques

Group scalar randomization

function Mult(G, k)

$r \xleftarrow{\$} \{0, 1, \ldots, 2^{32}\}$

return $[k + rn]G$

Additive splitting

function Mult(G, k)

$r \xleftarrow{\$} \mathbb{Z}_n^*$

return $[k - r]G + [r]G$

Euclidean splitting

function Mult(G, k)

$r \xleftarrow{\$} \{0, 1, \ldots, 2^{\lfloor \log_2(n)/2 \rfloor}\}$

$S \leftarrow [r]G$

$k_1 \leftarrow k \mod r$

$k_2 \leftarrow \lfloor \frac{k}{r} \rfloor$

return $[k_1]G + [k_2]S$

Multiplicative splitting

function Mult(G, k)

$r \xleftarrow{\$} \{0, 1, \ldots, 2^{32}\}$

$S \leftarrow [r]G$

return $[kr^{-1} \mod n]S$

Countermeasures

Advanced Example: Protected Ephemeral X25519

Input: A 255-bit scalar k and the x-coordinate x_P of some point P. **Output:** $x_{[k]P}$.

1: $ctr \leftarrow 0$ ▷ Initialize iteration counter
2: $x_P \xleftarrow{\$} \{0, \ldots, 2^{256} - 1\}$ ▷ Initialize output buffer to random bytes
3: $k \leftarrow \mathsf{clamp}(k)$
4: $k \leftarrow k/8$ ▷ Divide scalar k by 8 to account for initial 3 doublings
5: Increase(ctr)
6: $(X_P, Z_P) \leftarrow \mathsf{montdouble}(\mathsf{montdouble}(\mathsf{montdouble}(x_p, 1)))$ ▷ 3 doublings to multiply by co-factor 8
7: Increase(ctr)
8: **if** $Z_P = 0$ **then**
9: **go to** Line 23 ▷ Early-abort if input point is in order-8 subgroup
10: $x_P \leftarrow X_P \cdot Z_P^{-1}$ ▷ Return to affine x-coordinate
11: $X_1 \leftarrow 1, Z_1 \leftarrow 0$
12: $Z_2 \xleftarrow{\$} \{0, \ldots, 2^{255} - 20\}, X_2 \leftarrow x_P \cdot Z_2$ ▷ Initial randomization of projective representation
13: $k \leftarrow k \oplus 2k$ ▷ Precompute condition bits for cswap
14: Increase(ctr)
15: **for** i from 252 downto 1 **do** ▷ Main scalar-multiplication loop
16: $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{cswaprr}\ (X_1, Z_1, X_2, Z_2, k[i])$ ▷ Projective re-randomization merged with cswap
17: $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{ladderstep}\ (x_P, X_1, Z_1, X_2, Z_2)$
18: Increase(ctr)
19: $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{cswaprr}\ ((X_1, Z_1, X_2, Z_2), k[0])$
20: $x_P \leftarrow X_2 \cdot Z_2^{-1}$
21: Increase(ctr)
22: **if** ! Verify(ctr) **then** ▷ Detected wrong flow, including iteration count
23: $x_P \xleftarrow{\$} \{0, \ldots, 2^{256} - 1\}$ ▷ Set output buffer to random bytes
24: **return** x_P

Reverse-engineering

Goal: Extract implementation details from black-box ECC implementation using side-channels

- Groups of 2-3, knowledge of Python, SCA, ECC
- Work with **pyecsca** toolkit
- Mostly simulations, some work with captured **traces**

Quickstart

`start.ipynb`
Test the setup and familiarize with the toolkit

1. Start Jupyter notebook
2. Open `notebooks/start.ipynb`
3. Run the prepared cells
4. Solve the exercise

pyecsca

https://neuromancer.sk/pyecsca/

Quickstart

Solution

here

```
recovered_privkey = 1
for formula_call in subtree:
    if formula_call.formula.shortname == "add":
        recovered_privkey |= 1
    elif formula_call.formula.shortname == "dbl":
        recovered_privkey <<= 1

print(recovered_privkey)
```

ECC implementations

`implementations.ipynb`
Manually explore ECC implementations

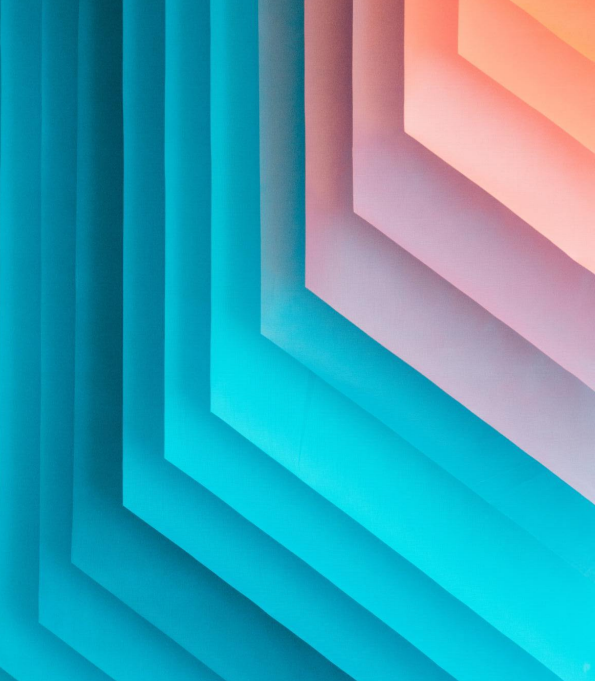
1. We have three implementations (of ECDH). Plot all of them and tell me what you can learn from how they look.
2. Now we concentrate on the implementation 1. Let's count how many iterations there are. Tell me what curve it can be.
3. Can we learn more from the scalar correlation? Is it left-to-right or right-to-left?
4. Let's see what we can learn from the results using correlation to the traces.
5. Can we verify whether the most significant bit is fixed to 1 using CPA? Can we confirm the arithmetic details?
6. Investigate a number of different possibilities to implement scalar multiplication.

ECC implementations

Solution

here

- Montgomery ladder, left-to-right
- 254 iterations
- Correlation on the result present after the scalarmult end $\Rightarrow$ conversion to affine
- The implementation is X25519 and the most significant scalar bit is 1
  - CPA worked under the assumption that the complete formulas "ladd-1987-m" are used
  - The traces come from: `https://github.com/sca-secure-library-sca25519/sca25519`
- The space of ECC implementations is very large



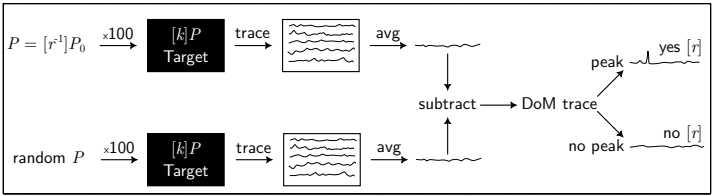
How?

General idea

- **Idea:** Use side-channel attacks and turn them around
  - ~~Assume knowledge of impl. and target key~~
  - Assume knowledge of key and target impl.
- Concretely "*special-point-based*" attacks: **RPA, ZVP, EPA**
- Can recognize when a special point appears in scalar multiplication
- **Idea:** Behavior of different implementations differs under these attacks

RPA

- **Refined Power Analysis** attack by Goubin
- Zero-coordinate points $P_0 = (x, 0)$ and $P_0 = (0, y)$
- 0 leaks!
- **Idea:** Introduce 0 conditionally on secret key
- $P = [r^{-1} \mod n]P_0$ as input to $[k]P$ computation (ECDH)
  - $[r]P$ computed $\Rightarrow$ 0 ☑
  - $[r]P$ not computed $\Rightarrow$ no 0 ✖



RPA-RE

`rpa.ipynb`
Reverse-engineer using RPA

1. Explore scalar multipliers
2. Practice the RPA attack
3. Use RPA for reverse-engineering

RPA-RE

Solution

here

- Scalar multipliers compute different chains of multiples
  - LTR and RTL never compute the same (2^i in RTL)
  - They get close for binary palindromes
- RPA attack works similarly to DPA
- Doing "*binary search*" for the secret multiplier works for RPA-RE

Conclusions

- Implementing elliptic curve cryptography is a non-trivial process requiring a range of implementation choices.
- Most of the attacks require knowledge of the details of the implementation.
- Most of the embedded implementations are black-box, and the attacker does not know the implementation.
- You learned how to
  - semi-manually use side-channel analysis to learn some details of the implementation;
  - to use automatic methods to learn the implementation details (based on Refined Power Analysis).

Conclusions

- Implementing elliptic curve cryptography is a non-trivial process requiring a range of implementation choices.
- Most of the attacks require knowledge of the details of the implementation.
- Most of the embedded implementations are black-box, and the attacker does not know the implementation.
- You learned how to
  - semi-manually use side-channel analysis to learn some details of the implementation;
  - to use automatic methods to learn the implementation details (based on Refined Power Analysis).
- Thank you for your attendance!

Thanks!

neuromancer.sk/pyecsca/

Icons from Noun Project & Font Awesome

Photos from Unsplash

References

- Louis Goubin;
  A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems
- Toru Akishita, Tsuyoshi Takagi;
  Zero-value point attacks on elliptic curve cryptosystem
- Tetsuya Izu, Tsuyoshi Takagi;
  Exceptional procedure attack on elliptic curve cryptosystems
- Vladimir Sedlacek, Jesús-Javier Chi-Domínguez, Jan Jancar, Billy Bob Brumley;
  A formula for disaster: a unified approach to elliptic curve special-point-based attacks
- Jan Jancar, Vojtech Suchanek, Petr Svenda, Vladimir Sedlacek, Łukasz Chmielewski;
  pyecsca: Reverse-engineering black-box elliptic curve cryptography via side-channel analysis

Resources

- https://unsplash.com/photos/9V5GssdEG-4
- https://unsplash.com/photos/wKc-i5zwfok
- https://unsplash.com/photos/Tk0B3Dfkf_4
- https://unsplash.com/photos/pAyN5zqdqDo