Elliptic Curve Cryptosystems in the Presence of Faults

Marc Joye
Elliptic Curve Cryptosystems in the Presence of Faults
Elliptic Curve Cryptography


- Useful for key exchange, encryption and digital signature
Fault Attacks

- Adversary induces faults during the computation
  - glitches (supply voltage or external clock)
  - temperature
  - light emission (white light or laser)
  - ...

![Diagram of a circuit with key, input, error, and toxic bottle]
This Talk

- Fault attacks and countermeasures for **elliptic-curve cryptosystems**
  - cryptographic primitives vs. cryptographic protocols
- Most known fault attacks are directed to cryptographic primitives
  - notable exception
    - skipping attacks [Schmidt and Herbst, 2008]
    - fault model experimentally validated

- List of research problems
Basics on Elliptic Curves (1/3)

Definition

An elliptic curve over a field $\mathbb{K}$ is the set of points $(x, y) \in E$

$$E : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6$$

along with the point $O$ at infinity

- $\text{Char } \mathbb{K} \neq 2, 3 \Rightarrow a_1 = a_2 = a_3 = 0$
- $\text{Char } \mathbb{K} = 2$ (non-supersingular case) $\Rightarrow a_1 = 1, a_3 = a_4 = 0$

Fact

The set $E(\mathbb{K})$ forms an additive group where

- $O$ is the neutral element
- the group law is given by the “chord-and-tangent” rule
Basics on Elliptic Curves (1/3)

Definition

An elliptic curve over a field $\mathbb{K}$ is the set of points $(x, y) \in E$

\[
E : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6
\]

along with the point $O$ at infinity

- $\text{Char } \mathbb{K} \neq 2, 3 \Rightarrow a_1 = a_2 = a_3 = 0$
- $\text{Char } \mathbb{K} = 2$ (non-supersingular case) $\Rightarrow a_1 = 1, a_3 = a_4 = 0$

Fact

The set $E(\mathbb{K})$ forms an additive group where

- $O$ is the neutral element
- the group law is given by the “chord-and-tangent” rule
Elliptic curves over $\mathbb{R}$

$y^2 = x^3 - 7x$

$P = (-2.35, -1.86)$, $Q = (-0.1, 0.836)$

$R = (3.89, -5.62)$
Elliptic curves over $\mathbb{R}$

- $y^2 = x^3 - 7x$
  - $P = (-2.35, -1.86)$
  - $Q = (-0.1, 0.836)$
  - $R = (3.89, -5.62)$

- $y^2 = x^3 - 3x + 5$
  - $P = (2, 2.65)$
  - $R = (1.11, 2.64)$
Basics on Elliptic Curves (3/3)

\[ E : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6 \]

- Let \( P = (x_1, y_1) \) and \( Q = (x_2, y_2) \)
- **Group law**
  - \( P + O = O + P = P \)
  - \( -P = (x_1, -y_1 - a_1 x_1 - a_3) \)
  - \( P + Q = (x_3, y_3) \) where
    \[ x_3 = \lambda^2 + a_1 \lambda - a_2 - x_1 - x_2, \quad y_3 = (x_1 - x_3) \lambda - y_1 - a_1 x_3 - a_3 \]
  - \( \lambda = \begin{cases} 
    \frac{y_1 - y_2}{x_1 - x_2} & \text{[addition]} \\
    \frac{3x_1^2 + 2a_2 x_1 + a_4 - a_1 y_1}{2y_1 + a_1 x_1 + a_3} & \text{[doubling]} 
  \end{cases} \)
Basics on Elliptic Curves (3/3)

Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$.

**Group law**

- $P + O = O + P = P$
- $-P = (x_1, -y_1 - a_1 x_1 - a_3)$
- $P + Q = (x_3, y_3)$ where

\[
x_3 = \lambda^2 + a_1 \lambda - a_2 - x_1 - x_2, \quad y_3 = (x_1 - x_3) \lambda - y_1 - a_1 x_3 - a_3
\]

with $\lambda = \begin{cases} 
\frac{y_1 - y_2}{x_1 - x_2} & \text{[addition]} \\
\frac{3x_1^2 + 2a_2x_1 + a_4 - a_1 y_1}{2y_1 + a_1 x_1 + a_3} & \text{[doubling]}
\end{cases}$
Basics on Elliptic Curves (3/3)

\[ E : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6 \]

- Let \( P = (x_1, y_1) \) and \( Q = (x_2, y_2) \)
- **Group law**
  - \( P + O = O + P = P \)
  - \( -P = (x_1, -y_1 - a_1 x_1 - a_3) \)
  - \( P + Q = (x_3, y_3) \) where
    \[ x_3 = \lambda^2 + a_1 \lambda - a_2 - x_1 - x_2, \quad y_3 = (x_1 - x_3) \lambda - y_1 - a_1 x_3 - a_3 \]
    with \( \lambda = \begin{cases} 
    \frac{y_1 - y_2}{x_1 - x_2} & \text{[addition]} \\
    \frac{3x_1^2 + 2a_2 x_1 + a_4 - a_1 y_1}{2y_1 + a_1 x_1 + a_3} & \text{[doubling]} 
    \end{cases} \)
Basics on Elliptic Curves (3/3)

\[ E : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6 \]

- Let \( P = (x_1, y_1) \) and \( Q = (x_2, y_2) \)
- **Group law**
  - \( P + O = O + P = P \)
  - \( -P = (x_1, -y_1 - a_1 x_1 - a_3) \)
  - \( P + Q = (x_3, y_3) \) where

\[ x_3 = \lambda^2 + a_1 \lambda - a_2 - x_1 - x_2, \quad y_3 = (x_1 - x_3) \lambda - y_1 - a_1 x_3 - a_3 \]

with \( \lambda = \begin{cases} \frac{y_1 - y_2}{x_1 - x_2} & \text{[addition]} \\ \frac{3x_1^2 + 2a_2 x_1 + a_4 - a_1 y_1}{2y_1 + a_1 x_1 + a_3} & \text{[doubling]} \end{cases} \)
EC Primitive

- EC primitive = point multiplication (a.k.a. scalar multiplication)
  \[ E(\mathbb{K}) \times \mathbb{Z} \rightarrow E(\mathbb{K}), \quad (P, d) \mapsto Q = [d]P \]

- one-way function

- Cryptographic elliptic curves
  - \( \mathbb{K} = \mathbb{F}_q \) with \( q = p \) (a prime) or \( q = 2^m \)
  - \( \#E(\mathbb{K}) = hn \) with \( h \in \{1, 2, 3, 4\} \) and \( n \) prime
  - typical size: \(|n|_2 = 224 \) (\( \approx |\mathbb{K}|_2 \))

**Definition (ECDL Problem)**

Let \( \mathcal{G} = \langle P \rangle \subseteq E(\mathbb{K}) \) a subgroup of prime order \( n \)

Given points \( P, Q \in \mathcal{G} \), compute \( d \) such that \( Q = [d]P \)
EC Digital Signature Algorithm (1/2)

- Elliptic curve variant of the Digital Signature Algorithm
  - a.k.a. Digital Signature Standard - DSS

- Domain parameters
  - finite field $\mathbb{F}_q$
  - elliptic curve $E/\mathbb{F}_q$ with $\#E(\mathbb{F}_q) = hn$
    - cofactor $h \leq 4$ and $n$ prime
  - cryptographic hash function $H$
  - point $G \in E$ of prime order $n$

$$\{\mathbb{F}_q, E, n, h, H, G\}$$
EC Digital Signature Algorithm (1/2)

- Elliptic curve variant of the Digital Signature Algorithm
  - a.k.a. Digital Signature Standard - DSS
- Domain parameters
  - finite field \( \mathbb{F}_q \)
  - elliptic curve \( E / \mathbb{F}_q \) with \( \#E(\mathbb{F}_q) = h n \)
    - cofactor \( h \leq 4 \) and \( n \) prime
  - cryptographic hash function \( H \)
  - point \( G \in E \) of prime order \( n \)

\[ \{ \mathbb{F}_q, E, n, h, H, G \} \]
EC Digital Signature Algorithm (2/2)

- **Key generation:** \( Y = [d]G \) with \( d \leftarrow \{1, \ldots, n - 1\} \)
  
  \( pk = \{G, Y\} \) and \( sk = \{d\} \)

- **Signing**
  
  **Input** message \( m \) and private key \( sk \)
  
  **Output** signature \( S = (r, s) \)

  1. pick a random \( k \in \{1, \ldots, n - 1\} \)
  2. compute \( T = [k]G \) and set \( r = x(T) \pmod{n} \)
  3. if \( r = 0 \) then goto Step 1
  4. compute \( s = (H(m) + d r)/k \pmod{n} \)
  5. return \( S = (r, s) \)

- **Verification**

  1. compute \( u_1 = H(m)/s \pmod{n} \) and \( u_2 = r/s \pmod{n} \)
  2. compute \( T = [u_1]G + [u_2]Y \)
  3. check whether \( r \equiv x(T) \pmod{n} \)
EC Digital Signature Algorithm (2/2)

- **Key generation:** \( Y = [d]G \) with \( d \leftarrow \{1, \ldots, n - 1\} \)
  \( pk = \{G, Y\} \) and \( sk = \{d\} \)

- **Signing**
  - **Input** message \( m \) and **private key** \( sk \)
  - **Output** signature \( S = (r, s) \)
  1. pick a random \( k \in \{1, \ldots, n - 1\} \)
  2. compute \( T = [k]G \) and set \( r = x(T) \pmod{n} \)
  3. if \( r = 0 \) then goto Step 1
  4. compute \( s = (H(m) + dr)/k \pmod{n} \)
  5. return \( S = (r, s) \)

- **Verification**
  1. compute \( u_1 = H(m)/s \pmod{n} \) and \( u_2 = r/s \pmod{n} \)
  2. compute \( T = [u_1]G + [u_2]Y \)
  3. check whether \( r \equiv x(T) \pmod{n} \)
EC Digital Signature Algorithm (2/2)

- **Key generation:** \( Y = [d]G \) with \( d \leftarrow \{1, \ldots, n - 1\} \)
  
  \[ pk = \{G, Y\} \text{ and } sk = \{d\} \]

- **Signing**
  
  **Input** message \( m \) and **private key** \( sk \)
  
  **Output** signature \( S = (r, s) \)

1. pick a **random** \( k \in \{1, \ldots, n - 1\} \)
2. compute \( T = [k]G \) and set \( r = x(T) \pmod{n} \)
3. if \( r = 0 \) then goto Step 1
4. compute \( s = (H(m) + d \, r) / k \pmod{n} \)
5. return \( S = (r, s) \)

- **Verification**

1. compute \( u_1 = H(m)/s \pmod{n} \) and \( u_2 = r/s \pmod{n} \)
2. compute \( T = [u_1]G + [u_2]Y \)
3. check whether \( r \equiv x(T) \pmod{n} \)
Public Key Validation

- For each received $pk = \{\text{domain params}, Y\}$, check that
  1. $Y \in E$
  2. $Y \neq O$
  3. (optional) $[n]Y = O$
EC Diffie-Hellman Key Exchange

- ECDH = Elliptic Curve Diffie-Hellman protocol
  - elliptic curve variant of the Diffie-Hellman key exchange

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td>$R_B$</td>
<td>$R_A$</td>
</tr>
</tbody>
</table>

$K_A = [a]R_B$

$K_B = [b]R_A$

- suffers from the man-in-the-middle attack
  - no data-origin authentication
  - exchanged messages should be signed

- ECMQV = Elliptic Curve Menezes-Qu-Vanstone protocol
  - implicit authentication
EC Diffie-Hellman Key Exchange

- ECDH = **Elliptic Curve Diffie-Hellman protocol**
  - elliptic curve variant of the Diffie-Hellman key exchange

Alice

| a | R_B |

Bob

| R_A | b |

\[
K_A = [a]R_B \\
K_B = [b]R_A
\]

- suffers from the **man-in-the-middle** attack
- no data-origin authentication
- exchanged messages should be signed

ECMQV = **Elliptic Curve Menezes-Qu-Vanstone protocol**
- implicit authentication
EC Diffie-Hellman Key Exchange

- ECDH = Elliptic Curve Diffie-Hellman protocol
  - elliptic curve variant of the Diffie-Hellman key exchange

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$R_A$</td>
</tr>
<tr>
<td>$R_B$</td>
<td>$b$</td>
</tr>
</tbody>
</table>

$K_A = [a]R_B$  $K_B = [b]R_A$

- suffers from the man-in-the-middle attack
  - no data-origin authentication
  - exchanged messages should be signed

- ECMQV = Elliptic Curve Menezes-Qu-Vanstone protocol
  - implicit authentication
EC Diffie-Hellman Key Exchange

- **ECDH = Elliptic Curve Diffie-Hellman protocol**
  - elliptic curve variant of the Diffie-Hellman key exchange

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$R_A$</td>
</tr>
<tr>
<td>$R_B$</td>
<td>$b$</td>
</tr>
</tbody>
</table>

- $K_A = [a]R_B$
- $K_B = [b]R_A$

- suffers from the **man-in-the-middle** attack
  - no data-origin authentication
  - exchanged messages should be signed

- **ECMQV = Elliptic Curve Menezes-Qu-Vanstone protocol**
  - implicit authentication
ECDH Augmented Encryption (1/2)

- **ECIES** = Elliptic Curve Integrated Encryption System
  - proposed by Michel Abdalla, Mihir Bellare and Phillip Rogaway in 2000
  - submitted to IEEE P1363a

- **Domain parameters**
  - finite field $\mathbb{F}_q$
  - elliptic curve $E/\mathbb{F}_q$ with $\#E(\mathbb{F}_q) = hn$
  - “special” hash functions
    - message authentication code $MAC_K(c)$
    - key derivation function $KD(T, \ell)$
  - symmetric encryption algorithm $Enc_K(m)$
  - point $G \in E$ of prime order $n$

\[
\{\mathbb{F}_q, E, n, h, MAC, KD, Enc, G\}\]
ECDH Augmented Encryption (1/2)

- **ECIES** = **Elliptic Curve Integrated Encryption System**
  - proposed by Michel Abdalla, Mihir Bellare and Phillip Rogaway in 2000
  - submitted to IEEE P1363a

- **Domain parameters**
  - finite field $\mathbb{F}_q$
  - elliptic curve $E/\mathbb{F}_q$ with $\#E(\mathbb{F}_q) = hn$
  - “special” hash functions
    - message authentication code $MAC_K(c)$
    - key derivation function $KD(T, \ell)$
  - symmetric encryption algorithm $Enc_K(m)$
  - point $G \in E$ of prime order $n$

\[ \{ \mathbb{F}_q, E, n, h, MAC, KD, Enc, G \} \]
Key generation: $Y = [d]G$ with $d \xleftarrow{\$} \{1, \ldots, n - 1\}$

$pk = \{G, Y\}$ and $sk = \{d\}$

ECIES encryption

1. pick a random $k \in \{1, \ldots, n - 1\}$
2. compute $U = [k]G$ and $T = [k]Y$
3. set $(K_1' || K_2') = KD(T, l)$
4. compute $c = Enc_{K_1}(m)$ and $r = MAC_{K_2}(c)$
5. return $(U, c, r)$

ECIES decryption

Input ciphertext $(U, c, r)$ and private key $sk$

Output plaintext $m$ or ⊥

1. compute $T' = [d]U$
2. set $(K_1' || K_2') = KD(T', l)$
3. if $MAC_{K_2}(c) = r$ then return $m = Enc_{K_1'}^{-1}(c)$
ECDH Augmented Encryption (2/2)

- **Key generation:** \( Y = [d]G \) with \( d \leftarrow \{1, \ldots, n - 1\} \)
  
  \( pk = \{G, Y\} \) and \( sk = \{d\} \)

- **ECIES encryption**
  1. pick a random \( k \in \{1, \ldots, n - 1\} \)
  2. compute \( U = [k]G \) and \( T = [k]Y \)
  3. set \((K_1||K_2) = KD(T, l)\)
  4. compute \( c = Enc_{K_1}(m) \) and \( r = MAC_{K_2}(c) \)
  5. return \((U, c, r)\)

- **ECIES decryption**
  1. Input ciphertext \((U, c, r)\) and private key \( sk\)
  2. Output plaintext \( m \) or \( \perp \)
  3. compute \( T' = [d]U \)
  4. set \((K'_1||K'_2) = KD(T', l)\)
  5. if \( MAC_{K'_2}(c) = r \) then return \( m = Enc_{K'_1}^{-1}(c) \)
ECDH Augmented Encryption (2/2)

- **Key generation:** \( Y = [d]G \) with \( d \overset{\$}{\leftarrow} \{1, \ldots, n - 1\} \)
  
  \[ \text{pk} = \{G, Y\} \text{ and } \text{sk} = \{d\} \]

- **ECIES encryption**
  1. pick a random \( k \in \{1, \ldots, n - 1\} \)
  2. compute \( U = [k]G \) and \( T = [k]Y \)
  3. set \( (K_1 || K_2) = KD(T, l) \)
  4. compute \( c = \text{Enc}_{K_1}(m) \) and \( r = \text{MAC}_{K_2}(c) \)
  5. return \((U, c, r)\)

- **ECIES decryption**
  
  **Input** ciphertext \((U, c, r)\) and **private key** \(\text{sk}\)
  
  **Output** plaintext \(m\) or \(\perp\)
  1. compute \( T' = [d]U \)
  2. set \( (K'_1 || K'_2) = KD(T', l) \)
  3. if \( \text{MAC}_{K'_2}(c) = r \) then return \( m = \text{Enc}_{K'_1}^{-1}(c) \)
Outline

1. Elliptic Curves
   - Basics on elliptic curves
   - Elliptic curve digital signature algorithm
   - Other algorithms

2. Attacks
   - Single-bit errors
   - Safe errors
   - Random errors
   - Skipping attacks

3. Countermeasures
   - Basic countermeasures
   - Scalar randomization
   - BOS\(^+\) algorithm
   - New algorithm

4. Conclusion
   - Research problems
Fault Attacks on ECC

- Bit-level vs. byte-level attacks
- Transient vs. permanent faults
- Private vs. public parameters
- Unsigned vs. signed representations
- Fixed vs. changing base point
- Basic vs. provably secure systems
Forcing-Bit Attack

Let \( d = \sum_{i=0}^{\ell-1} d_i 2^i \)

Forcing bit: \( d_j \to 0 \)

ECDSA

Check whether \( S = (r, s) \) is a valid signature

(Similarly applies when \( k_j \to 0 \) in Step 4)

ECIES
Forcing-Bit Attack

- Let \( d = \sum_{i=0}^{\ell-1} d_i 2^i \)
- Forcing bit: \( d_j \rightarrow 0 \)

**ECDSA**
- Check whether \( S = (r, s) \) is a valid signature
- (Similarly applies when \( k_j \rightarrow 0 \) in Step 4)

**ECIES**
Forcing-Bit Attack

- Let \( d = \sum_{i=0}^{\ell-1} d_i 2^i \)
- Forcing bit: \( d_j \rightarrow 0 \)

**ECDSA**

- Check whether \( S = (r, s) \) is a valid signature
  - if so, then \( d_j = 0 \)
  - if not, then \( d_j = 1 \)
- (Similarly applies when \( k_j \rightarrow 0 \) in Step 4)

**ECIES**
Forcing-Bit Attack

- Let $d = \sum_{i=0}^{\ell-1} d_i 2^i$
- Forcing bit: $d_j \rightarrow 0$

**ECDSA**

- Check whether $S = (r, s)$ is a valid signature
  - if so, then $d_j = 0$
  - if not, then $d_j = 1$
- (Similarly applies when $k_j \rightarrow 0$ in Step 4)

**ECIES**
Forcing-Bit Attack

- Let $d = \sum_{i=0}^{\ell-1} d_i 2^i$
- Forcing bit: $d_j \rightarrow 0$

ECDSA

- Check whether $S = (r, s)$ is a valid signature
  - if so, then $d_j = 0$
  - if not, then $d_j = 1$
  (Similarly applies when $k_j \rightarrow 0$ in Step 4)

ECIES

- Check the ciphertext validity
Forcing-Bit Attack

Let \( d = \sum_{i=0}^{\ell-1} d_i 2^i \)

Forcing bit: \( d_j \rightarrow 0 \)

**ECDSA**

Check whether \( S = (r, s) \) is a valid signature

- if so, then \( d_j = 0 \)
- if not, then \( d_j = 1 \)

(Similarly applies when \( k_j \rightarrow 0 \) in Step 4)

**ECIES**

Check the ciphertext validity
Forcing-Bit Attack

- Let \( d = \sum_{i=0}^{\ell-1} d_i 2^i \)
- Forcing bit: \( d_j \rightarrow 0 \)

**ECDSA**

- Check whether \( S = (r, s) \) is a valid signature
  - if so, then \( d_j = 0 \)
  - if not, then \( d_j = 1 \)
- (Similarly applies when \( k_j \rightarrow 0 \) in Step 4)

**ECIES**

- Check the ciphertext validity
  - if the output is \( m \) then \( d_j = 0 \)
  - if the output is \( \perp \) then \( d_j = 1 \)
Forcing-Bit Attack

- Let $d = \sum_{i=0}^{\ell-1} d_i 2^i$
- Forcing bit: $d_j \rightarrow 0$

**ECDSA**

- Check whether $S = (r, s)$ is a valid signature
  - if so, then $d_j = 0$
  - if not, then $d_j = 1$
- (Similarly applies when $k_j \rightarrow 0$ in Step 4)

**ECIES**

- Check the ciphertext validity
  - if the output is $m$ then $d_j = 0$
  - if the output is $\perp$ then $d_j = 1$
Forcing-Bit Attack

- Let \( d = \sum_{i=0}^{\ell-1} d_i 2^i \)
- Forcing bit: \( d_j \rightarrow 0 \)

**ECDSA**

- Check whether \( S = (r, s) \) is a valid signature
  - if so, then \( d_j = 0 \)
  - if not, then \( d_j = 1 \)
- (Similarly applies when \( k_j \rightarrow 0 \) in Step 4)

**ECIES**

- Check the ciphertext validity
  - if the output is \( m \) then \( d_j = 0 \)
  - if the output is \( \perp \) then \( d_j = 1 \)
Flipping-Bit Attack

Against ECDSA

- Let $d = \sum_{i=0}^{\ell-1} d_i 2^i$
- Flipping bit: $d_j \rightarrow \overline{d_j}$

  $$\Rightarrow \hat{s} = (r, \hat{s}) \text{ with } \begin{cases} \hat{s} = (H(m) + \hat{d} r) / k \pmod{n} \\ \hat{d} = (d_j - \overline{d_j}) 2^j + d \end{cases}$$

- Define $\hat{u}_1 = H(m) / \hat{s} \pmod{n}$ and $\hat{u}_2 = r / \hat{s} \pmod{n}$
- Compute $\hat{T} = [\hat{u}_1]G + [\hat{u}_2]Y$
- For $j = 0$ to $\ell - 1$ and $\sigma \in \{-1, 1\}$, check if
  $$x \left( \hat{T} + \left[ \frac{\sigma 2^j r}{\hat{s}} \right] G \right) = x([k]G) = r \Rightarrow \overline{d_j} - d_j = \sigma$$

  $$\Rightarrow d_j = \frac{1 - \sigma}{2}$$
Flipping-Bit Attack

Against ECDSA

- Let \( d = \sum_{i=0}^{\ell-1} d_i 2^i \)
- Flipping bit: \( d_j \rightarrow \overline{d_j} \)

\[
\Rightarrow \hat{S} = (r, \hat{s}) \text{ with } \begin{cases} 
\hat{s} = (H(m) + \hat{d} r) / k \pmod{n} \\
\hat{d} = (d_j - d_j) 2^j + d
\end{cases}
\]

- Define \( \hat{u}_1 = H(m) / \hat{s} \pmod{n} \) and \( \hat{u}_2 = r / \hat{s} \pmod{n} \)
- Compute \( \hat{T} = [\hat{u}_1]G + [\hat{u}_2]Y \)
- For \( j = 0 \) to \( \ell - 1 \) and \( \sigma \in \{-1, 1\} \), check if

\[
x \left( \hat{T} + \left[ \frac{\sigma 2^j r}{\hat{s}} \right] G \right) = x([k]G) = r \Rightarrow \overline{d}_j - d_j = \sigma \\
\Rightarrow d_j = \frac{1 - \sigma}{2}
\]
Flipping-Bit Attack

Against ECDSA

- Let $d = \sum_{i=0}^{\ell-1} d_i 2^i$
- Flipping bit: $d_j \rightarrow \overline{d_j}$

$$\Rightarrow \hat{S} = (r, \hat{s}) \text{ with } \begin{cases} \hat{s} = (H(m) + \hat{d} r)/k \pmod{n} \\ \hat{d} = (\overline{d_j} - d_j)2^j + d \end{cases}$$

- Define $\hat{u}_1 = H(m)/\hat{s} \pmod{n}$ and $\hat{u}_2 = r/\hat{s} \pmod{n}$
- Compute $\hat{T} = [\hat{u}_1]G + [\hat{u}_2]Y$
- For $j = 0$ to $\ell - 1$ and $\sigma \in \{-1, 1\}$, check if

$$x(\hat{T} + \left[\frac{\sigma 2^j r}{\hat{s}}\right]G) = x([k]G) = r \Rightarrow \overline{d_j} - d_j = \sigma \Rightarrow d_j = \frac{1-\sigma}{2}$$
Flipping-Bit Attack

Against ECDSA

- Let \( d = \sum_{i=0}^{\ell-1} d_i 2^i \)
- Flipping bit: \( d_j \rightarrow \overline{d_j} \)

\[ \Rightarrow \hat{s} = (r, \hat{s}) \text{ with } \begin{cases} \hat{s} = (H(m) + \hat{d} r)/k \pmod{n} \\ \hat{d} = (\overline{d_j} - d_j)2^j + d \end{cases} \]

- Define \( \hat{u}_1 = H(m)/\hat{s} \pmod{n} \) and \( \hat{u}_2 = r/\hat{s} \pmod{n} \)
- Compute \( \hat{T} = [\hat{u}_1]G + [\hat{u}_2]Y \)
- For \( j = 0 \) to \( \ell - 1 \) and \( \sigma \in \{-1, 1\} \), check if

\[ x \left( \hat{T} + \left[ \frac{\sigma 2^j r}{\hat{s}} \right] G \right) = x([k]G) = r \Rightarrow \overline{d_j} - d_j = \sigma \]

\[ \Rightarrow d_j = \frac{1-\sigma}{2} \]
### Flipping-Bit Attack

**Against ECDSA**

- Let \( d = \sum_{i=0}^{\ell-1} d_i 2^i \)
- Flipping bit: \( d_j \rightarrow \bar{d_j} \)

\[
\Rightarrow \hat{S} = (r, \hat{s}) \quad \text{with} \quad \begin{cases} 
\hat{s} = (H(m) + \hat{d} r)/k \pmod{n} \\
\hat{d} = (\bar{d_j} - d_j)2^j + d
\end{cases}
\]

- Define \( \hat{u}_1 = H(m)/\hat{s} \pmod{n} \) and \( \hat{u}_2 = r/\hat{s} \pmod{n} \)
- Compute \( \hat{T} = [\hat{u}_1]G + [\hat{u}_2]Y \)
- For \( j = 0 \) to \( \ell - 1 \) and \( \sigma \in \{-1, 1\} \), check if

\[
x \left( \hat{T} + \left[ \frac{\sigma 2^j r}{\hat{s}} \right] G \right) = x([k]G) = r \Rightarrow \bar{d_j} - d_j = \sigma \\
\Rightarrow d_j = \frac{1-\sigma}{2}
\]
Sign-Change Fault Attack

- Point inversion is inexpensive on elliptic curves
  \[ P = (x_1, y_1) \Rightarrow -P = (x_1, -y_1 - a_1 x_1 - a_3) \]
- Signed-digit point multiplication algorithms are preferred for computing
  \[ Q = [d]P \]
  - e.g., NAF-based method gives a speed-up factor of 11.11%
- \( d = \sum_{i=0}^{\ell} \delta_i 2^i \) with \( \delta_i \in \{0, 1, -1\} \)
- Signed-digit encoding: \( \delta_i = (\text{sign bit}, \text{value bit}) \),
  \[ 0 = (\star, 0), \quad 1 = (0, 1), \quad -1 = (1, 1) \]

Sign-change fault attack (specialized flipping-bit attack)

Induce a fault in the sign bit of \( \delta_i \)
- on the fly
- during exponent recoding
Safe-Error Attack (1/2)

- Double-and-add-\textit{always} algorithm
  - additive variant of the square-and-multiply-\textit{always}

\begin{align*}
\text{Input: } & \mathbf{U}, d = (d_{\ell-1}, \ldots, d_0)_2 \\
\text{Output: } & T = [d]\mathbf{U}
\end{align*}

1. $R_0 \leftarrow O; R_1 \leftarrow O$
2. For $i = \ell - 1$ downto 0 do
   - $R_0 \leftarrow [2]R_0$
   - $b \leftarrow 1 - d_i; R_b \leftarrow R_b + \mathbf{U}$
3. Return $R_0$

- when $b = 1$, there is a \textit{dummy} point addition
Safe-Error Attack (1/2)

- Double-and-add-\textit{always} algorithm
  - additive variant of the square-and-multiply-\textit{always}

\begin{itemize}
  \item Input: $U, d = (d_{\ell-1}, \ldots, d_0)_2$
  \item Output: $T = [d]U$
\end{itemize}

\begin{algorithm}
\begin{algorithmic}
\State $R_0 \leftarrow O$; $R_1 \leftarrow O$
\For{$i = \ell - 1$ \textbf{downto} 0}
\State $R_0 \leftarrow [2]R_0$
\State $b \leftarrow 1 - d_i$; $R_b \leftarrow R_b + U$
\EndFor
\State Return $R_0$
\end{algorithmic}
\end{algorithm}

- when $b = 1$, there is a dummy point addition
Timely induce a fault into the ALU during the add operation at iteration $i$

Check the output
- if an invalid ciphertext is notified (i.e., ⊥) then the error was effective
  $d_i = 1$
- if the result is correct then the point addition was dummy [safe error]
  $d_i = 0$

Re-iterate the attack for another value of $i$
Safe-Error Attack (2/2)

Against ECIES

- **Timely** induce a fault into the ALU during the add operation at iteration $i$
- Check the output
  - if an invalid ciphertext is notified (i.e., $\bot$) then the error was effective
    $\Rightarrow d_i = 1$
  - if the result is correct then the point addition was dummy [safe error]
    $\Rightarrow d_i = 0$
- Re-iterate the attack for another value of $i$
Safe-Error Attack (2/2)

<table>
<thead>
<tr>
<th>Against ECIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>- <strong>Timely</strong> induce a fault into the ALU during the add operation at iteration $i$</td>
</tr>
<tr>
<td>- Check the output</td>
</tr>
<tr>
<td>- if an invalid ciphertext is notified (i.e., $\perp$) then the error was effective $\Rightarrow d_i = 1$</td>
</tr>
<tr>
<td>- if the result is correct then the point addition was dummy [safe error] $\Rightarrow d_i = 0$</td>
</tr>
<tr>
<td>- Re-iterate the attack for another value of $i$</td>
</tr>
</tbody>
</table>
Safe-Error Attack (2/2)

Against ECIES

- **Timely** induce a fault into the ALU during the add operation at iteration $i$
- Check the output
  - if an invalid ciphertext is notified (i.e., ⊥) then the error was effective
    $\Rightarrow d_i = 1$
  - if the result is correct then the point addition was dummy [safe error]
    $\Rightarrow d_i = 0$
- Re-iterate the attack for another value of $i$
Safe-Error Attack (2/2)

Against ECIES

- **Timely** induce a fault into the ALU during the add operation at iteration $i$
- Check the output
  - if an invalid ciphertext is notified (i.e., ⊥) then the error was effective
    $\Rightarrow d_i = 1$
  - if the result is correct then the point addition was dummy [safe error]
    $\Rightarrow d_i = 0$
- Re-iterate the attack for another value of $i$
Safe-Error Attack (2/2)

Against ECIES

- **Timely** induce a fault into the ALU during the add operation at iteration $i$
- Check the output
  - if an invalid ciphertext is notified (i.e., $\bot$) then the error was effective
    \[ \Rightarrow d_i = 1 \]
  - if the result is correct then the point addition was dummy [safe error]
    \[ \Rightarrow d_i = 0 \]
- Re-iterate the attack for another value of $i$
Errors in Public Routines

- Digital signatures are often used for authentication purposes
  - e.g., only signed software can run on a given device
- Idea: inject a fault during the verification process

Public routines (parameters) should be checked for faults
Errors in Public Routines

- Digital signatures are often used for authentication purposes
  - e.g., only signed software can run on a given device
- Idea: inject a fault during the verification process

Public routines (parameters) should be checked for faults
Random Errors Against EC Primitive

Attack model
- EC parameters are in non-volatile memory
  - permanent faults in a unknown position, in any system parameter
  - transient fault during parameter transfer

Adversary’s goal
- Recover the value of $d$ in the computation of $Q = [d]P$
Key Observation (1/2)

\[ E : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6 \]

- Let \( P = (x_1, y_1) \) and \( Q = (x_2, y_2) \)
- \( P + Q = (x_3, y_3) \) where
  \[
  x_3 = \lambda^2 + a_1 \lambda - a_2 - x_1 - x_2, \quad y_3 = (x_1 - x_3) \lambda - y_1 - a_1 x_3 - a_3
  \]

  with \( \lambda = \begin{cases} 
  \frac{y_1 - y_2}{x_1 - x_2} & \text{[addition]} \\
  \frac{3x_1^2 + 2a_2 x_1 + a_4 - a_1 y_1}{2y_1 + a_1 x_1 + a_3} & \text{[doubling]}
  \end{cases} \)

- Parameter \( a_6 \) is not involved in point addition (or point doubling)
Key Observation (2/2)

\[ E : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6 \]

If a ‘point’ \( \tilde{P} = (\tilde{x}, \tilde{y}) \in \mathbb{F}_q \times \mathbb{F}_q \) but \( \tilde{P} \notin E \) then the computation of \( \tilde{Q} = [d] \tilde{P} \) will take place on the curve

\[ \tilde{E} : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + \tilde{a}_6 \]

where \( \tilde{a}_6 = \tilde{y}^2 + a_1 \tilde{x} \tilde{y} + a_3 \tilde{y} - \tilde{x}^3 - a_2 \tilde{x}^2 - a_4 \tilde{x} \)

Now if

1. \( \text{ord}_{\tilde{E}}(\tilde{P}) = t \) is small
2. discrete logarithms are computable in \( \langle \tilde{P} \rangle \)

then

\[ d \pmod{t} \]

can be recovered from \( \tilde{Q} \)
Chosen Input Point Attack

Construct a ‘point’ \( \tilde{P}_i = (\tilde{x}_i, \tilde{y}_i) \in \tilde{E}_i \) such that

1. \( \text{ord}_{\tilde{E}_i}(\tilde{P}_i) = t_i \) is small
2. discrete logarithms are computable in \( \langle \tilde{P}_i \rangle \)

Query the device with \( \tilde{P}_i \) and receive \( \tilde{Q}_i = [d]\tilde{P}_i \)

Solve the discrete logarithm and recover \( d \pmod{t_i} \)

Iterating the process gives

- \( d \pmod{t_i} \) for several \( t_i \)
- \( d \) by Chinese remaindering

(This attack can easily be prevented using the curve equation)
Faults in the Base Point

Recover $d$ in $Q = [d]P$ on $E_{/\mathbb{F}_p} : y^2 = x^3 + a_4x + a_6$

- Fault: $P = (x_1, y_1) \rightarrow \hat{P} = (\hat{x}_1, y_1) \in \tilde{E}$
- Device outputs $\hat{Q} = [d]\hat{P}$
- $\hat{Q} = [d](\hat{x}_1, y_1) = (\hat{x}_d, \hat{y}_d) \in \tilde{E}$
  $\Rightarrow \hat{a}_6 = \hat{y}_d^2 - \hat{x}_d^3 - a_4\hat{x}_d \pmod{p}$
- $\hat{x}_1$ is a root in $\mathbb{F}_p[X]$ of $X^3 + a_4X + \hat{a}_6 - y_1^2$
- Compute $d \pmod{t}$ from $\hat{Q} = [d]\hat{P}$

- Similar attack when the $y$-coordinate of $P$ is corrupted
- More assumptions are needed when both coordinates are corrupted
Faults in the Base Point

Recover $d$ in $Q = [d]P$ on $E_{/F_p} : y^2 = x^3 + a_4x + a_6$

- **Fault:** $P = (x_1, y_1) \rightarrow \hat{P} = (\hat{x}_1, y_1) \in \tilde{E}$
- Device outputs $\hat{Q} = [d]\hat{P}$
- $\hat{Q} = [d](\hat{x}_1, y_1) = (\hat{x}_d, \hat{y}_d) \in \tilde{E}$
  $\Rightarrow \tilde{a}_6 = \hat{y}_d^2 - \hat{x}_d^3 - a_4\hat{x}_d \pmod{p}$
- $\hat{x}_1$ is a root in $F_p[X]$ of $X^3 + a_4X + \tilde{a}_6 - y_1^2$
- Compute $d \pmod{t}$ from $\hat{Q} = [d]\hat{P}$

- Similar attack when the $y$-coordinate of $P$ is corrupted
- More assumptions are needed when both coordinates are corrupted
Faults in the Base Point

Recover $d$ in $Q = [d]P$ on $E_{/\mathbb{F}_p} : y^2 = x^3 + a_4x + a_6$

- **Fault:** $P = (x_1, y_1) \rightarrow \hat{P} = (\hat{x}_1, y_1) \in \tilde{E}$
- **Device outputs** $\hat{Q} = [d]\hat{P}$
  - $\hat{Q} = [d](\hat{x}_1, y_1) = (\hat{x}_d, \hat{y}_d) \in \tilde{E}$
  - $\Rightarrow \hat{a}_6 = \hat{y}_d^2 - \hat{x}_d^3 - a_4\hat{x}_d \pmod{p}$
- $\hat{x}_1$ is a root in $\mathbb{F}_p[X]$ of $X^3 + a_4X + \hat{a}_6 - y_1^2$
- **Compute** $d \pmod{t}$ from $\hat{Q} = [d]\hat{P}$

- Similar attack when the $y$-coordinate of $P$ is corrupted
- More assumptions are needed when both coordinates are corrupted
Faults in the Base Point

Recover \(d\) in \(Q = [d]P\) on \(E_{/\mathbb{F}_p} : y^2 = x^3 + a_4 x + a_6\)

- **Fault:** \(P = (x_1, y_1) \rightarrow \hat{P} = (\hat{x}_1, y_1) \in \tilde{E}\)
- **Device outputs** \(\hat{Q} = [d]\hat{P}\)
- \(\hat{Q} = [d](\hat{x}_1, y_1) = (\hat{x}_d, \hat{y}_d) \in \tilde{E}\)
  \[\Rightarrow \hat{a}_6 = \hat{y}_d^2 - \hat{x}_d^3 - a_4 \hat{x}_d \pmod{p}\]
- \(\hat{x}_1\) is a root in \(\mathbb{F}_p[X]\) of \(X^3 + a_4 X + \hat{a}_6 - y_1^2\)
- **Compute** \(d \pmod{t}\) from \(\hat{Q} = [d]\hat{P}\)

- Similar attack when the \(y\)-coordinate of \(P\) is corrupted
- More assumptions are needed when both coordinates are corrupted
Faults in the Base Point

Recover $d$ in $Q = [d]P$ on $E_{/\mathbb{F}_p}: y^2 = x^3 + a_4x + a_6$

- Fault: $P = (x_1, y_1) \rightarrow \hat{P} = (\hat{x}_1, y_1) \in \tilde{E}$
- Device outputs $\hat{Q} = [d]\hat{P}$
- $\hat{Q} = [d](\hat{x}_1, y_1) = (\hat{x}_d, \hat{y}_d) \in \tilde{E}$
  \[ \Rightarrow \hat{a}_6 = \hat{y}_d^2 - \hat{x}_d^3 - a_4\hat{x}_d \pmod{p} \]
- $\hat{x}_1$ is a root in $\mathbb{F}_p[X]$ of $X^3 + a_4X + \hat{a}_6 - y_1^2$
- Compute $d \pmod{t}$ from $\hat{Q} = [d]\hat{P}$

- Similar attack when the $y$-coordinate of $P$ is corrupted
- More assumptions are needed when both coordinates are corrupted
Faults in the Base Point

Recover $d$ in $Q = [d]P$ on $E_{/\mathbb{F}_p} : y^2 = x^3 + a_4x + a_6$

- Fault: $P = (x_1, y_1) \rightarrow \hat{P} = (\hat{x}_1, y_1) \in \tilde{E}$
- Device outputs $\hat{Q} = [d]\hat{P}$
- $\hat{Q} = [d](\hat{x}_1, y_1) = (\hat{x}_d, \hat{y}_d) \in \tilde{E}$
  $\Rightarrow \tilde{a}_6 = \hat{y}_d^2 - \hat{x}_d^3 - a_4\hat{x}_d \pmod{p}$
- $\hat{x}_1$ is a root in $\mathbb{F}_p[X]$ of $X^3 + a_4X + \tilde{a}_6 - y_1^2$
- Compute $d \pmod{t}$ from $\hat{Q} = [d]\hat{P}$

- Similar attack when the $y$-coordinate of $P$ is corrupted
- More assumptions are needed when both coordinates are corrupted
Faults in the Base Point

Recover $d$ in $Q = [d]P$ on $E/\mathbb{F}_p : y^2 = x^3 + a_4x + a_6$

- Fault: $P = (x_1, y_1) \rightarrow \hat{P} = (\hat{x}_1, y_1) \in \tilde{E}$
- Device outputs $\hat{Q} = [d]\hat{P}$
- $\hat{Q} = [d](\hat{x}_1, y_1) = (\hat{x}_d, \hat{y}_d) \in \tilde{E}$
  \[ \Rightarrow \hat{a}_6 = \hat{y}_d^2 - \hat{x}_d^3 - a_4\hat{x}_d \pmod{p} \]
- $\hat{x}_1$ is a root in $\mathbb{F}_p[X]$ of $X^3 + a_4X + \hat{a}_6 - y_1^2$
- Compute $d \pmod{t}$ from $\hat{Q} = [d]\hat{P}$

- Similar attack when the $y$-coordinate of $P$ is corrupted
- More assumptions are needed when both coordinates are corrupted
Faults in the Definition Field

Recover $d$ in $Q = [d]P$ on $E_{\mathbb{F}_p} : y^2 = x^3 + a_4x + a_6$

- Fault: $p \rightarrow \hat{p}$
- Device outputs $\hat{Q} = [d]\hat{P}$ with $\hat{P} = (\hat{x}_1, \hat{y}_1)$ and $\hat{x}_1 \equiv x_1 \pmod{\hat{p}}$ and $\hat{y}_1 \equiv y_1 \pmod{\hat{p}}$
- $\hat{Q} = [d](\hat{x}_1, y_1) = (\hat{x}_d, \hat{y}_d) \in \tilde{E}$
  \[ \Rightarrow \hat{a}_6 \equiv \hat{y}_d^2 - \hat{x}_d^3 - a_4\hat{x}_d \equiv \hat{y}_1^2 - \hat{x}_1^3 - a_4\hat{x}_1 \pmod{\hat{p}} \]
- $\hat{p}$ divides $(\hat{y}_d^2 - \hat{x}_d^3 - a_4\hat{x}_d) - (\hat{y}_1^2 - \hat{x}_1^3 - a_4\hat{x}_1)$
- Compute $d \pmod{t}$ from $\hat{Q} = [d]\hat{P}$

- Case where $p$ is a Mersenne prime; i.e., $p = 2^m \pm 2^t \pm 1$
Faults in the Curve Parameters

Recover $d$ in $Q = [d]P$ on $E_{\mathbb{F}_p} : y^2 = x^3 + a_4 x + a_6$

- Fault: $a_4 \rightarrow \hat{a}_4$
- Device outputs $\hat{Q} = [d]P$ on $\hat{E} : y^2 = x^3 + \hat{a}_4 x + \tilde{a}_6$
- $\hat{Q} = [d](x_1, y_1) = (\hat{x}_d, \hat{y}_d) \in \hat{E}$
- Two equations:

$$\begin{align*}
y_1^2 &= x_1^3 + \hat{a}_4 x_1 + \tilde{a}_6 \\
\hat{y}_d^2 &= \hat{x}_d^3 + \hat{a}_4 \hat{x}_d + \tilde{a}_6
\end{align*}$$

$\Rightarrow \hat{a}_4 = \ldots, \tilde{a}_6 = \ldots$

- Compute $d \pmod{t}$ from $\hat{Q} = [d]P$
Skipping Attack

Attack assumes that the attacker manages to skip a doubling operation. This can be seen as a random error at the bit level.

Algorithm 1 Double-and-add

Input: $G, k = (k_{\ell-1}, \ldots, k_0)_2$
Output: $Q = [k]G$

1. $R_0 \leftarrow O; R_1 \leftarrow G$
2. for $i = \ell - 1$ down to 0 do
3.   $R_0 \leftarrow [2]R_0$
4.   if $k_i = 1$ then $R_0 \leftarrow R_0 + R_1$
5. return $R_0$
Skipping Attack

Attack assumes that the attacker manages to skip a doubling operation, which can be seen as a random error at the bit level.

Algorithm 2 Double-and-add

Input: $G$, $k = (k_{\ell-1}, \ldots, k_0)_2$
Output: $Q = [k]G$

1. $R_0 \leftarrow O$; $R_1 \leftarrow G$
2. for $i = \ell - 1$ down to 0 do
3. \hspace{1em} $R_0 \leftarrow [2]R_0$
4. \hspace{1em} if $k_i = 1$ then $R_0 \leftarrow R_0 + R_1$
5. return $R_0$
Application to ECDSA

- doubling skipped at iteration $j$
- $T \rightsquigarrow \hat{T}$ where

$$\hat{T} = \sum_{i=j+1}^{\ell-1} [k_i 2^{i-1}]G + \sum_{i=0}^{j} [k_i 2^i]G$$

$$= \left[\frac{1}{2}\right] (T + [\tilde{k}]G)$$

with $\tilde{k} = (k_j, \ldots, k_0)_2$
- $(r, s) \rightsquigarrow (\hat{r}, \hat{s})$

Algorithm 3 Double-and-add

**Input:** $G$, $k = (k_{\ell-1}, \ldots, k_0)_2$

**Output:** $T = [k]G$

1. $R_0 \leftarrow O$; $R_1 \leftarrow G$
2. for $i = \ell - 1$ down to 0 do
3. $R_0 \leftarrow [2]R_0$
4. if $k_i = 1$ then $R_0 \leftarrow R_0 + R_1$
5. return $R_0$

Observation:

$$[\hat{u}_1]G + [\hat{u}_2]Y = \left[\frac{H(m)}{5}\right]G + [\hat{r}]Y = \left[\frac{H(m) + d\hat{r}}{5}\right]G = [k]G$$

$$\hat{r} \equiv x\left(\left[\frac{1}{2}\right] (T + [\tilde{k}]G)\right) \pmod{n} \quad \text{with} \quad T = [\hat{u}_1]G + [\hat{u}_2]Y \Rightarrow \tilde{k} = \ldots$$
Application to ECDSA

- doubling skipped at iteration $j$
- $T \leadsto \hat{T}$ where

$$
\hat{T} = \sum_{i=j+1}^{\ell-1} [k_i 2^{i-1}]G + \sum_{i=0}^{j} [k_i 2^i]G
$$

with $\tilde{k} = (k_j, \ldots, k_0)_2$
- $(r, s) \leadsto (\hat{r}, \hat{s})$

Algorithm 4 Double-and-add

Input: $G, k = (k_{\ell-1}, \ldots, k_0)_2$
Output: $T = [k]G$

1. $R_0 \leftarrow O; R_1 \leftarrow G$
2. for $i = \ell - 1$ down to 0 do
3. \hspace{1em} $R_0 \leftarrow [2]R_0$
4. \hspace{1em} if $k_i = 1$ then $R_0 \leftarrow R_0 + R_1$
5. \hspace{1em} return $R_0$

Observation:

$$
[\hat{u}_1]G + [\hat{u}_2]Y =\left[\frac{H(m)}{\hat{s}}\right]G + [\hat{r}]Y = \left[\frac{H(m) + dr\hat{r}}{\hat{s}}\right]G = [k]G
$$

$$\hat{r} \equiv x\left([\frac{1}{2}](T + [\tilde{k}]G)\right) \pmod{n} \quad \text{with} \quad T = [\hat{u}_1]G + [\hat{u}_2]Y \quad \implies \quad \tilde{k} = \ldots$$
Application to ECDSA

- doubling skipped at iteration $j$
- $T \rightarrow \hat{T}$ where

\[
\hat{T} = \sum_{i=j+1}^{\ell-1} [k_i 2^{i-1}]G + \sum_{i=0}^{j} [k_i 2^i]G
\]

with $\tilde{k} = (k_j, \ldots, k_0)_2$
- $(r, s) \rightarrow (\hat{r}, \hat{s})$

---

Algorithm 5 Double-and-add

Input: $G$, $k = (k_{\ell-1}, \ldots, k_0)_2$
Output: $T = [k]G$

1: $R_0 \leftarrow O; R_1 \leftarrow G$
2: for $i = \ell - 1$ down to 0 do
3: \hspace{1em} $R_0 \leftarrow [2]R_0$
4: \hspace{1em} if $k_i = 1$ then $R_0 \leftarrow R_0 + R_1$
5: \hspace{1em} return $R_0$

Observation:

\[
\begin{align*}
[\hat{u}_1]G + [\hat{u}_2]Y &= \left[\frac{H(m)}{s}\right]G + \left[\frac{\hat{r}}{s}\right]Y \\
&= \left[\frac{H(m)+d\hat{r}}{s}\right]G = [k]G
\end{align*}
\]

$\hat{r} \equiv x\left(\left[\frac{1}{2}\right](T + [\tilde{k}]G)\right) \pmod{n}$ with $T = [\hat{u}_1]G + [\hat{u}_2]Y \implies \tilde{k} = \ldots$
Outline

1. Elliptic Curves
   - Basics on elliptic curves
   - Elliptic curve digital signature algorithm
   - Other algorithms

2. Attacks
   - Single-bit errors
   - Safe errors
   - Random errors
   - Skipping attacks

3. Countermeasures
   - Basic countermeasures
   - Scalar randomization
   - BOS^+ algorithm
   - New algorithm

4. Conclusion
   - Research problems
Countermeasures

- Algorithmic countermeasures
  - memory checks, randomization, duplication, verification
  - Shamir’s trick (redundancy)
  - [rich] mathematical structure
- Basic vs. concrete systems
- Fixed vs. variable base point
- Infective computation
- BOS+ algorithm
Basic Countermeasures

- Add CRC checks
  - for private and public parameters
- Randomize the computation
  - e.g., \( d \leftarrow d + rn \) with \( n = \text{ord}_E(P) \)
- Compute the operations twice
  - doubles the running time
- Verify the signatures
  - ECDSA verification is slower than signing
- Check that the output point \( Q = [k]P \) is in \( \langle P \rangle \)
  - \( Q \in E \)
  - \( [h]Q \neq O \) (only implies of large order)
Basic Countermeasures

- Add CRC checks
  - for private and public parameters
- Randomize the computation
  - e.g., $d \leftarrow d + r n$ with $n = \text{ord}_E(P)$
- Compute the operations twice
  - doubles the running time
- Verify the signatures
  - ECDSA verification is slower than signing
- Check that the output point $Q = [k]P$ is in $\langle P \rangle$
  - $Q \in E$
  - $[h]Q \neq O$ (only implies of large order)
Basic Countermeasures

- Add CRC checks
  - for private and public parameters
- Randomize the computation
  - e.g., $d \leftarrow d + r \cdot n$ with $n = \text{ord}_E(P)$
- Compute the operations twice
  - doubles the running time
- Verify the signatures
  - ECDSA verification is slower than signing
- Check that the output point $Q = [k]P$ is in $\langle P \rangle$
  - $Q \in E$
  - $[h]Q \neq O$ (only implies of large order)
Basic Countermeasures

- Add CRC checks
  - for private and public parameters
- Randomize the computation
  - e.g., $d \leftarrow d + r \cdot n$ with $n = \text{ord}_E(P)$
- Compute the operations twice
  - doubles the running time
- Verify the signatures
  - ECDSA verification is slower than signing
- Check that the output point $Q = [k]P$ is in $\langle P \rangle$
  - $Q \in E$
  - $[h]Q \neq O$ (only implies of large order)
Basic Countermeasures

- Add CRC checks
  - for private and public parameters
- Randomize the computation
  - e.g., \( d \leftarrow d + r n \) with \( n = \text{ord}_E(P) \)
- Compute the operations twice
  - doubles the running time
- Verify the signatures
  - ECDSA verification is slower than signing
- Check that the output point \( Q = [k]P \) is in \( \langle P \rangle \)
  - \( Q \in E \)
  - \( [h]Q \neq O \) (only implies of large order)
Multiplier Randomization (1/2)

- Scalar $d$ should be randomized
- $d^* \leftarrow d + r \#E$ may not be a good solution
  - security issue

**Example (secp160k1)**

$p = 2^{160} - 2^{32} - 538D_{16}$  
$\#E = 01 \ 00000000 \ 00000000 \ 0001B8FA \ 16DFAB9A \ CA16B6B3_{16}$

$\Rightarrow d^* = d + r \#E = (r)_2 || d_{\ell-1} \cdots d_{\ell-t} ||$ some bits
Multiplier Randomization (1/2)

- Scalar $d$ should be randomized
- $d^* \leftarrow d + r \#E$ may not be a good solution
  - security issue

Example (secp160k1)

\[
p = 2^{160} - 2^{32} - 538_{16}
\]

[generalized] Mersenne prime

\[
\#E = 01\ 00000000\ 00000000\ 0001B8FA\ 16DFAB9A\ CA16B6B3_{16}
\]

\[
\Rightarrow d^* = d + r \#E = (r)_{2} \parallel d_{\ell-1} \cdots d_{\ell-t} \parallel \text{some bits}
\]
Multiplier Randomization (1/2)

- Scalar $d$ should be randomized
- $d^* \leftarrow d + r \#E$ may not be a good solution
  - security issue

### Example (secp160k1)

<table>
<thead>
<tr>
<th>$p = 2^{160} - 2^{32} - 538_{16}$</th>
<th>[generalized] Mersenne prime</th>
</tr>
</thead>
<tbody>
<tr>
<td>$#E = 01\ 00000000\ 00000000\ 0001B8FA\ 16DFAB9A\ CA16B6B3_{16}$</td>
<td></td>
</tr>
</tbody>
</table>

\[ \Rightarrow d^* = d + r \#E = (r)_2 \| d_{\ell-1} \cdots d_{\ell-t} \| \text{some bits} \]
Multiplier Randomization (2/2)

- Use splitting methods
  - additive:
    \[ [d]P = [d - r]P + [r]P \]
  - multiplicative:
    \[ [d]P = [d \cdot r^{-1}]([r]P) \]

```plaintext
Euclidean splitting
Write \( d = \lfloor d/r \rfloor r + (d \mod r) \) for a random \( r \)

\[ [d]P = [d \mod r]P + \lfloor \lfloor d/r \rfloor \rfloor ([r]P) \]
```
Multiplier Randomization (2/2)

- Use **splitting** methods
  - additive:
    \[[d]P = [d - r]P + [r]P\]
  - multiplicative:
    \[[d]P = [d r^{-1}]([r]P)\]

**Euclidean splitting**

Write \(d = \lfloor d/r \rfloor r + (d \mod r)\) for a random \(r\)

\[\Rightarrow [d]P = [d \mod r]P + \lceil [d/r] \rceil ([r]P)\]
Multiplier Randomization (2/2)

- Use splitting methods
  - additive:
    \[
    \lfloor d \rfloor P = \lfloor d - r \rfloor P + \lfloor r \rfloor P
    \]
  - multiplicative:
    \[
    \lfloor d \rfloor P = \lfloor d r^{-1} \rfloor (\lfloor r \rfloor P)
    \]

**Euclidean splitting**

Write \( d = \lfloor d/r \rfloor r + (d \mod r) \) for a random \( r \)

\[
\Rightarrow \lfloor d \rfloor P = \lfloor d \mod r \rfloor P + \lfloor \lfloor d/r \rfloor \rfloor (\lfloor r \rfloor P)
\]

Strauss-Shamir double ladder
Multiplier Randomization (2/2)

- Use splitting methods
  - additive:
    \[ [d]P = [d - r]P + [r]P \]
  - multiplicative:
    \[ [d]P = [d \cdot r^{-1}]([r]P) \]

Euclidean splitting

Write \( d = \lfloor d/r \rfloor r + (d \mod r) \) for a random \( r \)

\[ \implies [d]P = [d \mod r]P + [\lfloor d/r \rfloor]([r]P) \]

- Strauss-Shamir double ladder
Multiplier Randomization (2/2)

- Use **splitting** methods
  - additive:
    \[ [d]P = [d - r]P + [r]P \]
  - multiplicative:
    \[ [d]P = [d r^{-1}]([r]P) \]

### Euclidean splitting

Write \( d = \lfloor d/r \rfloor r + (d \mod r) \) for a random \( r \)

\[ \implies [d]P = [d \mod r]P + \lfloor [d/r] \rfloor ([r]P) \]

- Strauss-Shamir double ladder
Preventing Fault Attacks: The Case of RSA

Shamir’s countermeasure

1. Choose a (small) random integer \( r \)
2. Compute \( S^* = \hat{m}^d \mod rN \) and \( Z = \hat{m}^d \mod r \)
3. If \( S^* \equiv Z \pmod{r} \) then output \( S = S^* \mod N \), otherwise return error

Giraud’s countermeasure

1. Compute \( \hat{m}^d \mod N \) using Montgomery ladder and obtain the pair \((Z, S) = (\hat{m}^{d-1} \mod N, \hat{m}^d \mod N)\)
2. If \( Z \hat{m} \equiv S \pmod{N} \) then output \( S \), otherwise return error
Preventing Fault Attacks: The Case of RSA

Shamir’s countermeasure

1. Choose a (small) random integer \( r \)
2. Compute \( S^* = \hat{m}^d \mod rN \) and \( Z = \hat{m}^d \mod r \)
3. If \( S^* \equiv Z \pmod{r} \) then output \( S = S^* \mod N \), otherwise return error

Giraud’s countermeasure

1. Compute \( \hat{m}^d \mod N \) using Montgomery ladder and obtain the pair \((Z, S) = (\hat{m}^{d-1} \mod N, \hat{m}^d \mod N)\)
2. If \( Z \hat{m} \equiv S \pmod{N} \) then output \( S \), otherwise return error
Infective Computation

- Reminder:
  - Decisional tests should be avoided
  - Inducing a random fault in the status register flips the value of the zero flag bit with a probability of 50%

**Infective computation**

Make the decisional tests implicit and “infect” the computation in case of error detection

Example:

If \( T[a] = b \) then return a else error

\[ \Rightarrow \text{Return} \ (T[a] - b) \cdot r + a \]
Infective Computation

■ Reminder:
  ■ Decisional tests should be avoided
  ■ Inducing a random fault in the status register flips the value of the zero flag bit with a probability of 50%

Infective computation

Make the decisional tests implicit and “infect” the computation in case of error detection

Example:

\[ \text{If } (T[a] = b) \text{ then return } a \text{ else error} \]
\[ \Rightarrow \text{Return } (T[a] - b) \cdot r + a \]
Infective Computation

Reminder:

- Decisional tests should be avoided
- Inducing a random fault in the status register flips the value of the zero flag bit with a probability of 50%

Infective computation

Make the decisional tests implicit and “infect” the computation in case of error detection

Example:

\[ \text{If } (T[a] = b) \text{ then return } a \text{ else error} \]
\[ \Rightarrow \text{Return } (T[a] - b) \cdot r + a \]
Edwards Curves

\[ E_{/\mathbb{F}_p} : ax^2 + y^2 = 1 + bx^2y^2 \quad \text{where } ab(a - b) \neq 0 \]

- **Addition law**
  - \( O = (0, 1) \) [neutral element]
  - \(- (x_1, y_1) = (-x_1, y_1)\)
  - \((x_1, y_1) + (x_2, y_2) = (x_3, y_3) \) where
    \[
    x_3 = \frac{x_1y_2 + x_2y_1}{1 + bx_1x_2y_1y_2}, \quad y_3 = \frac{y_1y_2 - ax_1x_2}{1 - bx_1x_2y_1y_2}
    \]
  - \ldots also valid for point doubling (and \( O \))

- Addition law is **complete** if \( a \) is a square and \( b \) is a non-square
Shamir’s Trick for Elliptic Curve Cryptosystems

Let $\mathcal{R} = \mathbb{Z}/pr\mathbb{Z}$ for a (small) random prime $r$

1. Compute

   - $Q^* \leftarrow [d]P \in \mathcal{E}_{pr}(\mathbb{Z}/pr\mathbb{Z})$
   - $Y \leftarrow [d]P \in \mathcal{E}(\mathbb{F}_r)$

2. If $(Q^* \not\equiv Y \pmod{r})$ then return error

3. Return $Q^* \pmod{p}$
Shamir’s Trick for Elliptic Curve Cryptosystems

\[ P = (x_1, y_1) \in \mathcal{E}/\mathbb{F}_p : ax^2 + y^2 = 1 + bx^2y^2 \]

- Let \( \mathcal{R} = \mathbb{Z}/pr\mathbb{Z} \) for a (small) random prime \( r \)
  
  1. Compute
     - \( \mathcal{E}_{pr} \leftarrow \text{CRT}(\mathcal{E}, \mathcal{E}_r) \) where \( \mathcal{E}_{r/\mathbb{F}_r} : ax^2 + y^2 = 1 + brx^2y^2 \)
     - \( Q^* \leftarrow [d]P \in \mathcal{E}_{pr}(\mathbb{Z}/pr\mathbb{Z}) \)
     - \( Y \leftarrow [d]P_r \in \mathcal{E}_r(\mathbb{F}_r) \)
  
  2. If \( (Q^* \neq Y \mod r) \) then return error
  
  3. Return \( Q^* \mod p \)

**Idea #1**

Let \( b_r = (ax_1^2 + y_1^2 - 1)/(x_1^2y_1^2) \mod r \) so that \( P_r := P \mod r \in \mathcal{E}_r \)

- ... but completeness is not guaranteed (and \( \#\mathcal{E}_r \) is unknown)
Shamir’s Trick for Elliptic Curve Cryptosystems

$P = (x_1, y_1) \in \mathcal{E}_{/\mathbb{F}_p} : ax^2 + y^2 = 1 + bx^2y^2$

- Let $\mathcal{R} = \mathbb{Z}/pr\mathbb{Z}$ for a (small) random prime $r$
  1. Compute
     - $\mathcal{E}_{pr} \leftarrow \text{CRT}(\mathcal{E}, \mathcal{E}_r)$ and $P^* \leftarrow \text{CRT}(P, P_r)$
     - $Q^* \leftarrow [d]P^* \in \mathcal{E}_{pr}(\mathbb{Z}/pr\mathbb{Z})$
     - $Y \leftarrow [d \mod nr]P_r \in \mathcal{E}_r(\mathbb{F}_r)$
  2. If $(Q^* \not\equiv Y \mod r)$ then return error
  3. Return $Q^* \mod p$

Idea #2

Fix $E_r(\mathbb{F}_r) = \langle P_r \rangle$ so that addition is complete

- ... but $r$ is now a priori fixed and values must be pre-stored
BOS⁺ Algorithm

- Blömer, Otto, and Seifert (FDTC 2005)

Input: \( P \in \mathcal{E}, d \)
Output: \( Q = [d]P \)
In memory: \( \{\mathcal{E}_r, P_r \in \mathcal{E}_r, n_r = \#\mathcal{E}_r\} \)

1. Compute
   1. \( \mathcal{E}_{pr} \leftarrow \text{CRT}(\mathcal{E}, \mathcal{E}_r) \) and \( P^* \leftarrow \text{CRT}(P, P_r) \)
   2. \( Q^* \leftarrow [d]P^* \in \mathcal{E}_{pr} \)
   3. \( Y \leftarrow [d \pmod{n_r}]P_r \in \mathcal{E}_r \)

   \[
   \begin{align*}
   c_x &\leftarrow 1 + x_{pr} - x_r \pmod{r} \\
   c_y &\leftarrow 1 + y_{pr} - y_r \pmod{r} 
   \end{align*}
   \]

2. If \( Q^* \not\equiv Y \pmod{r} \) then return error
3. Return \( Q^* \pmod{p} \in \mathcal{E} \)
**BOS⁺ Algorithm**

Blömer, Otto, and Seifert (FDTC 2005)

---

**Input:** \( P \in \mathcal{E}, d \)

**Output:** \( Q = [d]P \)

**In memory:** \( \{\mathcal{E}_r, P_r \in \mathcal{E}_r, n_r = \#\mathcal{E}_r\} \)

1. **Compute**
   1. \( \mathcal{E}_{pr} \leftarrow \text{CRT}(\mathcal{E}, \mathcal{E}_r) \) and \( P^* \leftarrow \text{CRT}(P, P_r) \)
   2. \( Q^* \leftarrow [d]P^* \in \mathcal{E}_{pr} = (x_{pr}, y_{pr}) \)
   3. \( Y \leftarrow [d \pmod{n_r}]P_r \in \mathcal{E} = (x_r, y_r) \)
   4. \[ c_x \leftarrow 1 + x_{pr} - x_r \pmod{r} \]
      \[ c_y \leftarrow 1 + y_{pr} - y_r \pmod{r} \]

2. For a \( \kappa \)-bit random \( \rho \), compute \( \gamma \leftarrow \left\lfloor \frac{\rho c_x + (2^\kappa - \rho)c_y}{2^\kappa} \right\rfloor \)

3. Return \( Q = [\gamma]Q^* \pmod{p} \in \mathcal{E} \)
Shamir’s Trick for Elliptic Curve Cryptosystems ?!

\[ P = (x_1, y_1) \in \mathcal{E}/\mathbb{F}_p : ax^2 + y^2 = 1 + bx^2y^2 \]

1. Let \( R = \mathbb{Z}/pr\mathbb{Z} \) for a (small) random prime \( r \)
2. Compute
   - \( \mathcal{E}_{pr} \leftarrow \text{CRT}(\mathcal{E}, \mathcal{E}_r) \) and \( P^* \leftarrow \text{CRT}(P, P_r) \)
   - \( Q^* \leftarrow [d]P^* \in \mathcal{E}_{pr}(\mathbb{Z}/pr\mathbb{Z}) \)
   - \( Y \leftarrow [d \text{ (mod } n_r)]P_r \in \mathcal{E}_r(\mathbb{Z}/r\mathbb{Z}) \)
3. If \( Q^* \not\equiv Y \text{ (mod } r) \) then return error
4. Return \( Q^* \mod p \)

Idea #3 (???)

Choose \( \mathcal{E}_r(\mathbb{Z}/r\mathbb{Z}) = \langle P_r \rangle \), so that (i) addition is complete, (ii) \( n_r = \#\mathcal{E}_r \) is known, and (iii) no storage is required
New Algorithm

\[ \mathcal{E}_1(\mathbb{Z}/q^2\mathbb{Z}) = \{(\alpha q, 1) \mid \alpha \in \mathbb{Z}/q\mathbb{Z}\} \]

- Properties
  - \( \mathcal{E}_1 \simeq (\mathbb{Z}/q\mathbb{Z})^+, \ P_1 = (\alpha q, 1) \sim \alpha \)
  - \( \# \mathcal{E}_1 = q \)
  - \([d]P_1 = (dx_1, 1)\) where \( x_1 = \alpha q \)

- Addition law is complete

\[
(x_1, y_1) + (x_2, y_2) = \left( \frac{x_1 y_2 + x_2 y_1}{1 + bx_1 x_2 y_1 y_2}, \frac{y_1 y_2 - ax_1 x_2}{1 - bx_1 x_2 y_1 y_2} \right)
\]

whatever curve parameters \( a \) and \( b \)
New Algorithm

Input: \( P \in \mathcal{E}, d \)
Output: \( Q = [d]P \)

1. Choose a small random \( t \)
2. Define \( r \leftarrow t^2 \) and \( P_r \leftarrow (t, 1) \)
3. Compute
   - \( P^* \leftarrow \text{CRT}(P, P_r) \)
   - \( Q^* \leftarrow [d]P^* \in \mathcal{E}(\mathbb{Z}/pr\mathbb{Z}) \)
   - \( Y \leftarrow (dt \mod r, 1) \)
   - \[ \begin{align*}
   c_x &\leftarrow 1 + x_{pr} - x_r \pmod{r} \\
   c_y &\leftarrow y_{pr} \pmod{r}
   \end{align*} \]
4. If \( Q^* \not\equiv Y \pmod{r} \) then return error
5. Return \( Q^* \pmod{p} \in \mathcal{E}(\mathbb{F}_p) \)
New Algorithm

Input: \( P \in \mathcal{E}, d \)
Output: \( Q = [d]P \)

1. Choose a small random \( t \)
2. Define \( r \leftarrow t^2 \) and \( P_r \leftarrow (t, 1) \)
3. Compute
   
   \[ P^* \leftarrow \text{CRT}(P, P_r) \]
   \[ Q^* \leftarrow [d]P^* \in \mathcal{E}(\mathbb{Z}/pr\mathbb{Z}) \]
   \[ Y \leftarrow (dt \mod r, 1) \]
   \[ \begin{cases} 
     c_x \leftarrow 1 + x_{pr} - x_r \pmod{r} \\
     c_y \leftarrow y_{pr} \pmod{r} 
   \end{cases} \]
4. For a \( \kappa \)-bit random \( \rho \), compute \( \gamma \leftarrow \left\lfloor \frac{\rho c_x + (2^\kappa - \rho)c_y}{2^\kappa} \right\rfloor \)
5. Return \( Q = [\gamma]Q^* \pmod{p} \in \mathcal{E}(\mathbb{F}_p) \)
Outline

1. Elliptic Curves
   - Basics on elliptic curves
   - Elliptic curve digital signature algorithm
   - Other algorithms

2. Attacks
   - Single-bit errors
   - Safe errors
   - Random errors
   - Skipping attacks

3. Countermeasures
   - Basic countermeasures
   - Scalar randomization
   - BOS$^+$ algorithm
   - New algorithm

4. Conclusion
   - Research problems
Summary

- Always use ECC standards (ECDSA, ECIES, ECMQV)
- Protect private and public parameters
  - perform memory checks
- Protect public routines
- Avoid decisional tests and make use of infective computation
- Randomize the implementation
- Prefer the splitting methods
Summary

- Always use ECC standards (ECDSA, ECIES, ECMQV)
- Protect private and public parameters
  - perform memory checks
- Protect public routines
- Avoid decisional tests and make use of infective computation
- Randomize the implementation
- Prefer the splitting methods
Summary

- Always use ECC standards (ECDSA, ECIES, ECMQV)
- Protect private and public parameters
  - perform memory checks
- Protect public routines
  - Avoid decisional tests and make use of infective computation
  - Randomize the implementation
  - Prefer the splitting methods
Summary

- Always use ECC standards (ECDSA, ECIES, ECMQV)
- Protect private and public parameters
  - perform memory checks
- Protect public routines
- Avoid decisional tests and make use of infective computation
  - Randomize the implementation
  - Prefer the splitting methods
Summary

- Always use ECC standards (ECDSA, ECIES, ECMQV)
- Protect private and public parameters
  - perform memory checks
- Protect public routines
- Avoid decisional tests and make use of infective computation
- Randomize the implementation
- Prefer the splitting methods
Summary

- Always use ECC standards (ECDSA, ECIES, ECMQV)
- Protect private and public parameters
  - perform memory checks
- Protect public routines
- Avoid decisional tests and make use of infective computation
- Randomize the implementation
- Prefer the splitting methods
Further Research: Attacks
Further Research: Attacks

Research Problem #1

Mount fault attacks against randomized implementations of the EC primitive (e.g., using LLL)

Research Problem #2

Mount practical fault-attacks against elliptic curve schemes (i.e., beyond the primitive)

Research Problem #3

Combine classical attacks with fault attacks (i.e., exploit the extra info provided by the faults)
Further Research: Attacks

Research Problem #1

Mount fault attacks against randomized implementations of the EC primitive (e.g., using LLL)

Research Problem #2

Mount practical fault-attacks against elliptic curve schemes (i.e., beyond the primitive)

Research Problem #3

Combine classical attacks with fault attacks (i.e., exploit the extra info provided by the faults)
Further Research: Attacks

**Research Problem #1**
Mount fault attacks against randomized implementations of the EC primitive (e.g., using LLL)

**Research Problem #2**
Mount practical fault-attacks against elliptic curve schemes (i.e., beyond the primitive)

**Research Problem #3**
Combine classical attacks with fault attacks (i.e., exploit the extra info provided by the faults)
Further Research: Designs
Further Research: Designs

Research Problem #1

💡 Improve the **efficiency** of computations (speed-wise or memory-wise) and **security** — exploit the rich mathematical structure behind elliptic curves

Research Problem #2

💡 Explore scalar multiplication algorithms or design new ones having **invariants** (as in Giraud’s proposal)

Research Problem #3

💡 Develop countermeasures against **combined attacks** in an efficient way
Further Research: Designs

Research Problem #1

矿泉水 Improve the efficiency of computations (speed-wise or memory-wise) and security — exploit the rich mathematical structure behind elliptic curves

Research Problem #2

矿泉水矿泉水 Explore scalar multiplication algorithms or design new ones having invariants (as in Giraud’s proposal)

Research Problem #3

矿泉水矿泉水 Develop countermeasures against combined attacks in an efficient way
Further Research: Designs

Research Problem #1

🔥 Improve the **efficiency** of computations (speed-wise or memory-wise) and **security** — exploit the rich mathematical structure behind elliptic curves

Research Problem #2

🔥🔥 Explore scalar multiplication algorithms or design new ones having **invariants** (as in Giraud’s proposal)

Research Problem #3

🔥 Develop countermeasures against **combined attacks** in an efficient way
Comments/Questions?