Elliptic Curve Cryptosystems in the Presence of Faults

Marc Joye

technicolor

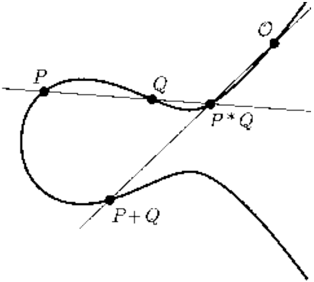
Elliptic Curve Cryptosystems in the Presence of Faults

Marc Joye

technicolor

Elliptic Curve Cryptography

- Invented [independently] by Neil Koblitz and Victor Miller in 1985

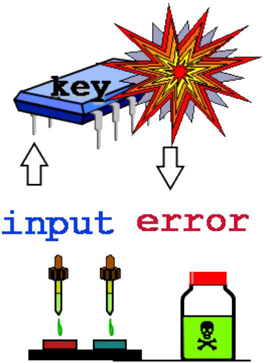


- Useful for key exchange, encryption and digital signature

technicolor

Fault Attacks

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



technicolor

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

Outline

technicolor

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 $\boldsymbol{O}$ at infinity

- Char $\mathbb{K} \neq 2, 3 \Rightarrow a_1 = a_2 = a_3 = 0$
- 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

- $\boldsymbol{O}$ 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 $\boldsymbol{O}$ at infinity

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

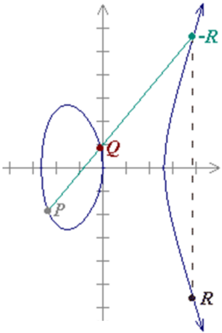
Fact

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

- $\boldsymbol{O}$ is the neutral element
- the group law is given by the “chord-and-tangent” rule

Basics on Elliptic Curves (2/3)

- Elliptic curves over $\mathbb{R}$



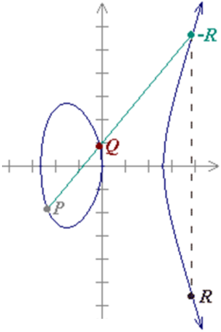
$$y^2 = x^3 - 7x$$

$\boldsymbol{P} = (-2.35, -1.86), \boldsymbol{Q} = (-0.1, 0.836)$
$\boldsymbol{R} = (3.89, -5.62)$

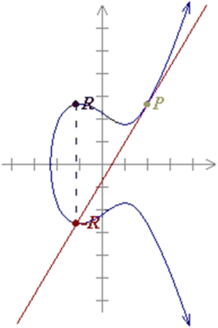
$\boldsymbol{P} = (2, 2.65)$
$\boldsymbol{R} = (1.11, 2.64)$

Basics on Elliptic Curves (2/3)

- Elliptic curves over $\mathbb{R}$



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



$$y^2 = x^3 - 3x + 5$$

$\boldsymbol{P} = (-2.35, -1.86)$, $\boldsymbol{Q} = (-0.1, 0.836)$
$\boldsymbol{R} = (3.89, -5.62)$

$\boldsymbol{P} = (2, 2.65)$
$\boldsymbol{R} = (1.11, 2.64)$

technicolor

Basics on Elliptic Curves (3/3)

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

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

$$x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2, \quad y_3 = (x_1 - x_3)\lambda - y_1 - a_1x_3 - a_3$$

$$\text{with } \lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{[addition]} \\ \dfrac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3} & \text{[doubling]} \end{cases}$$

technicolor

Basics on Elliptic Curves (3/3)

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

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

$$x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2, \quad y_3 = (x_1 - x_3)\lambda - y_1 - a_1x_3 - a_3$$

$$\text{with } \lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{[addition]} \\ \dfrac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3} & \text{[doubling]} \end{cases}$$

technicolor

Basics on Elliptic Curves (3/3)

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

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

$$x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2, \quad y_3 = (x_1 - x_3)\lambda - y_1 - a_1x_3 - a_3$$

$$\text{with } \lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{[addition]} \\ \dfrac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3} & \text{[doubling]} \end{cases}$$

technicolor

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 $\boldsymbol{P} = (x_1, y_1)$ and $\boldsymbol{Q} = (x_2, y_2)$
- Group law
  - $\boldsymbol{P} + \boldsymbol{O} = \boldsymbol{O} + \boldsymbol{P} = \boldsymbol{P}$
  - $-\boldsymbol{P} = (x_1, -y_1 - a_1\, x_1 - a_3)$
  - $\boldsymbol{P} + \boldsymbol{Q} = (x_3, y_3)$ where

$$x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2, \;\; y_3 = (x_1 - x_3)\lambda - y_1 - a_1 x_3 - a_3$$

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

technicolor

EC Primitive

- EC primitive = point multiplication (a.k.a. scalar multiplication)

$$E(\mathbb{K}) \times \mathbb{Z} \to E(\mathbb{K}), \quad (\boldsymbol{P}, d) \mapsto \boldsymbol{Q} = [d]\boldsymbol{P}$$

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

Definition (ECDL Problem)

Let $\mathbb{G} = \langle \boldsymbol{P} \rangle \subseteq E(\mathbb{K})$ a subgroup of prime order n
Given points $\boldsymbol{P}, \boldsymbol{Q} \in \mathbb{G}$, compute d such that $\boldsymbol{Q} = [d]\boldsymbol{P}$

technicolor

EC Digital Signature Algorithm (1/2)

- Elliptic curve variant of the Digital Signature Algorithm
  - a.k.a. Digital Signature Standard – DSS
  - included in IEEE P1363, ANSI X9.62, FIPS 186.2, SECG, and ISO 15946-2
- Domain parameters
  - finite field $\mathbb{F}_q$
  - elliptic curve $E/\mathbb{F}_q$ with $\#E(\mathbb{F}_q) = h\,n$
    - cofactor $h \leqslant 4$ and n prime
  - cryptographic hash function H
  - point $\boldsymbol{G} \in E$ of prime order n

$$\{\mathbb{F}_q, E, n, h, H, \boldsymbol{G}\}$$

technicolor

EC Digital Signature Algorithm (1/2)

- Elliptic curve variant of the Digital Signature Algorithm
  - a.k.a. Digital Signature Standard – DSS
  - included in IEEE P1363, ANSI X9.62, FIPS 186.2, SECG, and ISO 15946-2
- Domain parameters
  - finite field $\mathbb{F}_q$
  - elliptic curve $E/\mathbb{F}_q$ with $\#E(\mathbb{F}_q) = h\,n$
    - cofactor $h \leqslant 4$ and n prime
  - cryptographic hash function H
  - point $\boldsymbol{G} \in E$ of prime order n

$$\{\mathbb{F}_q, E, n, h, H, \boldsymbol{G}\}$$

technicolor

EC Digital Signature Algorithm (2/2)

- Key generation: $\boldsymbol{Y} = [d]\boldsymbol{G}$ with $d \xleftarrow{\$} \{1, \ldots, n-1\}$
  $pk = \{\boldsymbol{G}, \boldsymbol{Y}\}$ 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 $\boldsymbol{T} = [k]\boldsymbol{G}$ and set $r = \mathrm{x}(\boldsymbol{T}) \pmod{n}$
  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 $\boldsymbol{T} = [u_1]\boldsymbol{G} + [u_2]\boldsymbol{Y}$
  3. check whether $r \equiv \mathrm{x}(\boldsymbol{T}) \pmod{n}$

technicolor

EC Digital Signature Algorithm (2/2)

- Key generation: $\boldsymbol{Y} = [d]\boldsymbol{G}$ with $d \xleftarrow{\$} \{1, \ldots, n-1\}$
  $pk = \{\boldsymbol{G}, \boldsymbol{Y}\}$ 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 $\boldsymbol{T} = [k]\boldsymbol{G}$ and set $r = \mathrm{x}(\boldsymbol{T}) \pmod{n}$
  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 $\boldsymbol{T} = [u_1]\boldsymbol{G} + [u_2]\boldsymbol{Y}$
  3. check whether $r \equiv \mathrm{x}(\boldsymbol{T}) \pmod{n}$

EC Digital Signature Algorithm (2/2)

- Key generation: $\boldsymbol{Y} = [d]\boldsymbol{G}$ with $d \stackrel{\$}{\leftarrow} \{1, \ldots, n-1\}$
  $pk = \{\boldsymbol{G}, \boldsymbol{Y}\}$ 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 $\boldsymbol{T} = [k]\boldsymbol{G}$ and set $r = \mathrm{x}(\boldsymbol{T}) \pmod{n}$
  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 $\boldsymbol{T} = [u_1]\boldsymbol{G} + [u_2]\boldsymbol{Y}$
  3. check whether $r \equiv \mathrm{x}(\boldsymbol{T}) \pmod{n}$

technicolor

Public Key Validation

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

technicolor

EC Diffie-Hellman Key Exchange

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

Alice		Bob
a	$\xrightarrow{R_A=[a]G}$	R_A
R_B	$\xleftarrow{R_B=[b]G}$	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

Alice		Bob
a	$\xrightarrow{R_A = [a]G}$	R_A
R_B	$\xleftarrow{R_B = [b]G}$	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

Alice		Bob
a	$\xrightarrow{\mathbf{R_A}=[a]\mathbf{G}}$	$\mathbf{R_A}$
$\mathbf{R_B}$	$\xleftarrow{\mathbf{R_B}=[b]\mathbf{G}}$	b
$\mathbf{K_A} = [a]\mathbf{R_B}$		$\mathbf{K_B} = [b]\mathbf{R_A}$

  - 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

technicolor

EC Diffie-Hellman Key Exchange

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

Alice		Bob
a	$\xrightarrow{\boldsymbol{R_A}=[a]\boldsymbol{G}}$	$\boldsymbol{R_A}$
$\boldsymbol{R_B}$	$\xleftarrow{\boldsymbol{R_B}=[b]\boldsymbol{G}}$	b
$\boldsymbol{K_A} = [a]\boldsymbol{R_B}$		$\boldsymbol{K_B} = [b]\boldsymbol{R_A}$

  - 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

technicolor

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 $\#E(\mathbb{F}_q) = h\,n$
  - "special" hash functions
    - message authentication code $\mathrm{MAC}_K(c)$
    - key derivation function $\mathrm{KD}(\boldsymbol{T}, \ell)$
  - symmetric encryption algorithm $\mathtt{Enc}_K(m)$
  - point $\boldsymbol{G} \in E$ of prime order n

$$\{\mathbb{F}_q, E, n, h, \mathrm{MAC}, \mathrm{KD}, \mathtt{Enc}, \boldsymbol{G}\}$$

technicolor

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 $\#E(\mathbb{F}_q) = h\,n$
  - “special” hash functions
    - message authentication code $\mathsf{MAC}_K(c)$
    - key derivation function $\mathsf{KD}(\boldsymbol{T}, \ell)$
  - symmetric encryption algorithm $\mathtt{Enc}_K(m)$
  - point $\boldsymbol{G} \in E$ of prime order n

$$\{\mathbb{F}_q, E, n, h, \mathsf{MAC}, \mathsf{KD}, \mathtt{Enc}, \boldsymbol{G}\}$$

technicolor

ECDH Augmented Encryption (2/2)

- Key generation: $\boldsymbol{Y} = [d]\boldsymbol{G}$ with $d \xleftarrow{\$} \{1, \ldots, n-1\}$
  $pk = \{\boldsymbol{G}, \boldsymbol{Y}\}$ and $sk = \{d\}$
- ECIES encryption
  1. pick a random $k \in \{1, \ldots, n-1\}$
  2. compute $\boldsymbol{U} = [k]\boldsymbol{G}$ and $\boldsymbol{T} = [k]\boldsymbol{Y}$
  3. set $(K_1 \| K_2) = \mathrm{KD}(\boldsymbol{T}, l)$
  4. compute $c = \mathtt{Enc}_{K_1}(m)$ and $r = \mathrm{MAC}_{K_2}(c)$
  5. return $(\boldsymbol{U}, c, r)$

- ECIES decryption

  Input ciphertext $(\boldsymbol{U}, c, r)$ and private key sk

  Output plaintext m or $\perp$

  1. compute $\boldsymbol{T}' = [d]\boldsymbol{U}$
  2. set $(K_1' \| K_2') = \mathrm{KD}(\boldsymbol{T}', l)$
  3. if $\mathrm{MAC}_{K_2'}(c) = r$ then return $m = \mathtt{Enc}_{K_1'}^{-1}(c)$

ECDH Augmented Encryption (2/2)

- Key generation: $\boldsymbol{Y} = [d]\boldsymbol{G}$ with $d \xleftarrow{\$} \{1, \ldots, n-1\}$
  $pk = \{\boldsymbol{G}, \boldsymbol{Y}\}$ and $sk = \{d\}$
- ECIES encryption
  1. pick a random $k \in \{1, \ldots, n-1\}$
  2. compute $\boldsymbol{U} = [k]\boldsymbol{G}$ and $\boldsymbol{T} = [k]\boldsymbol{Y}$
  3. set $(K_1 \| K_2) = \mathrm{KD}(\boldsymbol{T}, l)$
  4. compute $c = \mathtt{Enc}_{K_1}(m)$ and $r = \mathrm{MAC}_{K_2}(c)$
  5. return $(\boldsymbol{U}, c, r)$

- ECIES decryption

  Input ciphertext $(\boldsymbol{U}, c, r)$ and private key sk

  Output plaintext m or $\perp$

  1. compute $\boldsymbol{T}' = [d]\boldsymbol{U}$
  2. set $(K_1' \| K_2') = \mathrm{KD}(\boldsymbol{T}', l)$
  3. if $\mathrm{MAC}_{K_2'}(c) = r$ then return $m = \mathtt{Enc}^{-1}_{K_1'}(c)$

technicolor

ECDH Augmented Encryption (2/2)

- Key generation: $\boldsymbol{Y} = [d]\boldsymbol{G}$ with $d \xleftarrow{\$} \{1, \ldots, n-1\}$
  $pk = \{\boldsymbol{G}, \boldsymbol{Y}\}$ and $sk = \{d\}$
- ECIES encryption
  1. pick a random $k \in \{1, \ldots, n-1\}$
  2. compute $\boldsymbol{U} = [k]\boldsymbol{G}$ and $\boldsymbol{T} = [k]\boldsymbol{Y}$
  3. set $(K_1 \| K_2) = \mathrm{KD}(\boldsymbol{T}, l)$
  4. compute $c = \mathtt{Enc}_{K_1}(m)$ and $r = \mathrm{MAC}_{K_2}(c)$
  5. return $(\boldsymbol{U}, c, r)$

- ECIES decryption

  Input ciphertext $(\boldsymbol{U}, c, r)$ and private key sk

  Output plaintext m or $\perp$

  1. compute $\boldsymbol{T'} = [d]\boldsymbol{U}$
  2. set $(K_1' \| K_2') = \mathrm{KD}(\boldsymbol{T'}, l)$
  3. if $\mathrm{MAC}_{K_2'}(c) = r$ then return $m = \mathtt{Enc}_{K_1'}^{-1}(c)$

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

technicolor

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

ECDSA

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

ECIES

ECIES

Forcing-Bit Attack

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

ECDSA

▸ ECDSA

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

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

ECIES

▸ ECIES

Forcing-Bit Attack

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

ECDSA

▸ 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 $k_j \to 0$ in Step 4)

ECIES

▸ ECIES

technicolor

Forcing-Bit Attack

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

ECDSA

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 $k_j \to 0$ in Step 4)

ECIES

ECIES

Forcing-Bit Attack

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

ECDSA

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 $k_j \to 0$ in Step 4)

ECIES

ECIES

- Check the ciphertext validity

technicolor

Forcing-Bit Attack

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

ECDSA

▸ 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 $k_j \to 0$ in Step 4)

ECIES

▸ 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

▸ 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

▸ ECIES

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

technicolor

Forcing-Bit Attack

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

ECDSA

▸ 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 $k_j \to 0$ in Step 4)

ECIES

▸ ECIES

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

technicolor

Forcing-Bit Attack

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

ECDSA

▸ 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 $k_j \to 0$ in Step 4)

ECIES

▸ ECIES

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

technicolor

Flipping-Bit Attack

Against ECDSA

▸ ECDSA

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

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

- 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

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

$$\Rightarrow d_j = \tfrac{1-\sigma}{2}$$

technicolor

Flipping-Bit Attack

Against ECDSA

▸ ECDSA

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

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

- 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

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

technicolor

Flipping-Bit Attack

Against ECDSA

▸ ECDSA

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

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

- 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

▸ ECDSA

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

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

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

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

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

Flipping-Bit Attack

Against ECDSA

▸ ECDSA

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

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

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

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

$$\Rightarrow d_j = \tfrac{1-\sigma}{2}$$

technicolor

Sign-Change Fault Attack

- Point inversion is inexpensive on elliptic curves

$$\boldsymbol{P} = (x_1, y_1) \quad \Rightarrow \quad -\boldsymbol{P} = (x_1, -y_1 - a_1 x_1 - a_3)$$

- Signed-digit point multiplication algorithms are preferred for computing $\boldsymbol{Q} = [d]\boldsymbol{P}$
  - 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 δ_i

- on the fly
- during exponent recoding

Safe-Error Attack (1/2)

- Double-and-add-*always* algorithm
  - additive variant of the square-and-multiply-*always*

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

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

- when $b = 1$, there is a dummy point addition

Safe-Error Attack (1/2)

- Double-and-add-*always* algorithm
  - additive variant of the square-and-multiply-*always*

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

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

  - when $b = 1$, there is a dummy point addition

Safe-Error Attack (2/2)

Against ECIES

▸ 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., $\perp$) 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

technicolor

Safe-Error Attack (2/2)

Against ECIES

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., $\perp$) 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

technicolor

Safe-Error Attack (2/2)

Against ECIES

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., $\perp$) 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

▸ 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., $\perp$) 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

technicolor

Safe-Error Attack (2/2)

Against ECIES

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., $\perp$) 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

technicolor

Safe-Error Attack (2/2)

Against ECIES

▸ 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., $\perp$) 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

technicolor

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

technicolor

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 $\boldsymbol{Q} = [d]\boldsymbol{P}$

Key Observation (1/2)

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

- Let $\boldsymbol{P} = (x_1, y_1)$ and $\boldsymbol{Q} = (x_2, y_2)$
- $\boldsymbol{P} + \boldsymbol{Q} = (x_3, y_3)$ where

$$x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2, \quad y_3 = (x_1 - x_3)\lambda - y_1 - a_1x_3 - a_3$$

$$\text{with } \lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{[addition]} \\ \dfrac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3} & \text{[doubling]} \end{cases}$$

- Parameter a_6 is not involved in point addition (or point doubling)

technicolor

Key Observation (2/2)

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

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

$$\tilde{E} : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + \tilde{a}_6$$

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

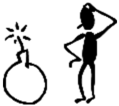
  then

$$d \pmod{t}$$

  can be recovered from $\tilde{\boldsymbol{Q}}$

technicolor

Chosen Input Point Attack



- Construct a 'point' $\tilde{\boldsymbol{P}}_{\boldsymbol{i}} = (\tilde{x}_i, \tilde{y}_i) \in \tilde{E}_i$ such that
  1. $\mathrm{ord}_{\tilde{E}_i}(\tilde{\boldsymbol{P}}_{\boldsymbol{i}}) = t_i$ is small
  2. discrete logarithms are computable in $\langle \tilde{\boldsymbol{P}}_{\boldsymbol{i}} \rangle$
- Query the device with $\tilde{\boldsymbol{P}}_{\boldsymbol{i}}$ and receive $\tilde{\boldsymbol{Q}}_{\boldsymbol{i}} = [d]\tilde{\boldsymbol{P}}_{\boldsymbol{i}}$
- 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 $\boldsymbol{Q} = [d]\boldsymbol{P}$ on $E_{/\mathbb{F}_p} : y^2 = x^3 + a_4 x + a_6$

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

- Similar attack when the y-coordinate of $\boldsymbol{P}$ is corrupted
- More assumptions are needed when both coordinates are corrupted

technicolor

Faults in the Base Point

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

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

- Similar attack when the y-coordinate of $\boldsymbol{P}$ is corrupted
- More assumptions are needed when both coordinates are corrupted

Faults in the Base Point

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

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

- Similar attack when the y-coordinate of $\boldsymbol{P}$ is corrupted
- More assumptions are needed when both coordinates are corrupted

technicolor

Faults in the Base Point

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

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

- Similar attack when the y-coordinate of $\boldsymbol{P}$ is corrupted
- More assumptions are needed when both coordinates are corrupted

Faults in the Base Point

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

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

- Similar attack when the y-coordinate of $\boldsymbol{P}$ is corrupted
- More assumptions are needed when both coordinates are corrupted

technicolor

Faults in the Base Point

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

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

- Similar attack when the y-coordinate of $\boldsymbol{P}$ is corrupted
- More assumptions are needed when both coordinates are corrupted

technicolor

Faults in the Base Point

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

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

- Similar attack when the y-coordinate of $\boldsymbol{P}$ is corrupted
- More assumptions are needed when both coordinates are corrupted

Faults in the Definition Field

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

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

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

technicolor

Faults in the Curve Parameters

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

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

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

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

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

technicolor

Skipping Attack

Attack assumes that the attacker manages to skip a doubling operation

- 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 ← O; R_1 ← G
2: for i = ℓ − 1 down to 0 do
3:     R_0 ← [2]R_0
4:     if k_i = 1 then R_0 ← R_0 + R_1
5: return R_0
```


technicolor

Skipping Attack

Attack assumes that the attacker manages to skip a doubling operation

- can be seen as a random error at the bit level

Algorithm 2 Double-and-add

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

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

technicolor

Application to ECDSA

ECDSA

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

$$\hat{T} = \sum_{i=j+1}^{\ell-1} [k_i\, 2^{i-1}]G + \sum_{i=0}^{j} [k_i\, 2^i]G$$
$$= [\tfrac{1}{2}](T + [\tilde{k}]G)$$

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: $T = [k]G$

```
R0 ← O; R1 ← G
for i = ℓ − 1 down to 0 do
    R0 ← [2]R0
    if k_i = 1 then R0 ← R0 + R1
return R0
```

Observation:

$$[\hat{u}_1]G + [\hat{u}_2]Y = [\tfrac{H(m)}{\hat{s}}]G + [\tfrac{\hat{r}}{\hat{s}}]Y = [\tfrac{H(m)+d\hat{r}}{\hat{s}}]G = [k]G$$

$$\hat{r} \stackrel{?}{=} x([\tfrac{1}{2}](T + [\tilde{k}]G)) \pmod{n} \quad \text{with } T = [\hat{u}_1]G + [\hat{u}_2]Y \implies \tilde{k} = \ldots$$

technicolor

Application to ECDSA

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

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

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

with $\tilde{k} = (k_j, \ldots, k_0)_2$

- $(r, s) \rightsquigarrow (\hat{r}, \hat{s})$

ECDSA

Algorithm 4 Double-and-add

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

```
R0 ← O; R1 ← G
for i = ℓ − 1 down to 0 do
    R0 ← [2]R0
    if k_i = 1 then R0 ← R0 + R1
return R0
```

Observation:

$$[\hat{u}_1]G + [\hat{u}_2]Y = [\tfrac{H(m)}{\hat{s}}]G + [\tfrac{\hat{r}}{\hat{s}}]Y = [\tfrac{H(m)+d\hat{r}}{\hat{s}}]G = [k]G$$

$$\hat{r} \stackrel{?}{=} x([\tfrac{1}{2}](T + [\tilde{k}]G)) \pmod{n} \quad \text{with } T = [\hat{u}_1]G + [\hat{u}_2]Y \implies \tilde{k} = \ldots$$

Application to ECDSA

▸ ECDSA

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

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

with $\tilde{k} = (k_j, \ldots, k_0)_2$

- $(r, s) \rightsquigarrow (\hat{r}, \hat{s})$

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: $\quad R_0 \leftarrow [2]R_0$
4: $\quad$ **if** $k_i = 1$ **then** $R_0 \leftarrow R_0 + R_1$
5: **return** R_0

Observation:

$$[\hat{u}_1]G + [\hat{u}_2]Y = [\tfrac{H(m)}{\hat{s}}]G + [\tfrac{\hat{r}}{\hat{s}}]Y = [\tfrac{H(m)+d\hat{r}}{\hat{s}}]G = [k]G$$

$$\hat{r} \overset{?}{\equiv} x([\tfrac{1}{2}](T + [\tilde{k}]G)) \pmod{n} \quad \text{with } T = [\hat{u}_1]G + [\hat{u}_2]Y \implies \tilde{k} = \ldots$$

technicolor

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

technicolor

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., $d \leftarrow d + r\,n$ with $n = \text{ord}_E(\boldsymbol{P})$
- Compute the operations twice
  - doubles the running time
- Verify the signatures
  - ECDSA verification is slower than signing
- Check that the output point $\boldsymbol{Q} = [k]\boldsymbol{P}$ is in $\langle \boldsymbol{P} \rangle$
  - $\boldsymbol{Q} \in E$
  - $[h]\boldsymbol{Q} \neq \boldsymbol{O}$ (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 $n = \mathrm{ord}_E(\boldsymbol{P})$
- Compute the operations twice
  - doubles the running time
- Verify the signatures
  - ECDSA verification is slower than signing
- Check that the output point $\boldsymbol{Q} = [k]\boldsymbol{P}$ is in $\langle \boldsymbol{P} \rangle$
  - $\boldsymbol{Q} \in E$
  - $[h]\boldsymbol{Q} \neq \boldsymbol{O}$ (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 $n = \mathrm{ord}_E(\boldsymbol{P})$
- Compute the operations twice
  - doubles the running time
- Verify the signatures
  - ECDSA verification is slower than signing
- Check that the output point $\boldsymbol{Q} = [k]\boldsymbol{P}$ is in $\langle \boldsymbol{P} \rangle$
  - $\boldsymbol{Q} \in E$
  - $[h]\boldsymbol{Q} \neq \boldsymbol{O}$ (only implies of large order)

technicolor

Basic Countermeasures

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

technicolor

Basic Countermeasures

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

technicolor

Multiplier Randomization (1/2)

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

Example (secp160k1)

$p = 2^{160} - 2^{32} - 538\mathrm{D}_{16}$ [generalized] Mersenne prime

$\#E = 01\ 00000000\ 00000000\ 0001\mathrm{B8FA}\ 16\mathrm{DFAB9A}\ \mathrm{CA16B6B3}_{16}$

$\Rightarrow d^* = d + r\,\#E = (r)_2 \parallel d_{\ell-1} \cdots d_{\ell-t} \parallel$ 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)

$p = 2^{160} - 2^{32} - 538\mathrm{D}_{16}$ [generalized] Mersenne prime

$\#E = 01\ 00000000\ 00000000\ 0001\mathrm{B8FA}\ 16\mathrm{DFAB9A}\ \mathrm{CA16B6B3}_{16}$

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

technicolor

Multiplier Randomization (1/2)

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

Example (secp160k1)

$p = 2^{160} - 2^{32} - \mathtt{538D}_{16}$ [generalized] Mersenne prime

$\#E = \mathtt{01\ 00000000\ 00000000\ 0001B8FA\ 16DFAB9A\ CA16B6B3}_{16}$

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

technicolor

Multiplier Randomization (2/2)

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

Euclidean splitting

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

$$\Longrightarrow [d]\boldsymbol{P} = [d \bmod r]\boldsymbol{P} + \left[\lfloor d/r \rfloor\right]([r]\boldsymbol{P})$$

technicolor

Multiplier Randomization (2/2)

- Use splitting methods
  - additive:

$$[d]\boldsymbol{P} = [d - r]\boldsymbol{P} + [r]\boldsymbol{P}$$

  - multiplicative:

$$[d]\boldsymbol{P} = [d\,r^{-1}]([r]\boldsymbol{P})$$

Euclidean splitting

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

$$\implies [d]\boldsymbol{P} = [d \bmod r]\boldsymbol{P} + [\lfloor d/r \rfloor]([r]\boldsymbol{P})$$

Multiplier Randomization (2/2)

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

Euclidean splitting

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

$$\Longrightarrow \; [d]\boldsymbol{P} = [d \bmod r]\boldsymbol{P} + \big[\lfloor d/r \rfloor\big]\big([r]\boldsymbol{P}\big)$$

Strauss-Shamir double ladder

Multiplier Randomization (2/2)

- Use splitting methods
  - additive:

$$[d]\boldsymbol{P} = [d - r]\boldsymbol{P} + [r]\boldsymbol{P}$$

  - multiplicative:

$$[d]\boldsymbol{P} = [d\,r^{-1}]\big([r]\boldsymbol{P}\big)$$

Euclidean splitting

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

$$\Longrightarrow \quad [d]\boldsymbol{P} = [d \bmod r]\boldsymbol{P} + \big[\lfloor d/r \rfloor\big]\big([r]\boldsymbol{P}\big)$$

- Strauss-Shamir double ladder

technicolor

Multiplier Randomization (2/2)

- Use splitting methods
  - additive:

$$[d]\boldsymbol{P} = [d - r]\boldsymbol{P} + [r]\boldsymbol{P}$$

  - multiplicative:

$$[d]\boldsymbol{P} = [d\,r^{-1}]\big([r]\boldsymbol{P}\big)$$

Euclidean splitting

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

$$\Longrightarrow \quad [d]\boldsymbol{P} = [d \bmod r]\boldsymbol{P} + \big[\lfloor d/r \rfloor\big]\big([r]\boldsymbol{P}\big)$$

- Strauss-Shamir double ladder

Preventing Fault Attacks: The Case of RSA

Shamir's countermeasure

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

Giraud's countermeasure

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

technicolor

Preventing Fault Attacks: The Case of RSA

Shamir's countermeasure

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

Giraud's countermeasure

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

technicolor

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:

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

$\Rightarrow Return\ (T[a] - b) \cdot r + a$

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:

$$\textit{If } (\mathrm{T}[a] = b) \textit{ then return } a \textit{ else } \mathtt{error}$$

$$\Rightarrow \textit{Return } (\mathrm{T}[a] - b) \cdot r + a$$

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:

$$\textit{If}\ (\mathrm{T}[a] = b)\ \textit{then return}\ a\ \textit{else}\ \mathtt{error}$$

$$\Rightarrow \textit{Return}\ (\mathrm{T}[a] - b) \cdot r + a$$

Edwards Curves

$$\mathcal{E}_{/\mathbb{F}_p} : ax^2 + y^2 = 1 + bx^2y^2 \quad \text{where } ab(a-b) \neq 0$$

- Addition law
  - $\boldsymbol{O} = (0,1)$ [neutral element]
  - $-(x_1, y_1) = (-x_1, y_1)$
  - $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = \frac{x_1y_2 + x_2y_1}{1 + bx_1x_2y_1y_2}, \quad y_3 = \frac{y_1y_2 - ax_1x_2}{1 - bx_1x_2y_1y_2}$$

  - ... also valid for point doubling (and $\boldsymbol{O}$)
- Addition law is *complete* if a is a square and b is a non-square

technicolor

Shamir's Trick for Elliptic Curve Cryptosystems

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

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

technicolor

Shamir's Trick for Elliptic Curve Cryptosystems

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

- Let $\mathcal{R} = \mathbb{Z}/pr\mathbb{Z}$ for a (small) random prime r
  1. Compute
     - $\mathcal{E}_{pr} \leftarrow \mathrm{CRT}(\mathcal{E}, \mathcal{E}_r)$ where $\mathcal{E}_{r_{/\mathbb{F}_r}} : ax^2 + y^2 = 1 + b_r x^2 y^2$
     - $\boldsymbol{Q^*} \leftarrow [d]\boldsymbol{P} \in \mathcal{E}_{pr}(\mathbb{Z}/pr\mathbb{Z})$
     - $\boldsymbol{Y} \leftarrow [d]\boldsymbol{P_r} \in \mathcal{E}_r(\mathbb{F}_r)$
  2. If $(\boldsymbol{Q^*} \not\equiv \boldsymbol{Y} \pmod{r})$ then return `error`
  3. Return $\boldsymbol{Q^*} \bmod p$

Idea #1

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

- . . . but completeness is not guaranteed (and $\#\mathcal{E}_r$ is unknown)

technicolor

Shamir's Trick for Elliptic Curve Cryptosystems

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

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

Idea #2

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

- . . . but r is now *a priori* fixed and values must be pre-stored

technicolor

BOS$^+$ Algorithm

- Blömer, Otto, and Seifert (FDTC 2005)

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

---

1. Compute
   1. $\mathcal{E}_{pr} \leftarrow \text{CRT}(\mathcal{E}, \mathcal{E}_r)$ and $\boldsymbol{P}^* \leftarrow \text{CRT}(\boldsymbol{P}, \boldsymbol{P}_r)$
   2. $\boldsymbol{Q}^* \leftarrow [d]\boldsymbol{P}^* \in \mathcal{E}_{pr}$ $\quad = (x_{pr}, y_{pr})$
   3. $\boldsymbol{Y} \leftarrow [d \pmod{n_r}]\boldsymbol{P}_r \in \mathcal{E}_r$ $\quad = (x_r, y_r)$
   4. $\begin{cases} c_x \leftarrow 1 + x_{pr} - x_r \pmod{r} \\ c_y \leftarrow 1 + y_{pr} - y_r \pmod{r} \end{cases}$
2. If $(\boldsymbol{Q}^* \neq \boldsymbol{Y} \pmod{r})$ then return `error`
3. Return $\boldsymbol{Q}^* \pmod{p} \in \mathcal{E}$

BOS$^+$ Algorithm

- Blömer, Otto, and Seifert (FDTC 2005)

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

1. Compute
   1. $\mathcal{E}_{pr} \leftarrow \mathrm{CRT}(\mathcal{E}, \mathcal{E}_r)$ and $\boldsymbol{P}^* \leftarrow \mathrm{CRT}(\boldsymbol{P}, \boldsymbol{P}_r)$
   2. $\boldsymbol{Q}^* \leftarrow [d]\boldsymbol{P}^* \in \mathcal{E}_{pr}$ $\quad = (x_{pr}, y_{pr})$
   3. $\boldsymbol{Y} \leftarrow [d \pmod{n_r}]\boldsymbol{P}_r \in \mathcal{E}_r$ $\quad = (x_r, y_r)$
   4. $\begin{cases} c_x \leftarrow 1 + x_{pr} - x_r \pmod{r} \\ c_y \leftarrow 1 + y_{pr} - y_r \pmod{r} \end{cases}$
2. For a κ-bit random ρ, compute $\gamma \leftarrow \left\lfloor \frac{\rho\, c_x + (2^\kappa - \rho) c_y}{2^\kappa} \right\rfloor$
3. Return $\boldsymbol{Q} = [\gamma]\boldsymbol{Q}^* \pmod{p} \in \mathcal{E}$

technicolor

Shamir's Trick for Elliptic Curve Cryptosystems ?!

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

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

Idea #3 (???)

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

technicolor

New Algorithm

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

- Properties
  - $\mathcal{E}_1 \simeq (\mathbb{Z}/q\mathbb{Z})^+$, $\boldsymbol{P_1} = (\alpha q, 1) \overset{\sim}{\mapsto} \alpha$
  - $\#\mathcal{E}_1 = q$
  - $[d]\boldsymbol{P_1} = (dx_1, 1)$ where $x_1 = \alpha q$
- Addition law is complete

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + x_2y_1}{1 + bx_1x_2y_1y_2}, \frac{y_1y_2 - ax_1x_2}{1 - bx_1x_2y_1y_2}\right)$$

whatever curve parameters a and b

technicolor

New Algorithm

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

1. Choose a small random t
2. Define $r \leftarrow t^2$ and $\boldsymbol{P}_r \leftarrow (t, 1)$
3. Compute
   1. $\boldsymbol{P}^* \leftarrow \mathrm{CRT}(\boldsymbol{P}, \boldsymbol{P}_r)$
   2. $\boldsymbol{Q}^* \leftarrow [d]\boldsymbol{P}^* \in \mathcal{E}(\mathbb{Z}/pr\mathbb{Z})$ $\quad = (x_{pr}, y_{pr})$
   3. $\boldsymbol{Y} \leftarrow (dt \bmod r, 1)$ $\quad = (x_r, y_r)$
   4. $\begin{cases} c_x \leftarrow 1 + x_{pr} - x_r \pmod{r} \\ c_y \leftarrow y_{pr} \pmod{r} \end{cases}$
4. If $(\boldsymbol{Q}^* \not\equiv \boldsymbol{Y} \pmod{r})$ then return `error`
5. Return $\boldsymbol{Q}^* \pmod{p} \in \mathcal{E}(\mathbb{F}_p)$

New Algorithm

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

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

Outline

technicolor

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

technicolor

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

technicolor

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

technicolor

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

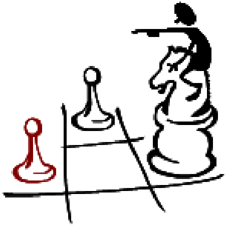
technicolor

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

technicolor

Further Research: Attacks



technicolor

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)

technicolor

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

technicolor

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

technicolor

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

technicolor

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

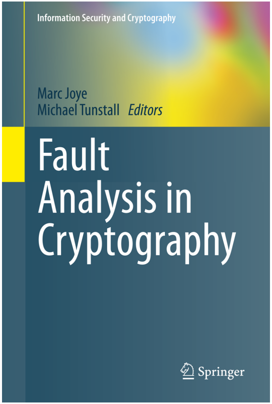
Explore scalar multiplication algorithms or design new ones having invariants (as in Giraud's proposal)

Research Problem #3

Develop countermeasures against combined attacks in an efficient way

technicolor

More Information



Comments/Questions?