Combined Attack on ECC using Points of Low Order

Benedikt Gierlichs
joint work with Junfeng Fan and Frederik Vercauteren

COSIC, Katholieke Universiteit Leuven, Belgium
• ECC: Elliptic curve over finite field
  – A set of points $P(x,y)$ and $\mathcal{O}$ at infinity
• $\mathcal{O}$ required to form an abelian group
• But in crypto you should never see $\mathcal{O}$
• $\mathcal{O}$ is not easy to deal with in implementation
• But it should never occur anyway

So how does /would your implementation deal with $\mathcal{O}$?
Outline

• Context: embedded security

• Background: elliptic curves and their use in cryptography

• Our attack: principle, toy examples, requirements

• Popular countermeasures for ECC implementations and our attack

• Conclusion
Context
Embedded cryptography

• 98% of processor market are embedded processors
  – In 2008: over 10 billion embedded devices
• Over 100 embedded processors in a single modern luxury car
• More and more applications with security context
Classical security model (simplified)

- Encryption and cryptographic operations in *black* boxes
- Attack on channel *between* communicating parties
- Protection by strong mathematic algorithms and protocols
Embedded security

- Can cryptographic functions alone assure security?

- Cryptographic functions put a barrier between what must be protected and an adversary
The system is as secure as its weakest link
Embedded security

- Can cryptographic functions alone assure security?
- Not if it is easy to bypass them
Embedded security

- Devices are not mathematical functions

- Subject to physics
  - Physical properties leak information about the secret key
  - Devices react to physical stimulation

- Device in possession of user
  - User can be malicious (or device stolen)

- Device under physical control of adversary
  - No time constraints
New security model (simplified)

- Attack channel \textit{and} endpoints
- Encryption and cryptographic operations in \textit{gray} boxes
- Need \textit{both}:
  - Protection by strong mathematic algorithms and protocols
  - Protection by secure implementation
Why research on attacks?

- We have no way to prove that implementation is secure
- Instead, test if implementation resists all known attacks
  - In research and in standardized real-world evaluations
  - Problem similar to symmetric cryptography
- Problem: one can not know all attacks

- Attacks lead to new countermeasures and vice versa
Background
Elliptic curves
Elliptic curves
Elliptic curves over finite fields

The elliptic curve $y^2 = x^3 + x + 3 \mod 23$
ECC versus RSA

• ECC has several advantages on embedded platforms
• For the same security level
  – Shorter keys
  – Smaller operands

Example:

80-bit security
  • RSA with 1248-bit keys
  • ECC with 160-bit keys

128-bit security
  • RSA with 3248-bit keys
  • ECC with 256-bit keys
Elliptic curves over finite fields

- $E$ over $F_p : y^2 = x^3 + ax + b \quad a, b \in F_p \quad 4a^3 + 27b^2 \neq 0$
- $E(K) := \{(x, y) \mid y^2 = x^3 + ax + b\} \cup \{O\}$
- $E(K)$ is abelian group
- $\#E(K) \simeq \#K$ with error $\leq 2\sqrt{\#K}$

- Use in crypto: scalar multiplication $k \cdot P$
  - EC discrete logarithm problem: given $P$ and $k \cdot P$, find $k$
  - Hard because order of $P$ huge on strong curves
  - Implemented as sequence of 'small' operations

$P \rightarrow k \cdot P$ scalar multiplication
Scalar multiplication on elliptic curves

- Given integer $k$ and point $P$, compute $Q = k \cdot P$
- Consider $k$ in its binary representation, e.g. $k = 10010...110_2$

---

**Algorithm 1: Double and Add Left-to-Right**

**Input:** $P$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$

**Output:** $Q = k \cdot P$

$R \leftarrow P$;

for $i \leftarrow n - 2$ down to 0 do

$R \leftarrow 2 \cdot R$;

if $(k_i = 1)$ then $R \leftarrow R + P$;

end

return $R$
Group operations: point addition
Group operations: point doubling
Group law and implementation

• Addition: \( P + Q = (x_3, y_3) \) with
\[
x_3 = \left( \frac{y_2 - y_1}{x_2 - x_1} \right)^2 - x_1 - x_2
\]
if \( P \neq \pm Q \) else \( \mathcal{O} \) appears
\[
y_3 = \left( \frac{y_2 - y_1}{x_2 - x_1} \right)(x_1 - x_3) - y_1
\]

• Doubling: \( 2P = (x_3, y_3) \) with
\[
x_3 = \left( \frac{3x_1^2 + a}{2y_1} \right) - x_1 - x_2
\]
if \( \text{ord}(P) > 2 \) else \( \mathcal{O} \) appears
\[
y_3 = \left( \frac{3x_1^2 + a}{2y_1} \right)(x_1 - x_3) - y_1
\]

• But these cases should never occur anyway
• Note that \( b \) is not used in the formulae
Group law and implementation (2)

• Implementations:
  – Coordinate system: affine, projective, Jacobian, etc.

• **Full domain correct:**
  – Implementation computes \( P+Q \) and \( 2 \cdot P \) correctly on the whole domain
  – For Weierstrass curves this typically requires IF statements

• **Partial domain correct**: not full domain correct
  – For some inputs, implementation
  • Crashes, e.g. division by zero for affine coordinates
  • No crash but gets stuck at fixed / invalid point for Jacobian and projective coord.
### Group law and implementation (3)

**Equation:**

\[ E(\mathbb{F}_p): y^2 = x^3 + ax + b \]

<table>
<thead>
<tr>
<th>Coordinate System</th>
<th>Operation</th>
<th>Using a</th>
<th>Using b</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projective</td>
<td>PA((P_1, P_2))</td>
<td>-</td>
<td>-</td>
<td>(P_1 = P_2)</td>
<td>(0,0,0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(P_1 = -P_2)</td>
<td>(0,*,0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(P_1 = (0,*,0))</td>
<td>(0,0,0)</td>
</tr>
<tr>
<td></td>
<td>PD((P_1))</td>
<td>+</td>
<td>-</td>
<td>Order((P_1) = 2)</td>
<td>(0,*,0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(P_1 = (0,*,0))</td>
<td>(0,0,0)</td>
</tr>
<tr>
<td>Jacobian</td>
<td>PA((P_1, P_2))</td>
<td>-</td>
<td>-</td>
<td>(P_1 = P_2)</td>
<td>(0,0,0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(P_1 = -P_2)</td>
<td>(<em>,</em>,0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(P_1 = (<em>,</em>,0))</td>
<td>(<em>,</em>,0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(P_1 = (0,0,0))</td>
<td>(0,0,0)</td>
</tr>
<tr>
<td></td>
<td>PD((P_1))</td>
<td>+</td>
<td>-</td>
<td>Order((P_1) = 2)</td>
<td>(<em>,</em>,0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(P_1 = (<em>,</em>,0))</td>
<td>(<em>,</em>,0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(P_1 = (0,0,0))</td>
<td>(0,0,0)</td>
</tr>
</tbody>
</table>

**Borderline cases for projective and Jacobian coordinates**
Our attack
The attack

- Setting: target computes $k \cdot P$ for any given $P$, $k$ is secret

- Idea: choose rogue, valid input $P$ s.t. a fault $\epsilon$ turns it into $P'$
  - $P'$ is point of very low order
Points with low-order neighbours

- Given curve $E: y^2 = x^3 + ax + b$, integers $l$ and $\delta$
- Construct $P(x_p,y_p)$ on $E$ s.t.
  - $\exists$ curve $E': y^2 = x^3 + ax + b'$
  - With $P'(x_p',y_p')$ of order $l$ on $E'$
  - Hamming dist. of bit representations $x_p || y_p$ and $x_p' || y_p'$ is $\delta$
- If $\delta = 1$ we call $P$ and $P'$ neighbours

- Input: $E$, $l$, $\delta$
- Output: $P$ and $P'$
Points with low-order neighbours

- NIST P-192 curve over $\mathbb{F}_p$ with $p = 2^{192} - 2^{64} - 1$, $a = -3$ and $b = 0x64210519E59C80E70FA7E9AB72243049FEB8DEECC146B9B1

<table>
<thead>
<tr>
<th>Order</th>
<th>$P$</th>
<th>bit-flip</th>
</tr>
</thead>
</table>
| 2     | $xP = 0x6D9D789820A2C19237C96AD4B8D86B87FB49D4D6C728B84F$  
|       | $yP = 0x1$  | 0 |
| 3     | $xP = 0x8E1EBDD009F11449C7BC2C02509F8E432ED15F10C2D33$  
|       | $yP = 0x7A568946EFA602B3624A61E513E57869CAF2AE854E1A17B$  | 2 |
| 4     | $xP = 0xB317D7BBD023E6293F1506221F5BC4A23D4BE2E05328C5F7$  
|       | $yP = 0xC70D48794F4065097620C0865B7D567329728C634CA6AE$  | 0 |
| 5     | $xP = 0xCC9BCC0061F64371E3C3BDE165DAD5380A7DC1919765940$  
|       | $yP = 0xCCB36B372834B8AFD7A9FCCFB40773E94A4178093458$  | 8 |
| 6     | $xP = 0xC3F76445E6A52138E283E485092F005BE0821C3F9E96B05E$  
|       | $yP = 0x535DBCCB593D72E7885B66E5FDF13A8FF9C57A8F8B91CE48$  | 1 |
| 7     | $xP = 0x5C003567728CCBC9F4C06620B9973193837BAEC67A29E43A$  
|       | $yP = 0x408D0C3135006B03EFF80961394D890F0E86D9FD1BA4E6C6$  | 3 |
| 8     | $xP = 0x74FD6A1AD39479C75A85305FA786E1DBDC845E03754E723E$  
|       | $yP = 0x6EF58ABFC071047BA4F425652B3EC1746EBE8FE16FEA1F5$  | 1 |
Attack against a toy implementation

• Full domain correct
  – $\mathcal{O}$ and all following computations will be handled correctly
• Double and add scalar multiplication

• Input $P$ with neighbour $P'$ of order 4, inject fault and measure
• Doubling: $2 \cdot P'$, $2 \cdot 2P'$, $2 \cdot 3P'$ or $2 \cdot \mathcal{O}$ (borderline cases)
• Addition: generates always odd multiples of $P'$, never $\mathcal{O}$
• $\mathcal{O}$ occurs only after 2 consecutive doublings
• If $\mathcal{O}$ occurs during processing of bit $k_i$, bit $k_{i+1}$ must be 0
• Uniquely identifies all 0 key bits (possibly except LSB)
Attack against a toy implementation

- Full domain correct
- Double and add scalar multiplication
- Input $P$ with neighbour $P'$ of order 4, inject fault and measure
- $k = 5405 = \overline{1010100011101}_2$

<table>
<thead>
<tr>
<th>$i$</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_i$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

| step 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| step 2 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

- Obtain all of $k$ with a single trace!
Attack against a toy implementation

• Affine coordinates, partial domain correct (crash at 1\textsuperscript{st} O)
• Double and add scalar multiplication

• First occurrence of O leaks, then no more information
• For P' of order \( l \), we obtain index I(\( l \)) s.t. the first I(\( l \)) bits of k form an integer divisible by \( l \)
  – Also information if not divisible by \( l \)

• Repeat with P' of increasing orders \( l \)
  – Requires several traces with the same k
• Incremental search algorithm, obtain almost all of k
Feasibility of attack

• Need to be able to choose input P s.t. P' is of low order
  – El Gamal encryption/decryption, static Diffie-Hellman, etc.
• Or: system with fixed base point where P is already rogue
  – Nice back-door: impossible to check all error patterns

• Fault injection: need a specific error ε
  – ε is 1 bit-flip, 256 random byte faults, only ε leads to P' and ◯
  – ε can be adjusted to any likely error pattern, in all coordinates
  – Precise timing

• Side channel: need leakage
  – We assume leakage by IFs, crashes, zero-value coordinates, etc.
• Group law implementation
  – Attack does not apply if all curve coefficients are used in PA/PD formulas
Our attack against protected implementations
Attacks on scalar multiplication and countermeasures

- **SPA**
  - Solution: regular algorithm / implementation (atomicity)

- **DPA**
  - Solution: key, field, curve and point randomization

- **Faults**
  - Solution: check output point and curve parameter validity

- **Low-order attack (weak curve attack)**
  - Solution: check input point validity
  - Small co-factor check (all NIST curves have co-factor 1)
Attack against protected implementations

• Input point validity check
  – No problem if we can inject fault after check but before mult

• Output point / curve parameters validity check
  – No problem, we already got the info
Attack against protected implementations

• Regular exponentiation algorithms / implementations to protect against SPA
  – Attack is fairly independent of scalar multiplication algorithm
  – Each algorithm computes some multiples of P that depend on k
  – If so, the attack applies

• Example: Montgomery powering ladder
Montgomery powering ladder

Algorithm 3: Montgomery powering ladder

Input: \( P, k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2 \)
Output: \( Q = k \cdot P \)

\[ R_0 \leftarrow P, \quad R_1 \leftarrow 2 \cdot P \]

for \( i \leftarrow n - 2 \) down to 0 do

\[ R_{-k_i} \leftarrow R_{k_i} + R_{-k_i}, \quad R_{k_i} \leftarrow 2 \cdot R_{k_i} \]

end

return \( R_0 \)

• 2 registers \( R_0 \) and \( R_1 = R_0 + P \)
  – Input \( P \) with neighbour \( P' \) of order 4
  – If 2 consecutive key bits are equal, \( R_0 \) or \( R_1 \) doubled twice, \( \bigcirc \) occurs
  – If 2 consecutive key bits are different, ordinary doublings
  – \( \bigcirc \) can never be the result of an addition

• Obtain almost all of \( k \) with a single trace
More in the paper

• Countermeasures we looked at
  – Random scalar splitting: \( k = k_1 + k_2, \ k \cdot P = k_1 \cdot P + k_2 \cdot P \)
  – Scalar blinding: \( k' = k + r \cdot \#E \)
  – Ephemeral keys
  – Coordinate randomization, e.g. random projective coordinates
  – Random elliptic curve isomorphisms
  – Base point blinding

• Binary curves
  – Applicability of attack depends on coordinate system
  – Affine and standard projective coord.: attack applies since only a used
  – Jacobian: attack does not apply since a and b are used
  – Lopez-Dahab: attack does not apply; only b is used but changing a
    results in isomorphic curve over its quadratic twist
Conclusion

• Our attack:
  – Input rogue $P$ and inject fault after initial checks
  – $P$ turns into $P'$ of low order
  – $k \cdot P'$ leads to ⊗ which can be detected via side channels
• Requires chosen inputs (or rogue fixed base point)
• Very powerful attack on full domain correct implementations
  – Defeats many countermeasures, requires only a single trace
• Combining countermeasures does not automatically protect against combined attacks
• Countermeasures that prevent our attack:
  – Sensors, concurrent validity checks, base point blinding, etc.
Thank you. Questions?

So how does / would your implementation deal with $\mathcal{O}$?