A Practical Second-Order Fault Attack against a Real-World Pairing Implementation

Peter Günther
joint work with
Johannes Blömer  Ricardo Gomes da Silva  Juliane Krämer  Jean-Pierre Seifert

University of Paderborn
Technical University of Berlin

FDTC 2014, September 23, 2014, Busan
The role of the final exponentiation

Two step pairing computation

\[ e : \mathcal{E}(\mathbb{F}_{p^k}) \times \mathcal{E}(\mathbb{F}_{p^k}) \rightarrow \mathbb{F}_{p^k}^* \]

\[ (P, Q) \mapsto f_{r,P}(Q)^d \]

1. \( \alpha \leftarrow f_{r,P}(Q) \) (Miller step)
2. \( \beta \leftarrow \alpha^d \) with \( d = (p^k - 1)/r \) (final exponentiation)

1. Computes non-degenerate, bilinear mapping to \( \mathbb{F}_{p^k}^*/(\mathbb{F}_{p^k}^*)^r \).
2. Maps equivalence classes \( \mathbb{F}_{p^k}^*/(\mathbb{F}_{p^k}^*)^r \) to unique representatives in \( \mu_r \).
Cryptanalysis of pairings

Inversion of both steps is required

1. Search secret $Q$ as one solution of $f_{r,P}(x, y) = \alpha$
   Problem: Function $f_{r,P}(x, y)$ has huge degree $r$

2. Inversion of final exponentiation $(\cdot)^d = \beta$
   Problem: Difficult to identify the correct $d$-th root $\alpha$

$\Rightarrow$ 2nd order attacks required

Cryptanalysis of pairings with fault attacks

1. Reduce degree of $f_{r,P}(x, y)$ by modification of $r, P, Q$

2. Make $(\cdot)^d$ as injective as possible
Fault attacks on pairings

Cryptanalysis of pairings

Inversion of both steps is required

1. Search secret $Q$ as one solution of $f_{r,P}(x, y) = \alpha$
   Problem: Function $f_{r,P}(x, y)$ has huge degree $r$

2. Inversion of final exponentiation $(\cdot)^d = \beta$
   Problem: Difficult to identify the correct $d$-th root $\alpha$

$\Rightarrow$ 2nd order attacks required

Our 2nd order attack

1. Round reduction of Miller loop: obtain $f_{r',P}(x, y)$ of degree $r' = 5$.

2. Skipping final exponentiation
Fault attacks on pairings

Outline of our attack

1. Assumption: attacker with physical access to target (especially CPU clock)
2. Trigger computation of $e(P, Q)$ on public argument (e.g. $P$) and secret argument (e.g. $Q$)
3. Distort computation of $e(P, Q)$ by clock glitch to obtain $\beta'$
4. Compute secret $Q$ from $\beta'$
Our attack: Schematic Hardware Setup

Glitcher
(delay,duration)
Queue
...

Timer

33 MHz
99 MHz

Queue
(delay,duration)

*.py

Host

*.log
*.py

Target

tgt_clk
tgt_io
tgt_rst

CPU

Mechanism: CPU clock glitching

Effect: Instruction skips

99 MHz (f)
33 MHz (f)

Peter Günther (UPB)
2nd order fault attack on PBC

FDTC 2014 5 / 16
Our attack: Schematic Hardware Setup

- Mechanism: CPU clock glitching
- Effect: Instruction skips

Glitcher

33 MHz

99 MHz

Timer

Queue

(delay,duration)

*.py

Host

*.log

*.py

Target

CPU

tgt_clk
tgt_io
tgt_rst

Mechanism: CPU clock glitching
Effect: Instruction skips
Our attack: Schematic Hardware Setup

- **Mechanism:** CPU clock glitching
- **Effect:** Instruction skips

99 MHz ($f_h$) ____________________________
33 MHz ($f_i$) ____________________________

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher

Glitcher
Our attack: Schematic Hardware Setup

- **Mechanism:** CPU clock glitching
- **Effect:** Instruction skips

Glitcher

```
33 MHz
99 MHz
```

Timer

```
(delay,duration)
```

Queue

```
*(delay,duration)*
```

Host

```
*.log
*.py
```

Target

```
33 MHz
99 MHz
tgt_clk
tgt_io
tgt_rst
```

CPU

```
tgt_clk
tgt_io
```

```
99 MHz \( (f_h) \)
33 MHz \( (f_i) \)
gl_clk
```

Peter Günther (UPB)
2nd order fault attack on PBC
FDTC 2014 5 / 16
Our attack: Schematic Hardware Setup

- Mechanism: CPU clock glitching
- Effect: Instruction skips

Mechanism: CPU clock glitching
Effect: Instruction skips
Our attack: Schematic Hardware Setup

- **Mechanism:** CPU clock glitching
- **Effect:** Instruction skips

Glitcher

- (delay, duration)
- \((t_1, d_1) = (3, 2)\)
- \((t_2, d_2) = (2, 1)\)

Queue

**Host**
- *.log
- *.py

**Target**
- tgt_clk
- tgt_io
- tgt_rst

**CPU**

- 33 MHz
- 99 MHz

**Mechanism:**

- 99 MHz \((f_h)\)
- 33 MHz \((f_l)\)

- gl_clk

- \(t_1\)
- \(d_1\)

- 0 1 2 3 4 5 6 7 8 9
Our attack: Schematic Hardware Setup

- **Mechanism:** CPU clock glitching
- **Effect:** Instruction skips

- Glitcher
  - (delay, duration)
  - (t_2, d_2) = (2, 1)

- Timer
  - 33 MHz
  - 99 MHz

- Queue

- Host
  - *.log
  - *.py

- Target
  - tgt_clk
  - tgt_io
  - tgt_rst

- CPU

- 99 MHz (f_h)
  - 33 MHz (f_i)
  - gl_clk

- (t_1, d_1) = (2, 1)
Our attack: Schematic Hardware Setup

- Mechanism: CPU clock glitching
- Effect: Instruction skips

99 MHz ($f_h$)          33 MHz ($f_i$)  
gl_clk

\[(t_2, d_2) = (2, 1)\]
Our attack: Schematic Hardware Setup

- Mechanism: CPU clock glitching
- Effect: Instruction skips

Mechanism: CPU clock glitching

Effect: Instruction skips

- Mechanism: CPU clock glitching
- Effect: Instruction skips

99 MHz \((f_h)\)

33 MHz \((f_l)\)

gl_clk

\(t_1\)

\(d_1\)

\(t_2\)

\(d_2\)

Peter Günther (UPB) 2nd order fault attack on PBC FDTC 2014
Our attack: Schematic Hardware Setup

- Mechanism: CPU clock glitching
- Effect: Instruction skips

99 MHz \( (f_h) \)  
33 MHz \( (f_l) \)  
gl_clk

\( t_1 \) \( d_1 \) \( t_2 \) \( d_2 \)
Our attack: Real Hardware Setup
Our target: Eta pairing of Relic toolkit on AVR

- Target hardware: Atmel AVR Xmega A1
- Target Software: Relic toolkit
  - Open source
  - Prime and Binary field arithmetic
  - Elliptic curves over prime and binary fields (NIST curves and pairing-friendly curves)
  - Bilinear maps and related extension fields
  - Cryptographic protocols
- Combination used on wireless sensor nodes as TinyPBC
- Unmodified code
  - No additional NOPs
  - No monitors
  - No triggers
Input $P, Q \in \mathcal{E}$, $r = (r_n \ldots r_0)$

Output $f_{r,P}(Q)$

1: $T \leftarrow [2]P$
2: $\alpha \leftarrow l_{P,P}(Q) \cdot l_{(r-1)P,P}(Q)$
3: for $j \leftarrow n - 2, \ldots, 1$ do
   4:   if $r_j = 1$ then
      5:     $T \leftarrow T + P$
      6:     $\alpha \leftarrow \alpha \cdot l_{T,P}(Q)$
   7:   end if
8:   $T \leftarrow [2]T$
9:   $\alpha \leftarrow \alpha^2 \cdot l_{T,T}(Q)$
10: end for
11: $\alpha \leftarrow \alpha^d$
12: return $\alpha$
The RELIC implementation

Input $P, Q \in \mathcal{E}, r = (r_n \ldots r_0)$
Output $f_{r,P}(Q)$

1: $T \leftarrow [2]P$
2: $\alpha \leftarrow l_P, P(Q) \cdot l_{(r-1)P, P}(Q)$
3: for $j \leftarrow n-2, \ldots, 1$ do
4: if $r_j = 1$ then
5: $T \leftarrow T + P$
6: $\alpha \leftarrow \alpha \cdot l_{T, P}(Q)$
7: end if
8: $T \leftarrow [2]T$
9: $\alpha \leftarrow \alpha^2 \cdot l_{T, T}(Q)$
10: end for
11: $\alpha \leftarrow \alpha^d$
12: return $\alpha$
The RELIC implementation

Input \( P, Q \in \mathcal{E} \), \( r = (r_n \ldots r_0) \)

Output \( f_{r,P}(Q) \)

1. \( T \leftarrow [2]P \)
2. \( \alpha \leftarrow l_{P,P}(Q) \cdot l_{r-1}P,P(Q) \)
3. **for** \( j \leftarrow n - 2, \ldots, 1 \) **do**
   4. **if** \( r_j = 1 \) **then**
      5. \( T \leftarrow T + P \)
      6. \( \alpha \leftarrow \alpha \cdot l_T,P(Q) \)
   7. **end if**
   8. \( T \leftarrow [2]T \)
   9. \( \alpha \leftarrow \alpha^2 \cdot l_T,T(Q) \)
10. **end for**
11. \( \alpha \leftarrow \alpha^d \)
12. return \( \alpha \).
The RELIC implementation

Input $P, Q \in E, r = (r_n \ldots r_0)$

Output $f_{r,P}(Q)$

1: $T \leftarrow [2]P$
2: $\alpha \leftarrow l_{P,P}(Q) \cdot l_{(r-1)P,P}(Q)$
4: if $r_j = 1$ then
5:     $T \leftarrow T + P$
6:     $\alpha \leftarrow \alpha \cdot l_{T,P}(Q)$
7: end if
8: $T \leftarrow [2]T$
9: $\alpha \leftarrow \alpha^2 \cdot l_{T,T}(Q)$
11: $\alpha \leftarrow \alpha^d$
12: return $\alpha$

.L2

... call fb4_mul_dx
subi r16,1
sbc r17, __zero_reg__
breq .+2
subi r28,36
... movw r22,r28
movw r24,r28
call etat_exp
pop r29
...

Peter Günther (UPB) 2nd order fault attack on PBC FDTC 2014 8 / 16
The RELIC implementation

Input \( P, Q \in \mathcal{E}, r = (r_n \ldots r_0) \)
Output \( f_{r,P}(Q) \)

1: \( T \leftarrow [2]P \)
2: \( \alpha \leftarrow l_{P,P}(Q) \cdot l_{(r-1)P,P}(Q) \)
3: for \( j \leftarrow n-2, \ldots, 1 \) do
4: if \( r_j = 1 \) then
5: \( T \leftarrow T + P \)
6: \( \alpha \leftarrow \alpha \cdot l_{T,P}(Q) \)
7: end if
8: \( T \leftarrow [2]T \)
9: \( \alpha \leftarrow \alpha^2 \cdot l_{T,T}(Q) \)
10: end for
11: return \( \alpha \)

.L2
... 
call fb4_mul_dxs
sbc r17,___zero_reg___
breq .+2
subi r28,36
... 
movw r22,r28
movw r24,r28
pop r29
...
The RELIC implementation

**Input**  \( P, Q \in \mathcal{E}, r = (r_n \ldots r_0) \)

**Output**  \( f_{r,P}(Q) \)

1:  \( T \leftarrow [2]P \)
2:  \( \alpha \leftarrow l_{P,P}(Q) \cdot l_{r-1,P,P}(Q) \)

3:  \textbf{if}  \( r_j = 1 \) \textbf{then}
4:      \( T \leftarrow T + P \)
5:      \( \alpha \leftarrow \alpha \cdot l_{T,P}(Q) \)
6:  \textbf{end if}
7:  \( T \leftarrow [2]T \)
8:  \( \alpha \leftarrow \alpha^2 \cdot l_{T,T}(Q) \)

9: \textbf{return} \( \alpha \)

```assembly
.L2
...  
call fb4_mul_dx
subi r16,1
sbc r17,___zero_reg___
breq .+2
...  
subi r28,36
...  
movw r22,r28
movw r24,r28
...  
pop r29
...  
```
1. Output with successful glitch:

\[ \beta' = (l_{P,P}(Q) \cdot l_{(n-1)P,P}(Q))^2 \cdot l_{2P,2P}(Q) \]

2. Capture secret as root of polynomial of degree 5:

\[ f(x, y) = \beta' - (l_{P,P}(x, y) \cdot l_{(r-1)P,P}(x, y))^2 \cdot l_{2P,2P}(x, y) \]

3. Compute simultaneous roots of \( f(x, y) \) and \( E : y^2 = x^3 - x \)

4. Test candidates \( Q' \) against result of correct pairing:

\[ e(P, Q') = e(P, Q)? \]
Timing of first fault is critical

The challenge for 2nd order attack

- The timings $t_1$, $t_2$ of the target instructions depend on unknown secret (e.g. $Q$)
- It is not possible to detect case where only one glitch is successful
- Many combinations have to be tested

Our strategy

1. Profiling: Determine probability distribution of $t_1$ and $t_2$ for randomized secret input (e.g. $Q$)
2. Attack:
   - Rank candidates for $t_1$ and $t_2$ according to their probability
   - Introduce fault as early as possible
3. Analysis: Full Automation
Output with successful glitch:

$$\beta' = (l_{P,P}(Q),P(Q))^{2} \cdot l_{2P,2P}(Q)$$

Capture secret as root of polynomial of degree 5:

$$f(x, y) = \beta' - (l_{P,P}(x,y))^{2} \cdot l_{2P,2P}(x,y)$$

Compute simultaneous roots of $f(x, y)$ and $\mathcal{E}: y^2 = x^3 - x$

Test candidates $Q'$ for correctness of correct pairing:

$$e(P, Q') = e(P, Q)?$$
Performance of the attack

- In < 10 seconds per experiment (average):
  - Self-tests
  - Configure glitcher
  - Restart target
  - Induce faults
  - Analyze result

- More than 10000 experiments per day
Conclusion

- Second order attacks on pairings possible
- Two stage computation: not enough protection
- Add dedicated countermeasures as protection against active attacks
Problem

Final exponentiation cannot always be skipped:

- Inlining at higher optimization levels
- Countermeasures that guarantee execution of function

Example (Inlining at higher optimization levels)

```
... movw r22,r28
movw r24,r28
call etat_exp
pop r29
...  
... movw r22,r28
movw r24,r28
jmp etat_exp
nop
...  
```
Our approach

- Skip part of final exponentiation to modify exponent:

\[ d = \frac{p^k - 1}{r} \rightarrow d' = \frac{p^k - 1}{r} + \delta \]

- Simplify mathematical inversion of final exponent \( d' \)
References

- Relic toolkit:
  http://code.google.com/p/relic-toolkit/

- Glitcher Die Datenkrake:
  https://www.usenix.org/conference/woot13/workshop-program/presentation/nedospasov
Background

The basic building block

Bilinear mapping:

\[ e : \mathcal{E}(\mathbb{F}_{p^k}) \times \mathcal{E}(\mathbb{F}_{p^k}) \rightarrow \mathbb{F}_{p^k}^{*} \]

\[ (P, Q) \mapsto f_{n,P}(Q)^d \]

- \( n, d \) are huge
- \( f_{n,P}(x, y) \): zero of order \( n \) at \( P \), degree \( > n \)

Interesting properties for application in cryptography

- Bilinearity: \( e(aP, bQ) = e(P, Q)^{ab} = e(bP, aQ) \)
- Hard to invert
- \( f_{n,P}(Q) \) is efficiently computable with Miller algorithm
An example Application: IBE

**Encryption**

\[ B = e(sP, rH("Bob")) \]

To: "Bob"

\[ \langle rP, M*B \rangle \]

**Decryption**

\[ M*B/e(rP, sH("bob")) \]

To: "Bob"
An example Application: IBE

Encryption

\[ B = e(sP, rH("Bob")) \]

To: "Bob"

\(<rP, M*B>\)

Decryption

ID of Bob

\[ M*B/e(rP, sH("bob")) \]

To: "Bob"

\(<rP, M*B>\)

The secret decryption key is one argument of the pairing.
Miller Algorithm (Victor Miller 1986)
Extending the elliptic curve double and add algorithm

\[ y^2 = x^3 - x \]

**Input**  \( P, Q \in \mathcal{E}, n = (n_{t-1} \ldots n_0) \)

**Output**  \( f_{n,P}(Q) \)

1. \( \alpha \leftarrow 1, T \leftarrow P \)
2. **for** \( j \leftarrow t - 2, \ldots, 0 \) **do**
3. \( \alpha \leftarrow \alpha^2 \cdot l_{T,T}(Q)/v_{2T}(Q) \)
4. \( T \leftarrow 2T \)
5. **if** \( n_j = 1 \) **then**
6. \( \alpha \leftarrow \alpha \cdot l_{T,P}(Q)/v_{T+P}(Q) \)
7. \( T \leftarrow T + P \)
8. **end if**
9. **end for**
10. **return** \( \alpha^d \)
Miller Algorithm (Victor Miller 1986)
Extending the elliptic curve double and add algorithm

Input  $P, Q \in \mathcal{E}$, $n = (n_{t-1} \ldots n_0)$
Output  $f_{n,P}(Q)$

1: $\alpha \leftarrow 1$, $T \leftarrow P$
2: for $j \leftarrow t-2, \ldots, 0$ do
3: \hspace{1em} $\alpha \leftarrow \alpha^2 \cdot l_{T,T}(Q)/v_{2T}(Q)$
4: \hspace{1em} $T \leftarrow 2T$
5: \hspace{1em} if $n_j = 1$ then
6: \hspace{2em} $\alpha \leftarrow \alpha \cdot l_{T,P}(Q)/v_{T+P}(Q)$
7: \hspace{1em} $T \leftarrow T + P$
8: \hspace{1em} end if
9: end for
10: return $\alpha^d$

$y^2 = x^3 - x$
Miller Algorithm (Victor Miller 1986)
Extending the elliptic curve double and add algorithm

Input \( P, Q \in \mathcal{E}, n = (n_{t-1} \ldots n_0) \)
Output \( f_{n,P}(Q) \)

1: \( \alpha \leftarrow 1, T \leftarrow P \)
2: for \( j \leftarrow t - 2, \ldots, 0 \) do
3: \( \alpha \leftarrow \alpha^2 \cdot l_{T,T}(Q)/v_{2T}(Q) \)
4: \( T \leftarrow 2T \)
5: if \( n_j = 1 \) then
6: \( \alpha \leftarrow \alpha \cdot l_{T,P}(Q)/v_{T+P}(Q) \)
7: \( T \leftarrow T + P \)
8: end if
9: end for
10: return \( \alpha^d \)

\[ y^2 = x^3 - x \]
Different delays/instructions, same effect

.L2
...
call fb4_mul_dxs // sets zero flag: Z=1
subi r16,1 // re-set zero flag: Z=0
sbc r17, __zero_reg__ // Z=Z
breq .+2 // Z=1: branch
rjmp .L2
subi r28,36
...
Different delays/instructions, same effect

.L2
...
call fb4_mul_dxs
    // sets zero flag: Z=1
subi r16,1
    // re-sets zero flag: Z=0
sbc r17,__.zero_reg__
    // Z=Z
breq .+2
    // Z=1: branch

subi r28,36
...

Peter Günther (UPB)
2nd order fault attack on PBC
FDTC 2014 20 / 16
Different delays/instructions, same effect

.L2
...
call fb4_mul_dxs //sets zero flag: Z=1
subi r16,1 //re-sets zero flag: Z=0
sbc r17,__zero_reg__ //Z=Z
breq .+2 //Z=1: branch
rjmp .L2
subi r28,36
...
Different delays/instructions, same effect

.L2
...
call fb4_mul_dxS // sets zero flag: Z=1
sbc r17, __zero_reg__ // Z=Z
breq .+2 // Z=1: branch
rjmp .L2
subi r28, 36
...
1st order attack
Group delays by output and locate rjmp .L2

subi rjmp .L2

Profiling Output

Output $\beta_4$  Output $\beta_2$  Output $\beta_2$  Output $\beta_1$

$t_1$
1st order attack
Group delays by output and locate `rjmp .L2`

```
subi rjmp .L2
```

Output $\beta_4$ matches pattern of profiling
⇒ $t_1 = 6$ is instruction of `rjmp .L2`
Proceed with 2nd order attack and correct setting of $t_1$