Side-Channel Analysis and Countermeasure Design for Implementation of Curve448 on Cortex-M4

HASP 2022

Mojtaba Bisheh-Niasar\textsuperscript{1}, Mila Anastasova\textsuperscript{1}, Abubakr Abdulgadir\textsuperscript{2}, Hwajeong Seo\textsuperscript{3}, and Reza Azarderakhsh\textsuperscript{1,2}

\textsuperscript{1}CEECS Department, Florida Atlantic University, FL, USA
\textsuperscript{2}PQSecure Technologies, LLC, Boca Raton, FL, USA
\textsuperscript{3}Hansung University, South Korea

October 1, 2022
Outline

1 Introduction

2 Proposed Architecture

3 Implementation Results and Comparison

4 SCA Evaluation

5 Conclusion
Introduction

- **Motivation**
  - Complexities of Curve448 with extended field size on IoT devices
  - Secure implementation challenges due to SCA leakages
  - Importance of hybrid cryptosystems to transition to PQC
  - Lack of Curve448 implementation

- **Curve448 Protected Implementation**
  - Addressing backdoor issues in other ECC constructions [1]
  - Considering Safe-Curve policies [2]
  - Offering 224-bit security for applications at a higher security level

---


Background

- Curve448 architecture on Cortex-M4 by Seo et. al. [1]
  - Performed 26 scalar multiplications per second at 168 MHz
  - Utilized extended affine and projective coordinates
  - Investigated constant-time algorithms

- Curve448 implementations
  - FPGA implementations and SCA evaluation by Sasdrich et. al. and Bisheh-Niasar et. al.
  - Ed448 implementation on AVR, MSP by Seo et. al.
  - Ed448 implementation on FPGA by Bisheh-Niasar et. al.

- Improve field arithmetic with careful memory management
- Employ efficient restricted-X coordinates
- Embed advanced security mechanisms to avoid DPA attacks

Our Contributions

- Optimize low-level field arithmetics:
  - Carry/borrow catcher technique
  - Refined-Operand Caching method
  - Interleaved reduction technique

- Exploit the special form of Curve448 prime

- Implement and evaluate side-channel and fault injection (FI) countermeasures
Our Contributions

- Optimize low-level field arithmetics:
  - Carry/borrow catcher technique
  - Refined-Operand Caching method
  - Interleaved reduction technique

- Exploit the special form of Curve448 prime

- Implement and evaluate side-channel and fault injection (FI) countermeasures

- Reduce memory access
- 18% low-level speedup
- 40% total speedup

Prevent leakage at the cost of 8%-22% overhead

A trade-off between performance and required protection
Our Contributions

- Optimize low-level field arithmetics:
  - Carry/borrow catcher technique
  - Refined-Operand Caching method
  - Interleaved reduction technique

Reduce memory access
- 18% low-level speedup
- 40% total speedup

- Exploit the special form of Curve448 prime

- Implement and evaluate side-channel and fault injection (FI) countermeasures

- Prevent leakage at the cost of 8%-22% overhead
- A trade-off between performance and required protection
Cortex-M4 Platform

- NIST recommended STM32F407-VG platform:
  - ARMv7-M architecture
  - 16 32-bit core registers
  - 32 32-bit FP registers
  - 1 cycle per instruction except memory access

**Implementation strategies:**
- Use the entire register set.
- Operate on larger operand sets.
- Re-organize the instruction flow for efficient design.

STM32F407-VG Discovery board
Modular Addition/Subtraction

- Implement a carry/borrow catcher technique
- Use reduced instruction set
- Two arithmetic operations on long integers in parallel
- Alternate the add/sub blocks
- Reduce the number of memory accessing instructions

Diagram showing the carry/borrow catcher technique and the alternating of add/sub blocks.
Modular Multiplication

- Refined-Operand Caching method
- Multi-precision multiplication achieved by UMAAL instruction
- Reduce the number of pipeline stalls by removing the interdependency
- Caching width = 4 words (128-bit)

- $A = (A[13], \ldots, A[1], A[0])$
- $B = (B[13], \ldots, B[1], B[0])$
- $C = (C[27], \ldots, C[1], C[0])$

448-bit wise multi-precision multiplication.
Features 1MB of flash and 192KB of RAM - 128KB SRAM and 64KB CCM RAM.

The 128KB of SRAM - not enough to run Curve448 scalar multiplication.

Reserve a region inside the CCM RAM to place part of the large data structures, residing into the stack.
Implementation Results

Implementation results for Curve448 scalar multiplication on STM32F4 platform:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Seo et. al. [1]</td>
<td>24</td>
<td>564</td>
<td>6.218</td>
<td>259</td>
<td>3.9</td>
<td>-</td>
</tr>
<tr>
<td>Double-and-always-Add</td>
<td></td>
<td>788</td>
<td>5.269</td>
<td>219</td>
<td>4.6</td>
<td>15.2</td>
</tr>
<tr>
<td>Montgomery ladder</td>
<td></td>
<td></td>
<td>3.740</td>
<td>155</td>
<td>6.4</td>
<td>39.8</td>
</tr>
<tr>
<td>Seo et. al. [1]</td>
<td>168</td>
<td>564</td>
<td>6.286</td>
<td>37.4</td>
<td>26.7</td>
<td>-</td>
</tr>
<tr>
<td>Double-and-always-Add</td>
<td></td>
<td>788</td>
<td>5.532</td>
<td>32.9</td>
<td>30.4</td>
<td>12.0</td>
</tr>
<tr>
<td>Montgomery ladder</td>
<td></td>
<td></td>
<td>3.917</td>
<td>23.3</td>
<td>42.9</td>
<td>37.6</td>
</tr>
</tbody>
</table>

Montgomery ladder:
- 29% performance improvement over the restricted X-coordinate
- 40% memory utilization penalty

1.6× speedup (43 scalar multiplications per second) at 168 MHz

Implementation Results

Implementation results on embedded processors:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>pre/post quantum</th>
<th>Cortex</th>
<th>Freq [MHz]</th>
<th>Latency [CC×10^3]</th>
<th>Time [ms]</th>
<th>Throughput [op/sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curve25519 [Fujii et. al.]</td>
<td>pre</td>
<td>M4</td>
<td>48</td>
<td>907</td>
<td>18.9</td>
<td>52.9</td>
</tr>
<tr>
<td>Secp256r1 [Lenngren]</td>
<td>pre</td>
<td>M4</td>
<td>64</td>
<td>994</td>
<td>15.5</td>
<td>64.3</td>
</tr>
<tr>
<td>FourQ [Liu et. al.]</td>
<td>pre</td>
<td>M4</td>
<td>168</td>
<td>511</td>
<td>3.0</td>
<td>328.8</td>
</tr>
<tr>
<td>Secp384r1 [Tschofenig et. al.]</td>
<td>pre</td>
<td>M3</td>
<td>100</td>
<td>20,200</td>
<td>202</td>
<td>4.9</td>
</tr>
<tr>
<td>Secp521r1 [Tschofenig et. al.]</td>
<td>pre</td>
<td>M3</td>
<td>100</td>
<td>35,100</td>
<td>351</td>
<td>2.8</td>
</tr>
<tr>
<td>SIKEp434 [Anastasova et. al.]</td>
<td>post</td>
<td>M4</td>
<td>24</td>
<td>68,260</td>
<td>2,844</td>
<td>0.3</td>
</tr>
<tr>
<td>Curve448 [Ours]</td>
<td>pre</td>
<td>M4</td>
<td>168</td>
<td>3,917</td>
<td>23.3</td>
<td>42.9</td>
</tr>
</tbody>
</table>

Higher security levels come with a performance penalty

Possibility of algorithmic improvements to reduce the required computation to break ECC.

Moving to a higher security level to keep a margin against unknown attack improvements
SCA-Protected Performance Results

Protected Cortex-M4 implementation results:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Unprotected</td>
<td>3.917</td>
<td>23.3</td>
<td>42.9</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Point Randomization</td>
<td>4.222</td>
<td>25.1</td>
<td>39.8</td>
<td>7.8</td>
<td></td>
</tr>
<tr>
<td>Scalar Blinding</td>
<td>4.417</td>
<td>26.3</td>
<td>38.0</td>
<td>12.7</td>
<td></td>
</tr>
<tr>
<td>Both countermeasures</td>
<td>4.789</td>
<td>28.5</td>
<td>35.1</td>
<td>22.2</td>
<td></td>
</tr>
</tbody>
</table>

- Montgomery ladder: countermeasure against timing, SPA, and sign change fault attacks
- Base point randomization: add one multiplication per ladder step
- Scalar blinding:
  - Countermeasure to avoid DPA, cross-correlation, safe-error, and differential fault analysis attacks
  - Extend the number of ladder step iterations
- Flow-counter countermeasure: protection against FI loop-abort attacks with negligible latency overhead
SCA Evaluation

- NewAE CW308T-STM32F
- Cortex-M4 at 25 MHz
- Capturing power traces via Picoscope 3000
- Sampling rate of 125 MS/s
- TVLA with pool of 10,000 traces

\[
\alpha = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}
\]

Leakage detection test on Curve448 after applying TVLA with a pool of 10,000 measurements:
(Up) \( t \)-test values for an unprotected implementation,
(Down) \( t \)-test values by enabling both countermeasures
Conclusion and future works

- Conclusion:
  - Implementing a secure design of Curve448 targeting a 224-bit security level for the 32-bit ARM Cortex-M4 architecture
  - Reducing the latency of scalar multiplication to 23 milliseconds at 168 MHz
  - Embedding different effective countermeasures at the cost of 8%-22% overhead
  - Evaluating our SCA protection with TVLA over 10,000 power measurements

- Future work:
  - Extending the design by fault attack countermeasures
  - Applying the method on Ed448
Thanks for your attention.