Efficient Hardware Implementations for Elliptic Curve Cryptography over Curve448

Mojtaba Bisheh Niasar\(^1\), Reza Azarderakhsh\(^1,2\), Mehran Mozaffari Kermani\(^3\)

\(^1\)CEECS Department, Florida Atlantic University
\(^2\)PQSecure Technologies, LLC, Boca Raton, FL, USA
\(^3\)CSE Department at University of South Florida

13-16 December 2020

IndoCrypt 2020
1. Introduction

2. ECDH Protocol

3. Proposed Architecture over Curve448

4. Implementation Results and Comparison

5. Conclusion
Introduction

- **Motivation**
  - Recently standardized by NIST
  - Offering higher security level after Curve25519 [1]
  - Suitable for hybrid key exchange with SIKEp434

- **Curve448**
  - Designed by Hamburg in 2015 [2]
  - Belongs to Safe-Curve [3]
  - Software-based design
  - Montgomery curve and also an untwisted Edwards curve (Edwards448) [4]

---


Background

- Work by [1] and [2] based on schoolbook multiplication
- LUT-based scheme [3]

Gaps:
- Few hardware implementations
- Exploration of the different trade-offs between resource utilization and performance
- Employing the Karatsuba-friendly property of Curve448

Our Contributions

- Investigate three design strategies:
  - **Lightweight architecture** targets area-constrained applications.
  - **High-performance architecture** targets time-constrained applications.
  - **Area-Time Efficient architecture** targets area-time trade-off applications.

- Employing various optimization techniques to increase efficiency
  - Refined Karatsuba multiplication
  - Redundant number presentation
  - Interleaved multiplication

- Performing a precise schedule corresponding to each architecture
- Variable-base-point multiplications
- Extended by side-channel countermeasures
X448 Computation

- ECPM: elliptic curve point multiplication $Q = k \cdot P$ over Curve448
- Finite field arithmetic with $p = 2^{448} - 2^{224} - 1$
- All arithmetic is performed on Montgomery ladder for efficiency
  - point addition (PA)
  - point doubling (PD)
- Employing projective coordinate

\[
\begin{align*}
X_{PD} &= (X_1 - Z_1)^2 \cdot (X_1 + Z_1)^2 \\
Z_{PD} &= 4X_1Z_1 \cdot (X_1^2 + 39081X_1Z_1 + Z_1^2) \\
X_{PA} &= 4(X_1X_2 - Z_1Z_2)^2 \\
Z_{PA} &= 4x_p(X_1Z_2 - Z_1X_2)^2
\end{align*}
\]
Montgomery ladder

Data dependency diagram for one step Montgomery ladder execution over Curve448

Latency (clock cycles)

S0: A
S1: M
S2: M
S3: M
S4: M
S5: M
S6: M
S7: M
S8: M
S9: M
S10: M
Total: A+10M

Addition
Subtraction
Multiplication
Squaring

Bisheh Niasar et al.
IndoCrypt 2020
X448 Side-Channel Protection

- Constant-time implementation against timing attack
- Secret-independent implementation against simple power analysis (SPA)
- Countermeasures against differential power analysis (DPA) attacks [1]
  - Point Randomization
    \[ P = (X, Z) \rightarrow P_r = (\lambda \cdot X, \lambda \cdot Z) \]
  - Scalar Blinding
    \[ Q = k \cdot P \rightarrow Q = k_r \cdot P \]

Hardware Architecture for Different Performance Levels over Curve448

Design I

Design II

Design III

Controller (FSM)

Program ROM

Modular Multiplication

Modular Addition

Dual Port RAM

P (base point)
Lightweight Architecture

- A 448-bit data is stored by splitting to 28 words in 16-bit chunks due to DSP block input size
- 16-bit datapath
- Program ROM consists of 2183 instruction lines
- Modular Addition/Subtraction:
  - Sequential addition by propagating carry/borrow to the next digit
- Modular Multiplication:
  - Based on the product scanning approach
  - Utilizing only one DSP
  - Interleaved reduction
Area-Time Efficient Architecture

- A 448-bit data is stored by splitting to 4 words in 112-bit chunks
- 128-bit datapath
- Program ROM consists of 480 instruction lines
- Fast Karatsuba multiplication [1] with golden ratio $\phi = 2^{224}$

$$A \times B = (A_0 B_0 + A_1 B_1) + 2^{224}(A_{10} B_{10} - A_0 B_0)$$

- Refined Karatsuba identity [2]

$$(a_0 + a_1 t^n) \cdot (b_0 + b_1 t^n) = (1 - t^n) \cdot (a_0 b_0 - t^n a_1 b_1) + t^n(a_0 + a_1) \cdot (b_0 + b_1)$$

Modular Multiplication:

- 64×64-bit multiplication using 16 DSPs

- High throughput:
  - a 64×64-bit per cycle
  - a 128×128-bit per 4 cycles
  - \( A_0 B_0 \) in 12 cycles

\[
A_0 B_0 = (1 - 2^{112}) \cdot (a_0 b_0 - 2^{112} a_1 b_1) + 2^{112} (a_0 + a_1) \cdot (b_0 + b_1) \\
= (1 - 2^{112}) \cdot (a_0 b_0 - 2^{112} a_1 b_1) + 2^{112} (a_{10} b_{10})
\]
(a) Optimized middle-level recombination: only five highlighted operations are performed.

(b) Optimized top-level recombination: applying interleaved reduction cancels the highlighted digits.
High-Performance Architecture

- Full 448-bit datapath
- Program ROM consists of 1554 instruction lines
- 5-level consecutive Karatsuba multiplication
  - reducing considerably the required cycles at the cost of expanding addition tree
- The first level is designed pipeline due to the number of available DSP blocks
  - \( Ch = A_1 B_1, \ Cl = A_0 B_0, \) and \( Cm = A_{10} B_{10} \)
- The rest of the multiplications are performed parallel using 81 DSP blocks
  - 5 clock cycles for all three required values
Performance Results

FPGA implementation results for different performance level architectures in terms of clock cycles and latency requirements

<table>
<thead>
<tr>
<th>Proposed Architecture</th>
<th>Design I</th>
<th>Design II</th>
<th>Design III</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Freq. [MHz]</td>
<td>250</td>
<td>127</td>
<td>95</td>
</tr>
<tr>
<td>(F_p)Operation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Addition</td>
<td>112</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>Subtraction</td>
<td>112</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>Multiplication</td>
<td>1,233</td>
<td>69</td>
<td>15</td>
</tr>
<tr>
<td>Inversion</td>
<td>620,047</td>
<td>32,272</td>
<td>6,917</td>
</tr>
<tr>
<td>Mont. step</td>
<td>14,346</td>
<td>708</td>
<td>158</td>
</tr>
<tr>
<td>Point Multiplication</td>
<td>7,047,055</td>
<td>349,546</td>
<td>77,702</td>
</tr>
<tr>
<td>Latency [ms]</td>
<td>28.19</td>
<td>2.75</td>
<td>0.82</td>
</tr>
</tbody>
</table>
## FPGA implementation results for different ECPM cores above 128-bit security

<table>
<thead>
<tr>
<th>Work</th>
<th>SCA</th>
<th>Area</th>
<th>Time</th>
<th></th>
<th></th>
<th>Total time</th>
<th>OP/s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LUTs</td>
<td>FFs</td>
<td>Slices</td>
<td>DSPs</td>
<td>BRAMs</td>
<td>Latency [CCs]</td>
</tr>
<tr>
<td>NIST P-384 (192-bit security)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alrimeih et al. (2014)</td>
<td>(+)</td>
<td>32900</td>
<td>-</td>
<td>11,200</td>
<td>289</td>
<td>128</td>
<td>-</td>
</tr>
<tr>
<td>Ananyi et al. (2009)</td>
<td>(-)</td>
<td>31946</td>
<td>-</td>
<td>20,793</td>
<td>32</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>NIST P-521 (260-bit security)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alrimeih et al. (2014)</td>
<td>(+)</td>
<td>32900</td>
<td>-</td>
<td>11,200</td>
<td>289</td>
<td>128</td>
<td>-</td>
</tr>
<tr>
<td>Ananyi et al. (2009)</td>
<td>(-)</td>
<td>31946</td>
<td>-</td>
<td>20,793</td>
<td>32</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>Curve448 (224-bit security)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sasdrich et al. (2017)</td>
<td>(-)</td>
<td>2,555</td>
<td>7,049</td>
<td>1,580</td>
<td>33</td>
<td>14</td>
<td>328,286</td>
</tr>
<tr>
<td>(-) Sasdrich et al. (2017)</td>
<td>(+)</td>
<td>3,583</td>
<td>7,423</td>
<td>1,648</td>
<td>35</td>
<td>14</td>
<td>473,926</td>
</tr>
<tr>
<td>Sasdrich et al. (2018)</td>
<td>(+)</td>
<td>4,624</td>
<td>8,209</td>
<td>1,985</td>
<td>33</td>
<td>14</td>
<td>499,344</td>
</tr>
<tr>
<td>Sasdrich et al. (2018)</td>
<td>(+)</td>
<td>4,624</td>
<td>8,209</td>
<td>1,985</td>
<td>33</td>
<td>14</td>
<td>547,728</td>
</tr>
<tr>
<td>Shah et al. (2020)</td>
<td>(-)</td>
<td>50,143</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>372,742</td>
</tr>
<tr>
<td>Design I (-)</td>
<td></td>
<td>321</td>
<td>174</td>
<td>137</td>
<td>1</td>
<td>2</td>
<td>7,047,055</td>
</tr>
<tr>
<td>Design I (+)</td>
<td></td>
<td>509</td>
<td>438</td>
<td>203</td>
<td>1</td>
<td>2</td>
<td>12,068,239</td>
</tr>
<tr>
<td>Design II (-)</td>
<td></td>
<td>2,233</td>
<td>1,152</td>
<td>760</td>
<td>16</td>
<td>9</td>
<td>349,546</td>
</tr>
<tr>
<td>Design II (+)</td>
<td></td>
<td>2,587</td>
<td>1,629</td>
<td>842</td>
<td>16</td>
<td>9</td>
<td>602,801</td>
</tr>
<tr>
<td>Design III (-)</td>
<td></td>
<td>13,132</td>
<td>4,035</td>
<td>4,354</td>
<td>81</td>
<td>0</td>
<td>77,702</td>
</tr>
<tr>
<td>Design III (+)</td>
<td></td>
<td>13,415</td>
<td>4,610</td>
<td>4,424</td>
<td>81</td>
<td>0</td>
<td>133,254</td>
</tr>
</tbody>
</table>
Efficiency comparison between FPGA-based ECPM architectures in terms of $A \cdot T$ (Area $\times$ Time) in a fixed low-frequency.

**red:** unprotected, **blue:** protected, **black:** high-protected
Comparison

- **Design I**
  - Saving 96% of resources with the competitive performance

- **Design II**
  - Improving 48% and 50% efficiency compared to [1] and [2]
  - Occupying 52% fewer resources

- **Design III**
  - Improving efficiency by 40% and 43% compared to [1] and [2]
  - Increasing 12% throughput


Conclusion

- Three hardware implementations for security level 224-bit
  - area-constrained applications
  - time-constrained applications
  - area and time trade-off applications

- Implemented on a mid-range Xilinx FPGA XC7Z7020

- Extended by side-channel countermeasures

- Computing 1219, 363, and 35 ECDH operations per second.
Thanks for your attention.