Efficient implementation of Elliptic Curve Cryptosystems over binary Galois Fields, GF(2^m), in normal and polynomial bases

ECE746, Final Project
Matthew Estes and Philip Hines
Contents

- Background
- Algorithm Selection
- Library Design Requirements
- Results
- Lessons Learned
- Conclusions
Background: ECC

- Elliptic Curve Cryptography
  - Based on the calculation of points along an elliptic curve in a finite field.
Background: Bases

Elliptic Curves

GF(p) – Prime Field

GF(2^m) – Binary Field

Polynomial Basis

Normal Basis

Optimal Normal Basis (ONB)

ONB Type 1

ONB Type 2
Background: Bases

- Polynomial Basis –
  \( f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_i x^i \)

- Normal Basis –
  \( f(x) = a_0 + a_1 x^p + a_2 x^{p^2} + a_3 x^{p^3} + a_i x^{p^i} \)

- Optimal Normal Basis is determined by field length \( m \) (in bits). Simplified Rules for ONB:
  - Type 1
    - 2 must be primitive in field \( Z^{(m+1)} \)
    - \( m+1 \) is prime
  - Type 2
    - 2 must be primitive in field \( Z^{(2m+1)} \)
    - \( 2m+1 \) is prime
Background: Elliptic Curve Layers

Cryptographic Protocols

ECC Operations
- Scalar Multiplication
  - Point Addition
  - Point Doubling
  - A2P or P2A

GF Operations
- Interchangeable
  - Inversion
  - Division
  - Addition
  - Multiplication
  - Squaring

Interchangeable operations are highlighted in green.
Algorithm Selection

- **Normal Basis Operations**
  - Rosing’s Fast Optimal Normal Basis algorithm.
  - Han/Dai Algorithm #1

- **Polynomial Basis Operations**
  - Selected Windowed Left-to-Right Combine for Multiplication
  - Selected an optimized Euclidean based algorithm for inversion
  - Selected custom reduction formulas optimized for NIST trinomials and pentanomials
Library Design Requirements

- Must support interchangeable operations for different bases
- Must abstract between field operations and elliptic curve operations
- Must be able to support multiple field sizes
- Must support structures allowing multiple instantiation (allowing for parallel operations)
Library Code Layers

Cryptographic Protocols

ECC Operations

Scalar Multiplication

Point Addition
Point Doubling
A2P or P2A

Point Subtraction
Point Copy
Point Solver $x \rightarrow y[2]$

GF Operations

Interchangeable

Add $a+b$
Add $a+b+c$
Add $a+b+c+d$
Add $a+b+c+d+e$
Add $a+b$ and $c+d$

Multiplication

Inversion

Squaring

Initialize
Precompute
Copy
Initialize
Fill Field

Initialize
Precompute
Copy
Initialize
Fill Field
Library Structures

To support multiple field lengths, all field operations and elliptic curve operations must be able to store/pass the following:

- **Field Parameters** – number of bits in field, words in a field, length of words in bits, etc.
- **Temporary space** – pre-allocated space of the correct field size for operations to use
- **Length Specific Structures** - Reduction Polynomials, Optimization Tables, and other length specific structures.
Library Usage

- Include “Ecclib.h”
- Choose Curve:
  \[ \text{Curve}_\text{ID} = \text{NORM}_\text{T2}_585; \]
- Initialize Ecclib:
  \[
  \text{FieldData}_\text{t}^* \ pFD = \text{Ecclib}_\text{Init}(\text{Curve}_\text{ID});
  \]
  - Creates Data Structures
  - Initializes Optimization Tables
  - Chooses Function Pointers for curve type:
    \[
    pFD->\text{pfnSquare} = \text{onb}_\text{sqr}_\text{copy};
    \]
Results

- The following slides will cover:
  - Inversion Algorithms
  - Polynomial Basis
  - Rosing ONB Sum
  - Rosing ONB Breakout
  - Polynomial vs. Normal Multiplication (including Fan Dai Algorithm)
  - Polynomial vs. Normal Inversion
    - (Fan Dai = Rosing = Almost Inverse)
  - Best Ecclib (Poly) vs. OpenSSL
Results: Inversion Algorithm

Inversion Algorithm Performance

Field Size (NIST Binary Fields)

Time (µs)

Itoh-Tsujii  Euclidean  Almost Inverse  Binary
Results: Polynomial Operations

Polynomial Algorithm Performance

Field Size (NIST fields)

Time (µs)

Addition
Squaring
Multiplication
Inversion
Results: Rosing ONB Sum

Why are some field sizes faster than others?
Results: Rosing ONB Breakout

Sum of time (us)

<table>
<thead>
<tr>
<th>type</th>
<th>operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONB 1</td>
<td>Inversion</td>
</tr>
<tr>
<td>ONB 2</td>
<td>Inversion</td>
</tr>
<tr>
<td></td>
<td>Multiplication</td>
</tr>
</tbody>
</table>

Legend:
- 1 - addition
- 1 - inversion
- 1 - multiplication
- 1 - squaring
- 2 - addition
- 2 - inversion
- 2 - multiplication
- 2 - squaring
- (blank) - (blank)
Results: Polynomial vs. Normal Multiplication

![Graph showing the comparison between Polynomial and Normal multiplication methods across different field sizes.]

- **Polynomial**
- **FanDai 1**
- **FanDai 2**
- **Rosing 1**
- **Rosing 2**
- 2 per. Mbx. Avg. (Poly - 0)

The graph illustrates the time (in microseconds) taken for multiplication across varying field sizes (in bits) for different methods.
Results: Polynomial vs. Normal Inversion

![Graph showing polynomial vs. normal inversion results.](Image)
Results: Ecclib vs. OpenSSL

ECC Scalar Multiplication

Field Size vs. Time (ms) for OpenSSL-Koblitz, OpenSSL-Random, Ecclib-Koblitz, Ecclib-Random.
Results: ECC ONB vs. Polynomial

![Graph showing the comparison of ECC ONB vs. Polynomial for various field sizes. The graph displays the time (in milliseconds) taken for different field sizes, with ONB Type 1, ONB Type 2, and Polynomial methods. The x-axis represents the field size, while the y-axis represents the time. The graph shows a clear increase in time for larger field sizes.]
Results: ECC ONB vs. Polynomial

![Graph showing the comparison between ECC ONB and Polynomial methods for different field sizes. The x-axis represents the field size, and the y-axis represents time in milliseconds. The graph compares ONB Type 1, ONB Type 2, and Polynomial methods. The ONB Type 2 method appears to have the fastest performance, followed by ONB Type 1, and then Polynomial.](image_url)
Lessons Learned

- Rosing and Fan/Dai ONB Code
  - Complicated Optimizations
  - Built as test code... extensive use of global variables and #defines
  - Implemented static at 113 bit fields
  - Mixed GF operations and EC operations
- Saving time initially by creating code not designed for flexibility cost more time in the end.
Conclusions

- Our Optimized Polynomial Basis implementation outperforms Rosing and Fan/Dai Normal Basis implementations significantly.
- Our Inversion implementations in ONB Type I outperform our Polynomial Basis inversion.
- Scalar multiplication over the polynomial basis outperforms the same over the ONB by one order of magnitude using our library.
- Our implementation of ECC scalar multiplication outperforms OpenSSL by approximately 50%.
- OpenSSL, LiDIA, MIRACL, Crypto++, and Borzoi all do not support Normal Basis field operations. Few, if any, implementations of Normal Basis libraries.
- Many theoretical algorithms are coded using inflexible coding practices.
Questions?
References

- (many others included in report)