Efficient Embedded Software Implementations of Public-Key Cryptography

Çetin Kaya Koç
koc@cs.ucsb.edu
Cryptographic Engineering 2012
Characteristics of Embedded Systems

- Low Computing Power
- Low Dynamic Memory for Data
- Restricted Stack Memory Usage
- Diverse Memory Allocation Techniques
- Different Memory Management & Organization
- Limitations on Program Code Size
- Difficulty in handling exceptions on System Calls
Challenges of Public Key Cryptography in Embedded Systems

- High Latency of Public Key Operations
- Excessive Demand on Dynamic Memory
- High Stack Memory Requirement
- High footprint
- Exception handling
- Secure Execution
- Removal of Sensitive Information from Memory
Elliptic Curve Cryptography

- Efficient field arithmetic
- Finite Field operations
- Point Operations
- Cryptographic schemes
- New techniques in curve arithmetic
- Fewer field operations
- Fast Signature & Small Implementation
Finite Fields for ECC Usage

- Prime fields, GF(p)
- Binary extension fields, GF(2^k).
- General Extension fields GF(p^k) - Not as common

**The Focus:** GF(p) & GF(2^k)

**The Goal:** Fast execution of finite field operations in embedded environment requiring low memory and footprint
Finite Field Operations in ECC

- Addition in GF(p) and GF(2^k)
  Inexpensive in terms of time and area
- Multiplicative inversion in GF(p) and GF(2^k)
  Prohibitively expensive in terms of time
  Possible to avoid some of them
- Multiplication in GF(p) and GF(2^k)
  Expensive in terms of time and area
  Most important operation
  Focus of many design efforts
Incomplete Arithmetic in GF(p)

- An effective method to enhance the time performance of arithmetic operations in software particularly when the field degree is not on the word boundary of underlying processor.
- For example, $2^{176} > p$ and $2^{175}$ when the word size of the computer is 32 bits.
- Based on a technique eliminating the need for reduction when the intermediary result does not exceed the word boundary.
- Up to 13% overall speedup.
Addition & Subtraction in GF(p)

- Takes up 15% plus of total computation time in an EC point multiplication operation with projective coordinates.
- Implemented in Assembly which improves both operations at least 100%.
- Incomplete arithmetic provides up to 40% and 25% performance improvement for addition and subtraction, respectively.
Multiplication in GF(p)

- Takes up 70% plus of total computation time in an EC point multiplication operation with projective coordinates
- Assembly implementation improves 150%
- Incomplete arithmetic provides up to 5%
- CIOS Montgomery algorithm seems to be the best performer when the prime p is arbitrarily chosen
Multiplicative Inversion in $\text{GF}(p)$

- Very important operation when affine EC coordinates are used
- The Montgomery inversion algorithm is used
- A new correction phase, which provides 1.5 overall speedup, is utilized
- No assembly is used
Arithmetic Operations in GF(2^k)

- No need to use Assembly language since there is no carry handling
- The subtraction operation is the same as the addition
- Addition is a trivial operation which does not need any special treatment in software
- All operations can be optimized for irreducible trinomials or pentanomials of any type
Multiplication in GF(2^k)

- Takes up about 70% of total computation time in an EC point multiplication operation with projective coordinates.
- Montgomery’s algorithm is not used for trinomials or pentanomials.
- Consists of two phases:
  - Polynomial Multiplication
  - Reduction with the irreducible polynomial.
Multiplication Phase

- Non-recursive Karatsuba-Offman algorithm is quite effective since redundant operations are eliminated.
- More improvement is possible with an increase in the code size and with acceleration tables.
- Standard Karatsuba-Offman algorithm provides around 25% overall speedup.
Reduction Phase

- Standard reduction method is the best when trinomials or pentanomials are used.
- $\text{GF}(2^k)$ version of Montgomery’s method performs better when randomly chosen irreducible polynomials are used.
- Acceleration tables of different amounts can provide different levels of speedup.
Elliptic Curve Point Operations

- Consists of a number of finite field operations
- For speed, mixed modified Jacobian coordinates offer the maximum performance
- For processors of low resources, affine coordinates provide a lean version of ECC
- Different acceleration techniques provide different levels of speedup at the expense of memory
- Simultaneous point multiplication with redundant representation improves signature verification
Special Curve Solutions

- All operations (field & point arithmetic) can be highly optimized if fixed special curves are chosen.
- For $\text{GF}(p)$, the speedup is up to 100% over the standard implementation.
- For $\text{GF}(2^k)$, the speedup is about 50%.
- For $\text{GF}(2^k)$, the speedup can increase as high as 400% if Koblitz curves are used.
Desired Characteristics of the Embedded Software (1)

- **Reentrant**
  A computer program or routine is described as reentrant if it is designed such that a single copy of the program's instructions in memory can be shared by multiple users or separate processes. The key to the design of a reentrant program is to ensure that no portion of the program code is modified by the different users/processes, and that process-unique information (such as local variables) is kept in a separate area of memory that is distinct for each user or process.

- **Thread-safe**
  A piece of code is thread-safe if it is reentrant or protected from multiple simultaneous execution by some form of mutual exclusion.
Desired Characteristics of the Embedded Software (2)

- Modularity and OS independence
- No OS or C calls within the library
- Low stack usage (under 1K is typical)
- Streamlined dynamic memory usage
- All memory allocation should (can) be made at one point by user
- Allocation can be made automatic with either user-supplied or standard memory manager
Desired Characteristics of the Embedded Software (3)

- Memory requirement for any curve can be reported by a function before any cryptographic operation is performed.
- Supports any curve of infinite precision with highly optimized manner.
- Configurable RNG.
- Safe execution.
- Removes any sensitive information from the memory after the cryptographic computations.
A set of acceleration techniques provide different levels of speedup.

Dynamic memory is divided into two parts: curve-specific or temporary. This feature allows several possibilities:

- Curve-specific memory can be shared between different threads using the same curve.
- Curve-specific memory can be allocated and initialized offline.
Performance on 80-MHz ARM7TDMI

<table>
<thead>
<tr>
<th>Bitsize</th>
<th>Signature (ms)</th>
<th>Memory (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PC0</td>
<td>PC1</td>
</tr>
<tr>
<td>192 – GF(p)</td>
<td>70</td>
<td>28</td>
</tr>
<tr>
<td>256 – GF(p)</td>
<td>145</td>
<td>53</td>
</tr>
<tr>
<td>163 – GF(2&lt;sup&gt;k&lt;/sup&gt;)</td>
<td>105</td>
<td>40</td>
</tr>
<tr>
<td>257 – GF(2&lt;sup&gt;k&lt;/sup&gt;)</td>
<td>274</td>
<td>109</td>
</tr>
</tbody>
</table>

- Random curves are used. Further improvement is possible with fixed curves
- PC0: No precomputation
- PC1: Precomputation Level 1, etc.
Performance on 20-MHz PALM

<table>
<thead>
<tr>
<th>Bitsize</th>
<th>Signature (s)</th>
<th>Memory (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PC0</td>
<td>PC1</td>
</tr>
<tr>
<td>192 – GF(p)</td>
<td>5.1</td>
<td>2.1</td>
</tr>
<tr>
<td>256 – GF(p)</td>
<td>11.1</td>
<td>4.3</td>
</tr>
<tr>
<td>163 – GF($2^k$)</td>
<td>4.9</td>
<td>2.0</td>
</tr>
<tr>
<td>257 – GF($2^k$)</td>
<td>12.0</td>
<td>4.6</td>
</tr>
</tbody>
</table>

- Random curves are used. Further improvement is possible with fixed curves
- PC0: No precomputation
- PC1: Precomputation Level 1, etc.
- Precomputation is restricted due to limited memory
References