MASTER THESIS PRESENTATION
Leuven, July 22nd 2008

SMART CARD IMPLEMENTATION OF ANONYMOUS CREDENTIALS

Josep Balasch
Escola Tècnica Superior d’Enginyeria de Telecomunicacions de Barcelona (UPC)
OUTLINE

1. Introduction
2. AVR Microcontrollers
3. Math and Software Specifications
4. Large Integers
5. Basic and Modular Arithmetic
6. Other Implementations
7. Measurements and Conclusions
INTRODUCTION (I)

- Anonymous Credential System
  - $S_U$: User’s master secret
  - $O_I$: Issuer’s key pair
  - $O_V$: Verifier’s key pair
  - $N$: User’s pseudonym
  - attr: Credential attributes

- Concerns:
  - Misuse of Credentials
  - Storage of Secret Keys

- Possible solution:
  - Tamper-resistant embedded devices
INTRODUCTION (II)

- Direct Anonymous Attestation (DAA)
  - Used by the Trusted Platform Module (TPM)
  - Outsourcing of non-critical operations to the Host
  - Goal: Implement a simplified version of the DAA Signing Protocol into a smart card device
ATMega2560 SMART CARD

- AVR 8-bit microcontrollers
  - High performance – low power
  - 32 general purpose registers
  - Same core architecture

- ATMega2560
  - 8 KBytes SRAM, 4 KBytes EEPROM
  - Set of 135 assembly instructions
  - Clock up to 16 MHz
  - No hardware arithmetic

- AVR Studio
  - Editor and simulator for AVR applications
  - Programming languages: C and assembly
SMART CARD IMPLEMENTATION OF ANONYMOUS CREDENTIALS

MATH SPECIFICATION

Signature of Knowledge

\[ SPK\{(e,v,f) : ZT_1^{-2^{l\epsilon - 1}} \equiv \pm T_1^e S^v R^f \ (mod\ n) \land f \in \{0,1\}^{l_f + l_\phi + l_H + 1} \land e \in \{0,1\}^{l_e + l_\phi + l_H + 2}\} (n_v || n_e || m) \]

\( (f,v) \quad \text{Secret values} \)

\( (R,S,n) \quad \text{Issuer’s Public Key} \)

2 random numbers

1 modular multiexponentiation

1 random number

2 hash computations

2 additions

2 multiplications
SOFTWARE SPECIFICATION

- **High-level functions**
  - RANDOM NUMBER GENERATOR
    - \( n \rightarrow \{0,1\}^n \)
  - HASH FUNCTION
    - \( m_1 \rightarrow H(m_1||m_2) \)
  - MULTIEXPONENTIATION
    - \( A \rightarrow A^x B^y \mod n \)
  - MULTIPLICATION & ADDITION
    - \( a \rightarrow a + b \cdot c \)

- **Bottom-up design approach**
  - Large integers
  - Basic and modular arithmetic
  - Other implementations
  - High-level functions
LARGE INTEGERS

- Structure for large integers: $\text{BigNum}$
  - Pointer to the first digit in memory
  - Number of digits

- Example: $0x111122 \ldots \text{EEFFFF}$ (256-bit)

- Word size $w = 8$ bits, Radix $b = 2^w = 2^8$
- Memory storage: Little Endian
- Memory positions multiples of 5 words
BASIC ARITHMETIC (I)

- Addition / Subtraction
  - Pencil-and-paper method (digit by digit)
  - Carry handling for each computation

- Division
  - Slowest of all basic arithmetic operations
  - Solution: Minimize the use of this operation in upper layers

- Multiplication
  - Most used basic arithmetic operation
  - Solution: Assembly low-level routine
BASIC ARITHMETIC (II)

🔗 Basic multiplication algorithms

**ROW WISE MULTIPLICATION**

```
  a2 a1 a0
  b2 b1 b0
```

**COLUMN WISE MULTIPLICATION**

```
  a2 a1 a0
  b2 b1 b0
```

```
  r5 r4 r3 r2 r1 r0
```
Hybrid Multiplication

- It merges the both basic methods
- Column wise with extended word size

Column width $d = 5$ digits
- Reason for reserving memory positions as multiples of 5 words
HYBRID MULTIPLICATION

COLUMN WISE MULTIPLICATION

A2 A1 A0
B2 B1 B0

x

A0 B0
A1 B0
A0 B1
A0 B0
A1 B1
A2 B2
A1 B0
A2 B1
A2 B2

R5 R4 R3 R2 R1 R0

ROW WISE MULTIPLICATION

\[
\begin{array}{cccccc}
  a_4 & a_3 & a_2 & a_1 & a_0 \\
  b_4 & b_3 & b_2 & b_1 & b_0 \\
\end{array}
\]

x

\[
\begin{array}{cccccccc}
  c_9 & c_8 & c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 \\
\end{array}
\]
MODULAR ARITHMETIC (I)

- Modular Reduction
  - \( R = X \mod M \)

- Barret Reduction
  - From the division definition:
    - \( X = Q \cdot M + R \rightarrow R = X - M \cdot Q \)
  - Find a good and fast quotient estimation \( Q' \)
    - \( R' = X - M \cdot Q' \)
  - Definition of quotient:
    \[
    Q = \left\lfloor \frac{X}{M} \right\rfloor
    \]
MODULAR ARITHMETIC (II)

- Barret’s estimation of the quotient:

\[ Q' = \left\{ \frac{X}{M} \cdot \frac{b^{2k}}{b^{2k}} \right\} = \left\{ \frac{X}{b^{k-1}} \cdot \frac{b^{2k}}{b^{k+1}} \right\} = \left\{ \frac{X}{b^{k-1}} \cdot \frac{\mu}{b^{k+1}} \right\} \]

- \( k \): the length in words of the modulus (1 word = 8 bits)
- \( b \): radix of the representation (\(2^8 = 256\))

- Precomputed value \( \mu \)
  - Only depends on the modulus \( \rightarrow \) constant

- Divisions by the radix \( b \) to some power
  - If the radix is a power of 2, they become shifts

- Estimation very precise: \( Q-2 \leq Q' \leq Q \)
- Maximum of two extra subtractions
Folding Technique

- Previous step in Barrett Reduction
- Find a number $X' < X$, such that:
  \[ X \mod M = X' \mod M \]

\[ X = X_H \cdot 2^f + X_L \]

\[ X' = X_H \cdot (2^f \mod M) + X_L \]

- Requires another precomputed value: $(2^f \mod M)$
- Folding point $f$
  - Between the length of the modulus and the length of $X$
  - One multiplication and one addition
MODULAR ARITHMETIC (IV)

- Modular Multiplication
  - R = X·Y mod M
  - Implementation of “classic algorithm”

- Multiplication: Hybrid Multiplication
- Modular Reduction: Barret with Folding
MODULAR ARITHMETIC (V)

- Modular Multiexponentiation
  - \( R = A^X \cdot B^Y \mod M \)
OTHER IMPLEMENTATIONS

- **Hash Function**
  - SHA-1 (Secure Hash Algorithm)
  - Input: blocks with length 512 bits
  - Output: message digest with length 160 bits
  - Implementation:
    - Only one block as input \( (m1||m2) < 447 \) bits

- **Pseudo Random Number Generator**
  - TT800: Twisted Generalized Feedback Shift Register
  - Suitable for embedded devices
  - Seed of 800 bits
  - Output of 32 bits
MEASUREMENTS

Computation times & Internal memory usage

<table>
<thead>
<tr>
<th>RANDOM NUMBER GENERATOR</th>
<th>Operands</th>
<th>Time@16Mhz</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>n = 1512 bits</td>
<td>3,2 ms</td>
<td>100 bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HASH FUNCTION</th>
<th>Operands</th>
<th>Time@16Mhz</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>m₁, m₂</td>
<td>m₁ = 160 bits, m₂ = 160 bits</td>
<td>3 ms</td>
<td>64 bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MULTIEXPONENTIATION</th>
<th>Operands</th>
<th>Time@16Mhz</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, n = 1024 bits</td>
<td>140 s</td>
<td>1 Kbyte</td>
<td></td>
</tr>
<tr>
<td>x = 1512 bits</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y = 400 bits</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MULTIPLICATION &amp; ADDITION</th>
<th>Operands</th>
<th>Time@16Mhz</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = 1512 bits, b = 160 bits, c = 1512 bits</td>
<td>5,4 ms</td>
<td>11 bytes</td>
<td></td>
</tr>
</tbody>
</table>
Multiexponentiation takes a long time
- Proposed solutions:
  - Decrease the lengths of the exponents
  - Problem: information not theoretically hidden
  - Smart card with cryptographic hardware

Design of a functional cryptographic library
- Easy to extend with new features:
  - Pseudonym support
  - Elliptic curve based on bilinear pairings
- Easy to implement new protocols
Thank You

QUESTIONS?