Software Library Implementation for Public Key Cryptosystems

Center for Information Security and Technologies (CIST)
Korea University
http://cist.korea.ac.kr

Tae Hyun Kim

Goals of our software library implementation

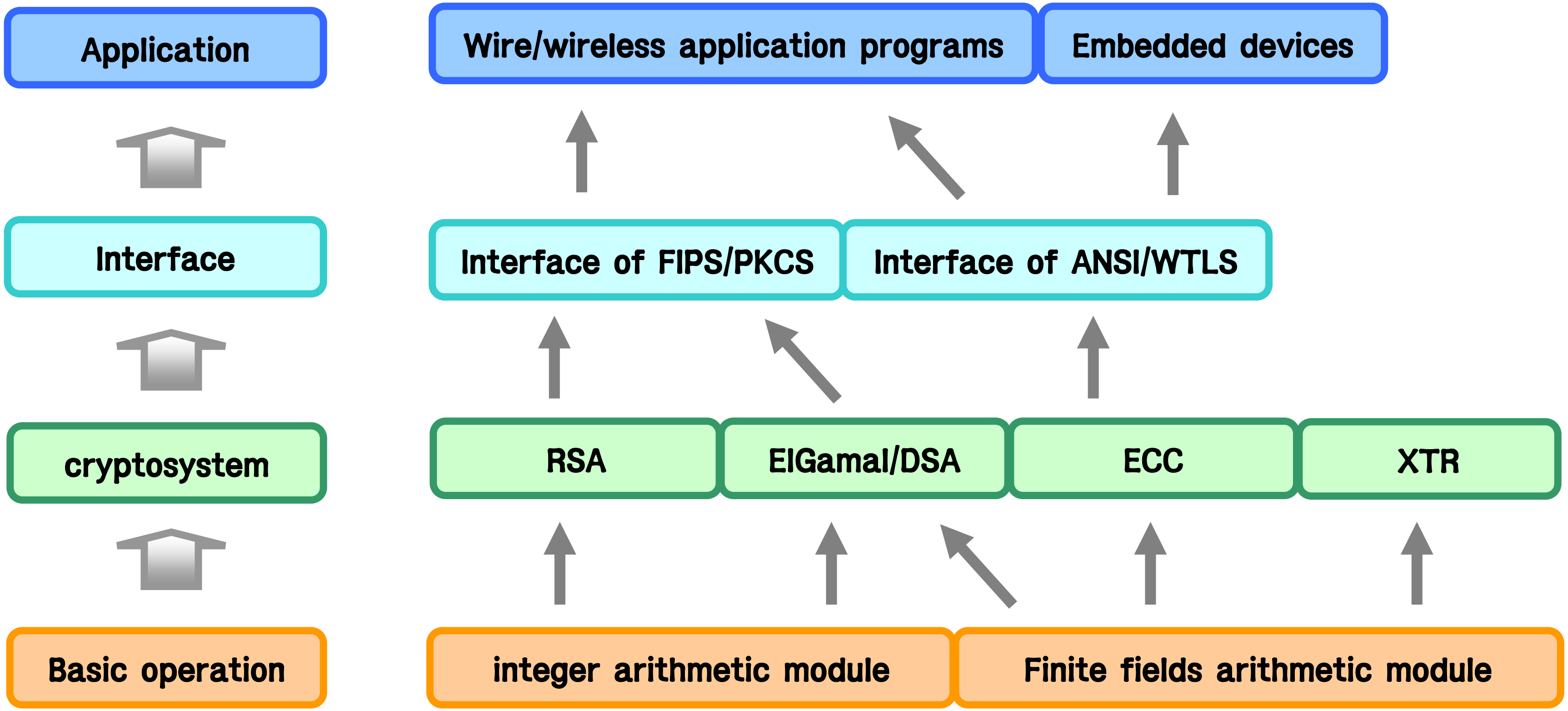
1. To support practical research for public key cryptosystems (PKCs)
   - Efficient implementation of RSA/ECC/XTR etc.
   - Use as tool to prove security

2. Application to industry of information security
   - Provision for a mandatory cryptographic modules

Contents of library

- Integer arithmetic module
  - PKCs based on integer arithmetic
    - RSA(PKCS#1) / ElGamal / DSA(FIPS 186-2)
- Finite field arithmetic module
  - PKCs based on finite field arithmetic module
    - Elliptic curve cryptosystem (ECC) / XTR
      - ANSI X9.62, ANSI X9.63, WTLS, SEC, etc.
      - Prime fields, binary fields and composite fields

Flow chart


Application
Wire/wireless application programs
Embedded devices
Interface
Interface of FIPS/PKCS
Interface of ANSI/WTLS
cryptosystem
RSA
ElGamal/DSA
ECC
XTR
Basic operation
integer arithmetic module
Finite fields arithmetic module

The merits of our library

- Our library is suitable to both 32-bit and 64-bit microprocessor and can be apply to some other microprocessors.
  - Use of flexible word-size

- Not only is our library applicable to various platform but also optimized.
  - In the case of ECC, it was optimized against some standard curves (WTLS #3, #5, #7).

RSA : the methods for high-speed

- ***Multiplication*** :
  - Karatsuba-ofman method
  - $(a_1X + a_0)\cdot(b_1X + b_0)$

    $= a_1b_1X^2 + (a_1b_0 + a_0b_1)X + a_0b_0$ (4 Mul)

    $= a_1b_1X^2 + [(a_1 + a_0)\cdot(b_1 + b_0) - a_1b_1 - a_0b_0]X + a_0b_0$ (3 Mul)

※ This method is efficient if the bit-size is larger than 512 bits.

CIST

RSA : the methods for high-speed

- ***Exponentiation*** :
  - Montgomery method
    - Montgomery Multiplication + Binary Exponentiation
    - No divisions
  - Sliding window method
    - Precomputation tables
  - Montgomery Multiplication with window method
    - No divisions
    - Precomputations tables

※ By our experiment, it is optimal in the case of Montgomery method with width-4 window method.

ECC : the methods for high-speed

- ***Squaring in GF(2^n)*** :
  - **Table lookup method : 8-bit precomputations**
    - If $a(x) = a_7x^7 + a_6x^6 \ldots + a_1x + a_0 = (a_7 \ldots a_1a_0)_2$
      then $a^2(x) = a_7x^{14} + a_6x^{12} + \ldots + a_1x^2 + a_0$
      $= (0a_70a_6 \ldots a_10a_0)_2$

Input: $(a_7 \ldots a_1a_0)_2$

Output: $(0a_70a_6 \ldots a_10a_0)_2$

※ Table[$(a_7 \ldots a_1a_0)_2$] = $(0a_70a_6 \ldots a_10a_0)_2$

ECC : the methods for high-speed

- ***Multiplication in GF(2^n)*** :
  - Right-to-left comb method
  - Left-to-right comb method
  - L-R comb method with windows of size 4

※ ***Algorithms of multiplication in GF(2^n) are similar to those of exponentiation.***

Algorithm 1. Right-to-left shift-and-add field multiplication

INPUT: Binary polynomials $a(x)$ and $b(x)$ of degree at most $m-1$.
OUTPUT: $c(x) = a(x) \cdot b(x) \bmod f(x)$.

1. If $a_0 = 1$ then $c \leftarrow b$; else $c \leftarrow 0$.
2. For i from 1 to $m-1$ do
   - 2.1 $b \leftarrow \boxed{b \cdot x} \bmod f(x)$.
   - 2.2 If $a_i = 1$ then $c \leftarrow \boxed{c + b}$.
3. Return(c).

Square

Multiplication

ECC : the methods for high-speed

- Example parameters are that the word-size is 32 and the field-size is 163.
  - Bit-scan order:
    - Right-to-left comb method : →
    - Left-to-right comb method : →

a_{31}	…	a_2	a_1	a_0	A[0]
a_{63}	…	a_{34}	a_{33}	a_{32}	A[1]
a_{95}	…	a_{66}	a_{65}	a_{64}	A[2]
a_{127}	…	a_{98}	a_{97}	a_{96}	A[3]
a_{159}	…	a_{130}	a_{129}	a_{128}	A[4]
		a_{162}	a_{161}	a_{160}	A[5]

Improvement for ECC

- In general, if parameters are fixed then algorithms can be modified in order to enhance performance.
  - If the size of field is decided then multiplication method and modular reductions can be efficient.

- So, we implemented on Pentium IV/2.0GHz. (32-bit μP; Windows 2000, MSVC) for WTLS #3, #5, #7.

WTLS #7 : Prime Field

- $p = 2^{160} - 2^{31} - 1$
  $= (\text{FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7FFFFFFF})_{16}$

- The modular reduction is performed by some additions and shifts.

$$2^{288} \equiv 2^{159} + 2^{128} \pmod p$$
$$2^{289} \equiv 2^{160} + 2^{129} \pmod p$$
$$\cdots$$
$$2^{319} \equiv 2^{190} + 2^{159} \pmod p$$

※ *Since the modular reduction is improved overall operations are also improved.*

WTLS #3, #5 : Binary Fields

- If an irreducible polynomial is trinomial or pentanomial then modular reductions are performed by one word at a time.
  - WTLS #3 : $p(x) = x^{163} + x^7 + x^6 + x^3 + 1$

$$x^{288} \equiv x^{132} + x^{131} + x^{128} + x^{125} \pmod{p(x)}$$

$$x^{289} \equiv x^{133} + x^{132} + x^{129} + x^{126} \pmod{p(x)}$$

$$\cdots$$

$$x^{319} \equiv x^{163} + x^{162} + x^{159} + x^{156} \pmod{p(x)}$$

  - WTLS #5 : $p(x) = x^{163} + x^8 + x^2 + x + 1$
- Since the size of fields is 163-bits, it is implemented without loop such as "for" or "while" statements.
  - L-R comb method with windows of size 4 is simply optimized.

Experimental results (Prime Fields)

- In WTLS #7, comparison of *modular reductions*
  - ① : the general modular reduction using division
  - ② : the high-speed modular reduction using special prime

	①	②	①	②	①	②
Iterations	100,000		500,000		1,000,000	
Time (Sec)	0.18700	0.03100	0.93700	0.17100	1.87500	0.34300

※ *We deduce that ② method is 6 times as fast as ① method.*

Experimental results (Prime Fields)

- In WTLS #7, comparison of *scalar multiplications*
  - ① : scalar multiplication with the general modular reduction using division
  - ② : scalar multiplication with the fast modular reduction

	①	②	①	②	①	②
Iterations	100		500		1,000	
Time (Sec)	0.781	0.375	3.812	1.828	7.782	3.750

※ *The above result shows that ② method is twice as fast as ① method.*

Experimental results (Binary Fields)

- In WTLS #3, #5, comparison of *modular reductions*
  - ① : the modular reductions by one bit at a time
  - ② : the high-speed modular reduction using trinomial and pentanomial (one word at a time)

	①	② (WTLS#3)	② (WTLS#5)	①	② (WTLS#3)	② (WTLS#5)
Iterations	100,000			500,000		
Time (Sec)	1.14000	0.01500	0.01500	5.65600	0.09300	0.09300

※ *We deduce that ② method is 70 times as fast as ① method.*

Experimental results (Binary Fields)

- In binary field, comparison of ***multiplications***
  - ① : Shift-and-add ② : Right-to-left comb ③ : Left-to-right comb
  - ④ : LR comb with windows of size 4
  - ⑤ : LR comb with windows of size 4 (optimized version of 163-bit)

		m = 163 (WTLS #3)		m = 233	m = 283
Iterations		50,000	100,000	100,000	100,000
Time (Sec)	①	0.23400	0.45300	0.76500	0.98400
	②	0.20300	0.40600	0.60900	0.76500
	③	0.21800	0.43700	0.62500	0.81200
	④	0.17100	0.35900	0.45300	0.48400
	⑤	0.03100	0.06200	--	--

※ ***We deduce that ⑤ method is 7 times as fast as ① method.***

CIST

Experimental results (Binary Fields)

- In WTLS #3, #5, comparison of ***scalar multiplications***

  ① : Shift-and-add ② : Right-to-left comb ③ : Left-to-right comb

  ④ : LR comb with windows of size 4

  ⑤ : LR comb with windows of size 4 (optimized version of 163-bit)

Reduction (m = 163)		one word at a time		one bit at a time	
Iterations : 1000		WTLS #3	WTLS #5	WTLS #3	WTLS #5
Time (Sec)	①	10.015	10.042	26.078	23.937
	②	8.859	8.897	24.578	26.453
	③	9.250	9.273	24.969	26.890
	④	7.203	7.225	22.765	24.657
	⑤	1.781	1.782	17.500	19.438

※ *The above result shows that ⑤ method in WTLS #3 is 6 times as fast as ① method*

CIST

Current & future research topic

- Efficient scalar multiplication for ECC
  - Generalized width-w JSF
- Side channel attacks on ECC
  - SPA/DPA
  - Development of efficient and secure countermeasure

Questions & Comments

Thank you

CIST