社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－CSEC－42（25）
2008 / 7 / 25

グループ署名回路のアーキテクチャ最適化

森岡 澄夫† 荒木 俊則† 一色 寿幸†
尾花 賢† 佐古 和恵† 寺西 勇†

† 日本電気 中央研究所 〒211-8666 川崎市中原区下沼部1753
E-mail: {s-morioka@ak, t-araki@ek, t-issiki@bx, obana@bx, k-sako@ab, teranisi@ah}.jp.nec.com

Architecture Optimization of a Group Signature Circuit

Sumio MORIOKA†, Toshinori ARAKI†, Toshiyuki ISSHIKI†,
Satoshi OBANA†, Kazue SAKO†, and Isamu TERANISHI†

† Central Research Labs., NEC Corporation
1753 Shimonumabe, Nakahara-ku, Kawasaki, Kanagawa 211-8666 Japan
E-mail: {s-morioka@ak, t-araki@ek, t-issiki@bx, obana@bx, k-sako@ab, teranisi@ah}.jp.nec.com

Abstract Group signature scheme is one of the most active research area in recent cryptographic algorithms/applications. Typical signature algorithm is a combination of dozens of elliptic curve (EC), modular, integer and hash arithmetic operations on data whose bit width exceeds 1,000 bits. A full-H/W IP core is desired for the use of the group signature in SoCs in slow-clock mobile devices. In order to construct a high performance and configurable group signature IP, connecting multiple modular / EC arithmetic units (sub-IPs) and a simple controller not by a wide-band bus but by a narrow-band bus is appropriate. While conventional behavioral synthesis from C-language was used, the development of an additional behavioral synthesizer for parallel scheduling of sub-IP level (C function-library level) operations was necessary. We explored an optimum H/W architecture for a typical group signature algorithm and found that at most 5 modular sub-IPs is enough. Practical H/W speed of less than 0.1 seconds at 100MHz on a 130nm standard cell ASIC library was achieved.

Key words Group Signature, Security H/W, IP core architecture, Behavioral synthesis, C function level parallelism

1. Introduction

Group signature scheme, first introduced by Chaum and Heyst [1], is one of the most active research area in recent cryptographic algorithms/applications [2]～[6]. In this scheme, users can sign messages anonymously, although there is an authority that can trace the signer. Group signatures have many practical applications such as e-voting, e-cash, fingerprinting, vehicular communication and so on [7], [8]. Regarding to the speed of group signature, a recent S/W implementation [4], [5] based on a typical signature algorithm described in [3] achieves 0.1-0.2 seconds on a 3GHz PC, and this speed is fast enough for practical use.

However, H/W (IP core) implementation has been desired for the use of group signature in LSIs in slow-clock (up to 300MHz or so) mobile devices. To the best of the authors' knowledge, there have been no report on H/W design of group signature algorithm.

Typical group signature algorithm [3]～[5] is a combination of dozens of primitive arithmetic operations which are also used in RSA and elliptic curve cryptography (ECC) [9], [10]. The H/W architectures such as "FSM + datapath" and "standard embedded CPU (ARM etc.) + IP core accelerators" are not suitable for group signature, because of design difficulty, slow speed, low portability, or vulnerability to side channel attacks [11], [12].

In this paper, we investigated an appropriate H/W architecture and design methodology for designing a high performance group signature, as follows. *Please note that our* design target is an IP core for SoC and we are not designing

a PC accelerator.

1. We found that amount of data transfer between primitive operations (elliptic curve (EC), modular, INT and hash operations) is little in the group signature. Therefore, the best way to achieve all of high performance, portability and configurability is to connect a data transfer controller and multiple sub-IPs, which execute each primitive operation, not by a wide-band bus but by a single narrow-band bus.

2. We applied a two-level behavioral synthesis design strategy. The use of C-based H/W modeling and behavioral synthesis [15]~[19] make efficient H/W development possible, by eliminating time-consuming RT-level simulation work (single group signature computation consumes 10M to 300M clock cycles). However, conventional behavioral synthesizers only support parallel scheduling of C-embedded operators ($+, -, *, /$, etc.) and cannot efficiently arrange the execution order of higher level (sub-IP level) functions whose operation mode can be changed at run time. Therefore, we made an additional custom behavioral synthesizer for the sub-IP level scheduling, in order to derive a paralleled computation sequence. This custom synthesizer is specific to typical group signature algorithm.

3. Using this custom synthesizer, we investigated a relationship between total computation speed and the number of sub-IPs. We found that the performance bottleneck is speed of modular exponentiation and the parallel use of at most 5 modular sub-IPs is enough, in both signature generation and verification.

As a result, practical H/W speed of less than 0.1 seconds at 100MHz (max 150-200MHz) on a 130nm standard cell ASIC library, using 350-500Kgate and 6-9KB SRAMs, was achieved.

This paper is organized as follows. In Section II, we introduce a fast group signature algorithm. In Section III, we explain the proposed circuit architecture and design methodology. In Section IV, we will show performance optimization results.

2. Group Signature Algorithm

2.1 Model

In this paper, we have implemented one of a typical and fast group signature algorithm described in [3]~[5]. Four entities join in a group signature scheme; *User, Issuer, Opener and User-Revocation manager*. The Issuer has the authority to add a User into a group, Opener has the authority to identify the signer, and User-Revocation manager has the authority to revoke a member (*User*) from a group.

While a group signature scheme has the procedures such as Key pair generation, Join, User revocation, Update, Sign, Verify and Open, only the Sign and Verify procedures have been implemented, because our IP will be embedded into LSIs in consumer electronics devices and the entity *User* will be the main user of our IP.

2.2 Security Parameters

We employ a set of security parameters $\kappa = (\kappa_n, \kappa_\ell, \kappa_e, \kappa_{e'}, \kappa_q, \kappa_c, \kappa_S)$, where κ_n, κ_ℓ, κ_e and $\kappa_{e'}$ are bit-length of n, ℓ, e and e', respectively, κ_q is bit-length of order value of an elliptic curve $\mathcal{G}$, κ_c is output bit-length of a hash function which is used for the Fiat-Shamir heuristic, and κ_S is bit-length such that when we pick r as $|a| + \kappa_S$-bit random number for any integer a, then $a + r$ and r are statistically indistinguishable.

For standard security level whose key length corresponds to RSA-1024 and ECC-160, the actual values of κ_n, κ_ℓ, κ_e, $\kappa_{e'}$, κ_q, κ_c and κ_S are 1024, 1024, 504, 60, 160, 160 and 60, respectively. Besides, for high security level whose key length corresponds to RSA-2048 and ECC-224, these values are 2048, 2048, 736, 60, 224, 224, 112, respectively.

2.3 Public Key and Secret Key

Let $\mathcal{G}$ denotes a finite group whose order q is a prime number, where bit-length of q is κ_q. Also, let $QR(n)$, Λ, $[c]G$, $+_e$ and $-_e$ denotes a quadratic residue modulo n, a set of integer values in a range $[0, 2^\lambda)$ where $\lambda = \kappa_n + \kappa_q + \kappa_S$, scalar multiplication on an elliptic curve, point addition and point subtraction, respectively.

The key pairs (public key and secret key) for each entity are as follows; (1) *Issuer's key pair* $\mathsf{ipk} = (n, a_0, a_1, a_2)$, $\mathsf{isk} = (p_1, p_2)$, where p_1 and p_2 are safe prime numbers whose bit-length are $\kappa_n/2$, $n = p_1 p_2$ and $a_0, a_1, a_2 \in \mathrm{QR}(n)$, (2) *Opener's key pair* $\mathsf{opk} = (q, G, H_1, H_2)$, $\mathsf{osk} = (y_1, y_2)$, satisfying $y_1, y_2 \in \mathbb{Z}_q$, $G \in \mathcal{G}$ and $(H_1, H_2) = ([y_1]G, [y_2]G)$, (3) *User-Revocation manager's key pair* $\mathsf{rpk} = (\ell, b, w)$, $\mathsf{rsk} = (\ell_1, \ell_2)$, where ℓ_1 and ℓ_2 are safe prime numbers whose bit-length are $\kappa_\ell/2$, $\ell = \ell_1 \ell_2$ and $b, w \in \mathrm{QR}(\ell)$, and (4) *The i-th user's (U_i) key pair* $\mathsf{mpk}_i = (h_i, A_i, e'_i, B_i)$, $\mathsf{msk}_i = x_i$, satisfying $x_i \in \Lambda$, $h_i = [x_i]G$, $B_i = b^{1/e'_i} \bmod \ell$, $e_i = 2^{\kappa_e} + e'_i$, and $a_0 a_1^{x_i} \equiv A_i^{e_i} \bmod n$.

2.4 Signature Generation Algorithm

The inputs of signature generation algorithm are ipk, rpk, opk, mpk_i, msk_i and a message m. Let $\mathsf{Hash} : \{0,1\}^* \rightarrow \{0,1\}^{\kappa_c}$ denotes a collision resistant hash function and $e_i = 2^{\kappa_e} + e'_i$. In this paper, we have used an elliptic curve specified in FIPS PUB 186-3 and SHA-1 [1]. The signature generation algorithm is shown below.

1. Choose $\rho_E \in \mathbb{Z}_q$, $(\rho_m, \rho_r) \in \{0,1\}^{\kappa_n/2} \times \{0,1\}^{\kappa_\ell/2}$ $\mu_x \in \{0,1\}^{\lambda+\kappa_c+\kappa_S}$, $\mu_s \in \{0,1\}^{\kappa_e+(\kappa_n/2)+\kappa_c+\kappa_S}$, $\mu_{e'} \in \{0,1\}^{\kappa_{e'}+\kappa_c+\kappa_S}$, $\mu_t \in \{0,1\}^{\kappa_{e'}+(\kappa_\ell/2)+\kappa_c+\kappa_S}$ and $\mu_E \in \mathbb{Z}_q$,

1: Circuit performance will not change even if some other standard hash functions, such as SHA-2, are used.

Sub-IP / operation mode	Bit length	CLK cycles	Times	Ratio
EC / scalar mult.	160	740,131	7	15.9%
EC / point add.	160	3,441	4	<0.1%
MOD / modular mult.	1024 x 1024	2,771	5	<0.1%
MOD / modular exp.	$(1024)^{280}$	1,476,441	2	9.0%
MOD / modular exp.	$(1024)^{512}$	2,696,661	2	16.5%
MOD / modular exp.	$(1024)^{792}$	4,170,281	1	12.8%
MOD / modular exp.	$(1024)^{1236}$	6,505,921	1	19.9%
MOD / modular exp.	$(1024)^{1464}$	7,705,661	1	23.6%
MOD / multiplicative inv.	1024	930,956	2	5.7%
INT / mult.	160 x 60	266	1	<0.1%
INT / mult.	160 x 160	326	1	<0.1%
INT / mult.	160 x 576	646	1	<0.1%
INT / mult.	160 x 1016	996	1	<0.1%
INT / mult.	160 x 1244	1,176	1	<0.1%
INT / mult.	512 x 50	406	1	<0.1%
INT / mult.	512 x 504	1,298	1	<0.1%
INT / modulo	321 % 160	3,466	1	<0.1%
INT / modulo	1464 % 160	27,346	1	<0.1%
HASH / SHA-1	(message len)	(<10,000)	1	<0.1%
PRNG / random num gen	160	853	2	<0.1%
PRNG / random num gen	280	991	1	<0.1%
PRNG / random num gen	512	1,261	2	<0.1%
PRNG / random num gen	792	1,391	1	<0.1%
PRNG / random num gen	1236	1,811	1	<0.1%
PRNG / random num gen	1464	2,081	1	<0.1%
Data transfer & control	--	*1,286,806*	--	*3.8%*
TOTAL (sequential exec.)	--	33,909,915	--	(100%)

Table 1 Example speed of primitive operations in signature generation (process independent, standard security level)

randomly.

2. Compute $\boldsymbol{E} = (E_0, E_1, E_2) = ([\rho_E]G, h_i + _e[\rho_E]H_1, h_i + _e [\rho_E]H_2)$ and $V_{\text{ConCipher}} = ([\mu_E]G, [\mu_x]G + _e [\mu_E]H_1, [\mu_x]G + _e [\mu_E]H_2)$.

3. Compute $(A_{\text{COM}}, B_{\text{COM}}) = (A_i a_2^{\rho_m} \bmod n, B_i w^{\rho_r} \bmod \ell)$ and $(V_{\text{ConMPK}}, V_{\text{ConRev}}) = (a_1^{\mu_x} a_2^{\mu_s} A_{\text{COM}}^{-\mu_{e'}} \bmod n,\ w^{\mu_t} B_{\text{COM}}^{-\mu_{e'}} \bmod \ell)$.

4. Compute $c = \mathsf{Hash}(\kappa, \text{ipk}, \text{opk}, \text{rpk}, \boldsymbol{E}, A_{\text{COM}}, B_{\text{COM}}, V_{\text{ConCipher}}, V_{\text{ConMPK}}, V_{\text{ConRev}}, m)$.

5. Compute $\tau_x = cx_i + \mu_x$, $\tau_s = ce_i\rho_m + \mu_s$, $\tau_t = ce'_i\rho_r + \mu_t$, $\tau_{e'} = ce'_i + \mu_{e'}$ and $\tau_E = c\rho_E + \mu_E \bmod q$.

6. The output signature is $(\boldsymbol{E}, A_{\text{COM}}, B_{\text{COM}}, c, \tau_x, \tau_s, \tau_{e'}, \tau_t, \tau_E)$.

2.5 Signature Verification Algorithm

The inputs of verification algorithm are ipk, opk, rpk, message m and the signature $\sigma = (\boldsymbol{E}, A_{\text{COM}}, B_{\text{COM}}, c, \tau_x, \tau_s, \tau_{e'}, \tau_t, \tau_E)$ which is attached to m.

1. Check if both $|\tau_x| \leqq \lambda + \kappa_c + \kappa_S$ and $|\tau_{e'}| \leqq \kappa_{e'} + \kappa_c + \kappa_S$ hold. If hold, then go to the next step. Otherwise, output reject.

2. Compute $V'_{\text{ConCipher}} = ([\tau_E]G - _e[c]E_0, [\tau_x]G + _e[\tau_E]H_1 - _e [c]E_1, [\tau_x]G + _e [\tau_E]H_2 - _e [c]E_2)$.

3. Compute $V'_{\text{ConMPK}} = a_0^c a_1^{\tau_x} a_2^{\tau_s} A_{\text{COM}}^{-(c2^{\kappa_e} + \tau_{e'})} \bmod n$ and $V'_{\text{ConRev}} = b^c w^{\tau_t} B_{\text{COM}}^{-\tau_{e'}} \bmod \ell$.

4. Check if $c = \mathsf{Hash}(\kappa, \text{ipk}, \text{opk}, \text{rpk}, \boldsymbol{E}, A_{\text{COM}}, B_{\text{COM}}, V'_{\text{ConCipher}}, V'_{\text{ConMPK}}, V'_{\text{ConRev}}, m)$ holds. If holds, output accept and otherwise, output reject.

Sub-IP / operation mode	Bit length	CLK cycles	Times	Ratio
EC / scalar mult.	160	740,131	7	16.0%
EC / point add.	160	3,441	5	<0.1%
EC / point negation	160	71	3	<0.1%
MOD / modular mult.	1024 x 1024	2,771	5	<0.1%
MOD / modular exp.	$(1024)^{160}$	844,741	2	5.2%
MOD / modular exp.	$(1024)^{280}$	1,476,441	1	4.5%
MOD / modular exp.	$(1024)^{986}$	3,507,241	1	10.8%
MOD / modular exp.	$(1024)^{793}$	4,180,751	1	12.9%
MOD / modular exp.	$(1024)^{1237}$	6,516,391	1	20.1%
MOD / modular exp.	$(1024)^{1465}$	7,716,137	1	23.8%
MOD / multiplicative inv.	1024	930,956	2	5.8%
INT / modulo	1465 % 160	27,206	1	<0.1%
HASH / SHA-1	672	260	1	<0.1%
Data transfer & control	--	*1,001,399*	--	*3.2 %*
TOTAL (sequential)	--	31,894,240	--	(100%)

Table 2 Example speed of primitive operations in signature verification (process independent, standard security level)

3. Group Signature IP-core Architecture

3.1 Features of the Algorithm from H/W Design Standpoint

Typical group signature algorithm has some significant features, from H/W design standpoint, as described below.

- The algorithm is a complicated combination (more than 70 steps in total) of primitive operations listed in Table 1 and 2. An appropriate parallel scheduling of different kinds of primitive operations, whose clock cycles are also much different, is an important key to speedup.

- Bit width of data is large and most of data should be stored not on registers but on SRAMs. In each primitive operation, most of the computation time is occupied by SRAM access cycles. Besides, only a small ratio (a few percents) of total computation time is consumed by data transfer between primitive operations (see Table 1 and 2).

- Although it is not yet well known what kinds of side channel attacks [11]~[14] are effective against group signature algorithm, careful implementation will be necessary because the algorithm involves large number of arithmetic operations compared to traditional encryption algorithms and therefore, much information on secret keys can be leaked to attackers via computation time or power waveform.

- Computation time of each primitive operation is independent of data/key value, when a countermeasure to timing attack [11] is incorporated. (It is always the case in cryptographic H/W implementation.)

3.2 Issues in Conventional H/W Architecture

There have been two standard approaches to implement conventional security algorithms into H/W. The first approach is a "datapath + control FSM" architecture in Fig. 1. Typical ECC and RSA circuits have this structure [9] and the FSM part can have a hierarchical structure [10]. However, this architecture is not suitable for group signature H/W, because parallel execution of multiple operations of varying kinds and with different execution times is almost impossible.

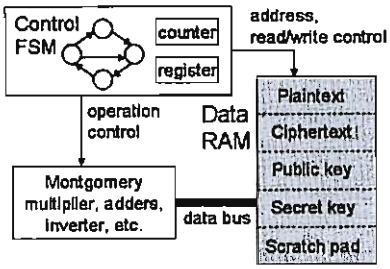
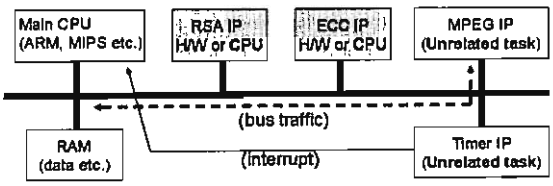


Fig. 1 Typical public key cryptographic H/W



Issue 1: Interferences in computation timing and power waveform because of unpredictable bus traffic / interrupt by unrelated tasks.
Issue 2: Difficult to reuse design as is, in other LSIs whose bus architecture and memory map are different.
Issue 3: Occupies relatively large main-CPU time.

Fig. 2 Conventional security function implementation in SoC

The second approach is to implement the group signature algorithm on a standard SoC architecture in Fig. 2, where an embedded CPU and some IP accelerators are connected to on-chip bus (multi-CPU architecture can be used). Main issues in using this approach are: (i) reuse of the H/W in other SoC is not easy because bus architecture and memory map are often very different, (ii) the group signature algorithm may consume too long CPU time as a single computation[2], and (iii) precise management of computation timing and power waveform is difficult because there are much interference by bus traffic and interrupt from unrelated SoC tasks. Such interference can be used by side channel attackers in order to make analyzing key values easy.

3.3 The Proposed IP-core Architecture

We have decided to enclose all of group signature computation into a single IP-core, as shown in Fig. 3. Our group signature IP (Fig. 4) has an internal structure where multiple fast arithmetic units (sub-IPs) and a data transfer controller are connected by a local bus.

The use of a single narrow-band bus is specific to group signature algorithm. Neither multiple buses nor wide-band bus is necessary, because amount of traffic on the bus is little as mentioned in Section 3.1. This simple bus architecture enables high configurability of number of sub-IPs and high routability at back-end design process.

2: In some mobile devices, CPU time assigned to a task is strictly restricted.

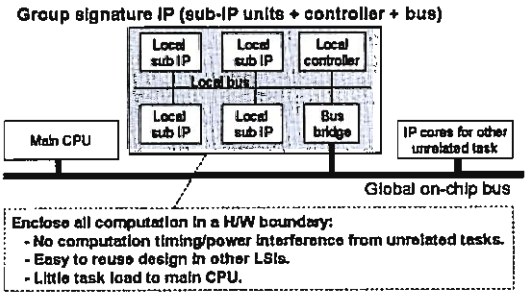


Fig. 3 Use of our group signature IP in a SoC

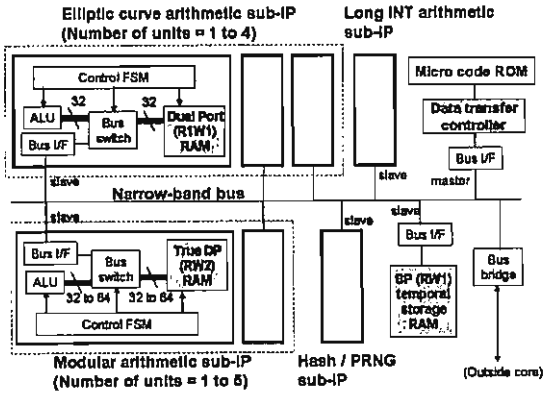


Fig. 4 The proposed group signature IP architecture

Each sub-IP corresponds to elliptic curve (EC), modular, long-bit integer and hash arithmetic operation respectively, and has multiple function modes for varying kinds of operations and data bit width as shown in Table 1 and 2. Internal structure of sub-IP is almost the same with the structure in Fig. 1. For fast computation, modular arithmetic is done on Montgomery domain [9] and EC arithmetic is done on Jacobian coordinate of Montgomery number representation. The number of SRAM ports in each sub-IP is equal to the maximum number of simultaneous R/W accesses.

In Fig. 4, a compact custom-made controller is used in order to transfer data between each sub-IP, issue computation start commands and monitor computation progress in the sub-IPs. This controller never perform arithmetic computation. Micro-code (firmware) for the controller is stored in a ROM or a combinational logic circuit.

4. Design Flow of Group Signature IP-core

4.1 Full Use of Behavioral Synthesis and FPGA Prototyping

Our methodology of designing group signature IP is summarized in Fig. 5. Our IP is constructed by conventional building-block approach, i.e., each sub-IP is made and tested

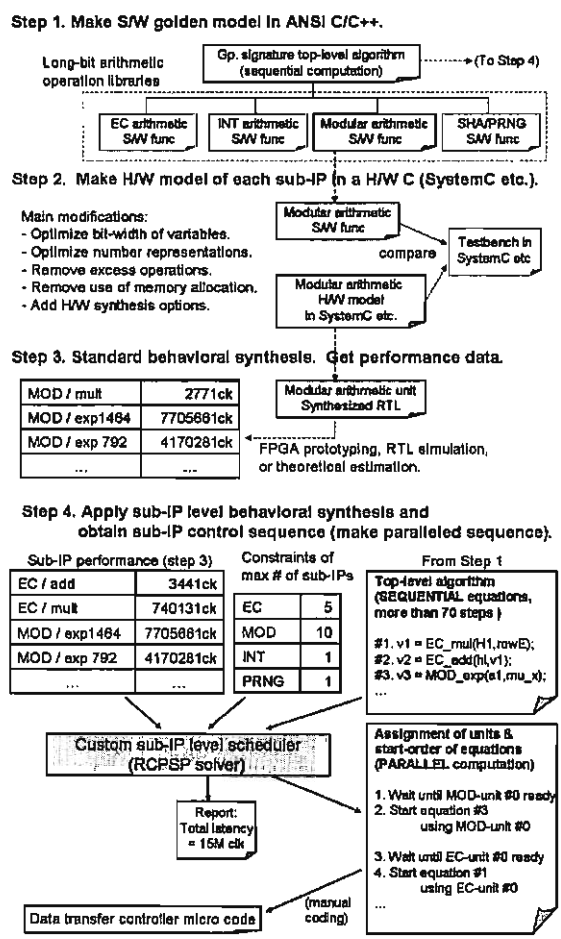


Fig. 5 Proposed design flow with two-level behavioral synthesis

separately prior to total IP integration. As shown in Step 1, 2, 3 and 5 in Fig. 5, we fully used a combination of C-based H/W modeling, behavioral synthesis [15]~[19] and FPGA prototyping, because of the following reasons:

- Efficient development of each sub-IP becomes possible by avoiding use of RTL simulation. Simulating only one of sub-IP functions (e.g., modular exponentiation) can take a few hours on standard PC, while behavioral level simulation and behavioral synthesis are finished within several minutes.

- Achieve high configurability of the total performance. As mentioned in Section 3.1, performance of each sub-IP is determined by SRAM type. If RTL design entry is used, changing SRAM type becomes a difficult and time consuming work.

- Even though building-block approach is used, full verification of entire group signature IP is still necessary in order to debug micro-code (firmware) of the data transfer controller. For this purpose, RTL simulation cannot be used at all because of too slow speed. FPGA prototyping is appropriate.

4.2 Modification of S/W C codes into H/W C

Even though there has been significant progress in behavioral synthesis techniques in recent years, it is still difficult to synthesize an efficient sub-IP H/W from pure S/W C codes without any modification. In fact, we could not reuse our fast S/W code [5] as is, and some amount of rewriting work, as shown below, was necessary in Step 2 in Fig. 5:

- Data bit-width optimization and removing excess operations. Not only group signature but also public key cryptographic S/W usually use special function libraries, which execute long-bit arithmetic operations by combination of 32-bit or 64-bit C-embedded integer operations. Careful overflow control is also implemented in S/W[3]. However, direct computation on arbitrary bit-width is possible in H/W.

- Optimizing arithmetic operators and number representation. While only integer operations on 2's complement representation can be used in S/W, flexible construction of any operations on any number representations (e.g., GF operations on residual number system) is possible in H/W.

- Removing dynamic memory allocation, dynamic pointers[4] and recursion. If S/W-like data types such as balance tree and linear list are used, they have to be removed too.

- Adding an appropriate behavioral synthesis control options into the source code. A sample behavior description of Montgomery multiplication [9] in SystemC[5] is shown in Table 3. One of the most frequently used options is "loop folding" [16], which is attached to the inner loop. In this example, the loop-iteration is forced to start every 2 clock cycles and multiple iterations are executed simultaneously, in overlapped manner. The purpose of using this option is to issue access to SRAM every clock cycle.

The resulting performance of synthesized units (Table 1 and 2) is comparable to that of conventional hand-made RTL of RSA and ECC. The number of clock cycles is independent of ASIC/FPGA process libraries[6].

3 : For instance, an expression "64-bit number × 64-bit number" does not return full 128-bit value in standard C language. In S/W, the expression has to be divided into four 32-bit multiplications.

4 : Use of pointers whose destinations are changed dynamically.

5 : The real code is written not in SystemC but in another extended-C,

```
/* s = Montgomery_mult(x, y, n) */
void modulo_unit::mont_mult(
 data_sram MEM; // OPTION: MAP TO DP-SRAM, LATENCY=2CK
 sc_uint<9> x, y, n, s, /* memory address */
 sc_uint<CTRL_LEN> blklen
)
{
  sc_biguint<96> acc; // OPTION: MAP TO REGISTER
  sc_uint<32>    x_ini, ss;
  ....
  for (i = 0; i < blklen; i++) {
    ....
    /* Without using loop folding, 5 clock cycles
       will be necessary for one iteration. */
    // OPTION : FOLD THIS LOOP, OVERLAP INTERVAL=2CK
    for (j = i; j < blklen; j++) \{
      /* 96-bit operation can be done without
         separating into 32-bit operations. */
      acc = acc.range(95, 32) + MEM[s+j]
            + x_ini * MEM[y+j] + ss * MEM[n+j];
      MEM[s+j-1] = acc.range(31, 0);
    }
    ....
  }
}
```

Table 3 Main part of Montgomery multiplication written in SystemC

4.3 Use of a Custom Sub-IP Level Behavioral Synthesizer

As will be discussed in Section 5., the total IP performance is mostly determined by how many sub-IP operations are executed in parallel. However, standard behavioral synthesizers [15], [16] only support parallel scheduling of C-embedded operators ($+, -, \times, /, \%$) and does not support neither scheduling of higher level functions nor scheduling of functions whose operation mode is determined at run time.

Therefore, as shown in Step 4 in Fig. 5, a custom sub-IP level (C function-library level) behavioral synthesizer was made and used. Inputs of the custom synthesizer are (i) a sequential description of entire algorithm (equations in Section 2.4 and 2.5) (ii) number of sub-IPs, and (iii) clock cycle counts of every operation in sub-IPs which are obtained by RTL simulation, FPGA prototyping or theoretical estimation. The custom synthesizer outputs a paralleled computation sequence, where start order of each sub-IP operation and assignment of sub-IP unit-number are specified. This output is used to write a micro-code of the data transfer controller.

The custom synthesizer solves a RCPSP (Resource Constrained Project Scheduling Problem) using a heuristic such that an operation start as soon as a corresponding sub-IP is available and an operation whose execution time is longer starts earlier. This custom synthesizer is specific to the group signature, and many general ESL/TLM synthesis issues [17]~[19] are omitted in the following aspects, because of features in Section 3.1; (1) data-value analysis is unnecessary, (2) data transfer time between sub-IPs can be ignored, and (3) exploring local-bus topology and/or synthesizing macro pipeline are unnecessary.

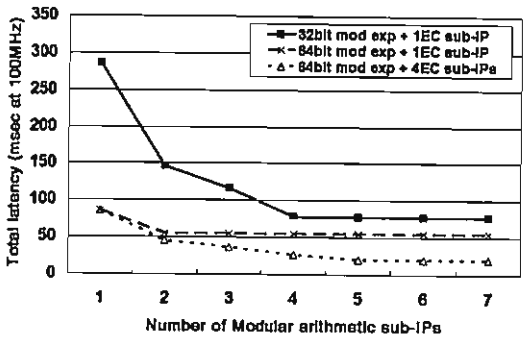


Fig. 6 Performance of signature generation (standard security level)

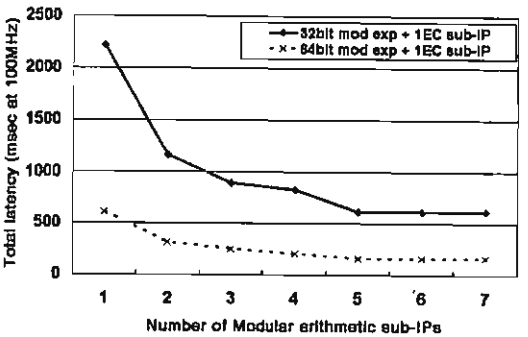


Fig. 7 Performance of signature generation (high security level)

5. Architecture Optimization Result

Using the custom synthesizer in Section 4.3, we investigated relationship between total computation speed and number of sub-IPs. Results of signature generation/verification speed in standard/high security levels are shown in Fig. 6, 7, 8 and 9. Although these results are obtained from the absolute values of clock cycle counts in Table 1 and 2, simular results will be obtained even if process libraries or target devices are changed, because the number of clock cycles is independent of libraries (see Section 4.2).

yet there is no major difference other than syntax.

6: Besides, maximum clock frequency is much affected by process libraries.

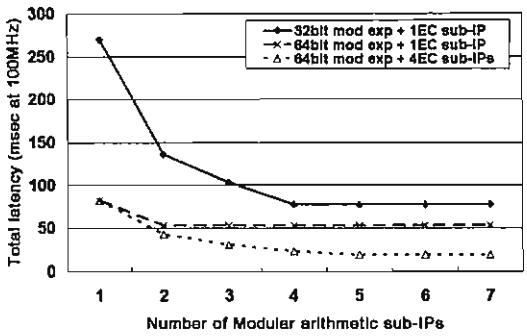


Fig. 8 Performance of signature verification (standard security level)

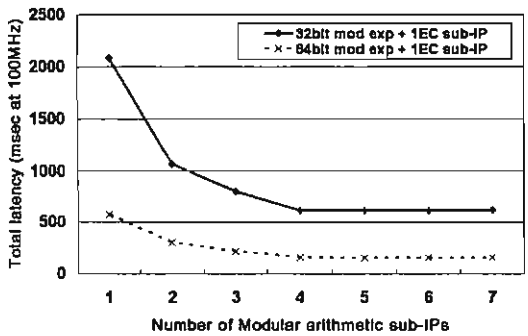


Fig. 9 Performance of signature verification (high security level)

If all operations are executed in serial manner, more than 80% of total computation time is occupied by modular exponentiation (Table 1 and 2). Therefore, increasing number of the modular arithmetic sub-IP significantly improves total speed, and maximum speed is reached when 4 or 5 modular sub-IPs are used. Besides, increasing number of the other sub-IPs has no effect except when a 64-bit modular exponentiation is used in standard security level. Also, increasing bit width of RAM and Montgomery multiplier in the modular arithmetic sub-IP is effective. The number of clock cycles consumed by modular exponentiation is $O((\frac{n1}{m})^2 \cdot n2)$ where m is bit with of RAM, $n1$ is bit width of base (1024 or 2048) and $n2$ is bit width of exponent. Maximum clock frequency is slowed down ($O(\frac{1}{logm})$) if m increases, yet the effect is not so large compared to the reduction in clock cycles.

Another important observation is that the optimum number of sub-IPs is the same between signature generation and verification. The same H/W can be used in generation and verification without dropping performance.

Example circuit size of each sub-IP (standard security level), when mapped to a 130nm standard cell ASIC is shown in Table 4. The AHB bus was tentatively used as a local bus. The total circuit size, if number of each sub-IP is one, is 350K

Sub-IP	Gate count	Memory size/port
EC arithmetic + Bus I/F	67.8K	85 words, Rx1 Wx1
Modular arithmetic + Bus I/F	52.5K	270 words, RWx2
INT arithmetic + Bus I/F	44.6K	277 words, Rx1 Wx1
Hash/PRNG + data transfer ctrl	180.1K	80 words, RWx1
Common RAM	-	748 words, RWx1
TOTAL (If number of each sub-IP is 1)	345.0K	

Table 4 Size of each sub-IP when mapped on a 130nm standard cell ASIC (standard security level)

ASIC gates. One additional modular arithmetic unit will increase the total circuit size about 50K gates. If high security level is selected, number of logic gates remains the same and the amount of SRAM is doubled. Maximum clock frequency was 150-200MHz under worst delay condition. We mapped the same circuit on an FPGA device and confirmed that signature generation and verification can be done correctly within an expected time.

6. Conclusion

In this paper, we investigated an appropriate H/W architecture and design methodology of group signature IP for SoC in mobile devices. The speed of modular exponentiation operation determines total performance and a custom sub-IP level behavioral synthesizer, which is specific to group signature algorithm, was necessary to achieve highly parallel computation. To the best of the authors' knowledge, this is the first report of high performance IP implementation of a typical group signature algorithm.

Acknowledgment

This work was (partly) supported by Ministry of Internal Affairs and Communications.

References

[1] D.Chaum and E.van Heyst, "Group signatures," EUROCRYPT '91, LNCS Vol.547, pp.257-265, 1991.

[2] J.Camenisch and M.Michels, "Separability and Efficiency for Generic Group Signature Schemes," CRYPTO 1999, pp.413-430, 1999.

[3] J.Camenisch and J.Groth, "Group signatures: Better efficiency and new theoretical aspects," SCN 2004, LNCS Vol. 3352, pp.120-133, 2004.

[4] I.Teranishi, J.Furukawa, and K.Sako, "k-Times anonymous authentication," ASIACRYPT2004, LNCS Vol.3329, pp.308-322, 2004.

[5] T.Isshiki, K.Mori, K.Sako, I.Teranishi, and S.Yonezawa, "Using Group Signature for Identity Management and its Implementation," DIM2006, 2006.

[6] J.Groth, "Fully Anonymous Group Signatures Without Random Oracles," Proc. ASIACRYPT 2007, LNCS Vol.4833, pp.164-180, 2007.

[7] C.Popescu, "Applications of group signatures to anonymous fingerprinting schemes," VIPromCom-2002, 4th EURASIP - IEEE Intl. Symp. on Video/Image Processing and Multimedia Communications, pp.177-182, 2002.

[8] X.Sun, X.Lin and P.H.Ho, "Secure Vehicular Communications Based on Group Signature and ID-Based Signature

Scheme," IEEE International Conference on Communications 2007 (ICC '07), pp.1539-1545, 2007.
[9] C.K.Koc, "High-speed RSA implementation," Technical Report TR201, RSA Laboratories, 1994.
[10] P.Schaumont and I.Verbauwhede, "Domain Specific Tools and Methods for Application in Security Processor Design," Kluwer Journal for Design Automation of Embedded Systems, pp. 365-383, 2002.
[11] P.Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," CRYPTO 1996, pp.104-113, 1996.
[12] C.Giraud, "An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis," IEEE Transactions on Computers, Vol.55, No.9, pp.1116-1120, 2006.
[13] A.E.Cohen and K.K.Parhi, "A New Side Channel Resistant Scalar Point Multiplication Method for Binary Elliptic Curves," Fortieth Asilomar Conference on Signals, Systems and Computers, 2006 (ACSSC '06), pp.1205-1209, 2006.
[14] D.Stebila and N.Theriault, "Unified Point Addition Formulae and Side-Channel Attacks," Proc. CHES 2006, LNCS Vol.4249, pp.354-368, 2006.
[15] K.Wakabayashi and T.Okamoto, "C-Based SoC Design Flow and EDA Tools: An ASIC and System Vendor Perspective," IEEE trans. on CAD, vol.19, no.12, pp.1507-1522, 2000.
[16] K.Wakabayashi, "CyberWorkBench: integrated design environment based on C-based behavior synthesis and verification," IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test, 2005 (VLSI-TSA-DAT), pp.173-176, 2005.
[17] A.Habibi and S.Tahar, "Design and verification of SystemC transaction-level models," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol.14, No.1, pp.57-68, 2006.
[18] S.Murali, L.Benini, G.Micheli, "An Application-Specific Design Methodology for On-Chip Crossbar Generation," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol.26, No.7, pp.1283-1296, 2007.
[19] Y.Hwang, S.Abdi and D.Gajski, "Cycle-approximate Retargetable Performance Estimation at the Transaction Level," Design, Automation and Test in Europe, 2008 (DATE '08), pp.3-8, 2008.