Flexible Architecture Optimization and ASIC Implementation of Group Signature Algorithm using a Customized HLS Methodology

Sumio Morioka † Toshiyuki Isshiki ‡ Satoshi Obana ‡ Yuichi Nakamura † Kazue Sako ‡
† System IP Core Research Laboratories, NEC Corporation
‡ Information and Media Processing Laboratories, NEC Corporation
1753 Shimonumabe, Nakahara-ku, Kawasaki, Kanagawa 211-8666, Japan
{s-morioka@ak, t-issiki@bx, obana@bx, yuichi@az, k-sako@ab}.jp.nec.com

***Abstract*— Group signature is one of the main theme in recent digital signature studies. Typical signature algorithm is a combination of more than 70 elliptic curve (ECC), modular (RSA), long-bit integer and hash arithmetic functions. A full H/W IP core is strongly desired for the use of group signature in SoCs in slow-clock and low-power mobile devices and embedded systems. Flexible adjustment of H/W speed and size, depending on different systems and LSI process technologies, is also required. However, for designing and verifying H/W, the group signature algorithm is too complicated to use a standard RTL (Register Transfer Level) design methodology nor any recent HLS (High Level Synthesis). Therefore, we incorporated a two-level behavioral synthesis approach, where an optimized macro-architecture is explored by a custom-made scheduler, after a database of multiple number of micro-architectures are effectively constructed by conventional HLS. We implemented the signature algorithm on a low-cost 0.25um gate-array. The H/W size is approximately 1M gates and our chip can compute a group signature at the equivalent speed (0.135 seconds@100MHz clock) with 3GHz PC S/W, while the power consumption is two orders of magnitude lower (425mW@100MHz).**

I. 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], [3], [4]. In this scheme, users can sign messages anonymously, although there is an authority that can trace the signer (Fig. 1). Group signatures have many practical applications such as e-voting, e-cash, fingerprinting, vehicular communication and so on[5], [6]. Regarding to the speed of group signature, a recent S/W implementation[4] 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 (LSI) implementation has been desired for the use of group signature in LSIs in slow-clock (up to 300MHz or so) and low-power ($<$ 1W) mobile devices end embedded systems. In these systems, not only high H/W performance and low-power but also high flexiblity of adjusting H/W architecture is quite important, because all of the required throughput, operating-clock frequency, LSI process technologies, and the other requirements are significantly different between systems. To the best of the authors' knowledge, there have been few reports on H/W implementation of group signature algorithm.

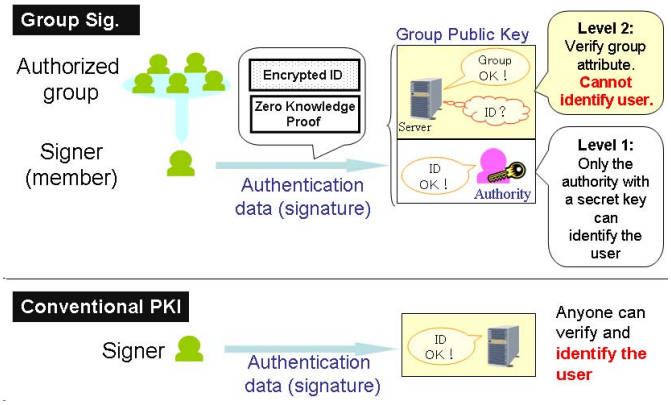


Fig. 1. Difference between group signature and conventional digital signature

The typical signature algorithm[3], [4] is a combination of more than 70 primitive functions which are also used in RSA, elliptic curve cryptography (ECC) and hash functions[7], [8], [9]. The standard "datapath + FSM (Finite State Machine)" H/W architecture and RTL (Register Transfer Level) design entry, which are used in traditional cryptographic H/W designs such as AES, ECC, RSA and SHA, are not suitable for group signature, because of design difficulty, slow H/W performance and low portability. In particular, RTL design-entry highly restricts flexible and quick macro/micro-architecture reconstruction when target system specification and/or target LSI process are changed.

In this paper, we investigated an appropriate architecture and design methodology for designing a high performance and low-power group signature H/W, as follows.

1. We found that in group signature computation, amount of data transfer between primitive functions is little and therefore, the best way to achieve all of high performance, portability and configurability is to connect a data transfer controller and multiple *cores* (perform elliptic curve, modular, INT and hash functions, respectively) 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 HLS (High Level Synthesis)[1], where optimized RTLs are synthesized from sequential algorithm

[1]Also called as behavioral synthesis, while HLS involves behavioral synthesis. In this paper, we do not distinguish them.

978-1-4577-1058-2/11/$26.00 ©2011 IEEE

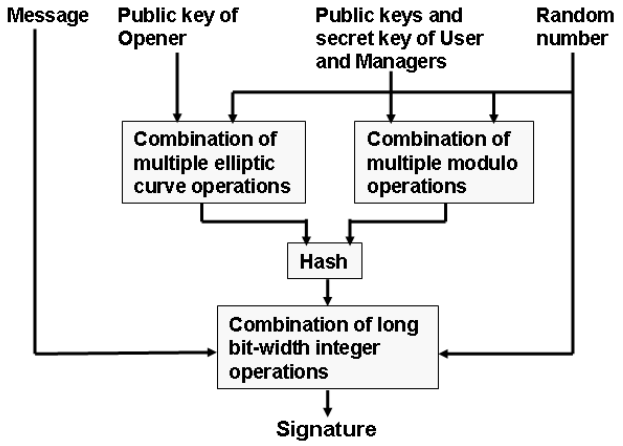


Fig. 2. Data flow of group signature generation

descriptions in C language[11], [12], is suitable in order to avoid time consuming RTL design/simulation works. However, current HLS and cannot directly treat rather large algorithms such as group signature, because parallel scheduling of core-level functions or tasks, i.e. RSA, ECC and hash functions, is not yet supported by various difficulties on H/W architecture synthesis algorithm. Therefore, we made a custom upper level behavioral synthesizer for exploring *macro-architecture*, i.e. how to arrange cores. The conventional HLS is used in order to synthesize low level H/W (*micro-architecture*) inside cores. This two-level synthesis strategy is not generic but specific to group signature.

3. We investigated a relationship between total computation speed and the number of cores.

We implemented a sample 1024-bit group signature ASIC on a low-cost 0.25um gate-array, in order to confirm that our design methodology works well. Practical H/W speed of 0.135 seconds at 100MHz, which is the same speed with S/W on 3GHz PC, was achieved with very low power consumption, 425mW. If a newer LSI process is used, less than 100mW power consumption is easily possible.

This paper is organized as follows. In Section II, we explain a typical group signature algorithm. In Section III, we explain the proposed H/W architecture and design methodology. In Section IV, we will show architecture optimization and sample ASIC implementation results.

II. Group Signature Algorithm

A. Model

In this paper, we have implemented one of a typical and fast group signature algorithm described in [3], [4]. 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 the main user of group signature will be *User*.

Primitive function	Bit length	CLK cycles	Times	Ratio
EC Scalar Multiplication	160	970,055	7	31.4%
EC Point Addition	160	4,861	4	<0.1%
Modular Multiplication	1024 x 1024	1,841	5	<0.1%
Modular Exponentiation	$(1024)^{280}$	521,439	2	4.8%
Modular Exponentiation	$(1024)^{512}$	943,291	2	8.7%
Modular Exponentiation	$(1024)^{792}$	1,500,187	1	6.9%
Modular Exponentiation	$(1024)^{1236}$	2,341,599	1	10.8%
Modular Exponentiation	$(1024)^{1464}$	2,804,294	1	13.0%
Modulo Mult Inv.	1024	1,713,022	2	15.8%
INT Multiplication	160 x 60	84	1	<0.1%
INT Multiplication	160 x 160	125	1	<0.1%
INT Multiplication	160 x 576	301	1	<0.1%
INT Multiplication	160 x 1016	489	1	<0.1%
INT Multiplication	160 x 1244	584	1	<0.1%
INT Multiplication	512 x 50	239	1	<0.1%
INT Multiplication	512 x 504	736	1	<0.1%
INT Modulo	321 % 160	4,611	1	<0.1%
INT Modulo	1464 % 160	34,146	1	0.16%
SHA-224	(message len)	(<10,000)	1	<0.1%
Random number gen	160	29,720	2	0.2%
Random number gen	280	35,608	1	0.16%
Random number gen	512	35,688	2	0.33%
Random number gen	792	47,475	1	0.22%
Random number gen	1236	53,342	1	0.25%
Random number gen	1464	69,070	1	0.32%
Data transfer & control	––	*1,286,806*	––	*5.9%*
Total (SEQUENTIAL)	––	21,633,992	––	100%

TABLE I

Examples of the speed of primitive functions in group signature generation

B. 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 $\mathscr{G}$, 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, 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, these values are 2048, 2048, 736, 60, 224, 224, 112, respectively.

C. Public Key and Secret Key

Let $\mathscr{G}$ denotes a finite group and order q is a prime number whose bit-length is κ_q. Also, let Λ, $[c]G$, $+_e$ and $-_e$ denotes 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 $y_1, y_2 \in \mathbb{Z}_q$, $G \in \mathscr{G}$ 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* $\mathsf{mpk}_i = (h_i, A_i, e'_i, B_i)$, $\mathsf{msk}_i = x_i$,

Primitive function	Bit length	CLK cycles	Times	Ratio
EC Scalar Multiplication	160	970,055	7	33.7%
EC Point Addition	160	4,861	5	0.12%
EC Point Negation	160	48	3	<0.1%
Modular Multiplication	1024 x 1024	1,841	5	<0.1%
Modular Exponentiation	$(1024)^{160}$	299,568	2	3.0%
Modular Exponentiation	$(1024)^{280}$	521,439	1	2.6%
Modular Exponentiation	$(1024)^{665}$	1,230,148	1	6.1%
Modular Exponentiation	$(1024)^{793}$	1,475,235	1	7.3%
Modular Exponentiation	$(1024)^{1237}$	2,298,414	1	11.4%
Modular Exponentiation	$(1024)^{1465}$	2,727,297	1	13.5%
Modulo Mult Inv.	1024	1,713,022	2	17.0%
INT Modulo	1465 % 160	34,473	1	0.17%
SHA-224	672	260	1	<0.1%
Data transfer & control	––	*1,001,399*	––	*5.0%*
Total (SEQUENTIAL)	––	20,137,884	––	100%

TABLE II

EXAMPLES OF THE SPEED OF PRIMITIVE FUNCTIONS IN GROUP SIGNATURE VERIFICATION

satisfying $x_i \in \Lambda$, $B_i = b^{1/e'_i} \bmod \ell$, $e_i = 2^{\kappa_e} + e'_i$, and $a_0 a_1^{x_i} \equiv A_i^{e_i} \bmod n$.

D. 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-224. The signature generation algorithm is shown below (see also Fig. 2).

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 $\mu_E \in \mathbb{Z}_q$, randomly.

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

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

4. Compute $c = \mathsf{Hash}(\kappa, \mathsf{ipk}, \mathsf{opk}, \mathsf{rpk}, E, A_{\mathrm{COM}}, B_{\mathrm{COM}}, V_{\mathtt{ComCipher}}, V_{\mathtt{ComMPK}}, V_{\mathtt{ComRev}}, m)$.

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 $(E, A_{\mathrm{COM}}, B_{\mathrm{COM}}, c, \tau_x, \tau_s, \tau_{e'}, \tau_t, \tau_E)$.

E. Signature Verification Algorithm

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

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

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

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

4. Check if $c = \mathsf{Hash}(\kappa, \mathsf{ipk}, \mathsf{opk}, \mathsf{rpk}, E, A_{\mathrm{COM}}, B_{\mathrm{COM}}, V'_{\mathtt{ComCipher}}, V'_{\mathtt{ComMPK}}, V'_{\mathtt{ComRev}}, m)$ holds. If holds, output accept and otherwise, output reject.

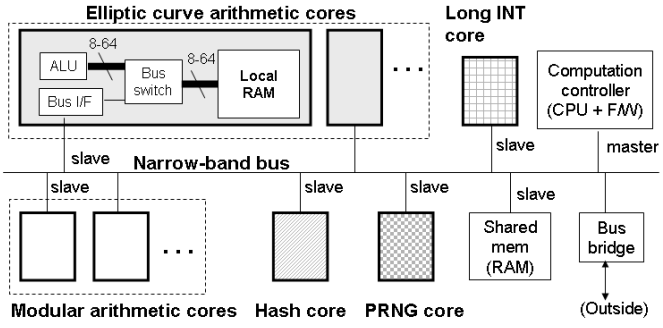


Fig. 3. The proposed macro-architecture of group signature H/W

III. SoC-like Group Signature H/W

A. Feature of the Algorithm from H/W Design Standpoint

Comparing with traditional digital signatures, the group signature algorithm have some significant features from H/W design standpoint, as described below.

- The algorithm is a complicated combination (> 70 steps in total) of primitive functions listed in Table I and II.

- Bit width of data is large and most of data should be stored not on registers but on memories (SRAMs). Most of total computation time is occupied by memory access cycles in each primitive function. Besides, only a small ratio (a few percents) is consumed by data transfer between primitive functions, as shown in Table I and II.

- The standard "single datapath + single FSM" architecture, which is common in traditional cryptographic H/Ws[7], [8], cannot be used, because parallel execution of multiple functions of varying kinds and with different execution times is almost impossible.

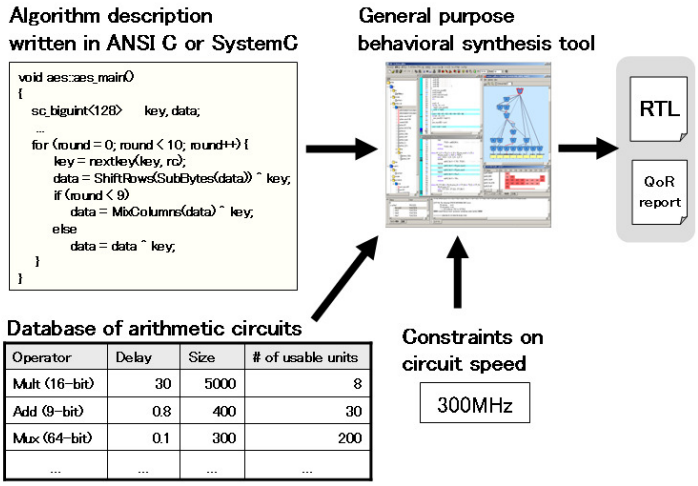
- Verification and debugging of entire signature computation on RT-level is quite time consuming. Because RTL simulation of traditional public key algorithms even consume several hours, simulation of entire group signature (consumes 10 millions to 300 millions of clock cycles) will take more than a day.

B. The Proposed SoC-like Macro-architecture

We have decided to use a *macro-architecture* shown in Fig. 3. Multiple number of cores are connected each other by a local bus. Each core corresponds to elliptic curve (EC), modular (RSA), long-bit INT and hash function respectively, and has multiple modes for varying kinds of arithmetic operations and data bit width as shown in Table I and II.

Under this macro-architecture, we can easily configure H/W performance and size by (1) changing number of cores and (2) changing configuration of cores. Although the speed of fastest core is only shown in Table I and II, we can make a lot of configurations of cores whose speed and size are different, by chaging H/W structure inside cores (*micro-architecture*).

While this macro-architecture is similar with top level structure of conventional SoCs, an important difference is that neither wide-band bus nor multiple number of buses is necessary even in the highest speed implementation. Single narrow-band bus is enough because amount of traffic on the



Operator	Delay	Size	# of usable units
Mult (16-bit)	30	5000	8
Add (9-bit)	0.8	400	30
Mux (64-bit)	0.1	300	200
...	...	...	...

Fig. 4. Conventional general purpose behavioral synthesis (or HLS) method

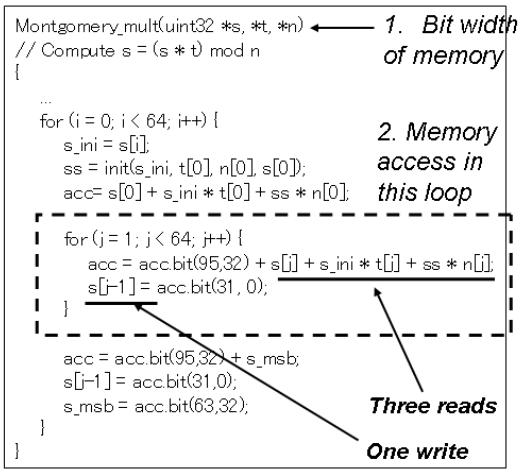


Fig. 5. Micro-architecture optimization items

Fast RSA 2048-bit core	HLS [12]	Hand-made RTL
Latency (@100MHz)	1ms	1ms
Max clock frequency	230MHz	300MHz
Gate count	70K	60K
Code size	500lines (SystemC)	1900lines (Verilog)

AES-CBC 128-bit core	HLS [12]	Hand-made RTL
Latency (@100MHz)	11clock	11clock
Max clock frequency	200MHz	220MHz
Gate count	25K	22K
Code size	760lines (SystemC)	860lines (Verilog)

TABLE III
COMPARISON OF HLS AND CONVENTIONAL RTL ENTRY (0.13UM STANDARD CELL)

bus is little, as mentioned in Section III-A. Most of computation time is consumed inside each core and communication overhead is very small.

In Fig. 3, a compact custom-made CPU[2] is used in order to transfer data between cores, issue computation-start commands and monitor computation progress in cores.

IV. ARCHITECTURE OPTIMIZATION BY A CUSTOMIZED HLS METHODOLOGY

A. Full Use of HLS and FPGA Emulation for Flexible Architecture Change

We fully used a combination of C-based H/W modeling, HLS (or behavioral synthesis)[11], [12] and FPGA emulation, because of the following reasons:

- Achieve high configurability of H/W by reducing RTL implementation/FPGA emulation cost. As described later, we prepared multiple configurations (implementations) of the same function cores whose micro-architectures are very different, in order to explore macro-architecture. If RTL design entry was used, changing micro-architecture a lot of times was too time consuming and was an unrealistic work.
- Enable functional verification of total H/W by avoiding use of RTL simulation. Even though a fairly simple bus-architecture (Fig. 3) is adopted, full verification of entire H/W cannot be eliminated because checking firmware of CPU (the data transfer controller) is necessary. For this purpose, RTL simulation cannot be used at all because of too slow speed. This issue is common with SoC's top-level verification. Combination of simulation in behavioral/transaction level and FPGA emulation is appropriate.

B. Limitation of Current HLS

Typical HLS tools generate RTL from sequential algorithm descriptions (often written in C language)[11], [12] (Fig. 4). As shown in Table III, recent HLS tools are so powerful and reliable that the synthesis quality is comparable with hand-made RTL, when tools are applied to traditional cryptographic algorithms such as AES, TDES, ECC, RSA and SHA[10], [12]. This is the reason why we accept to use HLS in this work.

However, it is still difficult to synthesize a 'usable' H/W from larger S/W codes. In fact, we could not reuse our fast group signature S/W[4] as is. One of the main reasons is that none of current HLS tools can perform macro-architecture synthesis involving large-function level or task level scheduling and pipelining (so called macro-pipelining) [3]. Current HLS tools only support parallel scheduling of small C-embedded operators ($+, -, \times, /, \%$ etc.) and micro-architecture synthesis.

C. Evaluating Multiple Micro-architectures by Conventional HLS and FPGA Emulation

Regarding to H/W implementation of modulo (RSA), ECC and hash functions, a lot of research have been already done in the past [7], [8], [9].

In the group signature H/W, the speed of modulo and EC functions are critical as shown in Table I and II. Because these functions treat long-bit data (keys, plaintext and ciphertext) on local memories, adjusting bandwidth of the memories are important. For instance, modulo exponentiation operation is performed by a famous combination of binary method and Montgomery multiplication (Fig. 5)[7]. The speed of H/W is determined by the bit width of memory−ALU (Montgomery multiplier, in this case)

[2]Any commercial embedded processors can be used, too.

[3]The C-code of group signature algorithm is 20 or more times larger than RSA or AES code.

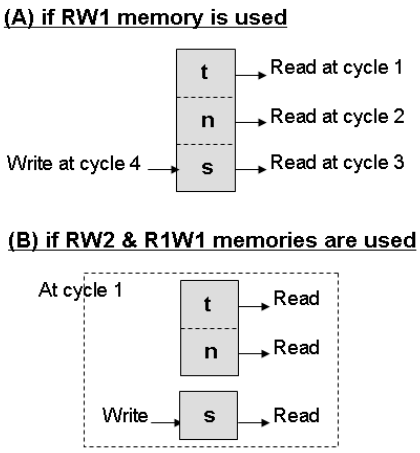


Fig. 6. Micro architecture optimization; adjusting number of memory ports

Type of modulo (RSA) core	Clock cycles (1024^{280})	Size (LUTs)	Average dynamic power (mW@70MHz)
RW2+R1W1 mem, 32-bit port	539,241	15,915	104
R1W1+R1W1 mem, 32-bit port	856,256	16,975	117
R1W1+RW1 mem, 32-bit port	865,248	16,453	127
RW1+RW1 mem, 32-bit port	868,012	16,359	101
RW2+R1W1 mem, 16-bit port	1,486,665	11,249	78
R1W1+R1W1 mem, 16-bit port	2,696,464	11,419	76
R1W1+RW1 mem, 16-bit port	2,696,464	11,435	69
RW1+RW1 mem, 16-bit port	2,701,916	11,081	98
RW2+R1W1 mem, 8-bit port	5,161,929	8,612	54
R1W1+R1W1 mem, 8-bit port	9,919,728	8,705	52
R1W1+RW1 mem, 8-bit port	9,919,728	8,588	46
RW1+RW1 mem, 8-bit port	9,930,556	7,230	55

TABLE IV
EXAMPLE OF DIFFERENT CONFIGURATIONS OF MODULO CORE
(VIRTEX-6 XC6VLX760 FPGA)

connection and the possible number of simultaneous memory access, as summarized in Fig. 6. Of course, H/W size becomes larger if number of memory ports and/or ALU bit width are increased.

By using conventional HLS and fast FPGA emulation, we can efficiently gather the actual data of speed, size and even power consumption of different configurations of cores. An example is shown in Table IV.

D. Macro-architecture Exploration using a Group Signature Specific HLS

If all of the functions in Table I and II are executed in serial manner, total H/W speed becomes unacceptably low. An appropriate parallel scheduling of function executions, whose clock cycles are much different, is essential. Selection of number and configuration of cores outlines not only the H/W speed but also the size. In addition, unlike sequential S/W computation, it is unnecessary to use the fastest cores for *all* functions when parallel computation is possible. The cores on non-critical paths should be slowed down in order to reduce H/W size.

Therefore, a custom higher level (core-function level) behavioral synthesizer was made and used (Fig. 7). Inputs of this custom synthesizer are (i) a sequential description of

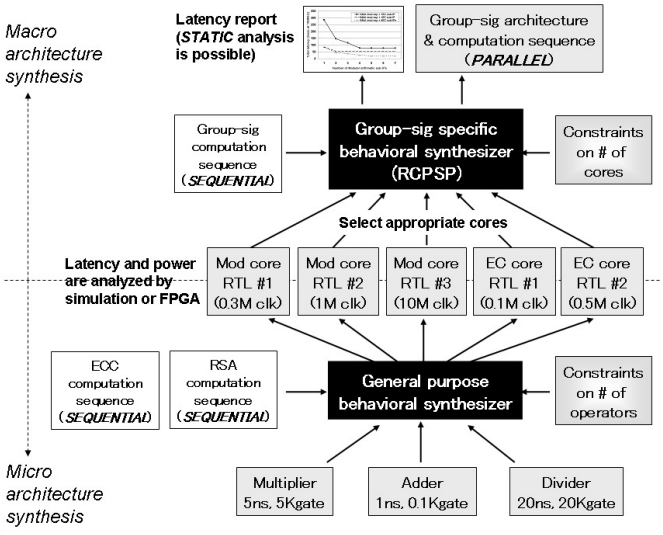


Fig. 7. The proposed two-level synthesis method customized for group signature circuit

entire group signature algorithm (equations in Section II), (ii) maximum number of cores, and (iii) core performance data gathered by conventional HLS (Table IV). The custom synthesizer outputs a paralleled computation sequence, where start order of each function computation and assignment of core-unit number are specified.

This custom synthesizer evaluates H/W speed not by dynamic simulation but by a static method; it solves a RCPSP (Resource Constrained Project Scheduling Problem) using a heuristics such that a function execution start as soon as a corresponding core is available and a function whose execution time is longer starts earlier.

Our two-level synthesis approach is very specific to group signature algorithm and it is inappropriate to adopt to general SoC designs. We can use this 'static' method because all of the following conditions hold in the group signature case; (1) there is no stream-type pipelined computation and H/W performance can be computed by simply summing latencies of cores on the critical path, (2) latency is independent of data, (3) overhead of data transfer between cores is very small and can be ignored, and (4) exploring and synthesizing bus topology are unnecessary. If one or more conditions above is not satisfied, 'dynamic' simulation is necessary for evaluating H/W speed, in general.

V. EXPERIMENTAL RESULTS

A. Architecture Exploration Result

Using the proposed two-level synthesis methodology in Section IV, we investigated how H/W performance vary when number and configuration of cores are changed (Fig. 8 and Fig. 9). Clearly, increasing number of the modulo core significantly improves total speed, and maximum speed is reached when 4 or 5 modulo cores are used. Besides, increasing number of the other cores has much smaller effect. Also, increasing band width of memory−Montgomery multiplier connection in the modulo core is effective. The number of clock cycles consumed by modular exponentiation

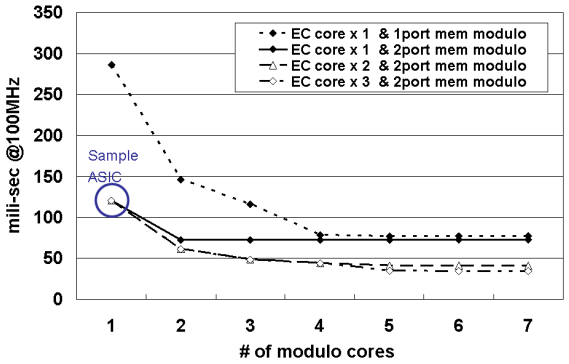


Fig. 8. Performance analysis result (1024-bit signature generation)

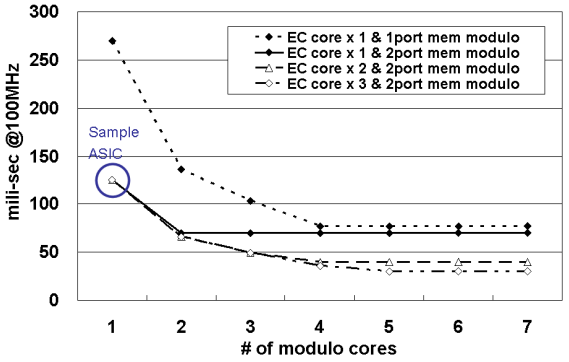


Fig. 9. Performance analysis result (1024-bit signature verification)

is $O((\frac{n_1}{m})^2 \cdot n_2)$ where m is bit width of this connection, n_1 is bit width of base and n_2 is bit width of exponent. Maximum clock frequency is slowed down ($O(\frac{1}{logm})$) if m increases.

Another important observation is that the optimum number of cores is the same between signature generation and verification. This means that the same H/W can be used between these computations without dropping performance.

B. ASIC Implementation Result

We successfully implemented a sample 1024-bit group signature H/W on a low-cost 0.25um ASIC (Fig. 10). In this sample ASIC, number of modulo, EC, long-bit INT and hash core is set to all 1. While the number of local memory ports is maximized (= 2 ports), bit width of ALU is limited to 32 for achieving 100MHz clock frequency.

The ASIC implementation result is summarized in Table V. The actual latency is almost the same with the predicted value by RCPSP solver in Section IV-D. No significant issue is found in timing closure, DFT synthesis and the other back-end procedures.

Even though HLS was fully used, we were able to control the synthesis so that the speed of modulo and EC functions is independent of secret-key values, as a countermeasure to basic side channel attacks such as timing attack.However, further investigation on the vulnerability of group signature H/W to recent advanced attacks is still remaining as a future work.

VI. Conclusion

In this paper, we investigated a H/W architecture and design methodology of group signature IP for SoCs in mobile devices. While use of HLS is essential, a custom macro-architecture synthesizer is necessary because capability of current HLS is limited to micro-architecture synthesis. To the best of the authors' knowledge, this is the first report of high performance and low-power ASIC implementation of group signature algorithm.

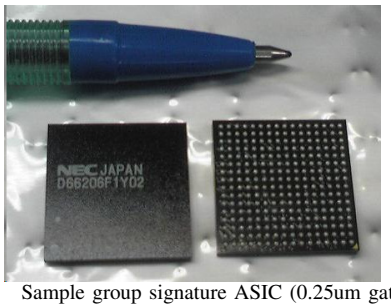


Fig. 10. Sample group signature ASIC (0.25um gate array)

Maximum clock speed	100MHz (worst condition)
Latency of signature generation	0.135 seconds (100MHz)
Latency of signature verification	0.135 seconds (100MHz)
Gate size (logic part)	81M gate (after DFT) 65M gate (before DFT)
On-chip SRAM amount	total 320K bits
Test coverage	>95% by ATPG
Average power consumption	425mW (100MHz, 1.5V)

TABLE V

Sample ASIC implementation result (0.25um gate array)

Acknowledgement

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] T.Isshiki, K.Mori, K.Sako, I.Teranishi, and S.Yonezawa, "Using Group Signature for Identity Management and its Implementation," DIM2006, 2006.
[5] 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.
[6] 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.
[7] C.K.Koc, "High-speed RSA implementation," Technical Report TR201, RSA Laboratories, 1994.
[8] 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.
[9] A.Satoh and K.Takano, "A Scalable Dual-Field Elliptic Curve Cryptographic Processor," IEEE trans. on Computers, vol.52, no.4, pp.449-460, 2003.
[10] Sumio Morioka and Akashi Satoh, "A 10Gbps Full-AES Crypto Design with a Twisted-BDD S-Box Architecture," 2002 IEEE International Conference on Computer Design (ICCD 2002), pp.98-103, Sep 2002.
[11] 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.
[12] Sumio Morioka, "The Inevitable Use of Behavioral Synthesis in Advanced Security Hardware Designs," TUTORIAL 2: High-Level Synthesis for ESL Design: Fundamentals and Case Studies, The 46th Design Automation Conference (DAC), July 2009.