Architecture Optimization of Group Signature Circuits for Cloud Computing Environment

Sumio Morioka Jun Furukawa Yuichi Nakamura Kazue Sako

Central Research Laboratories, NEC Corporation
1753 Shimonumabe, Nakahara-ku, Kawasaki, Kanagawa 211–8666 Japan
{s-morioka@ak, j-furukawa@ay, yuichi@az, k-sako@ab}.jp.nec.com

Abstract— Group signature is one of the main theme in recent digital signature studies. The signature algorithm is a combination of more than 30 elliptic curve (ECC), modular (RSA), long-bit integer (INT) and hash arithmetic operations. Low-power and fast H/W accelerators are strongly desired in cloud computing environment where a lot of client devices (mobile devices, embedded systems, sensor devices and etc.) are connected to servers in data center via network. In this paper, we propose a H/W macro-architecture for servers in data center, and will compare it with the architecture for client devices. While these architectures are completely different, we can use the same H/W design methodology where the architectures are explored automatically by a custom-made HLS (High Level Synthesis) tool.

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). While a lot of applications of group signatures have been proposed and studied[5, 6, 7], we consider that the use in cloud computing environment in order to provide secure SaaS (Software as a Service) and PaaS (Platform as a Service) will be a main application in the very near future.

Regarding to the speed of group signature, a recent S/W implementation based on a typical signature algorithm[3] achieves 0.1-0.2 seconds on 3GHz PC[4]. The achieved speed is fast enough under a rather limited situation such that a high power consumption PC (> 100W) can be used and only one signature computation is required at a time.

However, fast and low-power H/W (LSI) accelerators have been desired for the use of group signature not only in slow-clock client devices (mobile devices, embedded systems, sensor devices and etc.) but also in servers in data center (Fig. 2). Server acceleration is particularly

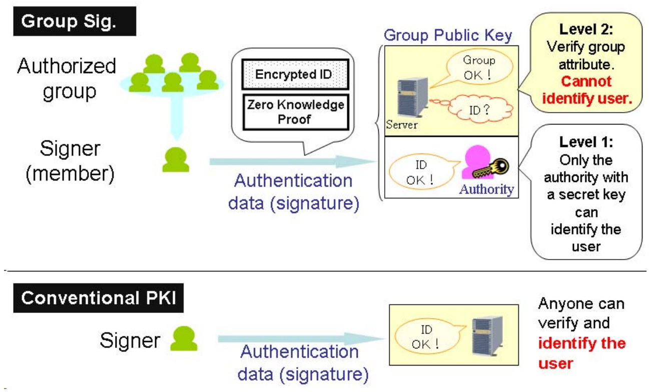

Group Sig.
Authorized group
Signer (member)
Encrypted ID
Zero Knowledge Proof
Authentication data (signature)
Group Public Key
Group OK !
ID ?
Server
ID OK !
Authority
Level 2: Verify group attribute. Cannot identify user.
Level 1: Only the authority with a secret key can identify the user
Conventional PKI
Signer
Authentication data (signature)
ID OK !
Anyone can verify and identify the user

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

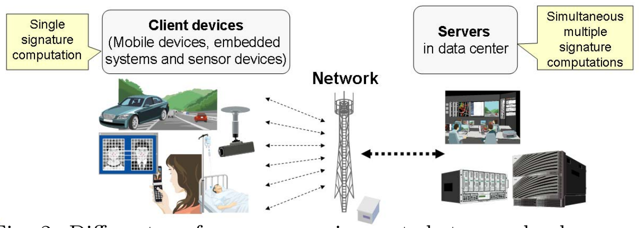

Single signature computation
Client devices (Mobile devices, embedded systems and sensor devices)
Network
Servers in data center
Simultaneous multiple signature computations

Fig. 2. Different performance requirements between cloud servers and clients

important because a lot of signature requests should be processed simultaneously under strict restrictions of TAT, power consumption and total number of servers.

The typical group signature algorithm[3, 4] is a combination of more than 30 primitive operations which are also used in RSA, ECC and hash functions[8, 9]. For implementing the complicated group signature algorithm into H/W, optimization of higher level architecture (so called *macro-architecture*) and applying ESL (Engineering System Level) design methodology are necessary.

In this paper, we propose a H/W macro-architecture for servers in data center, and will compare it with the architecture for client devices[14]. A main difference of these architectures is the necessity of simultaneous processing of multiple signature requests. In this paper, we will show the following items:

1. A custom two-level architecture optimization

methodology (presented in [14]). The group signature algorithm is too large to be handled by the current HLS (High Level Synthesis) tools[1], while they can be used for exploring H/W architecture (*micro-architecture*) of smaller functions such as modulo (RSA), ECC and hash. Therefore, we made a custom upper level behavioral synthesizer for exploring *macro-architecture*, i.e. how to arrange modulo/ECC/hash *cores*.

2. Macro-architecture for client devices (presented in [14]). Practical H/W speed of 0.135 seconds at 100MHz, which is the same speed with S/W on 3GHz PC, was achieved.

3. A new macro-architecture for servers in data centers. We found that the power efficiency is wrong if the above H/W accelerator for client devices are simply arranged in parallel. Therefore we adopted a different architecture such that multiple cores are connected and each core can be assigned to any signature request. After optimizing the number of cores, we were able to reduce the total H/W size almost 50%, compared to the simple parallel architecture.

This paper is organized as follows. In Section II, we explain a typical group signature algorithm. In Section III, we explain the proposed two-level architecture optimization methodology. In Section IV, we will show the results of architecture optimization.

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*.

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 $\mathcal{G}$, 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,

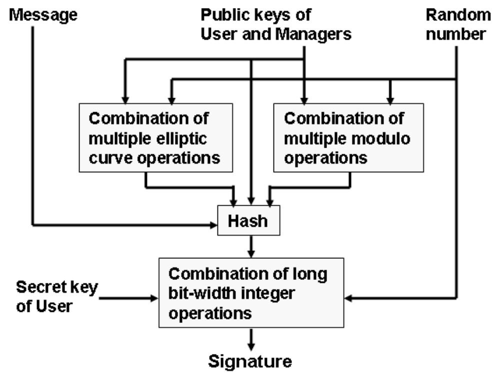

Message
Public keys of User and Managers
Random number
Combination of multiple elliptic curve operations
Combination of multiple modulo operations
Hash
Secret key of User
Combination of long bit-width integer operations
Signature

Fig. 3. Data flow of group signature generation

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 $\mathcal{G}$ 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 $y_1, y_2 \in \mathbb{Z}_q$, $G \in \mathcal{G}$ 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 $x_i \in \Lambda$, $B_i = b^{1/e'_i} \bmod \ell$, $e_i = 2^{\kappa_e} + e'_i$, 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 $\mathsf{Hash} : \{0,1\}^* \to \{0,1\}^{\kappa_c}$ 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. 3).

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

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

[1]Standard HLS tools automatically synthesize optimized RTLs from sequential algorithm descriptions in C language[10, 11, 12, 13].

Primitive func.	Bit length	CLK cycles	Times	Ratio
EC Scalar Mult.	160	970,055	7	31.4%
EC Point Add.	160	4,861	4	<0.1%
Modular Mult.	1024 x 1024	1,841	5	<0.1%
Modular Exp.	$(1024)^{280}$	521,439	2	4.8%
Modular Exp.	$(1024)^{512}$	943,291	2	8.7%
Modular Exp.	$(1024)^{792}$	1,500,187	1	6.9%
Modular Exp.	$(1024)^{1236}$	2,341,599	1	10.8%
Modular Exp.	$(1024)^{1464}$	2,804,294	1	13.0%
Modulo Mult Inv.	1024	1,713,022	2	15.8%
INT Mult.	160 x 60	84	1	<0.1%
INT Mult.	160 x 160	125	1	<0.1%
INT Mult.	160 x 576	301	1	<0.1%
INT Mult.	160 x 1016	489	1	<0.1%
INT Mult.	160 x 1244	584	1	<0.1%
INT Mult.	512 x 50	239	1	<0.1%
INT Mult.	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	(msg len)	(<10,000)	1	<0.1%
PRNG (rand)	160	29,720	2	0.2%
PRNG	280	35,608	1	0.16%
PRNG	512	35,688	2	0.33%
PRNG	792	47,475	1	0.22%
PRNG	1236	53,342	1	0.25%
PRNG	1464	69,070	1	0.32%
Data transfer	−−	*1,286,806*	−−	*5.9%*
Total	−−	21,633,992	−−	100%

TABLE I
Examples of the speed of primitive functions in group signature generation

$w^{\mu_t} B_{\mathrm{COM}}^{-\mu_{e'}} \bmod \ell$).

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

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 ($\boldsymbol{E}$, A_{COM}, B_{COM}, c, τ_x, τ_s, $\tau_{e'}$, τ_t, τ_E).

E. Signature Verification Algorithm

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

1. Check if both $|\tau_x| \leq \lambda + \kappa_c + \kappa_S$ and $|\tau_{e'}| \leq \kappa_{e'} + \kappa_c + \kappa_S$ 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 $c = \mathsf{Hash}(\kappa, \mathsf{ipk}, \mathsf{opk}, \mathsf{rpk}, \boldsymbol{E}, A_{\mathrm{COM}}, B_{\mathrm{COM}}, V'_{\mathtt{ComCipher}}, V'_{\mathtt{ComMPK}}, V'_{\mathtt{ComRev}}, m)$ holds. If holds, output accept and otherwise, output reject.

III. The Proposed Architecture Exploration Methodology

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 (> 30 steps in total) of primitive functions listed in Table I.

- 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.

- Only a small ratio (a few percents) is consumed by data transfer between primitive functions, as shown in Table I.

- Verification and debugging of entire signature computation on RT-level is quite time consuming. A simulation of entire group signature (consumes 10 millions to 300 millions of clock cycles) will take more than a day.

B. 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)[10, 13], 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. 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. Combination of simulation in behavioral/transaction level and FPGA emulation is appropriate.

C. Limitation of Current HLS

As shown in Table II, recent HLS tools[10, 13] 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[13]. 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). Current HLS tools only support parallel scheduling of small C-embedded operators ($+, -, \times, /, \%$ etc.) and micro-architecture synthesis.

Fast RSA 2048-bit core	HLS [10, 11]	Handmade RTL
Latency (@100MHz)	1ms	1ms
Max clock frequency	230MHz	300MHz
Gate count	70K	60K
Code size	500lines (SystemC)	1900lines (Verilog)

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

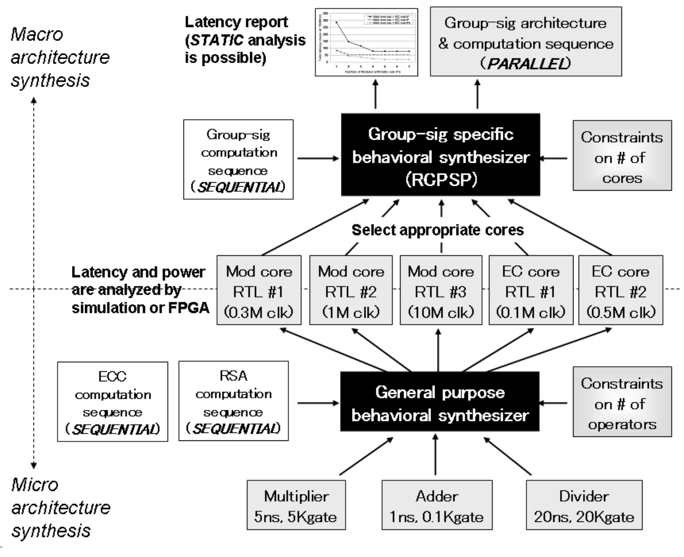

Macro architecture synthesis
Latency report (STATIC analysis is possible)
Group-sig architecture & computation sequence (PARALLEL)
Group-sig computation sequence (SEQUENTIAL)
Group-sig specific behavioral synthesizer (RCPSP)
Constraints on # of cores
Select appropriate cores
Latency and power are analyzed by simulation or FPGA
Mod core RTL #1 (0.3M clk)
Mod core RTL #2 (1M clk)
Mod core RTL #3 (10M clk)
EC core RTL #1 (0.1M clk)
EC core RTL #2 (0.5M clk)
ECC computation sequence (SEQUENTIAL)
RSA computation sequence (SEQUENTIAL)
General purpose behavioral synthesizer
Constraints on # of operators
Micro architecture synthesis
Multiplier 5ns, 5Kgate
Adder 1ns, 0.1Kgate
Divider 20ns, 20Kgate

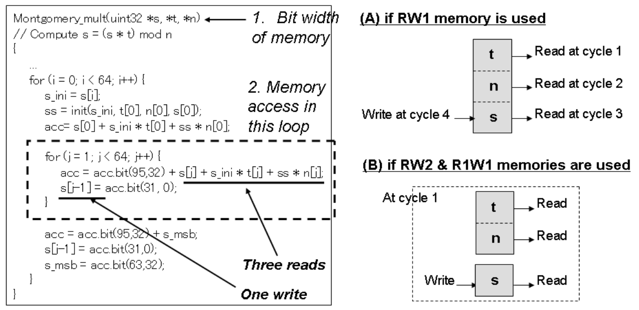
Fig. 4. The proposed two-level synthesis method customized for group signature circuit

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

If all of the functions in Table I 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.

Therefore, a custom higher level (core-function level) behavioral synthesizer was made and used (Fig. 4)[14]. Inputs of this custom synthesizer are (i) a sequential description of entire group signature algorithm (equations in Section II.), (ii) maximum number of cores, and (iii) core performance data gathered by conventional HLS (Table III). 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).

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.



```
Montgomery_mult(uint32 *s, *t, *n)
// Compute s = (s * t) mod n
{
    ...
    for (i = 0; i < 64; i++) {
        s_ini = s[i];
        ss = init(s_ini, t[0], n[0], s[0]);
        acc= s[0] + s_ini * t[0] + ss * n[0];

        for (j = 1; j < 64; j++) {
            acc = acc.bit(95,32) + s[j] + s_ini * t[j] + ss * n[j];
            s[j-1] = acc.bit(31 , 0);
        }

        acc = acc.bit(95,32) + s_msb;
        s[j-1] = acc.bit(31,0);
        s_msb = acc.bit(63,32);
    }
}
```

Fig. 5. Micro-architecture optimization items

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

TABLE III
RESULTS OF MICRO-ARCHITECTURE EXPLORATION (VIRTEX-6 XC6VLX760 FPGA)

E. *Micro-architecture Exploration using Conventional HLS and FPGA emulation*

In the group signature H/W, the speed of modulo and EC functions are critical as shown in Table I. Because these functions treat long-bit data (keys, plaintext and ciphertext) on local memories, adjusting bandwidth of the memories are important[8, 9]. For instance, as shown in Fig. 5, the speed of Montgomery multiplication is determined by the bit width of memory−ALU (multiplier, in this case) connection and the possible number of simultaneous memory access. 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[14]. An example is shown in Table III.

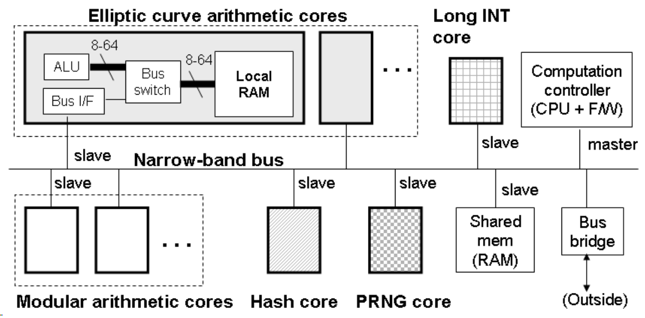


Fig. 6. The proposed macro-architecture for client devices

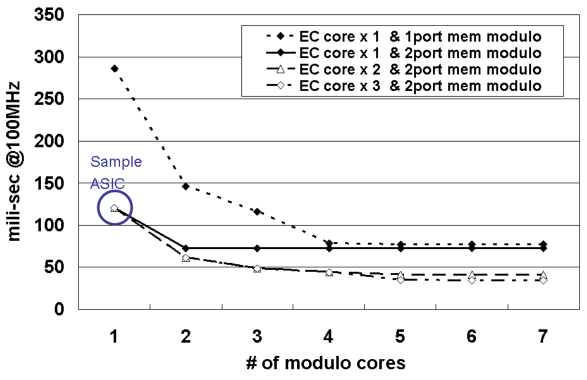


Fig. 7. Early performance estimation of the accelerator for client devices (1024-bit signature generation)

IV. Architecture Exploration Results

A. Macro-architecture of H/W Accelerators for Client Devices

For slow-clock client devices (mobile devices, embedded systems, sensor devices and etc.), we used a macro-architecture shown in Fig. 6[14]. Elliptic curve, modular, long-bit INT and hash cores are connected each other via local bus. 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.

The result of macro-architecture exploration is shown in Fig. 7. Clearly, increasing number of the modulo core significantly improves total speed, and maximum speed is reached when 4 or 5 cores are used [2]. Besides, increasing number of the other cores has much smaller effect. Regarding to the micro-architecture selection, we suppose that using the fastest cores for functions is better to achieve high performance and high power efficiency.

In order to prove that our design methodology works appropriately and the circuit can clear all back-end ASIC fabrication procedures, we implemented a sample accelerator H/W in Section A., on a low-cost 0.25um ASIC. The

[2]The optimum number of cores is the same between signature generation and verification, while performance graph of the signature verification is omitted here.

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)	810K gate (after DFT) 650K gate (before DFT)
On-chip SRAM amount	total 320K bits
Test coverage	>95% by ATPG
Average power consumption	425mW (100MHz, 1.5V)

TABLE IV
Sample ASIC implementation results (0.25um gate array)

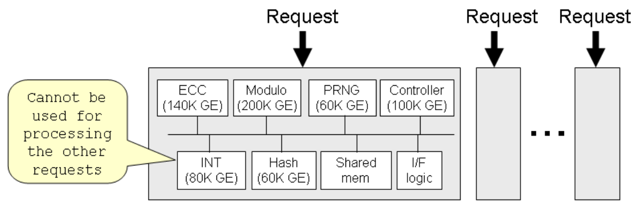


Fig. 8. An issue in simple parallel architecture for server accelerators

implementation result is summarized in Table IV. The actual latency is almost the same with the estimated value by RCPSP solver in Section D., and we were able to confirm that macro-architectures can be evaluated correctly in our two-level HLS methodology.

B. Macro-architecture of H/W Accelerators for Servers

We also investigated accelerator architecture for servers in data centers. In cloud computing environment, these servers have to process multiple group signature requests simultaneously. For this purpose, we would like to avoid to use multiple servers because of too high power consumption (> 1000W).

However, if multiple H/Ws in Section A. are arranged in parallel (Fig. 8), the power efficiency is still wrong. The reason is that the cores other than modulo and ECC are not running during most of the computation (see Table I also), although they occupies 40-50% of total circuit area

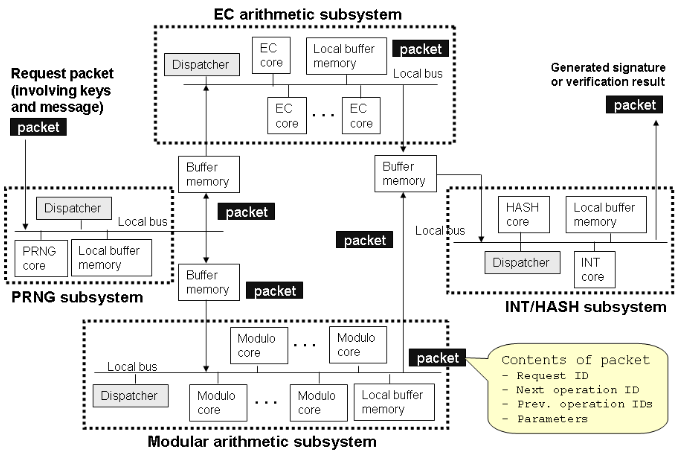


Fig. 9. The proposed H/W architecture for server accelerators

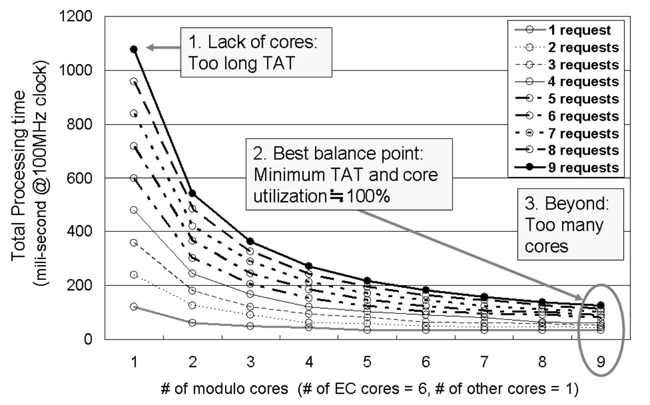


Fig. 10. Performance analysis of server accelerators

and consumes the same ratio of static power and clock tree power.

Therefore, as shown in Fig. 9, we adopted a different architecture such that arithmetic subsystems, which involve multiple cores, are connected in a pipelined manner. In this system, multiple signature requests can be received even when the other signature computation is in progress. The accepted request is propagated through the pipeline as a packet that records which operation should be done in the next. The dispatcher in each subsystem assigns a core and execute the operations.

We analyzed the total performance of this system by almost the same method in Section D., while one difference is that a sequential sequence description of multiple signature computation is given to the RCPSP solver. We examined approximately 1000 different system configurations, by changing number of cores[3]. The best result is shown in Fig. 10. In this figure, the number of modulo cores is changed and those of the other cores are fixed. The TAT is improved as the number of modulo cores increases, yet at some point (9 modulo cores), the improvement is stopped because core utilization start to reduce. At the optimum point where both minimum TAT and best core utilization is achieved, the ratio of number of modulo, ECC, INT, HASH and PRNG cores is 9:6:1:1:1, and the worst TAT is the same with the simple parallel architecture in Fig. 8 (modulo:ECC:INT:HASH:PRNG = 9:9:9:9:9)[4]. In this configuration, according to the core size shown in Fig. 8, we can reduce the total H/W size almost 50%.

V. Conclusion

In this paper, we investigated different group signature H/W architectures for client devices and servers in cloud computing environment. We confirmed that fast, compact and lower power accelerators can be designed by our custom two-level HLS design methodology.

[3]Regarding to micro-architecture, we selected the fastest modulo and ECC cores in Table III, because the power efficiency is also the highest.

[4]The number of simultaneous signature requests is 9. In order to process more number of requests, we simply arrange the circuit in Fig. 9 in parallel.

Acknowledgements

References

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

[2] M.Bellare, D.Micciancio and B.Warinschi, "Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions," EUROCRYPT 2003, LNCS Vol.2656, 2003.

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

"k-Times anonymous authentication," ASIACRYPT2004, LNCS Vol.3329, pp.308-322, 2004.

[4] T.Isshiki, K.Mori, K.Sako, I.Teranishi, and S.Yonezawa, "Using Group Signature for Identity Management and its Implementation," ACM CCS2006 Workshop on Digital Identity Management (DIM 2006), pp.73-78, 2006.

[5] W.Qiu, H.Guan, X.Jiang and Z.Huang, "Group Oriented Secure Routing Protocol of Mobile Agents," International Conference on Computational Intelligence and Security Workshops, 2007 (CISW 2007), pp.526 - 529, 2007.

[6] J.Guo, J.P.Baugh and S.Wang, "A Group Signature Based Secure and Privacy-Preserving Vehicular Communication Framework," 2007 Mobile Networking for Vehicular Environments, pp.103-108, 2007.

[7] 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.

[8] C.K.Koc, "High-speed RSA implementation," Technical Report TR201, RSA Laboratories, 1994.

[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] 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.

[11] 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.

[12] 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.

[13] 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.

[14] S.Morioka, T.Isshiki, S.Obana and K.Sako, "Flexible Architecture Optimization and ASIC Implementation of Group Signature Algorithm using a Custimized HLS Methodology," IEEE Int. Symp. on Hardware-Oriented Security and Trust (HOST 2011), pp.57-62, 2011.