# Design and Implementation of a Private and Public Key Crypto Processor and Its Application to a Security System

HoWon Kim, *Member, IEEE,* and Sunggu Lee, *Member, IEEE*

**Abstract** — *This paper presents the design and implementation of a crypto processor, a special-purpose microprocessor optimized for the execution of cryptography algorithms. This crypto processor can be used for various security applications such as storage devices, embedded systems, network routers, security gateways using IPSec and SSL protocol, etc. The crypto processor consists of a 32-bit RISC processor block and coprocessor blocks dedicated to the AES, KASUMI, SEED, triple-DES private key crypto algorithms and ECC and RSA public key crypto algorithm. The dedicated coprocessor block permits fast execution of encryption, decryption, and key scheduling operations. The 32-bit RISC processor block can be used to execute various crypto algorithms such as Hash and other application programs such as user authentication and IC card interface. The crypto processor has been designed and implemented using an FPGA, and some parts of crypto algorithms have been fabricated as a single VLSI chip using 0.5* $\mu m$ *CMOS technology. To test and demonstrate the capabilities of this chip, a custom board providing real-time data security for a data storage device has been developed.*

**Index Terms — Crypto Processor, Security, AES, KASUMI, SEED, Triple-DES, ECC, RSA.**

## I. Introduction

Nowadays, nearly all companies, government agencies and home users depend on computer systems and communication systems such as the Internet and Intranet. The advent of computers and networks has completely changed the way in which we live and work. The expansion of the worldwide communication network such as the Internet and the increased dependency on digitized information in our society makes information more vulnerable to abuse. If there are security problems in these information systems, users will fear that their sensitive information may be monitored and business secrets stolen. For these reasons, it is important to make information systems secure by protecting data and resources from malicious acts — crypto (cryptography) algorithms are the core of such security systems[1].

uses

By encoding a message using crypto algorithms, users can make information transmitted over communication systems almost impossible to read, even if such information is intercepted for malicious purposes. It is fairly easy to implement crypto algorithms in software, but such algorithms are typically too slow for real-time applications such as storage devices, embedded systems, network routers, etc. For this reason, it becomes necessary to implement crypto algorithms in hardware.

Because our crypto processor supports various private and public key crypto algorithms, our crypto processor can be applied to VPN(Virtual Private Network) and secure web servers which use the IPSec(IP Security) and SSL(Secure Sockets Layer) protocol. The IPSec and SSL protocol require RSA(the Rivest, Shamir, and Adleman algorithm), AES(Advanced Encryption Standard), and triple-DES(Data Encryption Standard) crypto algorithms for key exchange and data encryption[12]. We can also use our crypto processor for an IMT-2000 RNC(Radio Network Controller) switch, wireless applications and various security applications because our crypto processor supports KASUMI[7], ECC(Elliptic Curve Cryptography), and other crypto algorithms.

In our crypto processor design, we have implemented the crypto processor with an FPGA. After verifying the correct operation of the FPGA implemented crypto processor, we have implemented some of the crypto algorithms with an ASIC. The reason we did not implement all of the crypto algorithms in our ASIC chip is because of the area constraints of the selected fabrication process.

Our crypto processor has a RISC processor and coprocessor blocks dedicated to crypto algorithms. The dedicated crypto block of the crypto processor permits fast execution of encryption, decryption, and key scheduling operations for AES, KASUMI, SEED[16], and triple-DES[13] private key crypto algorithms and ECC and RSA public key crypto algorithm. Also, the 32-bit RISC processor block can execute other crypto algorithms including hash algorithms (SHA-1, MD5, etc.) and control the dedicated crypto block and I/O buffers.

Various crypto algorithms can be programmed and executed in our crypto processor without interrupting the host node's work. If crypto-related operations were executed in the host node's CPU without the use of a special crypto processor, it would consume the processing power of the host node's processor and hence slow down the entire system. We have designed and implemented this crypto processor with the design philosophy of making certain crypto algorithms as fast as possible while providing reasonably high performance for other crypto algorithms.

- HoWon Kim is with the Electronics and Telecommunication Research Institute(ETRI), DaeJeon, 305-350, Republic of Korea.(e-mail: khw@etri.re.kr or khw@ieee.org)
- Sunggu Lee is with the Department of Electronic and Electrical Engineering, Pohang University of Science and Technology (POSTECH), San 31, HyoJaDong, NamGu, Pohang, 790-784, Republic of Korea.(e-mail: slee@postech.ac.kr)

Contributed Paper
Manuscript received November 17, 2003

This paper is organized as follows. In Section II, the architecture of the crypto processor is briefly described; this includes the dedicated crypto block for AES, KASUMI, SEED, triple-DES, ECC, RSA and the 32-bit RISC processor. In Section III, the FPGA and VLSI design methodology of the crypto processor is described. In Section IV, the implementation results and performance evaluation of the crypto processor design are reported. Section V presents an example application of the crypto processor as a means for providing real time data security for a storage device. Finally, concluding remarks are presented in Section VI.

## II. The Crypto Processor Architecture

### *A. The architecture of the crypto processor*

The block diagram of our crypto processor is shown in Fig. 1. This single chip crypto processor has a PCI interface logic, dual port memory, memory controller, register file, datapath controller and dedicated crypto blocks for the AES, KASUMI, SEED, triple-DES, ECC and RSA crypto algorithms.

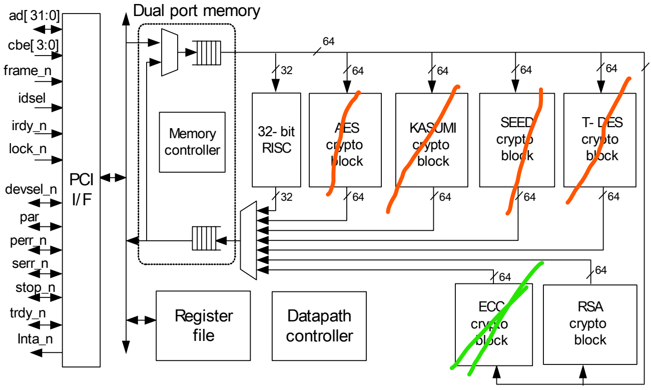


**Fig. 1. Block diagram of the private and public key crypto processor.**

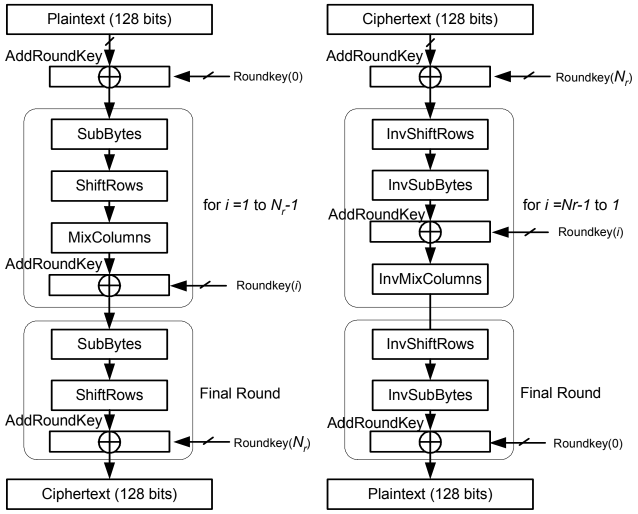
The 32-bit RISC type crypto controller controls the dedicated crypto block and performs the interface operations with external devices such as memory and an I/O bus interface controller. It can also execute various crypto algorithms such as MD5 and SHA-1 (hash algorithms) and other application programs such as a user authentication program and an IC card interface program.

The dedicated crypto block results in fast execution of the encryption, decryption and key scheduling operations for the AES, KASUMI, SEED, and triple-DES algorithms, and enables fast scalar multiplication and exponentiation operations for the ECC and RSA crypto algorithms. The PCI interface logic permits our crypto processor to be easily applied to practical environments (any application which has a PCI I/O bus).

### *B. The dedicated crypto block for the private key crypto algorithm*

#### *1) AES crypto block*

In September of 1997 the National Institute of Standards and Technology (NIST) issued a request for possible candidates for a new Advanced Encryption Standard(AES) to replace the DES. Then in October of 2000, NIST announced the cipher Rijndael, which is developed by Joan Daemen and Vincent Rijmen, as an AES algorithm. Rijndael is a block cipher using 128, 192, 256-bit input/output and keys. The sizes of data blocks and keys can be chosen independently. The number of rounds depends on both of these parameters and is given in [2].



(a) Encryption (b) Decryption

**Fig. 2. Structure of AES Cipher (a) encryption (b) decryption.**

In this paper, 128bits for both I/O block and user keys are assumed. Therefore, the cipher in all configurations presented operates in $N_r = 10$ rounds.

Fig. 2 shows the encryption and decryption structure of the AES algorithm. After the initial roundkey addition, $N_r$ rounds of encryption are performed. The first $N_r - 1$ rounds are the same, with a small difference in the final round. As shown in Fig.2(a), each of the first $N_r - 1$ rounds consists of 4 transformations: SubBytes, ShiftRows, MixColumns and AddRoundKey. The final round excludes the MixColumns transformation. The 4 transformations listed are as follows :

- SubBytes: This transformation is performed on each byte of the State using a substitution table(S-box). The S-box is constructed of the compositions of two transformations: multiplicative inverse in $GF(2^8)$ with irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$, and an affine mapping over $GF(2)$. In the decryption process, the inverse S-box is used. The inverse S-box is constructed by first applying the inverse of the affine transformation and then computing the multiplicative inverse in $GF(2^8)$.
- ShiftRows: In this transformation, the rows of the State shift cyclically to the left with different offsets[2]. In the decryption process, the shifting offsets have different values.
- MixColumns: The MixColumns transformation is performed on the State column-by-column. Each column is considered as a four-term polynomial over $GF(2^8)$ and multiplied by $a(x)$ modulo $x^4 + 1$ ,

where $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + 1$ for encryption and $a(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$ for decryption.

- AddRoundKey: In this transformation, a round key is added to the State using a bitwise Exclusive-OR(XOR) operation. AddRoundKey is the same for the decryption process.

The decryption algorithm uses a different ordering of the inverse forms of the transformations used in the encryption algorithm as shown in Fig. 2(b) [2].

Fig.3 shows the block diagram of the AES crypto block. It consists of an AES encryption/decryption core, a key generator, a register file and control logic. The data path for encryption is as follows: MUXA → AddRoundKey → RegA_128 → S_Box(SubBytes) → SR for Enc(ShiftRows) → MixColumn(skipped in the final round) → MUXEnc → MUXB →MUXA. The data path for decryption is as follows: MUXA → AddRoundKey → InvMixCol(skipped in the first round) → MUXDec → SR for Dec(InvShiftRows) → SI Box(InvSubBytes) → MUXB → MUXA. The AES crypto block is an one-round based architecture. Although other architectures which use pipelining, sub-pipelining, and loop unrolling for high performance are possible[17], we have selected the most simple architecture because the AES algorithm can achieve high performance with this simple architecture. In other crypto blocks, we have used sub-pipelining and pipelining techniques to implement the KASUMI, SEED, and triple-DES crypto blocks.

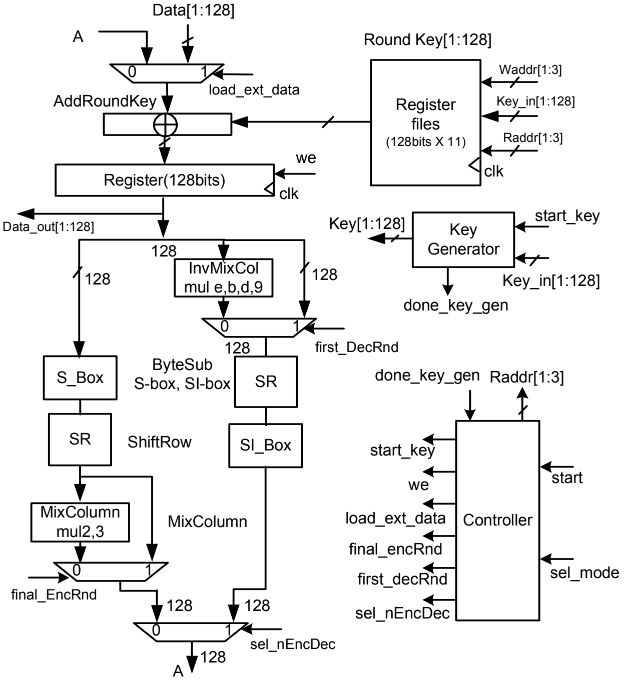


Fig. 3. Block diagram of the AES crypto block.

We have implemented the SubBytes block (S-box) with a ROM instead of calculating multiplicative inverse and affine transform for simple and high performance. We have used a 1Kbyte (256×8 bits×16/8×2) ROM for the S-box and SI-box used in the implementation of the AES crypto block. It requires eight repetitions of the S-box and SI-box for encryption and decryption, respectively. Key values are pre-computed and saved in REG_FILE so that they can be used for encryption and decryption without recalculation of key expansion. The multiplication logic for MixColums and InvMixColums is based on the following calculations: $\{02\}X$ can be implemented by shifting and exclusive OR; i.e., $\{x_7, x_6, x_5, x_4, x_3 \oplus x_7, x_2 \oplus x_7, x_1, x_0 \oplus x_7, 0\}$. Also, $\{03\}X$ is $\{02\}X \oplus X$. $\{09\}X$ can be implemented by $\{x_4 \oplus x_7, x_3 \oplus x_6 \oplus x_7, x_2 \oplus x_5 \oplus x_6 \oplus x_7, x_1 \oplus x_4 \oplus x_5 \oplus x_6,$ $x_0 \oplus x_3 \oplus x_5 \oplus x_7, x_2 \oplus x_6 \oplus x_7, x_1 \oplus x_5 \oplus x_6, x_0 \oplus x_5 \oplus x_7\}$. The expression for $\{0b\}X$, $\{0d\}X$, and $\{0e\}X$ can be found in [17].

### *2) KASUMI crypto block*

KASUMI is a block cipher algorithm which has a Feistel structure and is based on the MISTY1[18] algorithm. KASUMI is a core cipher of the confidentiality algorithm (*f8*) and the integrity algorithm (*f9*) which have been standardized by the ETSI/SAGE regarding the 3rd Generation Partnership Program (3GPP) Security of Data. A block diagram of the KASUMI is depicted in Fig. 4. It operates on a 64-bit data block and uses a 128-bit key with eight operation rounds. The 64-bit input $I$ is divided into two 32-bit strings $L_0$ and $R_0$, where $I = L_0 \| R_0$. Then for each integer $i$ with $1 \le i \le 8$, we define: $R_i = L_{i-1}, L_i = R_{i-1} \oplus f_i(L_{i-1}, RK_i)$.

This constitutes the $i^{th}$ round function of KASUMI, where $f_i$ denotes the round function with $L_{i-1}$ and round key $RK_i$ as inputs. The round key comprises the subkey triplet of $(KL_i, KO_i, KI_i)$. For odd rounds, the $f_i$ function is as defined below: $f_i(I, RK_i) = FO(FL(I, KL_i), KO_i, KI_i)$. For even rounds, the $f_i$ function is defined as follows:

$$f_i(I, K_i) = FL(FO(I, KO_i, KI_i), KL_i)$$

The result of the KASUMI algorithm is equal to the 64-bit string $(L_8 \| R_8)$ offered at the end of the eighth round[7].

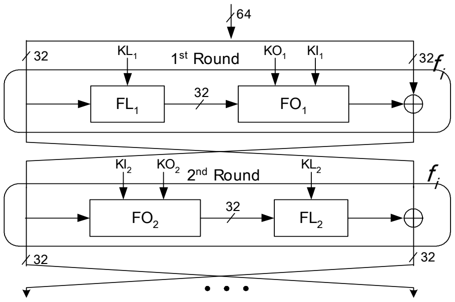


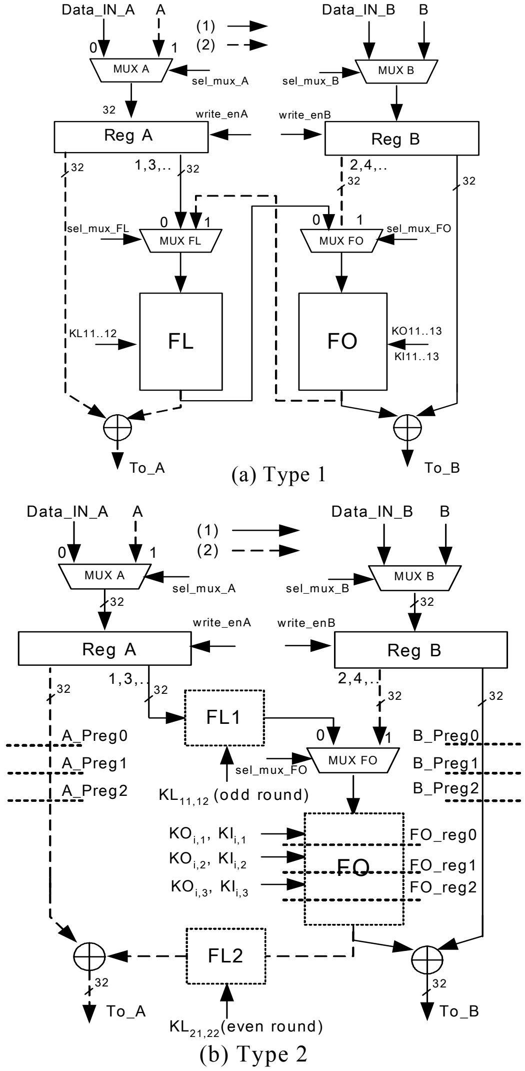
Fig. 4. Block diagram of the KASUMI crypto algorithm.

For implementation of the KASUMI algorithm as a hardware block, we have considered two architectures, the low power version (Type 1) and high performance version(Type 2), which are applicable to ME(Mobile Equipment) and RNC(Radio Network Controller) switch in a 3GPP system, respectively.

Fig.5(a) shows the block diagram of the KASUMI algorithm for low power consumption. It has a simple but efficient architecture for executing KASUMI. One round of KASUMI is repeated 8 times to complete the KASUMI operation. This architecture has simple hardware complexity and low power consumption at the cost of a sacrifice in performance.

For odd rounds, the data in Reg $A$ is processed by the $FL$ function and then the $FO$ function with proper key values. The results are fed back to Reg $B$ after an exclusive-OR (XOR) with the data in Reg $B$. The data path for this operation is depicted with solid lines in Fig.5(a). For even rounds, the data in Reg $B$ is processed by the $FO$ function and then the $FL$ function. The exclusive-OR of this result and the contents of Reg A are saved in Reg $A$.

To make the KASUMI algorithm applicable to an RNC switch, which requires high performance cryptographic operations, we have designed another architecture (Type 2) as shown in Fig.5(b). It has a four-stage pipeline for one round of the hardware logic (the sub-pipelining technique in [17]). This architecture achieves high performance with a low hardware complexity. The four-stage pipeline means that we can execute encryption (or decryption) operations for up to four messages simultaneously. This architecture is suitable for an RNC switch, which needs to be able to handle a lot of message traffic.



(a) Type 1

(b) Type 2

**Fig. 5. Block diagram of the low power KASUMI crypto block (Type 1) and high performance KASUMI crypto block (Type 2).**

Since the major delay time is due to the $FO$ function block (which is composed of three $FI$ blocks, with each $FI$ block composed of two $S9$ boxes and two $S7$ boxes), we have divided the $FO$ block into four parts by inserting three pipeline registers ( $FO\_reg0$ , $FO\_reg1$ $FO\_reg2$ ) in the $FO$ block as shown in Fig. 5(b). Also, one pipeline register is inserted on each $FI$ block as shown in Fig.6.

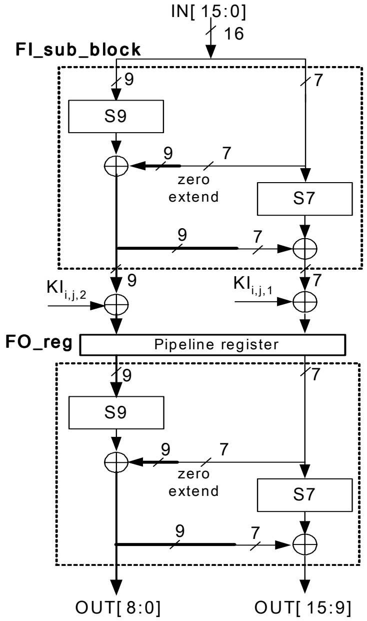


**Fig. 6. Block diagram of the sub-pipelined FI block.**

The number of pipeline registers and the positions of those registers are determined by calculating the critical timing delay of the KASUMI hardware and by considering tradeoffs between performance and hardware complexity.

The four-stage pipeline structure of the KASUMI algorithm results in a high operating frequency and a high throughput. Ideally, it executes 4 times faster than the Type 1 architecture. In order to avoid $FL$ block conflicts with other data, we have inserted another $FL$ block, $FL2$, as shown in Fig.5 (b).

For key scheduling in the KASUMI crypto block, we have developed an efficient on-the-fly key scheduling hardware block for the four-stage pipeline architecture. The scheduling block has registers to store constant values and key values, a hardwired right rotation block, XOR gates and registers for pipeline synchronization. Also, the key scheduling block is extensible to any number of pipeline stage with the addition of a few key storage and synchronization registers [5].

In general, there are two methods to implement the S-box in hardware; the LUT (Look Up Table) ROM design method and the combinational logic design method [11]. In the S-box implementation of the KASUMI crypto block, we have selected the combinational logic method instead of the ROM-based method. By implementing the KASUMI S-box using combinational logic, we can get a short delay time.

Up to now, we have described the details about the private key crypto blocks such as AES and KASUMI. Since the hardware design techniques presented in the previous sections are also applicable to other private key crypto blocks (such as SEED and Triple-DES, etc.), we will now briefly describe the architecture of the SEED and Triple-DES crypto block in following sections.

### *3) SEED crypto block*

The SEED algorithm[16] is a block cipher that operates on 128-bit blocks of data and uses a 128-bit key. It has a 16-round Feistel structure.

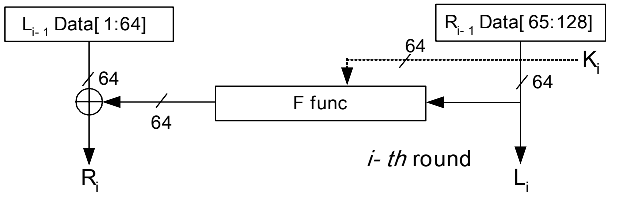


**Fig. 7. Block diagram of the i-*th* round of SEED crypto algorithm.**

Fig.7 shows one stage of the SEED algorithm. The SEED algorithm uses two 8 × 8 S-boxes (for substitution), permutations, rotations, and basic modulo-arithmetic operations such as modulo-2 addition (exclusive OR) and modulo-$2^{32}$ addition. As with other Feistel ciphers, the SEED algorithm has an *F* function, which takes a 64-bit data value and 64-bit key values. A 64-bit input block of the round function is divided into two 32-bit blocks and wrapped with 4 phases: a mixing phase of two 32-bit subkey blocks and 3 layers of *G* functions with additions for mixing two 32-bit blocks. The *G* function is composed of two layers of 8 X 8 S-boxes and permutation logic to provide good characteristics against DC (Differential Cryptanalysis) and LC (Linear Cryptanalysis) attacks. More information on the SEED algorithm can be found in [16].

From a hardware implementation viewpoint, the SEED crypto algorithm is not an efficient algorithm. The *G* function and modulo- $2^{32}$ addition logic in the *F* function and key scheduling logic make the SEED slow. Furthermore, from a security viewpoint, the last layer of the *G* function in the *F* function is a redundant component because it does not provide any security strength (for providing security strength, the last layer of the *G* function should have an XOR operation with key values like the previous two *G* functions).

To implement the SEED crypto block, we have instantiated one stage (round) and divided it into 5 stages of pipeline logic. That is, we have implemented the SEED crypto block with the sub-pipelining technique [17]. We have inserted two 64bit-sized pipeline registers into the *F* function and two 64bit-sized pipeline registers into the key generation logic as shown in Fig. 8. After scrutinizing the critical path of the SEED crypto block, we have decided on the pipeline insertion location. The complexity of the *G* function (which consists of four 8X8 S-boxes and some combinational logic) and the long delay time of modulo- $2^{32}$ addition results in high latency of the SEED crypto block.

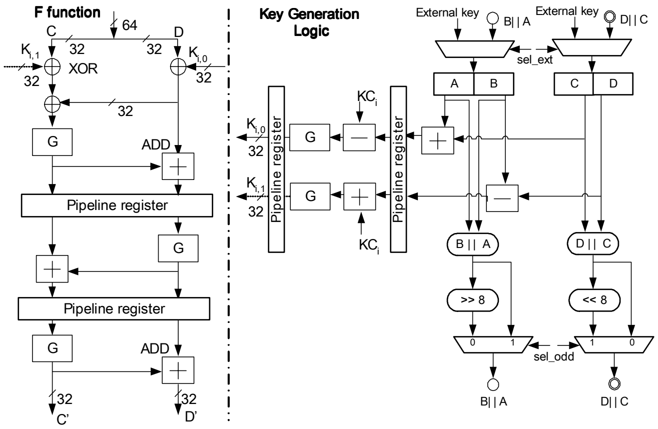


**Fig. 8. Block diagram of the pipelined *F* function block(left) and sub-pipelined key generation block(right).**

In the SEED crypto block, the key values are generated simultaneously with the encryption or decryption process. The reasons for using on-the-fly key scheduling instead of the pre-computation method are as follows. First, the SEED crypto algorithm has a good key scheduling structure for on-the-fly key scheduling. Second, if we adapt the pre-computation method to the SEED crypto block, it requires a large amount of key storage capacity because of its 5-stage pipeline.

### *4) Triple-DES crypto block*

DES(Data Encryption Standard)[13] is a block cipher which uses a 64-bit key and operates on 64-bit blocks of data. Because every 8th bit of the 64-bit key is used for parity checking, DES has a 56-bit key. In the DES algorithm, there are 16 rounds of identical operations such as non-linear substitutions and permutations. In each round, 48-bit subkeys are generated, and substitutions using S-box, bitwise shift, and XOR operations are performed.

The 56-bit key length is relatively small by today's standards. For increased security, the DES operation can be performed three consecutive times, which expands the effective key length to 112 bits. Using DES in this manner is referred to as triple-DES[15]. In this section, we only describe the DES crypto block because the expansion to triple-DES is trivial.

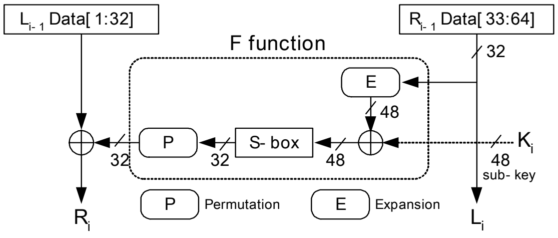


**Fig. 9. Block diagram of the DES crypto algorithm.**

Fig. 9 shows one round of the DES algorithm. The left and right halves of each 64-bit input data operand are treated as separate 32-bit data operands, $L_{i-1}$ and $R_{i-1}$. The 32-bit right halves of the data are passed to the next left halves of the data

($L_{i-1} = R_{i-1}$), and the 32-bit left halves of the data are processed in the following manner: $R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$.

As shown in Fig. 9, the $F$ function of the DES algorithm is composed of an expansion permutation table (block $E$), modulo-2 addition with the $i-th$ sub-key($K_i$) or round key, substitution with the S-box, and permutation with the $P$ table (block $P$).

In our DES crypto block implementation, we have implemented the DES crypto block with a 4-stage pipeline as shown in Fig. 10. This architecture has the advantage that 4 data-key pairs can be processed simultaneously. Thus, it has high performance at the expense of an increase in the hardware overhead. The reasons for selecting the pipelining technique instead of sub-pipelining or one-round repetition are as follows. First, because the DES crypto block should be used as a triple-DES (which requires 3 rounds of DES) for security reasons, the use of the sub-pipelining technique for the DES architecture would result in many clock cycles to complete the triple-DES operation. Second, the triple-DES crypto block can be implemented with low hardware complexity because of the hardware simplicity of the DES crypto block.

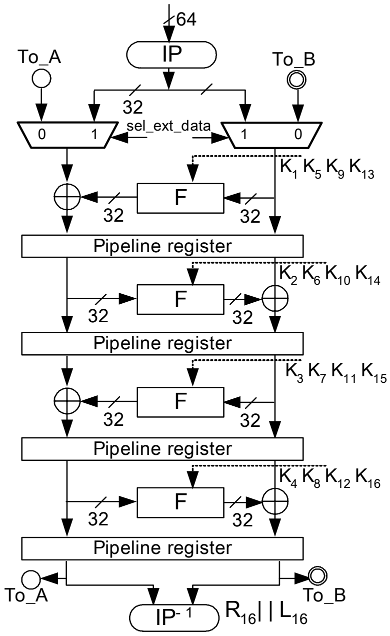


**Fig. 10. Block diagram of the pipelined DES crypto block.**

### *C. The dedicated crypto block for the public key crypto algorithm*

#### *1) ECC crypto block*

The ECC crypto system was proposed independently in 1985 by Vitor Miller and Neal Koblitz. The ECC crypto system is based on the difficulty of solving the discrete logarithm problem. In general, ECC has advantages over RSA in that ECC has higher security per key bit, higher speed, lower power consumption, and better storage efficiencies than RSA. For these reasons, ECC is particularly beneficial in applications with bandwidth, processor capacity, power availability, or storage constraints such as IC cards, mobile devices, etc.[4],[10].

The elliptic curve used in our crypto processor is defined by Weierstrass equations as $y^2 + xy = x^3 + ax^2 + b$, where $a, b \in GF(2^m)$ and $b \neq 0$. Since the ECC over a normal basis is efficient in hardware implementations, we have chosen a curve such as $GF(2^{146})$ over a normal basis. A normal basis of $GF(2^m)$ over $GF(2)$ has a basis set of $NB = \{\beta, \beta^2, \beta^{2^2}, ..., \beta^{2^{(m-1)}}\}$ for some $\beta \in GF(2^m)$.

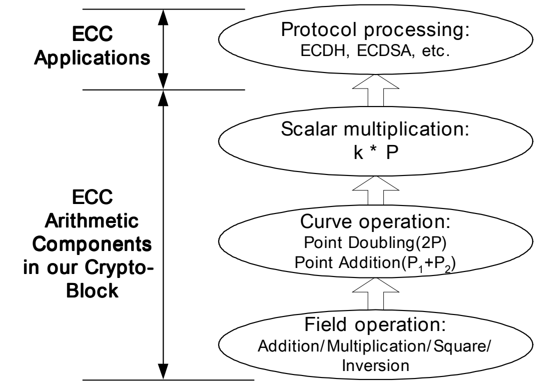


**Fig. 11. The Hierarchy of ECC arithmetic operations.**

The hierarchy of ECC arithmetic operations is shown in Fig. 11. ECC application protocols such as ECDH and ECDSA are performed using scalar multiplication at the highest level of the ECC crypto system. The scalar multiplication is done by repeated curve operations such as point doubling and point addition with proper algorithm such as binary method [12].

| Addition equation: | Doubling equation: |
|---|---|
| $P_3(x_3, y_3) = P_1(x_1, y_1) + P_2(x_2, y_2)$ | $P_3(x_3, y_3) = 2P_2(x_2, y_2)$ |
| $\lambda = (y_1 + y_2)/(x_1 + x_2);$ | $\lambda = x_2 + y_2 / x_2;$ |
| $x_3 = \lambda^2 + \lambda + a + x_1 + x_2;$ | $x_3 = \lambda^2 + \lambda + a;$ |
| $y_3 = (x_1 + x_3)\lambda + x_3 + y_1;$ | $y_3 = {x_2}^2 + (\lambda + 1)x_3;$ |

**Fig. 12. The equations for ECC addition and doubling.**

The most basic operation in ECC is the field operation.

The addition and subtraction operation at the field level are the simple XOR (modulo-2 addition) operation, and the squaring operation is also trivial (simple cyclic shift operation) as shown below. If an element $A$ is represented as $A = \sum_{i=0}^{m-1} a_i \beta^{2^i}$, then $A^2$ is represented as follows:

$A^2 = a_0\beta^2 + a_1\beta^{2^2} + a_2\beta^{2^3} + ... + a_{m-2}\beta^{2^{(m-1)}} + a_{m-1}\beta^{2^m}$, where $(\beta^{2^i})^2 = \beta^{2^{i+1}}$ and $\beta^{2^m} = \beta$ by Fermat's theorem. If $A$ is represented as a vector such as $A = [a_0, a_1, ..., a_{m-1}]$, then $A^2 = [a_{m-1}, a_0, a_1, ..., a_{m-2}]$. However, since the multiplication in a normal basis is rather complex, we have used the Massey and Omura multiplication algorithm for our design [8].

Let $A = \sum_{i=0}^{m-1} a_i \beta^{2^i}$ and $B = \sum_{j=0}^{m-1} b_j \beta^{2^j}$, and let

$C = A\times B = \sum_{k=0}^{m-1} c_k \beta^{2^k}$ . The multiplication of $A$ and $B$ is :

$$C = A\times B = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_i b_j \beta^{2^i}\beta^{2^j}.$$

If we let $\beta^{2^i}\beta^{2^j} = \sum_{k=0}^{m-1}\lambda_{ij}^{(k)}\beta^{2^k}$, $\lambda_{ij}^{(k)} \in \{0,1\}$, then comparing the coefficient of $\beta^{2^k}$ in $C$ yields the formula

$$c_k = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_i b_j \lambda_{ij}^{(k)},\quad 0 \le k \le m-1 \cdot$$

```
1: $s = xy$                  5: $w = v/R$
2: $u = s(m')\bmod R$        6: if $(w \ge m)$ then $w = w - m$
3: $t = um$                  7: Ret($w$)
4: $v = s + t$
```

**Fig. 13. The single-precision Montgomery multiplication algorithm.**

This equation can be written in a form that requires only $\lambda_{ij}^{(0)}$ [6]. Thus, $c_k$ can be written as follows:

$$c_k = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_{i+k} b_{j+k} \lambda_{ij}^{(0)}$$

This equation means that if we build a logic circuit with inputs $A$ and $B$ to compute the product digit $c_0$, we can easily compute the $c_k$ value using simple cyclic shifts of the vector representations of $A$ and $B$ [9].

Also, for the inversion operation, we have used a recursive inversion algorithm based on Fermat's theorem, which is shown in [14]. In this section, we only describe the core of the inversion algorithm as follows.

In Fermat's theorem, $a^{-1} = a^{2^m - 2}$ , where $a \in GF(2^m)$ . The exponent can be factored as $2^m - 2 = 2(2^{m-1} - 1)$ . This means $a^{2(2^{m-1}-1)} = [a^{2^{m-1}-1}]^2$. Let $m - 1 = r_0$ , and suppose $r_0$ is even. Then, $2^{r_0} - 1 = [2^{r_0/2} - 1][2^{r_0/2} + 1]$ . This equation can be reduced recursively and computing $a^{2^{r_0/2}+1}$ is easy. It requires a squaring operation and a multiplication operation. Also, if $r_0$ is odd, $2^{r_0} - 1$ can be recursively described as follows:

$$2^{r_0} - 1 = 2[2^{(r_0-1)/2} - 1][2^{(r_0-1)/2} + 1] + 1 .$$

That is, if $r_0$ is odd, we require an additional squaring and multiplication operation. More detailed algorithms can be found in [14]. The calculation of point addition and doubling requires the inversion, multiplication, squaring, and addition operations over the field $GF(2^m)$ as shown in Fig.12.

Since our ECC crypto block is implemented at the level of scalar multiplication, we can use our ECC crypto block directly for ECC applications such as ECDH and ECDSA. To perform scalar multiplication, we have used the binary method [4].

### *2) RSA crypto block*

The RSA public key crypto system was invented in 1978 by Rivest *et al.*[19], and it is now the most widely deployed public key crypto system. It is used for securing web traffic and e-mail in the SSL protocol. The security of the RSA cryptosystem depends on the difficulty of factoring a large integer, the published modulus value.

In this section, we describe of the core algorithm and its implementation in our crypto processor architecture (the details of the RSA algorithm can be found in [12],[15]). The core arithmetic operation in RSA is exponentiation, which is accomplished by a series of modular multiplications. Therefore, fast modular multiplication is key to achieving fast execution of RSA cryptosystems.

Contrary to a classical modular multiplication algorithm, the Montgomery multiplication algorithm does not need the division operation [1]. This method is based on an ingenious representation of the residue class modulo $m$ , and it replaces division by $m$ operations with division by a power of 2 (if we use $R = 2^n$ ). The single-precision Montgomery multiplication algorithm is shown in Fig. 13.

Inputs are integers $x, y (0 \le x, y < m), R = 2^n$, $n = \lceil \log_2 m \rceil$, with $\gcd(R,m) = 1$, $R > m$, and $m' = -m^{-1} \bmod 2$. The output is $xyR^{-1} \bmod m$.

In Fig. 13, if we assume $w$ is a multiple of $R = 2^n$, then Step 5 can be solved efficiently with a simple shift operation. The assumption can be verified as follows: Since we have assumed Step 5 is a $k-multiple$ of $R$, $v = kR \rightarrow$ $v \bmod R = (s + um) \bmod R = 0$. This equation is solved as $u \equiv -sm^{-1}(\bmod R) = sm'(\bmod R)$, which is equivalent to Step 2. Thus, the Montgomery multiplication algorithm can be solved without a complex division operation if we select $R = 2^n$.

```
1: $A = 0.(A = (a_n a_{n-1} \ldots a_1 a_0)_2.)$
2: For $i$ from 0 to $n-1$ do
    2.1: $u_i = (a_0 + x_i y_0)m' \bmod 2.$
    2.2: $A = (A + x_i y + u_i m)/2.$
3: If $A \ge m$ then $A = A - m.$
4: Ret($A$).
```

**Fig. 14. The multiple-precision Montgomery multiplication algorithm.**

Fig. 14 shows the multiple-precision Montgomery multiplication algorithm. Since we have implemented 1024-bit RSA in our crypto processor, the value of $n$ is 1024 and $x, y,$ and $m$ are represented by multiple-precision.

This algorithm is adequate for hardware implementation because it is composed of simple operations such as an $n$ bit by one bit multiplication operation ( $x_i y$ and $u_i m$ ), shift operation (division by 2), and addition. Fig. 15 shows the hardware block diagram for the multiple-precision Montgomery multiplication algorithm. It is composed of input registers (for $x, y, m, m'$), a $u_i$ calculation block, a Montgomery multiplication core block, and control logic. The $x_i y$ and $u_i m$ multiplication operations are equivalently implemented with multiplexers (MUX1 and MUX2) and division by 2 is implemented with a shifter. We have used the CSA (Carry Save Adder) for addition operations.

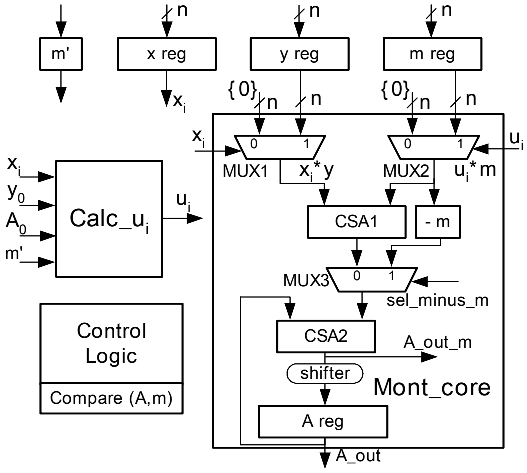


**Fig. 15. Block diagram of the RSA crypto block.**

Now let us consider the computation of modular exponentiation, the RSA's core operation, by using the Montgomery multiplication algorithm. Assuming we wish to compute $x^a \bmod m$, the input $x$ is first transformed into the Montgomery representation (m-residue of $x$) as $\bar{x} = xR \bmod m$. This is done using a Montgomery multiplication with $x$ and $(R^2 \bmod m)$ as inputs; i.e.,

$$\bar{x} = MontMul(x, R^2) = (x \times R^2 \bmod m)R^{-1} \bmod m = xR \bmod m\,.$$

The m-residue representation is stable over Montgomery multiplication:

$$\bar{x}^2 = (xR \bmod m \times xR \bmod m)R^{-1} \bmod m = x^2 R \bmod m.$$

$$\bar{x}^3 = (x^2 R \bmod m \times xR \bmod m)R^{-1} \bmod m = x^3 R \bmod m.$$

□ □ □

$$\bar{z} \equiv \bar{x}^a = x^a R \bmod m.$$

The output of the exponentiation $\bar{z}$ is converted back into the expected value( $z \equiv x^a \bmod m$ ) by using the Montgomery multiplication algorithm as follows:

$$z = MontMul(\bar{z}, 1) = (x^a R \times 1 \bmod m)R^{-1} \bmod m = x^a \bmod m.$$

Since multiple Montgomery multiplications are computed over the m-residues before the result is converted back to the original representation, the efficiency of the exponentiation relies on the ability to perform the Montgomery modular multiplication algorithm.

### *D. The 32-bit RISC processor block*

In our crypto processor, we have used the ARM7TM processor, the 32-bit RISC type processor block with a three-stage pipeline [3]. It controls the operation of the dedicated crypto block during encryption, decryption, and key scheduling, and also performs the operations required to interface with external devices such as the input buffer, output buffer, memory, and IC card interface logic. Since the RISC processor block is fully programmable, it can execute various crypto algorithms, protocols and application programs with a high degree of freedom. The programmability of the crypto processor makes the crypto processor applicable to embedded systems which require standalone programmability.

The 32-bit RISC processor block has features (such as a barrel shifter, a Booth multiplier block, register file, and a 16-bit and 32-bit data memory architecture) that enable it to achieve high performance and savings in memory when executing crypto algorithms.

The 32-bit barrel shifter implements a shift/rotate of its input data by any amount to produce an output within a fixed time period. It has associated logic to allow values to be arithmetic shifted or rotated through the carry bit. The barrel shifter boosts the performance of crypto algorithms, such as most symmetric key crypto algorithms, which require multiple-bit shift operations. The Booth multiplier block assists in the implementation of the multiply and multiply-and-add instructions. This instruction is useful for implementing hash algorithms.

TABLE I
ARCHITECTURAL CHARACTERISTICS OF THE PRIVATE KEY CRYPTO BLOCKS

| Feature | AES | KASUMI | SEED | Triple-DES |
|---|---|---|---|---|
| Architecture | One-round | Sub-pipelining | Sub-pipelining | pipelining |
| S-box | ROM based | Combinational logic based | ROM based | ROM based |
| Pipeline stage | 0 | 4 | 5 | 4 |

TABLE II
ARCHITECTURAL CHARACTERISTICS OF THE PUBLIC KEY CRYPTO BLOCKS

| Feature | ECC | RSA |
|---|---|---|
| Architecture | Scalar multiplication<br>( $kP$ ) | Exponentiation<br>( $x^a \bmod m$ ) |
| Security bit | 146 bits | 1024 bits<br>(2048 bits on CRT) |
| Basis or core algorithm | Normal Basis | Montgomery multiplication |

## III. FPGA AND VLSI DESIGN METHODOLOGY

Our crypto processor was modeled using VHDL (VHSIC Hardware Description Language) language and then implemented as an ASIC chip after verification with an FPGA implementation. Modeling the processor using VHDL facilitates quick prototyping and modification of the target design while considering various possible trade-offs in different implementations of the crypto algorithms with differing speed and area characteristics.

After verifying the functionality and performance of the crypto processor, we implemented the crypto processor with an ASIC. The target process technology is 0.5 $\mu m$ CMOS technology.

## IV. PERFORMANCE EVALUATION OF THE CRYPTO PROCESSOR

### *A. FPGA Implementation*

In previous sections, we have described the architecture and design methodology for our crypto processor. In this section,

TABLE III
CHARACTERISTICS OF THE AES, KASUMI,SEED, AND TRIPLE-DES CRYPTO BLOCKS

| Feature | AES | KASUMI | SEED | Triple-DES |
|---|---|---|---|---|
| Frequency (MHz) | 58 | 71 | 56 | 50 |
| Logic Size (Slices) | 1,689 | 1,174 | 1,809 | 732 |
| Performance (Mbps) | 390 | 568 | 358 | 267 |

TABLE IV
CHARACTERISTICS OF THE ECC AND RSA CRYPTO BLOCKS

| Feature | ECC(146bits) | RSA(1024bits) |
|---|---|---|
| Frequency (MHz) | 50 | 28 |
| Logic size (Slices) | 3,036 | 4,595 |
| Execution time | 7.28*msec* (scalar multiplication) | 6.69*msec* (encryption with 16-bit sized key)<br>58.9*msec* (decryption with 1024-bit sized key) |

we present and analyze the implementation results of our crypto processor. Table I and II show the architectural characteristics of the crypto blocks in our crypto processor.

The AES crypto block was implemented with a one-round based architecture and its S-boxes were implemented with FPGA's internal memory. Also, we have selected the sub-pipelining technique for the KASUMI and SEED crypto blocks, and the pipelining technique for the Triple-DES crypto block. The S-box of the KASUMI crypto block was implemented with combinational logic due to the constraints of the target FPGA's memory size. The reason we have used only the pipelining technique for Triple-DES is due to the fact that Triple-DES requires too many rounds(16 × 3). If we implemented Triple-DES with a sub-pipelining technique for high performance, we would have had to wait much longer to get the result.

Table II shows the architectural characteristics of the 146-bit ECC and 1024-bit RSA crypto blocks. We can get 2048-bit RSA operations when we applying the Chinese Remainder Theorem to the 1024-bit RSA crypto block. The scalar multiplication level designed ECC crypto block and the exponentiation level designed RSA crypto block provide the user with easy interface to their applications such as ECDH, ECDSA, and RSA encryption/decryption. The ECC crypto block has field operation units (addition, multiplication, squaring, and inversion) and curve operation units (point doubling and addition) as its internal blocks.

Tables III and IV show features of our crypto blocks when implemented using an FPGA. Although the AES crypto block was implemented with an one-round based architecture, it achieves 390Mbps, which is superior to SEED and Triple-DES crypto blocks, which were implemented with a sub-pipelining and pipelining technique, respectively. The logic size of the AES crypto block is rather high because it's S-box and SI-boxes are implemented in the FPGA's internal memory (only two S-boxes and two SI-boxes are implemented). It is also possible to reduce the logic size if we sacrifice performance.

The KASUMI crypto block(Type 2, the high performance version) achieves the highest performance as shown in Table III. This is because KASUMI has a highly parallelizable encryption/decryption body and key scheduling block as described in the preceding section. When we compare the characteristics between KASUMI and SEED, which are implemented with a similar architecture (sub-pipelining), we can easily see KASUMI is superior to SEED in operating frequency, hardware complexity, and performance. Since the confidentiality algorithm f8 and integrity algorithm f9 in a 3GPP system are simple applications based on the KASUMI crypto algorithm[7], we can easily implemented f8/f9 algorithms with low overhead.

The SEED crypto block achieves 358Mbps in spite of its 5 stage sub-pipelines. This is due to of its architectural inefficiency; the $G$ function and the modulo-$2^{32}$ adder, which are the core of the SEED crypto algorithm, produce a long delay time. The SEED crypto block has high hardware complexity in crypto blocks when compared to other crypto blocks with a similar architecture such as KASUMI and Triple-DES. The Triple-DES crypto block shows the worst performance in our crypto blocks. This is because Triple-DES requires 48 round operations (three repetitions of DES) to complete its encryption/decryption operation.

Table IV shows the characteristics of the ECC and RSA crypto blocks implemented in the FPGA chip. The public key crypto blocks operate at about the range of 28 to 50MHz. The execution time of the ECC crypto block is 7.28*msec*, which corresponds to a throughput of about 20Kbps. The execution time of the ECC crypto block consists of the time for computing scalar multiplication, $kP$, where $P$ is defined in $GF(2^{146})$ and $k$ is a random 146-bit value. The execution time of the RSA crypto block is 6.69*msec* for encryption and 58.9*msec* for decryption, which correspond to 153Kbps and 17Kbps throughput respectively.

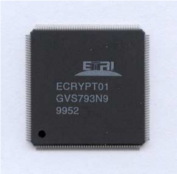


**Fig. 16. Photograph of the crypto processor.**

TABLE V
MAIN FEATURES OF THE CRYPTO PROCESSOR

| Parameter | Value |
|---|---|
| Technology | 0.5 $\mu m$ CMOS |
| Package Type | PQFP |
| Gate Counts | 200K (with I/O PADS) |
| Chip Size | 8.1*mm* X 8.1*mm.* |
| Operating Frequency | 33MHz (PCI bus interface) |
| Number of I/O Pins | 176pins |
| VDD and VSS | 5V for VDD, and 0V for VSS |

The critical point in the performance of the ECC crypto block is the efficiency of the multiplicative inversion algorithm and the scalar multiplication algorithm. Also, the critical path in our RSA crypto block is due to the 1024-bit addition block. Since there are other algorithms that achieve better performance than those used in our crypto blocks[1],[4], we will devise more efficient algorithms in implementing the next version of our crypto processor.

### *B. ASIC Implementation*

After verification of our design with an FPGA implementation, we have laid out and fabricated the crypto processor using 0.5 $\mu m$ CMOS technology[1].

Fig.16 shows a photograph of the crypto processor, and Table V summarizes the main features of the crypto processor. Note that a photograph of the layout is not presented as the circuit was synthesized using a standard cell library (the layout picture is not very meaningful as it simply consists of a few rectangular boxes). Also, in the ASIC version of our crypto processor, we have implemented the S-boxes of crypto blocks with combinational logic (and not by ROMs) because the target process (0.5 $\mu m$ CMOS process) does not support embedded ROMs.

## V. A Crypto Processor Application: Real-time Data Security for a Storage Device

To evaluate the usability of the crypto processor, we have developed an RTDS (Real Time Data Security) system for storage devices. The RTDS system is composed of control and monitoring software with a GUI(Graphical User Interface) environment, a device driver, and an RTDS board. Fig.17 shows the block diagram of the RTDS system, and Fig.18 shows a photograph of the RTDS board with the crypto processor.

In order to make a section of the hard disk area secure, a user can configure a specific directory to be a secure directory by using the control and monitoring software. Data which is written (read) to (from) the secure hard disk area is automatically encrypted (decrypted) by the crypto processor in real-time.

1. A user process wants to write data into the secure area of a hard disk(a).
2. The CPU reads data from a certain area of the memory and sends it to the hard disk via the I/O bus(b).
3. The device driver, which is a part of a RTDS system, catches the hard disk write event, and forwards data to the crypto processor(c).
4. In the crypto processor, an encryption task is performed in real-time(d).
5. The crypto processor, which has completed its encryption task, sends the encrypted data to the hard disk(e).
6. The hard disk receives the encrypted data and completes the write procedure(f).

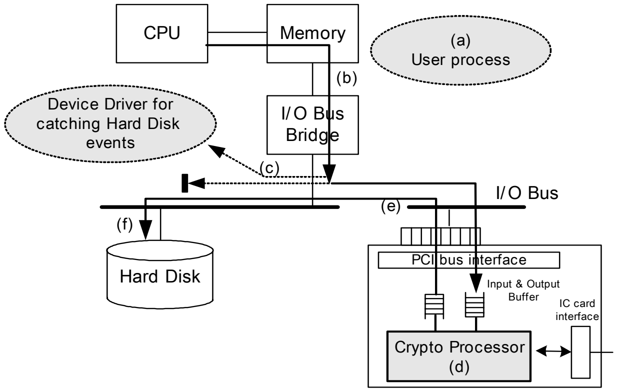


**Fig. 17. Block diagram of the Real Time Data Security System for storage devices.**

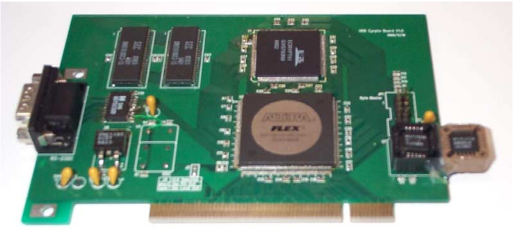


**Fig. 18. Photograph of the RTDS board.**

The RTDS board, shown in Fig.18, is mainly composed of a PCI interface controller, an SRAM buffer, a FPGA chip and an ASIC chip. The performance of the crypto processor and the PCI interface controller is high —267□ 568 Mbps and 1056 Mbps, respectively — and the average access time of the hard disk that is used for our test environment, is low — 12 $ms$ in our system. Therefore, the RTDS system operates in real-time.

## VI. Conclusions and Future Works

In this paper, we have presented the design and implementation of a crypto processor composed of a 32-bit RISC processor and coprocessor blocks dedicated to the AES, KASUMI, SEED, triple-DES, ECC and RSA crypto algorithms. The dedicated block of the crypto processor accelerates private and public key crypto algorithms and the programmability of the crypto controller makes possible fast execution of various security applications (such as SHA-1 and protocol processing etc.). Some parts of the crypto processor were also implemented as an ASIC chip using 0.5 $\mu m$ CMOS technology after verification with an FPGA implementation. Simulations, formal verification, and static timing analysis were used to fully verify the ASIC design before fabrication.

The crypto processor was evaluated by constructing an RTDS (Real-Time Data Security) system for storage devices.

[1] In ASIC version of our crypto processor, we have only implemented the 32-bit RISC processor, SEED and Triple-DES crypto block because of the chip size constraint.

This application board was used to thoroughly test and verify the functionality of the crypto processor. The crypto processor in the RTDS system performs data encryption and decryption in real-time. The high performance and high flexibility of the crypto processor design makes it applicable to various security applications such as storage devices, embedded systems, network routers, security gateways for IPSec and SSL protocol processing, etc.

For future work, we plan to develop additional high performance public key crypto blocks. Also, to enhance the security of our crypto processor, we will devise side channel attack resistant techniques in the private and public key crypto blocks.

## References

[1] Paul C. van Oorschot, Alfred J. Menezes, and Scott A. Vanstone, *Handbook of applied cryptography,* CRC press Inc., Florida, 1996.

[2] Joan Daemen and Vincent Rijmen, The design of Rijndael: *AES – the Advanced Encryption Standard*, Springer-Verlag, Berlin, 2002.

[3] ARM corp., *ARM7 Data Sheet*, 1996.

[4] I. F. Blake, G. Seroussi, and N. P. Smart, *Elliptic curves in cryptography*, London Mathematical Society Lecture Notes, Cambridge University Press, Cambridge, 2000.

[5] H. W. Kim, Y. J. Choi, M. S. Kim, and H. S. Ryu, "*Hardware Implementation of 3GPP KASUMI Crypto Algorithm*," ITC-CSCC, Vol. 1, pages 317-320, July 16 – 19, 2002, Phuket, Thailand.

[6] Alfred Menezes, *Elliptic curve public key cryptosystems,* Kluwer Academic Publishers, 1993.

[7] ETSI/SAGE, *Specification of the 3GPP confidentiality and integrity algorithms document 2: KASUMI specification*, December 1999.

[8] Hasan, Wange, and Bhargava, *A modified massey-omura parallel multipler for a class of finite fields*, IEEE Transactions on Computers 42(1993), 1278-1280.

[9] G. B. Agnew, R. C. Mullin, I. M. Onyszchuck, and S. A. Vanstone, "*An Implementation for a Fast Public-Key Cryptosystem*,"Journal of Cryptology, vol. 3, pp. 63-79, 1991.

[10] Don B. Johnson, Alfred J. Menezes, and Scott Vanstone, *Elliptic curve digital signature algorithm(ECDSA)*, available at http://www.certicom.com.

[11] Konstas marinis, Nikos K. Moshopoulos, Foris Karoubalis, and Kiamal Z. Pekmestzi*, On the hardware implementation of the 3GPP confidentiality and integrity algorithms*, ISC 2001, Lecture Notes in Computer Science 2200, 2001.

[12] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanston(eds.), *Handbook of applied cryptography*, CRC Press, 1996.

[13] National Institute of Standards and Technology, *FIPS publication 46-2: Data Encryption Standard*, National Institute for Standards and Technology, Gaithersburg, MD, USA, December 1993.

[14] Machael Rosing, *Implementing elliptic curves in cryptography*, Manning, 1998.

[15] Bruce Schneier, *Applied cryptography(2nd ed.)*, John Wiley and Sons, Inc., New York, 1996.

[16] TTA, *128-bit Symmetric Block Cipher(SEED)*, Telecommunications Technology Association(TTA), Seoul, Korea, June 1999.

[17] Xinmiao Zhang and Keshab K. Parhi, "Implementation Approaches for the Advanced Encryption Standard Algorithm," *IEEE Circuits and Systems Magazine*, Vol. 2, Issue. 4, pp. 24-46, Fourth Quarter 2002.

[18] Mitsuru Matsui, "New block encryption algorithm MISTY," *Fast Software Encryption '97*, vol. 1267 of LNCS, pages 54-68, Springer-Verlag, 1997.

[19] R. L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and pubic-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120-126, Feb. 1978.

**Ho Won Kim** (M'93) received his B.S.E.E. degree from KyungPook National University, DaeGu, Korea, in 1993 and the M.S and Ph.D. degrees in Electronic and Electrical Engineering from Pohang University of Science and Technology(POSTECH), Pohang, Korea, in 1995 and 1999, respectively. He is currently a senior researcher at the Electronics and Telecommunications Research Institute(ETRI), DaeJeon, Korea. His research interests include information security and computer architecture. Currently, his main research focus is on private and public key crypto processor design and IC card security issues. He is a member of the IEEE and the IEEE Computer Society.

**Sunggu Lee** (M'83) received the B.S.E.E. degree with highest distinction from the University of Kansas, Lawrence, in 1985 and the M.S.E. and Ph.D. degrees from the University of Michigan, Ann Arbor, in 1987 and 1990, respectively. He is currently an Associate Professor in the Department of Electronic and Electrical Engineering at the Pohang University of Science and Technology (POSTECH), Pohang, Korea. Prior to this appointment, he was an Assistant Professor in the Department of Electrical Engineering at the University of Delaware in Newark, Delaware, U.S.A. From June 1997 to June 1998, he spent one year as a Visiting Scientist at the IBM T. J. Watson Research Center. His research interests are in parallel computing using clusters, fault-tolerant computing, and real-time computing.