An Area-Efficient Universal Cryptography Processor for Smart Cards

Yadollah Eslami, *Member, IEEE*, Ali Sheikholeslami, *Senior Member, IEEE*, P. Glenn Gulak, *Senior Member, IEEE*, Shoichi Masui, *Member, IEEE*, and Kenji Mukaida

***Abstract*—Cryptography circuits for smart cards and portable electronic devices provide user authentication and secure data communication. These circuits should, in general, occupy small chip area, consume low power, handle several cryptography algorithms, and provide acceptable performance. This paper presents, for the first time, a hardware implementation of three standard cryptography algorithms on a universal architecture. The microcoded cryptography processor targets smart card applications and implements both private key and public key algorithms and meets the power and performance specifications and is as small as 2.25 mm^2 in 0.18-μm 6LM CMOS. A new algorithm is implemented by changing the contents of the memory blocks that are implemented in ferroelectric RAM (FeRAM). Using FeRAM allows nonvolatile storage of the configuration bits, which are changed only when a new algorithm instantiation is required.**

***Index Terms*—Computer security, cryptography, microprocessors, smart cards.**

I. Introduction

THE rapid growth of portable electronic devices with limited power and area has opened a vast area of low-power and compact circuit design opportunities and challenges for VLSI circuit designers. Cellular phones, PDAs, and smart cards are examples of portable electronic products that are becoming an integral part of everyday life. The popularity of these devices necessitates special considerations for their security subsystems. Unlike computer network security systems that impose less stringent limitations on the area and power consumption but put more emphasis on high throughput (several Gigabit/s), portable applications demand security hardware with more restrictions on area and power and less on throughput (several hundred kilobit/s to a few Megabit/s). This difference in requirements dictates a different approach in the design and implementation of the security systems for these devices.

Since next-generation, multipurpose smart cards will be used for a wide range of applications, their security system must implement both private (symmetric) and public (asymmetric) key algorithms, to accommodate various application requirements. Private key algorithms with high throughput are suitable for data communication, while public key algorithms with much lower throughput are suitable for private key exchange and authentication. Among all available algorithms, data encryption standard (DES), advanced encryption standard (AES), and elliptic curve cryptography (ECC), which are approved by standards organizations [1]–[3], are selected for this application. DES, for past compatibility, and AES, for high security and throughput, are the major candidates for private key algorithms, and ECC is the best candidate for the public key algorithm for its encryption efficiency. RSA, which is also a standard public key algorithm, is not considered in this design for three reasons. First, it is believed that 160-b ECC provides the same level of security as 1024-b RSA. Thus, ECC will be a better choice when implementation area is a key factor in the design. Second, RSA uses binary addition of large numbers and needs binary adders that are either slow for carry propagation or large for look-ahead carry generation. Third, a larger number of bits in RSA means wider buses, which adds to the area and power consumption of the design, both of which are scarce resources in smart cards.

A cryptography system can be implemented in either software or hardware. Software implementations allow multiple algorithms to be supported on the same hardware platform, but they are usually slow and cannot meet the required specifications. Moreover, they are considered to be more vulnerable to side-channel attacks compared to other implementations. Side-channel attacks use physical measurements on the device, for example, the power consumption of the processor, to detect the encryption/decryption key [4]–[6]. On the other hand, hardware implementations which support high throughput do not allow for flexibility and, hence, are not suitable for smart cards. Flexible field-programmable gate array (FPGA) implementations are not a good choice either, because they need large area and power which cannot be supported on smart cards.

Based on the physical constraints for smart cards set by the International Standard Organization (ISO) [7], the maximum chip area on a smart card is limited to 25 mm^2. Considering the nonvolatile and volatile memories, CPU, and other peripheral circuits required on the chip, it is desirable to fit the security subsystem in as small an area as possible. One of the objectives of this study is to investigate the minimum required chip area for the implementation of the security circuits satisfying the algorithm agility, power consumption, and throughput requirements. Rows 3, 8, and 9 of Table I show the area and power consumption of recently published ASIC implementations of several cryptography algorithms. Although these implementations mostly target high-throughput network applications,

Manuscript received June 25, 2004; revised on February 3, 2005 and May 6, 2005. This work was supported in part by Fujitsu Laboratories Ltd., Japan, and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

Y. Eslami is with the Department of DRAM Research and Development, Micron Technology Inc., Boise, ID 83707 USA (e-mail: yeslami@micron.com).

A. Sheikholeslami and P. G. Gulak are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: ali@eecg.utoronto.ca; gulak@eecg.utoronto.ca).

S. Masui and K. Mukaida are with Fujitsu Laboratories Ltd., Kawasaki, 211-8588 Japan (e-mail: masui.shoichi@jp.fujitsu.com; mukaida@fram.ed.fujitsu.co.jp).

Digital Object Identifier 10.1109/TVLSI.2005.863188

TABLE I
SOME RECENT CRYPTOGRAPHY ALGORITHM IMPLEMENTATION SPECIFICATIONS (*: BASED ON THE AREA OF A TWO-INPUT NAND GATE WITH FANOUT OF FOUR IN 0.18-μm CMOS)

	ALGORITHM	HW/ SW	ASIC/FPGA	CLOCK (MHZ)	AREA/GATE COUNT	THROUGHPUT	POWER	TECHNOLOGY
[15] *2000*	*DES RSA Blowfish SAFER*	*HW*	*ASIC*	*77*	-	*44.0 MBytes/s 300 Kbps*	*No Silicon*	*2μm*
[30] *2001*	*ECC*	*SW*	-	-	-	*44.8ms/PM (160bits)*	-	*ARM (32 bit)*
[10] *2001*	*RSA ECC*	*HW*	*ASIC*	*50*	*2.9mm x 2.9mm ($8.41mm^2$)*	*8.2ms (512bit) 6.95ms/PM (176bit)*	*75mW @2V*	*0.25μm*
[31] *2001*	*AES*	*HW*	*ASIC*	*333*	*0.1 mm^2*	*24Mbps*	*No Silicon*	*0.18μm*
[32] *2002*	*ECC*	*SW*	-	*13*	-	*~22ms/PM (163bits)*	-	*ARM (32bit)*
[11] *2002*	*ECC*	*HW*	*FPGA (Variable n)*	*30*	*936 CLBs (155bit)*	*10.8 ms (155bit)*	-	*Xilinx XCV1000-6*
[12] *2002*	*ECC*	*HW*	*ASIC*	*13.56*	*14434 gates (160bit) ($0.5mm^2$)**	*15ms/PM (160bit)*	*No Silicon*	*0.35μm*
[14] *2003*	*AES*	*HW*	*ASIC*	*154*	*173000 gates ($5.9mm^2$)**	*1.6Gbps (AES-128)*	*56mW @1.8V*	*0.18μm*
[29] *2003*	*AES*	*HW*	*ASIC*	*465*	*28626 gates (1.0 mm^2)* + 4Kbit RAM + 128Kbit ROM*	*1.64Gb/s*	*314mW @1.8V*	*0.18μm*
Our work	*DES AES ECC*	*HW*	*ASIC*	*13.56*	*2.25mm^2*	*3.5Mbps 1.83(Enc), 0.85(Dec)Mbps 417ms/PM (155b)*	*15.9mW 16.3mW 18.3mW @1.8V*	*0.18μm 6LM*

they clearly indicate the challenge in fitting three separate dedicated circuits—to implement the three mentioned algorithms (i.e., DES, AES, and ECC)—in a small area. Moreover, the cryptography scircuits should be able to provide encryption/decryption of the data at the maximum data transfer rate of 847.5 kb/ps $(f_{\mathrm{CLK}}/16)$ at a clock frequency of 13.56 MHz set for contactless smart cards [8]. Furthermore, in the case of contactless smart cards, since the power is transferred to the IC via RF signals, the power transfer is limited to 10–20 mW, so the cryptography circuits should ideally be power-aware. RISC processors that implement the cryptography algorithms in software provide a very attractive throughput, area, and power consumption solution [9]. However, since users and manufacturers are still concerned about the vulnerability of software implementations of the cryptography algorithms to side-channel attacks, it is important to design hardware circuits that meet the required specifications and are more immune to these attacks. Considering all of the limitations discussed so far, it is quite challenging to design and implement an algorithm-agile and area-and-power-efficient cryptoprocessor for smart cards with acceptable performance and security.

This paper introduces, for the first time, a hardware implementation of a cryptography coprocessor (cryptoprocessor) that implements two standard private-key algorithms (DES and AES), and one standard public-key algorithm (ECC), in a single design that satisfies the power consumption and

throughput requirements of smart cards and occupies 2.25 mm^2 in 0.18-μm 6LM CMOS. The published works in this area address only a subset of these issues. For example, [10] introduces an energy-efficient reconfigurable cryptography processor that considers public key cryptography only. References [11] and [12] consider ECC only, [13] implements DES, and [14] implements AES only. The FPGA implementation in [15] is a processor that does not support AES and ECC. A reconfigurable architecture to implement six candidate algorithms of AES competition is presented in [16].

This paper is organized as follows. Section II presents an overview of the three cryptography algorithms implemented in this study. A mathematical background for these algorithms can be found in [3] and [17]–[20]. Section III introduces the proposed cryptography engine. The designed cryptoprocessor specifications are presented in Section IV, followed by the implementation details in Section V. Area and power requirements of the design are discussed in Section VI and its performance in Section VII. Possible enhancements to the design presented in this paper are discussed in Section VIII. The conclusions are provided in Section IX.

II. Cryptography Algorithms

In this section, a brief introduction of the three implemented algorithms which are essential in the understanding of the remainder of the paper are provided. Interested readers are referred to [1]–[3], and [17]–[19] for additional details.

A. Data Encryption Standard (DES)

This is a well-established algorithm that has been used for more than two decades (since 1977) in military and commercial data exchange and storage. The algorithm is designed to encipher and decipher blocks of data consisting of 64 b using a 56-b key. A block to be enciphered is subjected to an initial permutation (IP), then to 16 rounds of a complex key-dependent permutation, and, finally, to another permutation which is the inverse of the IP, IP^{-1}, as shown in Fig. 1. The function f() in this figure is the heart of this algorithm and consists of an expansion, XOR, lookup table (LUT), and permutation, as depicted in Fig. 2. To decipher, it is necessary to apply the very same algorithm to an enciphered message block, using the same key.

Since the processing power of current computers is much higher than those of two decades ago, a brute-force attack (checking all possible key combinations to decipher an encrypted ciphertext) to this algorithm is possible in a relatively short time (possibly as short as a few minutes [21]). For this reason, this algorithm is no longer considered to be a secure algorithm for many applications by the National Institute of Standards and Technology (NIST). A more secure algorithm based on DES which is still supported by NIST is called the triple data encryption algorithm (Triple DES, 3DES, or TDEA) depicted in Fig. 3. In this figure, DES represents encryption and DES^{-1} represents decryption. 3DES involves applying DES, then DES^{-1}, followed by a final DES to the plain text using three different key options [1], which results in a cipher text that is much harder to break.

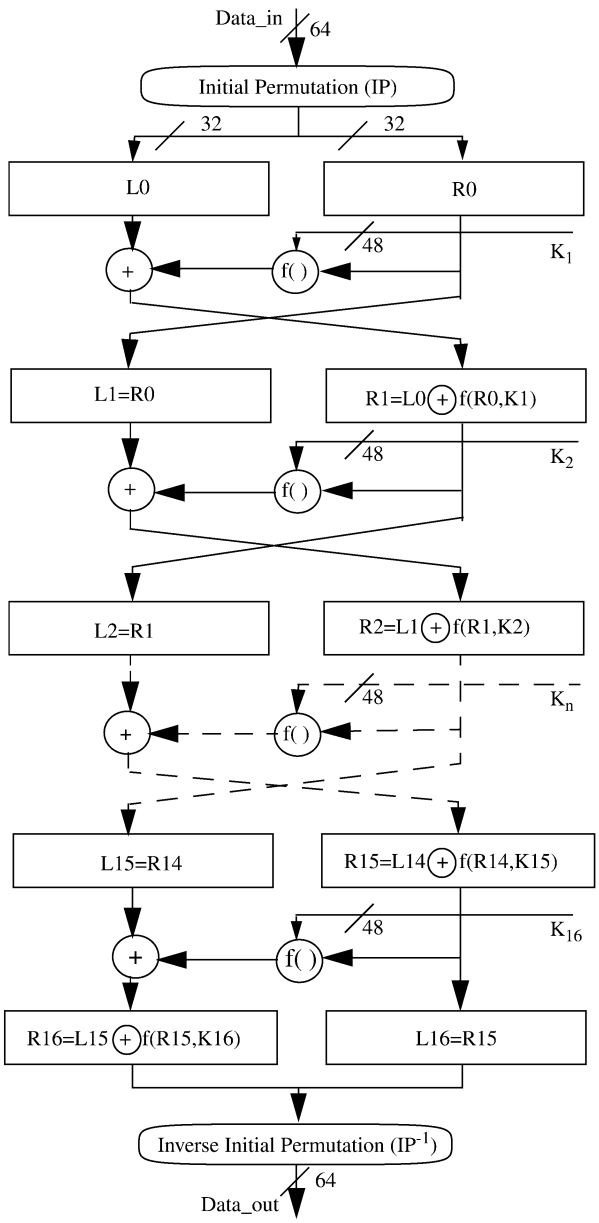


Fig. 1. DES block diagram (K1, K2,$\cdots$, K16 refer to the key values used in rounds 1–16).

The implementation of DES needs four basic operations only, namely, the XOR, shift, LUT, and permutation, which are relatively simple to implement in hardware. The TDEA also uses the same set of operations as DES.

B. Advanced Encryption Standard (AES)

AES, also known as Rijndael, is a block encryption algorithm which encrypts blocks of 128 b using a unique key for both encryption and decryption [2]. A block diagram representation of the algorithm is shown in Fig. 4.

Three versions of the algorithm are available differing only in the key generation procedure and in the number of rounds the data is processed for a complete encryption (decryption) [2]. AES-128 uses a 128-b key and needs 10 rounds. AES-192 and AES-256, respectively, need 192-b and 256-b keys and 12 and 14 rounds for processing a block of data.

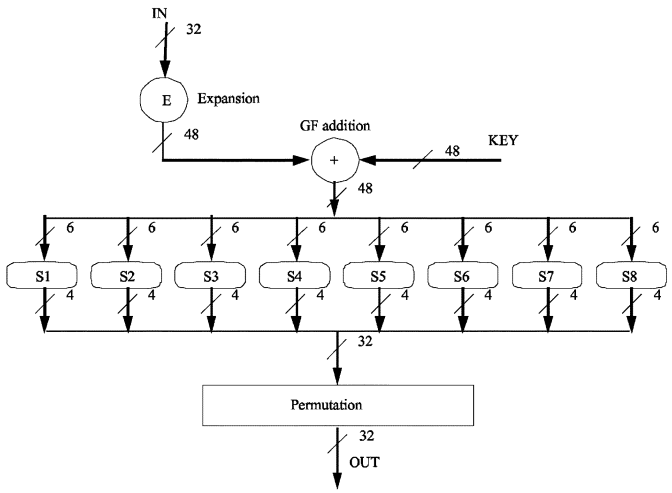


Fig. 2. DES f()-box details.

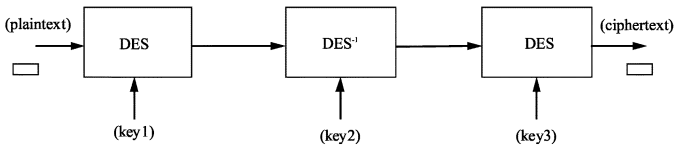


Fig. 3. 3DES (TDEA) block diagram.

The 128-b input data is considered as a 4×4 array of 8-b bytes (also called "state" in the algorithm). The state undergoes four different operations in each round, except for the final round which has only three operations. These operations are "*ByteSub*," "*ShiftRow*," "*MixColumn*," and "*AddRoundKey*" operations. "*MixColumn*" is omitted in the final round. Each round of the algorithm needs a 128-b key, which is generated from the input key to the algorithm. The key-scheduler block (not shown in Fig. 4) consists of two sections: the key expansion unit, which expands the input key bits to the maximum number of bits required by the algorithm, and the key selection unit, which selects the required number of bits from the expanded key, for every round [2]. As mentioned before, aside from the key values, all of the steps in all of the rounds are the same except for the last round that *MixColumn* is not present.

Each byte in the state matrix is an element of a Galois Field $\mathrm{GF}(2^8)$, and all of the operations can be expressed in terms of the field operations [2]. In simple terms, $\mathrm{GF}(2^n)$ is a set of 2^n elements each represented by an n-bit string of 0's and 1's and two basic operations: addition and multiplication. These two operations are defined such that the closure, associativity, and other field properties are satisfied [20].

From the implementation point of view, *ByteSub* operation can be implemented by LUT. The *ShiftRow* can be implemented using a circular shifter. The *MixColumn* is the most complicated operation in this algorithm and needs $\mathrm{GF}(2^8)$ field multiply and add operations. Due to the specific choices of the parameters of the algorithm, this operation can be expressed as a matrix multiplication, which can be implemented using shift and XOR operations. A more detailed analysis of the implementation options of this block are presented in [22]. *AddRoundKey* is just a logical XOR operation.

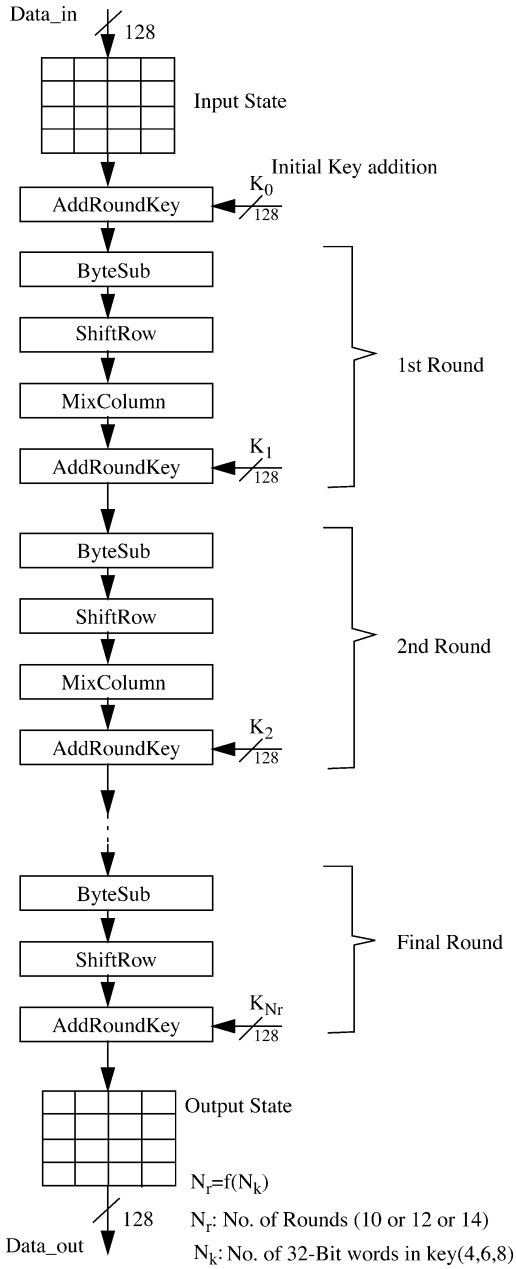


Fig. 4. AES block diagram.

C. Elliptic Curve Cryptography (ECC)

The set of all (x, y) pairs satisfying the nonsupersingular elliptic curve equation

$$y^2 + xy = x^3 + a_2x^2 + a_6$$

are called points on the elliptic curve E, where x, y, a_2, and a_6 are elements of the $\mathrm{GF}(2^n)$. The *point addition* $(S = P + Q)$ and *multiplication* $(R = k \cdot P$, where k is a constant) operations are defined such that both S and R are also points on the elliptic curve E. Moreover, knowing R and P, it is practically impossible to find k. This property forms the fundamental foundation of ECC [17], [18].

Elliptic curves can be used in different forms in cryptography. As an example, we will explain one of the basic applications,

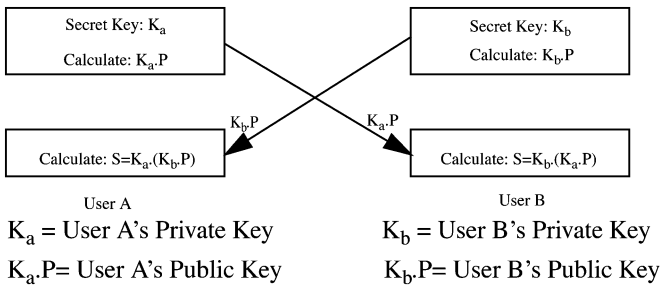


Fig. 5. ECC secret-key-exchange algorithm block diagram.

which is the secret key exchange. The basic secret-sharing algorithm (ECKAS-DH in [3]), also known as the Diffie–Hellman protocol for key exchange, is pictured in Fig. 5. In brief, both users, A and B, agree on the elliptic curve E, a point P on E, and a mathematical basis, such as polynomial basis or normal basis (NB). Each user then chooses a secret key from $\mathrm{GF}(2^m)$, K_a and K_b, and calculates her/his own public key ($PK_{\mathrm{a}} = K_a$. P and $PK_b = K_b \cdot P$) and sends it to the other user. At this point both users can calculate the secret point $S(x_s, y_s)$

$$S(x_s, y_s) = K_a \cdot PK_b = K_b \cdot PK_a = K_a \cdot K_b \cdot P.$$

Note that, although both x_s and y_s are available, only one of them should be used for higher security [3], [17]–[19].

III. Cryptography Engine

Our procedure for designing this domain-specific processor starts by listing all of the required functional blocks for the selected algorithms followed by the selection of a minimum set of the required blocks that can implement all algorithms. Then, a cost function, which is the area in this case, but could be any other parameter such as power or speed or any combination of several parameters, is defined and calculated for the selected set. If the cost is larger than the available budget, the complex functional blocks are expressed in terms of simpler blocks by revising the algorithm primitives or by selecting a new mathematical interpretation of the operation. This should be followed by the selection of the minimum set of functional blocks and calculation of the new cost. The procedure should be repeated until the cost criterion is met or the simplest form of the functional blocks is achieved.

The list of operations required by the three algorithms DES, AES, and ECC is shown in Table II. The list is directly derived from the primitives of the algorithms as defined in Section II. Referring to Table I, a design including three independent circuits, each implementing one of these algorithms, needs an area that is much larger than the available area for this application. A reconfigurable architecture at this level is not feasible either, because most of the operations are complex and dedicated to each algorithm, and there are only few operations which are common among the three algorithms.

Our approach to address this problem is to express the high-level operations of each algorithm in terms of basic arithmetic and logic operations to maximize the number of common operations among the algorithms and, hence, minimize the overall

TABLE II
Operations Required by DES, AES, and ECC for Direct Implementation

	Operand Size					
	Expand/ Permute	Substitute	Logic XOR	GF SQR	GF MULT	GF INV
DES	35,56, 48, 64	6in => 4out	32, 48	-	-	-
AES	-	8in => 8out	8, 128	-	8	-
ECC	-	-	m	m	M	m

(GF: Galois Field m: ECC key length in bits)

required chip area. This is challenging for two reasons. First, for DES, the data and the key lengths are fixed at 64 and 56 b, for AES the data length is fixed at 128 b, but the key length could be any of 128, 192, or 256 b, and for ECC both the data and the key lengths are variable (m-bits). Second, DES uses a set of simple logical functions, substitution, and permutation operations, and AES adds to this set a more complicated $\mathrm{GF}(2^8)$ multiplication which can be implemented using shift and XOR operations, while ECC is based on complex mathematical finite field operations in GF(2) and $\mathrm{GF}(2^m)$.

The field elements used in ECC can be represented in either polynomial basis (PB) or normal basis (NB) [3], [17]–[19]. The hardware implementation of the field operations when using PB requires large silicon area. By using NB, $\mathrm{GF}(2^m)$ squaring (GF SQR in Table II) is achieved by a simple circular shift left operation. Furthermore, $\mathrm{GF}(2^m)$ multiplication is simplified to a series of LUTs, AND, and XOR operations, and $\mathrm{GF}(2^m)$ inversion is simplified to a series of shift and field multiplications [17].

Optimal normal basis (ONB) representation is a subset of NB which further reduces the implementation complexity and increases the throughput. The major advantage of using ONB instead of NB is that the multiplication matrix [3] used in field multiplication in ONB has at most two nonzero elements per row/column in comparison to up to m nonzero values in NB. To share more blocks among the three algorithms and minimize the hardware and increase the throughput, we use an ONB representation of field elements for ECC in our design. This choice affects the design in two ways. First, the multiply and invert operations are performed faster compared to NB and require less power. Second, this limits the ECC implementation to the field sizes that support ONB only [3], because all field sizes do not have ONB representations. This is not considered a serious drawback because there are enough field sizes with ONB representation that can be used for ECC implementation. Our design implements ECC with five field sizes for low (83 b), medium (131, 155), and high (239, 254) security applications.

Table III shows the list of all required functional blocks for the desired algorithms after expressing the basic primitives in terms of simple operations. The table reveals that the algorithms may be implemented using much simpler operations that also maximize the number of common blocks among them. Based on the information in Table III, we are proposing the cryptoprocessor

TABLE III
OPERATIONS REQUIRED BY DES, AES, AND ECC AFTER REORGANIZATION OF PRIMITIVES

	Operand Size				
	Expand/ Permute	Table look up	Logic XOR	Logic AND	Circular/Logical Right/Left Shift
DES	35,56, 48, 64	6in => 4out	32, 48	-	28 (by 1 or 2 bits)
AES	-	8in => 8out	8, 128	-	32 (by 8 bits)
ECC	-	-	m	m	m (by x bits, 0<x<m)

(m: ECC key length in bits)

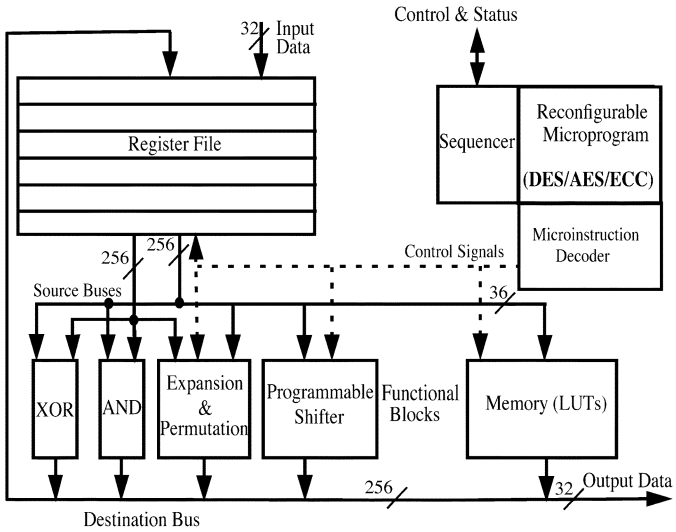


Fig. 6. Detailed cryptoprocessor architecture.

depicted in Fig. 6, which uses a microprogrammed processor architecture [25], [26]. This design implements all of the required microoperations for the three algorithms and provides algorithm agility by updating the contents of its memory blocks.

IV. CRYPTOPROCESSOR SPECIFICATIONS

This section includes three subsections. Section IV-A explains the architecture of the designed cryptoprocessor, which is followed by the details of its initialization and interface with the host CPU in Section IV-B. Section IV-C introduces the Microinstruction set and the Microprograms for the implemented algorithms in the cryptoprocessor.

A. Architecture

A microprogrammed control unit (Fig. 7) controls the data path and implements the algorithm agility. The design uses a 32-b bidirectional data bus and a 9-b address bus and a 13-b control bus to communicate to the host CPU. The host CPU uses these signals to write/read the algorithm inputs/outputs to/from the memory-mapped cryptoprocessor registers in addition to the updating the configuration memories (microprogram and LUTs) that define the function of the cryptoprocessor.

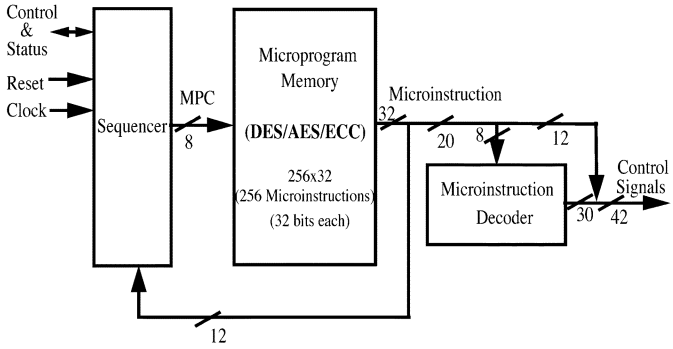


Fig. 7. Cryptoprocessor controller and its subblock interconnections.

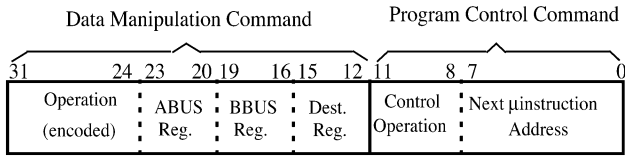


Fig. 8. Cryptoprocessor microinstruction format.

TABLE IV
TYPICAL FERAM AND SRAM CHARACTERISTICS IN 0.18-μM CMOS TECHNOLOGY (PLANAR CAPACITOR ASSUMED FOR FERAM)

	Type	Read_mode	Size	Area (mm^2)	Power ($mW/V^2/MHz$)	Access Time (ns)
FeRAM	NonVol.	Dest.	256X32	0.26	0.0213	50
SRAM	Vol.	NonDest.	256X32	0.35	0.0163	1.9

Dest. : Destructive Vol.: Volatile

Microinstruction encoding (Fig. 8) is used to minimize the number of bits in each microinstruction. Note that the 8-b operation code field in the instruction code is used to have the word length of the memory at 32 b and to use standard memory blocks for the microprogram memory. This field could be reduced to 5 b (only 32 microinstructions are required for the current implementation, as shown in Table V) and a custom-designed memory module with 29-b word length could be used to further reduce the chip area.

LUTs are used to realize the substitution operations in DES and AES and to implement the multiply operation in ECC. For the case of ECC, LUTs are used to implement the $m \times m$ multiplication matrix [3] using two $m \times \log_2 m$ arrays to save area. The most notable feature of this design is that reconfigurability is achieved by changing the contents of the microprogram memory and the LUTs. A maximum of 24 kb needs to be updated for an algorithm change, which can be done in less than 30 ms (using 847.5-kb.s data rate for contactless cards [8]).

Both volatile (SRAM) and nonvolatile memories (e.g., EEPROM, Flash, and FeRAM) can be used for the memory blocks in this cryptoprocessor, but nonvolatile memories have the advantage of keeping their contents, and hence the cryptoprocessor configuration, intact, even when the power is removed from the circuit. Therefore, the cryptoprocessor does not need to be reconfigured before each use, unless a change of algorithm is required. Among the nonvolatile memories,

TABLE V
DAT A MANIPULATION MICROINSTRUCTIONS OF THE CRYPTOPROCESSOR

Mnemonic	Operation	Comments
NOP		*No Operation*
PF	*out <--f_block_Permuted(in)*	*Permutation used in f() block of DES*
IP0	*out <--Initial_Permutation(in)*	*Initial Permutation used in DES(First 32 bit)*
IP1	*out <--Initial_Permutation(in)*	*Initial Permutation used in DES(Second 32 bit)*
IPM10	*out <--Inverse_Initial_Permutation(in)*	*Inverse Initial Permutation used in DES(First 32 bits)*
IPM11	*out <--Inverse_Initial_Permutation(in)*	*Inverse Initial Permutation used in DES(Second 32 bits)*
EXP0	*out <--Expanded(in)*	*Expansion function used in f() in DES(First 32 bits)*
EXP1	*out <--Expanded(in)*	*Expansion function used in f() in DES(Second 32 bits)*
PC10	*out <----Permutation_choice1(in)*	*Permutation Choice1 used in DES key(First 32 bits)*
PC11	*out <--Permutation_choice1(in)*	*Permutation Choice1 used in DES key(Second 32 bits)*
PC20	*out <--Permutation_choice2(in)*	*Permutation Choice2 used in DES key(First 32 bits)*
PC21	*out <--Permutation_choice2(in)*	*Permutation Choice2 used in DES key(Second 32 bits)*
MOV	*out <--in*	*Move Registers*
SHTL	*out <--Shifted_Left(in)*	*Shift Left operation*
SHTR	*out <--Shifted_Right(in)*	*Shift Right operation*
SPSHFW	*out <--(Special) Shifted(in)*	*Special Shift Left mode used in AES (Normal output)*
SPSHFR	*out <--(Special_extension) Shifted(in)*	*Special Shift Left mode used in AES (Extended output)*
LUT0	*out <--M[in]*	*Memory look up operation for every 8-bit of in and out*
LUT1	*out <--M[in]*	*Same as LUT0, but uses the upper 256 bytes of memories*
BMOV	*out[8*i-1: 8*9i-1)] < --in[8*i-1: 8*9i-1)]*	*Byte MOVe used in AES ShiftRows() operation*
XOR	*out <--(ABUS) XOR (BBUS)*	*XOR operation*
AND	*out <--(ABUS) AND (BBUS)*	*AND operation*
SHTR-LUT	*out <--Shifted(in)*	*Shift Right (ECC mode, uses LUT result as shift count)*
SHTL-WIN	*out <--Shifted(in)*	*Shift Left(ECC mode, uses Win_Reg as shift count)*

FeRAM is a better choice, because of its short *write access time* and low *power* and *voltage write operation* [13], which results in fast instantiation of the algorithm with minimum power. Differential capacitance read scheme (DCRS) [27] can be used for the FeRAM read operation to achieve short access time (less than 50 ns) that satisfies the available time budget for smart card applications ($T_{\rm cp} = 74$ ns approximately, which corresponds to $f_{\rm CLK} = 13.56$ MHz). The main disadvantage of FeRAM over other nonvolatile memories is its higher power consumption for read operation. One remedy to this problem is to use shadow SRAM memory blocks, which can add up to 10% to the overall chip area. Table IV compares the basic characteristics of typical FeRAM and SRAM memories in 0.18-μm CMOS technology.

The microcode is optimized such that at most 16 live variables need to be present in the cryptoprocessor at any given time. These variables are stored in 16 registers that comprise the register file. Source operands of the instructions are provided by the registers of the register file, and the result of the operation is written back to one of the registers at the end of the same clock cycle. To accommodate a variable number of data and shift counts by the shifter, a logarithmic shifter is used as the core of the shifter block. To minimize the area and power consumption, the width of the internal buses and the registers of the cryptoprocessor is limited to 256 b. Since a 256-b ECC provides higher security than a 2048-b RSA [18], the cryptoprocessor will provide sufficient security for smart card applications. Moreover, for applications that do not use all 256 b (e.g., DES, AES, and ECC with key length less than 256 b), the unused bits are deactivated to further reduce power consumption.

B. Initializations

The operational mode of the cryptoprocessor is controlled by a RESET signal. When the RESET signal is at logic "0," the cryptoprocessor is in reset mode and the host CPU can read and

TABLE VI
PROGRAM CONTROL MICROINSTRUCTIONS OF THE CRYPTOPROCESSOR

Mnemonic	Operation	Comments
JMPA	*MPC <--NIAD*	*Jump Always*
JNXLRA	*MPC <--MPC+1, Rnd_Cntr <--NIAD, m_Reg <--m-1, Win_Reg <--0*	*Jump Next, and Load Round Counter*
JPLRm	*MPC <--NIAD, Rnd_Cntr <-- m_Reg*	*JumP and Load Round counter from m_Register*
DRJNZ	*Rnd_Cntr <--Rnd_Cntr-1, If(Rnd_Cntr -1 !=0) Then MPC <--NIAD,*	*Decrement Round counter and Jump if NonZero*
RJW1	*{Win_Reg,m_Reg} <--shifted_left({Win_Reg,m_Reg, 1); If(shifted_bit ==1) Then MPC <--NIAD,*	*Shift left {Win_reg, m_Reg}by one bit and jump if shifted bit is "1", continue otherwise*
JNXLTA	*MPC <--MPC+1, Temp_Reg <--NIAD*	*Jump NeXt and Load Temp reg from address*
JPALTR	*MPC <--NIAD, Temp_Reg <--Rnd_Cntr*	*JumP And Load Temp reg from Rnd_Cntr*
JPALTW	*MPC <--NIAD, Temp_Reg <--Win_Reg*	*JumP And Load Temp reg from Win_Reg*
CALL	*MPC <--NIAD, SBR1<--MPC+1, SBR2<--SBR1*	*CALL subroutine*
RET	*MPC<--SBR1, SBR1<--SBR2*	*RETurn from subroutine*
DTJNZ	*Temp_Reg <--Temp_Reg -1, If(Temp_reg -1 !=0) Then MPC <--NIAD, Else MPC <--MPC+1*	*Decrement Temp register and Jump if NonZero*
JMPA	MPC <--NIAD	Jump Always
JNXLRA	MPC <--MPC+1, Rnd_Cntr <--NIAD, m_Reg <--m-1, Win_Reg <--0	Jump Next, and Load Round Counter
JPLRm	MPC <--NIAD, Rnd_Cntr <-- m_Reg	JumP and Load Round counter from m_Register
DRJNZ	Rnd_Cntr <--Rnd_Cntr-1, If(Rnd_Cntr -1 !=0) Then MPC <--NIAD, Else MPC <--MPC+1	Decrement Round counter and Jump if NonZero
RJW1	{Win_Reg,m_Reg} <--shifted_left({Win_Reg,m_Reg, 1); If(shifted_bit ==1) Then MPC <--NIAD, Else MPC <--MPC+1	Shift left {Win_reg, m_Reg}by one bit and jump if shifted bit is "1", continue otherwise
JNXLTA	MPC <--MPC+1, Temp_Reg <--NIAD	Jump NeXt and Load Temp reg from address
JPALTR	MPC <--NIAD, Temp_Reg <--Rnd_Cntr	JumP And Load Temp reg from Rnd_Cntr
JPALTW	MPC <--NIAD, Temp_Reg <--Win_Reg	JumP And Load Temp reg from Win_Reg
CALL	MPC <--NIAD, SBR1<--MPC+1, SBR2<--SBR1	CALL subroutine
RET	MPC<--SBR1, SBR1<--SBR2	RETurn from subroutine
DTJNZ	Temp_Reg <--Temp_Reg -1, If(Temp_reg -1 !=0) Then MPC <--NIAD, Else MPC <--MPC+1	Decrement Temp register and Jump if NonZero
JMPA	MPC <--NIAD	Jump Always
JNXLRA	MPC <--MPC+1, Rnd_Cntr <--NIAD, m_Reg <--m-1, Win_Reg <--0	Jump Next, and Load Round Counter
JPALTR	*MPC <--NIAD, Temp_Reg <--Rnd_Cntr*	*JumP And Load Temp reg from Rnd_Cntr*
JPALTW	*MPC <--NIAD, Temp_Reg <--Win_Reg*	*JumP And Load Temp reg from Win_Reg*
CALL	*MPC <--NIAD, SBR1<--MPC+1, SBR2<--SBR1*	*CALL subroutine*
RET	*MPC<--SBR1, SBR1<--SBR2*	*RETurn from subroutine*
DTJNZ	*Temp_Reg <--Temp_Reg -1, If(Temp_reg -1 !=0) Then MPC <--NIAD, Else MPC <--MPC+1*	*Decrement Temp register and Jump if NonZero*
SKEQ	*if(Rnd_Cntr == NIAD) Then MPC <--MPC+1 Else MPC<--MPC+2*	*SKip if EQual*
SKCY	*if(CY_in==0) Then MPC<--MPC+1, Else MPC<--MPC+2*	*Store carry input and sKip if CY_in is "1"*
JNCF	*if(CF=="0") Then MPC<--NIAD, Else MPC<--MPC+1*	*Jump if Not Carry Flag set*
DONE		*Assert DONE signal -Processor halt*

write the memory and register contents using the 32-b bidirectional data bus, 9-b address bus, and the four control signals. When the RESET signal is at logic "1," the cryptoprocessor is in run mode and acts as an ASIC, implementing one of the three algorithms based on the preloaded contents of the memory blocks.

The private and public keys are kept in the registers of the register file of the cryptoprocessor and are available to the host CPU in reset mode only. Thus, in a typical scenario, the cryptoprocessor, which is configured as an ECC engine, receives a private key using one of the key exchange protocols (such as

Diffie–Hellman) and stores it in one of its registers. Then, the cryptoprocessor is reconfigured in AES (or DES) mode by the host CPU, and the stored key is used for a data transfer session in a private-key data-exchange mode (DES or AES). Note that, for this implementation, there is no need to move the keys among the registers by the main processor, and, hence, less information will leak compared to the software implementations for the attackers who use the side-channel information to detect the algorithm keys.

C. Microinstruction Set and Microprograms

There are 24 data manipulation and 15 program control microinstructions defined for the designed cryptoprocessor, which are presented in Tables V and VI, respectively. Data manipulation microinstructions may need up to three operands (two source operands and one destination operand) which are always found in the cryptoprocessor register file, D0 to D15.

Each line of the microcode fits in one word (32 b) of the microprogram memory. The DES microcode is the shortest and has 46 lines of code (23 for encryption and 23 for decryption). The microcode for AES needs 150 lines of code (68 for encryption and 82 for decryption) and is the longest microcode among the three, mainly because of the key scheduler. It uses one level of subroutine call. ECC needs 60 lines of microcode to implement the point multiplication algorithm and uses two levels of subroutine call. The amount of time it takes to do a point multiplication depends on the field size. Table VII compares the time requirements of the three algorithms. The numbers presented for ECC are based on the assumption that all of the bits in the point multiplier k are "1." Obviously, this is not a valid assumption during normal operation and provides very pessimistic performance results, but presents the performance in the case where the processor needs to pretend that all the bits in k are "1" to prevent any side-channel attack. The anti-side-channel attack feature is not implemented in this version of the cryptoprocessor, but can easily be incorporated should it be confirmed that the current implementation leaks information about the key value.

V. Implementation Details

This section highlights the implementation techniques used in the hardware and/or the microcodes of DES, AES, and ECC algorithms.

A. DES

The parameters used in the main flow of DES depicted in Fig. 1 are 32 b wide and the implementation of the operations in each round is straightforward [1]. However, the operations required for the f()-box shown in Fig. 2 are slightly modified to match the architecture. Note that the key scheduler is also modified to have its output match the changes in the f()-box.

B. AES

Section II along with Fig. 4 introduced AES operations. This section focuses on the subtleties of three fundamental operations (and their inverses) that are critical to their area-efficient implementation.

TABLE VII
No. of Clock Cycles Required Per Algorithm.

	Encryption	Decryption
DES	248	248
AES	951	2036
ECC-83	1,367,114	
ECC-83	3,414,850	
ECC-83	5,739,898	
ECC-83	15,945,058	
ECC-83	19,297,332	

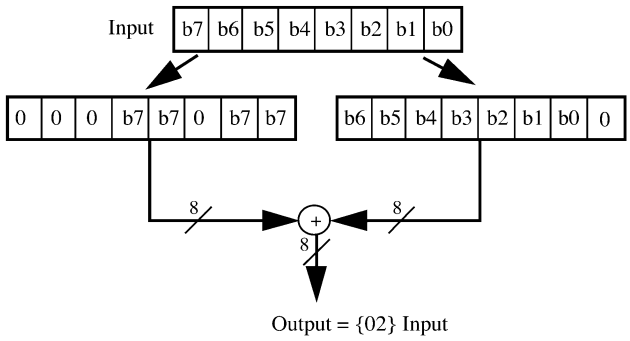


Fig. 9. AES X-times ({2} times) operation.

1) AddRoundKey and InvAddRoundKey: The key generation for AES encryption (decryption) can be done either in one step or in every round [2]. The former approach generates all of the required key bits for all rounds and needs some extra space to store them until they are consumed. The benefit is that the generated bits can be reused for consecutive blocks. The latter approach produces the key bits of a round when they are required and hence does not need extra space but needs to reproduce the key bits from the input key for each block. Our design uses the latter approach to keep the area as small as possible. Obviously, this approach degrades the throughput and the degradation is more severe for decryption than for encryption, but the throughput is still within the specifications for this application. Once the key bits for a round are ready, the AddRoundKey (InvAddRoundKey) operation is performed on all columns of the state matrix per clock using the XOR operation.

2) ShiftRow and InvShiftRow: Both of these operations are performed by ByteMove (BMOV) microinstruction that allows byte transfers among the registers in the register file.

3) MixColumn and InvMixColumn: These operations are the most complex operations in this algorithm and their direct implementation needs an 8-b Galois field multiplier, $\mathrm{GF}(2^8)$. However, due to special selection of the parameters of the operation used in the algorithm, the operations can be implemented using shift and XOR operations only. For example, the

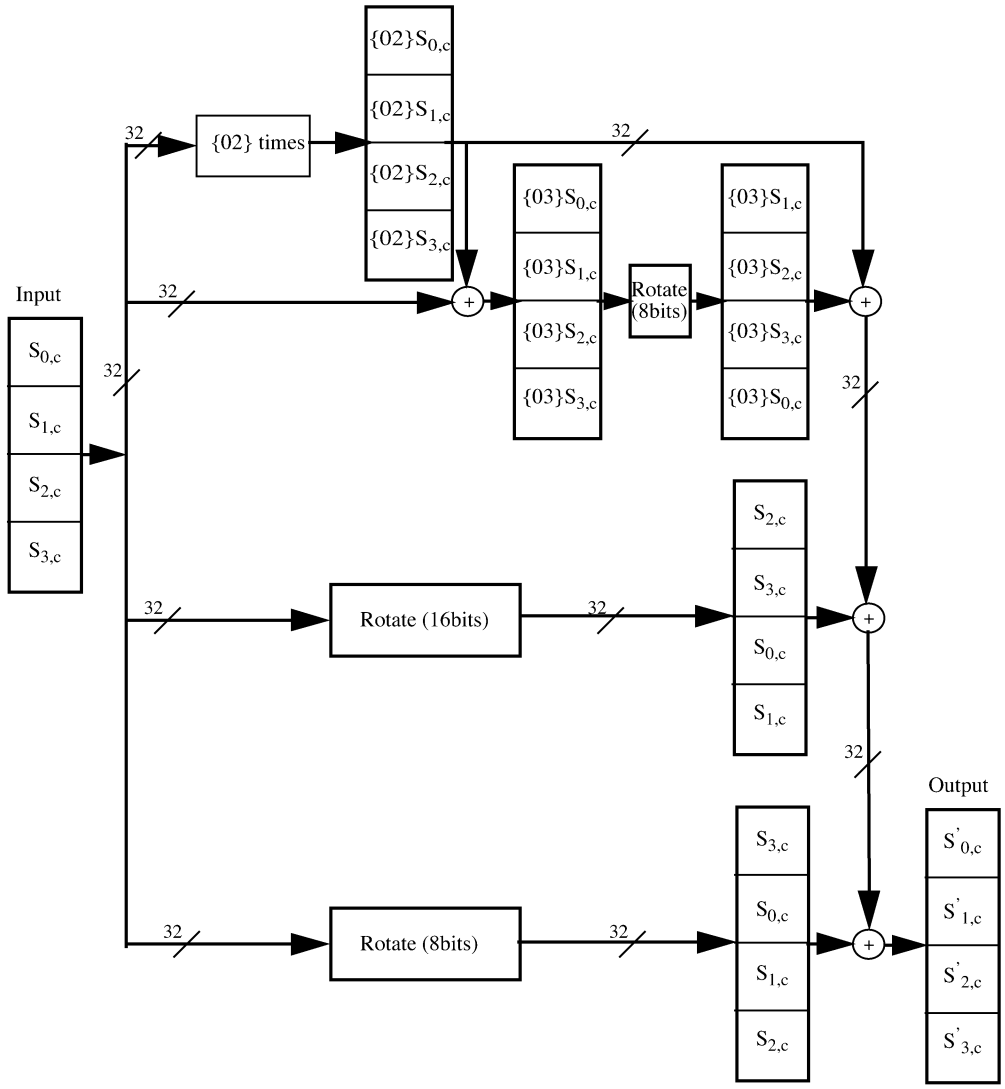


Fig. 10. AES MixColumn implementation on one column.

MixColumn operation on column c (c $= 0, 1, 2,$ and 3) of the state can be represented by the following matrix multiplication:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}.$$

It can be seen that, to evaluate a new column, both $\{02\}$ times and $\{03\}$ times of the column in addition to the original column are required. Note that $\{02\}$ times (or X-times) operation in AES is a special operation which is visualized in Fig. 9. We have implemented the MixColumn operation as depicted in Fig. 10. It can be seen that the microoperations are performed on a column (32 b) and not on a byte at a time which utilizes the hardware more efficiently and preforms the operation faster than a direct byte-by-byte evaluation. Parhi *et al.* [22] have also proposed a method for sharing the required hardware for the evaluation of each element $S'_{i,c}$ when each element is evaluated separately. In our implementation, however, we are using a fixed hardware that is optimized to keep the overall area of the design small. The InvMixColumn is also implemented using the same approach but takes longer because we need to evaluate $\{09\}$, $\{0b\}$, $\{0d\}$, and $\{0e\}$ (all values in Hexadecimal) times each column to evaluate the new state column. Each of these terms is evaluated in a systematic manner using the $\{02\}$ times and XOR operations. For example, $\{0b\}$ times any operand a is calculated, using the expansion $\{0b\} = \{08\} \oplus \{02\} \oplus \{01\} = (\{02\}(\{02\}(\{02\}))) \oplus \{02\} \oplus \{01\}$, as $\{0b\}a = (\{02\}(\{02\}(\{02\}a))) \oplus \{02\}a \oplus a$, where $\oplus$ represents the XOR operation.

C. ECC

The point multiplication $Q = K \cdot P$ in ECC is the basic operation of this algorithm, and, instead of k times the addition of the point P to itself, it is more efficient to use the binary expansion of k and use a set of doubling and addition to find Q, as presented in [11]. The point-doubling operation $(Q = Q+Q)$ is carried out by the point-doubling microcode and the point addition operation $(Q = P + Q)$ is performed by the point addition microcode. Both of these microcodes use field multiplication and inversion subroutine microcodes. Field addition and doubling is cheap (XOR and Circular shift left) and is done in the point multiplication main routine. It should be emphasized that,

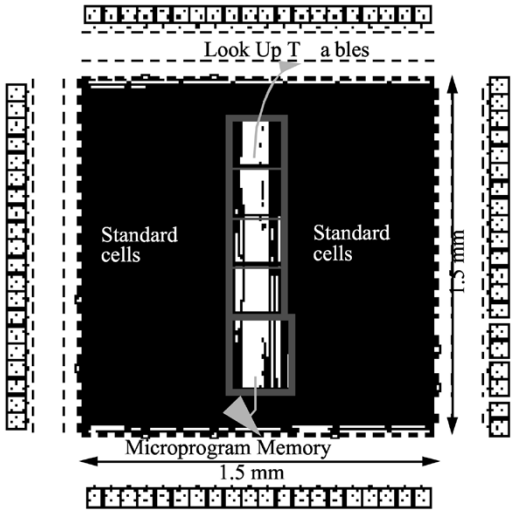


Fig. 11. Cryptoprocessor chip layout—SRAM version. (TSMC 0.18-μm CMOS).

TABLE VIII
CRYPTOPROCESSOR CORE POWER CONSUMPTION ESTIMATES PER ALGORITHM (POST-LAYOUT @VDD = 1.8 V)

	Core power Consumption (mW) @fCLK=13.56MHz	Core power Consumption (mW) for 847.5 Kbps throughput @fCLK<13.56MHz
DES	15.9	3.85
AES	16.3	7.55 (Encryption), 16.3 (Decryption)
ECC	18.3	not applicable

since all of the computations are performed in a single basis (ONB), there is no need for the basis conversion, and, hence, no basis conversion overhead is associated with this design.

VI. AREA AND POWER REQUIREMENTS

We used the Verilog Hardware Description Language to realize the cryptoprocessor and Synopsys to synthesize the RTL code using TSMC 0.18-μm CMOS standard cell library and SRAM memory blocks. In addition to five memory blocks (Microprogram+4 LUTs), 27 874 standard cells and 18 350 nets are present in the design. Cadence First Encounter was then used to place and route the layout of the cryptoprocessor, resulting in the layout shown in Fig. 11 with a core area of 2.25 mm^2 (1.5 mm $\times$ 1.5 mm), which is 9% of total available chip area.

The estimated core power consumption of the design for different algorithms is presented in Table VIII for a 1.8-V power supply and 13.56-MHz clock frequency. The last column in this table presents the core power consumption at a reduced clock frequency (lower than 13.56 MHz) that provides the required throughput of 847.5 kb/s for contactless smart cards. The power consumption is estimated by the Synopsys Power Analysis tool, using the postlayout netlist of the cryptoprocessor along with the node activity data for each algorithm. The power consumption can be further reduced by running the processor at lower voltages than the nominal voltage of 1.8 V (as long as the speed and throughput requirements are satisfied). In Section VIII, we will discuss how the power consumption can be further reduced in future versions of the cryptoprocessor.

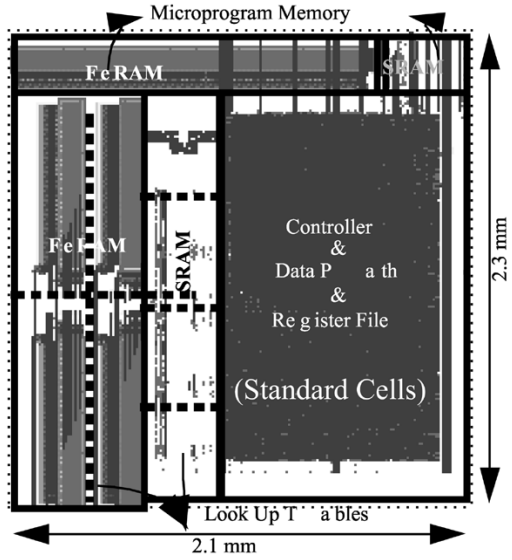


Fig. 12. Cryptoprocessor chip layout—FeRAM version. (Fujitsu 0.18-μm CMOS + 0.35 μm Ferro).

To obtain an estimate of the chip area using FeRAM technology, the layout of the same design using Fujitsu's 0.18-μm CMOS + 0.35 μm Ferro process with FeRAM as the main memory and SRAM as shadow back-up was generated with the same set of tools. The resulting layout is shown in Fig. 12. The core area for this implementation is 4.83 mm^2 (2.1 mm $\times$ 2.3 mm), which is 19.32% of total available chip area. Compiled FeRAM memory modules were not available at the time of developing this layout, and hence experimental FeRAM blocks which contain test structures and SRAM blocks were used instead. It is possible to further reduce the core area of the FeRAM implementation to less than 2.5 mm^2 (less than 10% of total chip area) for more area-constrained applications by removing the SRAM blocks and replacing the present FeRAM blocks with production FeRAM blocks. As mentioned before, the FeRAM implementation of the cryptoprocessor enjoys from the nonvolatile configuration storage. For both implementations, a serialization of the operation of the main processor and the cryptoprocessor can satisfy the power requirements. Obviously, a power-optimized version of the cryptoprocessor can be run concurrently with the main processor.

VII. DESIGN PERFORMANCE

We used Verilog-XL to simulate both the RTL and gate-level netlists of the designed cryptoprocessor. The simulated performance of the post-layout design is summarized in Table IX (for all three algorithms) and in Fig. 13 (for ECC only). The first two rows of Table IX show that the throughput of encryption and decryption of DES and encryption of AES algorithms are much higher than the maximum data transfer rate for contactless smart cards, at 847.5 kb/s [8]; hence, for more power-restricted applications, the clock frequency can be lowered for these cases to reduce the power consumption. Also, note that,

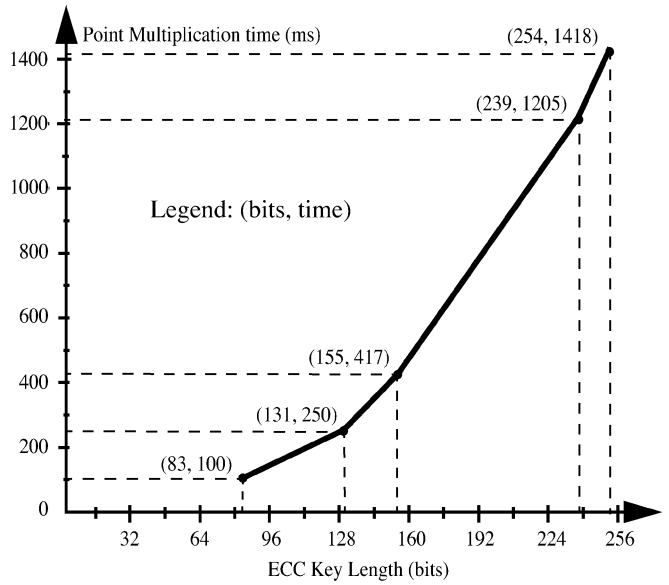


Fig. 13. ECC point-multiplication time for various key lengths.

TABLE IX
THROUGHPUT ESTIMATES FOR THE CRYPTOPROCESSOR OPERATING AT $f_{CLK} = 13.56$ MHz

	Encryption	Decryption
DES	3.50Mb/s	3.50Mb/s
AES	1.83Mb/s	0.85Mb/s
ECC-83	9.92 PM/s	
ECC-83	3.97PM/s	
ECC-83	2.36PM/s	
ECC-83	0.85PM/s	
ECC-83	0.70PM/s	

(PM/s: No. of Point Multiplications per Second)

for ECC, the performance is measured in point multiplication per second (PM/s) rather than bits per second (b/s), because ECC is mainly used for secret key exchange and authentication purposes which both use point multiplication as the basic operation. The ECC performance plot in Fig. 13 shows that highly secure applications (239 254 b ECC) need to spend around 1.5 s for secret key exchange or a point multiplication, while medium (131, 155 b) and low (83 b) security applications can perform the same operations in less than a second. Moreover, it is possible to trade performance with area and power in this implementation. For example, higher performance can be obtained by running the processor at higher frequencies—up to 25 MHz for the current design—(increasing power consumption) and/or using pipelining (increasing area), for more performance-demanding applications. It is worth mentioning that the performance listed in Table IX and Fig. 13 for ECC are based on the worst case scenario mentioned in Section IV-C.

As mentioned in Section I, there is no published work of a single hardware implementation that supports DES, AES, and ECC available for comparison. Table I lists several recent publications of hardware and software implementations of cryptography algorithms contrasted with the implementation described in this paper. Moreover, the original proposal documents of Rijndael to the AES selection committee by the algorithm designers indicates that the fastest implementation of AES-128 on Intel 8051 needs 3168 CPU cycles (each CPU cycle is 12 clock cycles) [33] and 1016 bytes of code, and its implementation on Motorola 68HC08 needs 8390 clock cycles and 919 bytes of memory [33]. Our implementation, however, needs 248 clock cycles and uses 92 bytes of microcode only, which is abut 40 times faster and 10 times more code-size efficient compared to the referenced implementations.

VIII. DISCUSSION

In this section, we address a few extra features that can be added to the design in the future to enhance its capabilities.

First, the design currently uses a 256×32 microprogram memory, while the maximum number of microcode words needed by DES, AES, and ECC are 46, 150, and 60, respectively. Thus, in the current design, the microcode for all three algorithms can be stored in the microprogram memory and the algorithm instantiation is accomplished by changing the LUT contents only. In a more area-limited application, the memory size can be reduced to 150 words ($41\% \cong 0.05$ mm^2 reduction). In this case, the microprogram memory contents, in addition to the LUT contents, need to be updated by the host CPU during an algorithm change.

Second, the ECC multiplication is currently performed in a firmware loop which degrades the performance due to the regular fetching of the loop microcode. A hardware implementation of this instruction can improve the ECC performance by 200%–300%, with negligible area overhead. This can also reduce the power consumption by reducing the memory access rate.

Third, the current implementation of AES decryption is much slower than AES encryption, mainly because of the inverse-key-generation algorithm. This could be greatly improved by reconsidering the inverse-key-generation algorithm and by making the registers in the register file byte-addressable.

Fourth, although the cryptoprocessor is designed to support DES, AES, and ECC only, it provides basic micro-operations that are used in other cryptography algorithms. Therefore, other algorithms such as Serpent (which was also a NIST finalist during AES selection [28]) that use the same basic micro-operations can also be implemented on the same hardware by developing a suitable microcode for each algorithm.

Finally, for the current design, we have been mainly concerned about the feasibility, area requirements, and the performance of the cryptoprocessor, which are important for all types of smart cards, and less concerned about the power consumption, which is a constraint specific to contactless smart cards only. However, we have used several implementation techniques, such as using keeper cells on the bus lines and clock gating for the registers in the register file, to keep the power

consumption low. To reduce power consumption further, the following techniques are available to the designers: 1) reducing the bus wire lengths by using a hierarchical implementation, instead of a flat implementation and 2) using a selective clock signal for the registers to clock the active portion of the registers for cases where operands occupy less than the full register width.

IX. Conclusion

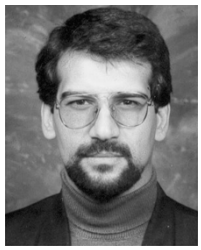
This design presents, for the first time, a universal cryptography processor for smart-card applications that supports both private and public key cryptography algorithms. We achieved this by expressing the primitives of three important algorithms for smart cards (DES, AES, and ECC) in terms of simple logical operations that maximize the number of common blocks among them. This approach resulted in a cryptoprocessor that meets the power consumption and performance specifications of smart cards and occupies 2.25 mm^2 in 0.18-μm CMOS when SRAM memory blocks are used. This area represents just 9% of the maximum available smart-card die area of 25 mm^2. Using FeRAM instead of SRAM memory blocks provides nonvolatile configuration at no extra area overhead.

Acknowledgment

The authors would like to thank the Canadian Microelectronics Corporation (CMC) for technical support and the use of the Advanced Digital Test Collaboratory located at the University of Toronto, Toronto, ON, Canada.

References

[1] *Data Encryption Standard (DES)*, Oct. 1999. Fed. Inf. Process. Standards Pub..
[2] *Advanced Encryption Standard (AES)*, Nov. 2001. Fed. Inf. Process. Standards Pub..
[3] *IEEE Standard Specifications for Public-Key Cryptography*, Jan. 2000.
[4] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Investigation of power analysis attacks on smartcards," in *Proc. USENIX Workshop Smartcards Technology*, Chicago, IL, May 1999, p. 151 and 161.
[5] K. Okeya and K. Sakurai, "A multiple power analysis breaks the advanced version of the randomized addition-subtraction chains countermeasure against side channel attacks," in *Proc. IEEE Inf. Theory Workshop*, 2003, pp. 175–178.
[6] S. B. Ors, F. Gurkaynak, E. Oswald, and B. Preneel, "Power-analysis attack on an ASIC AES implementation," in *Proc. Inf. Technol.: Coding Computing*, vol. 2, 2004, pp. 546–552.
[7] *Smart Cards Standards*, 1995–2004. Int. Standard Org..
[8] *International Standard Organization/International Electrotechnical Commission ISO/IEC 14 443 standard*.
[9] [Online]. Available: http://www.mips.com/ProductCatalog/P_MIPS324KFamily/productBrief
[10] J. Goodman and A. P. Chandrakasan, "An energy-efficient reconfigurable public-key cryptography processor," *IEEE J. Solid-State Circuits*, vol. 36, no. 11, pp. 1808–1820, Nov. 2001.
[11] P. H. W. Leong and I. K. H. Leung, "A microcoded elliptic curve processor using FPGA technology," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 10, no. 5, pp. 550–559, Oct. 2002.
[12] J. H. Kim and D. H. Lee, "A compact finite field processor over GF(2 m) for elliptic curve cryptography," in *Proc. ISCAS*, vol. 2, pp. 340–343.
[13] S. Masui, T. Ninomiya, M. Oura, W. Yokozeki, K. Mukaida, and S. Kawashima, "A ferroelectric memory-based secure dynamically programmable gate array," *IEEE J. Solid-State Circuits*, vol. 38, no. 5, pp. 715–725, May 2003.
[14] I. Verbauwhede, P. Schaumont, and H. Kuo, "Design and performance testing of a 2.29-GB/s rijndael processor," *IEEE J. Solid-State Circuits*, vol. 38, no. 3, pp. 569–572, Mar. 2003.
[15] S. S. Raghuram and C. Chakrabarti, "A programmable processor for cryptography," in *Proc. ISCAS*, pp. V-685–V-688.
[16] K. Eguro and S. Hauck, "Issues and approaches to coarse-grain reconfigurable architecture development," in *Proc. 11th Annu. IEEE Symp. Field-Programmable Custom Computing Machines*, 2003, pp. 111–120.
[17] M. Rosing, *Implementing Elliptic Curve Cryptography*. Greenwich, CT: Manning, 1999.
[18] I. Blake *et al.*, *Elliptic Curves in Cryptography*. Cambridge, U.K.: Cambridge Univ. Press, 1999.
[19] A. Menesez, *Elliptic Curve Public Key Cryptosystems*. Dordrecht, The Netherlands: Kluwer, 1993, ch. 6, pp. 83–99.
[20] N. Biggs, *Discrete Mathematics*. Oxford, U.K.: Oxford Univ. Press, 2002.
[21] M. J. Wiener, "Efficient DES key search—An update," *RSA Labs. Cryptosyst.*, vol. 3, no. 2, pp. 6–8, 1997.
[22] X. Zhang and K. K. Parhi, "Implementation approaches for the advanced encryption standard algorithm," *IEEE Circuits Syst. Mag.*, vol. 2, no. 4, pp. 24–46, Apr. 2002.
[23] G.V.S. Raju and R. Akbani, "Elliptic curve cryptosystem and its applications," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, vol. 2, 2003, pp. 1540–1543.
[24] H. Eberle, N. Gura, and S. Chang-Shantz, "A cryptographic processor for arbitrary elliptic curves over GF(2 m)," in *Proc. IEEE Int. Conf. Application-Specific Syst., Architectures, Processors*, 2003, pp. 444–454.
[25] V. C. Hamacher, Z. G. Vranesic, and S. G. Zaky, *Computer Organization*, 5th ed. New York: McGraw-Hill, 1996, pp. 429–445.
[26] D. A. Patterson and J. L. Hennessy, *Computer Organization & Design—The Hardware/Software Interface*, 2nd ed. San Mateo, CA: Morgan Kaufmann, 1998, pp. 336–432.
[27] Y. Eslami, A. Sheikholeslami, S. Masui, T. Endo, and S. Kawashima, "Circuit implementations of the differential capacitance read scheme (DCRS) for ferroelectric random-access memories (FeRAM)," *IEEE J. Solid-State Circuits*, vol. 39, no. 11, pp. 2024–2031, Nov. 2004.
[28] [Online]. Available: http://www.cl.cam.ac.uk/~rja14/serpent.html
[29] N. S. Kim, T. Mudge, and R. Brown, "A 2.3 Gb/s fully integrated and synthesizable aes rijndael core," in *Proc. IEEE Custom Integrated Circuits Conf.*, 2003, pp. 193–196.
[30] M. Aydos, T. Yanik, and C. K. Koc, "High-speed implementation of an ECC-based wireless authentication protocol on an ARM microprocessor," *Proc. IEE Commun.*, vol. 148, no. 5, pp. 273–279, Oct. 2001.
[31] E. Trichina, M. Bucci, D. De Seta, and R. Luzzi, "Supplemental cryptographic hardware for smart cards," *IEEE Micro*, pp. 26–35, Nov.–Dec. 2001.
[32] J. H. Han, Y. J. Kim, S. I. Jun, K. I. Chung, and C. H. Seo, "Implementation of ECC/ECDSA cryptography algorithms based on java card," in *Proc. 22nd Int. Conf. Distrib. Comput. Syst. Workshop*, 2002.
[33] *Document Version*2. 03/09/99.



Yadollah Eslami (S'00–M'05) received the B.Sc. degree in electrical engineering from Shiraz University, Shiraz, Iran, in 1985, the M.Sc. degree in communication systems from Isfahan University of Technology, Isfahan, Iran, in 1987, and the Ph.D. degree in electrical and computer engineering from the University of Toronto, Toronto, ON, Canada, in 2005.

He was a Lecturer with the Electrical and Computer Engineering Department, Isfahan University of Technology, from 1987 to 1999. He is currently a Design Engineer with the Department of DRAM Research and Development, Micron Technology Inc., Boise, ID. His research interests are in the areas of VLSI memories, VLSI implementation of cryptography algorithms, and microprocessor architecture.

Dr. Eslami was the recipient of the Ontario Graduate Scholarship in Science and Technology, the University of Toronto Open Fellowship, and the Edwards S. Rogers Sr. Scholarship from 1999 to 2005.

Ali Sheikholeslami (S'98–M'99–SM'02) received the B.Sc. degree from Shiraz University, Shiraz, Iran, in 1990 and the M.A.Sc. and Ph.D. degrees from the University of Toronto, Toronto, ON, Canada, in 1994 and 1999, respectively, all in electrical and computer engineering.

In 1999, he joined the Department of Electrical and Computer Engineering, University of Toronto, where he is currently an Associate Professor. His research interests are in the areas of analog and digital integrated circuits, high-speed signaling, VLSI memory design (including SRAM, DRAM, and CAM), and ferroelectric memories. He has collaborated with industry on various VLSI design projects in the past few years, including work with Nortel, Canada, in 1994, with Mosaid, Canada, since 1996, and with Fujitsu Laboratories, Japan, since 1998. He is currently spending the first half of his sabbatical year with Fujitsu Laboratories, Kawaski, Japan. He presently supervises three active research groups in the areas of ferroelectric memory, content-addressable memory (CAM), and high-speed signaling. He has coauthored several journal and conference papers (in all three areas), in addition to two U.S. patents on CAM and one U.S. patent on ferroelectric memories.

Dr. Sheikholeslami has served on the Memory Subcommittee of the IEEE International Solid-State Circuits Conference (ISSCC) from 2001 to 2004 and on the Technology Directions Subcommittee of the same conference from 2002 to 2005. He presented a tutorial on ferroelectric memory design at the ISSCC 2002. He was the Program Chair for the 34th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2004) held in Toronto, Toronto, ON, Canada. He is a Registered Professional Engineer in the Province of Ontario, Canada. He was the recipient of the Best Professor of the Year Award in 2000, 2002, and 2005 by the popular vote of the undergraduate students in the Department of Electrical and Computer Engineering, University of Toronto.

P. Glenn Gulak (S'82–M'83–SM'96) received the Ph.D. degree from the University of Manitoba, Winnipeg, MB, Canada.

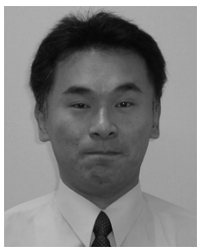
While at the University of Mannitoba, he held a Natural Sciences and Engineering Research Council of Canada Postgraduate Scholarship. He is a Professor with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada. His present research interests are in the areas of algorithms, circuits, and system-on-chip architectures for digital communications. He has authored or coauthored more than 100 publications in refereed journal and refereed conference proceedings. In addition, he has received numerous teaching awards for undergraduate courses taught in both the Department of Computer Science and the Department of Electrical and Computer Engineering at the University of Toronto. He held the L. Lau Chair in Electrical and Computer Engineering for the five-year period 1999–2004. He currently holds the Canada Research Chair in Signal Processing Systems. From January 1985 to January 1988, he was a Research Associate with the Information Systems Laboratory and the Computer Systems Laboratory, Stanford University, Stanford, CA. From March 2001 to March 2003, he was the Chief Technical Officer and Senior Vice President of LSI Engineering, a fabless semiconductor startup headquartered in Irvine, CA, with $85M USD of financing that focused on wireline and wireless communication ICs.

Dr. Gulak served on the ISSCC Signal Processing Technical Subcommittee from 1990 to 1999, was ISSCC Technical Vice-Chair in 2000, and served as the Technical Program Chair for ISSCC 2001. He was the recipient of the IEEE Millenium Medal in 2001.

Shoichi Masui (M'90) received the B.S. and M.S. degrees from Nagoya University, Nagoya, Japan in 1982 and 1984, respectively.

From 1984 to 1999, he was with Nippon Steel Corporation, Sagamihara, Japan, where he was engaged in research on SOI devices, nonvolatile memory circuit design, and its application to radio frequency identification (RFID) ICs. From 1990 to 1992, he was a Visiting Scholar with Stanford University, Stanford CA, where he was involved with research on substrate-coupling noise in mixed-signal ICs. In 1999, he joined Fujitsu Ltd., and, since 2000, he has been with Fujitsu Laboratories Ltd., Kawasaki, Japan, where he is currently a Research Fellow engaged in design of ferroelectric random access memory (FeRAM) for smart cards, RFIDs, and reconfigurable logic LSIs. In 2001, he was a Visiting Scholar with the University of Toronto, Toronto, ON, Canada, where he researched FeRAM design and its application to reconfigurable logic LSIs.

Mr. Masui was the recipient of a commendation by the Minister of Education, Culture, Sports, Science, and Technology, Japan, in 2004 for his research achievements on FeRAM.



Kenji Mukaida received the B.S. degree from the Science University of Tokyo, Tokyo, Japan, in 1990.

From 1990 to 1999, he was with Nippon Steel Corporation, Sagamihara, Japan, where he was engaged in the design of logic LSIs and computer systems. In 1999, he joined Fujitsu Laboratories Ltd., Kawaski, Japan, where, since 2001, he has been engaged in the design of reconfigurable logic LSIs.