# DESIGN AND IMPLEMENTATION OF A PRIVATE AND PUBLIC KEY CRYPTO PROCESSOR FOR NEXT-GENERATION IT SECURITY APPLICATIONS

***Mohamed Khalil Hani, Hau Yuan Wen, Arul Paniandi***
VLSI-ECAD Research Laboratory
Faculty of Electrical Engineering
Universiti Teknologi Malaysia
81310 Skudai, Johor
Malaysia
Email: khalil@fke.utm.my

***ABSTRACT***

*The growing problem of breaches in information security in recent years has created a demand for earnest efforts towards ensuring security in electronic systems. The successful deployment of these electronic systems for e-commerce, Internet banking, government online services, VPNs, mobile commerce, Public Key Infrastructure (PKI), etc., is dependent on the effectiveness of the security solutions. These security concerns are further compounded when resource-constrained environments and real-time speed requirements have to be considered in next-generation applications. Consequently, these IT security issues have been a subject of intensive research in areas of computing, networking and cryptography these last few years. This paper presents the design and implementation of a crypto processor, a special-purpose embedded system optimized for the execution of cryptographic algorithms in hardware. This cryptosystem can be used in wide range of electronic devices, which include PCs, PDAs, wireless handsets, smart cards, hardware security modules, network appliances, such as routers, gateways, firewalls, storage and web servers. The proposed crypto processor consists of a 32-bit RISC processor block and several IP cores that accelerates private and public key crypto computations, LZSS data compression, SHA-1 hashing, and wide-operand modular arithmetic computation. These dedicated crypto IP cores, which are implemented as coprocessors, permit high-speed execution of the compute-intensive operations in AES encryption, ECC and RSA-based digital signature, and other PKI-enabling functions. The proposed embedded system is designed using SoC technology, with hardware described in VHDL and the embedded software coded in C. The resulting cryptohardware is implemented into a single Altera Stratix FPGA microchip. The operating system frequency is set to 40 MHz. A demonstration application prototype in the form of a real-time secure e-document application has been developed to verify the functionality and validate the embedded system.*

***Keywords: Embedded system, Cryptography, Data Security, AES, RSA, ECC.***

## 1.0 INTRODUCTION

Nowadays, it is difficult to open a newspaper, watch a television program, or even have a conversation without some mention of the Internet, e-commerce, smart cards, m-commerce and government online systems. The rapid progress in wireless communication system, personal communication system, and smart card technology in our society makes information more vulnerable to abuse - the content of the communication may be exposed to an eavesdropper, or system services can be used fraudulently. It is critical that these information systems are made secure before these systems are deployed extensively in society. These security concerns are further compounded when next-generation system requirements of resource-constraints and real-time speed are taken into account. Consequently, in the last few years, these IT security issues have been a subject of intensive research in areas of computing, networking and cryptography.

Cryptography offers a robust solution for IT security by providing security services in terms of confidentiality, data integrity, authenticity and non-repudiation. These services form the core operations in Public Key Infrastructure (PKI), which is an essential framework for managing digital certificates and encryption keys for people, programs and systems [1] [2]. PKI-enabling functions are required in secure electronic systems applied in e-commerce, e-health systems, e-government (e.g. *MyKad)* and secure military communications. The crypto processor proposed in this work supports comprehensively these functions, which include 128-bit AES, 163-bit ECC, 1024-bit RSA, LZSS data compression, SHA-1 hashing, wide-operand modular arithmetic. AES (Advanced Encryption Standard) is replacing DES and 3DES as the standard for private key cryptography. RSA (Rivest, Shamir & Adleman is currently the legacy and widely installed public key cryptography, which will migrate to ECC (Elliptive Curve Cryptography) in next-generation security devices [3].

Cryptographic (crypto) algorithms can be implemented either in hardware or software. It is relatively easy to implement crypto algorithms completely in software, but due to increasing data rates and complexity of security protocols, such approach is typically too slow for applications such as embedded systems, networks routers, online databases, *etc*. These shortcomings are most felt in systems that need to process a large number of transactions at very high speed (e.g. network routers, firewalls, web servers, online databases), and systems with resource-constrained environments (e.g. PDAs, cell phones and smart cards). In addition, new techniques for breaking security, such as power analysis and fault analysis, require that the system implementation itself be secure even when it is physical accessed by malicious entities. Resistance to these attacks can be ensured if suitable tamper-proof or clone-free features are built in the designs.

To achieve optimal system performance while maintaining physical security, it is desirable to implement crypto algorithms in hardware. Hardware-based solutions add specific custom crypto hardware components in order to offload time-consuming computations and to reduce bottleneck. While custom hardware techniques address the security-processing gap to some extent, their implementations are expensive and their flexibility and ability to handle evolving standards is limited. Hence the emerging trend is to design hardware/software platforms, utilizing a combination of programmable embedded components and hardware for security processing. The solution implements only specific parts of the security protocol, that is, the time-consuming crypto algorithms in hardware. This leaves the book-keeping, packet processing, control functionality and other non-computation-intensive processing to be performed by the system's host processor.

Today's systems, particularly those networked, also need to be flexible if they are to be commercially viable. In this regard, a desirable feature in most modern security protocol is algorithm agility required to support algorithm independent protocol. Hence it is desirable to implement crypto algorithms within reconfigurable devices such as Field Programmable Gate Arrays (FPGAs). Besides providing for dynamic system evolution, FPGA architectures also allow complex arithmetic operations that are not suited to general purpose CPUs to be implemented more efficiently [4]. They also offer a more cost-effective solution than traditional VLSI/ASIC hardware, which has a much longer design lead-time. The fast prototyping development time of an FPGA design allows modifications to be implemented with relative ease [5]. System-on-chip (SoC) design methodology is employed in the hardware design of the proposed cryptosystem. SoC is designed as a programmable platform that integrates most of the functions of the end product into a single chip. It incorporates at least one processing element (microprocessor, DSP, etc.) that executes the system embedded software. The SoC system employs a bus-based architecture, which also includes peripherals, random logic and interfaces to the outside world. The proposed embedded system is prototyped on an Altera Nios Prototyping Board containing a Stratix EP1S40F780C5 FPGA chip. To the authors' knowledge, the design and single-chip implementation of the combination of the cryptographic algorithms and hybrid systems considered in this paper have not been reported in existing literature.

This paper is organized as follows. In Section 2, the hardware architecture of the cryptosystem is presented. This is followed by a discussion on an application demonstration prototype of a secure e-document transfer via Internet, provided in Section 3. The design of each crypto co-processor is described in Section 4, which is followed by a discussion of the FPGA implementation and performance evaluation of the proposed crypto processors. Finally, the conclusion and recommendations for future work are presented in Section 6.

## 2.0 THE EMBEDDED CRYPTOSYSTEM ARCHITECTURE

The block diagram of our embedded cryptosystem is shown in Fig. 1. This single chip cryptosystem has a RS-232 UART serial interface logic, on-chip memory, timer, PIO (parallel input/output), bus and interface logic to communicate with memory external to the FPGA chip. The crypto co-processors for the ECC, SHA, RSA, AES, LZSS and a modular arithmetic processor are also configured in the same microchip. The proposed crypto embedded system consists of three main components: (1) Hardware processing blocks, (2) Device Drivers, and (3) Application Programming Interface (API).

The Hardware Processing Block consists of a Control CPU and six IP cores (coprocessors) that perform the 128-bit AES symmetric cipher, 1024-bit RSA and 163-bit ECC public key cryptography, SHA-1 algorithm, LZSS data compression and 163-bit Modular Arithmetic Processing (MAP) unit. The dedicated hardware accelerators result in high-speed execution of the crypto algorithms. The embedded control CPU is the Altera Nios processor. This processor is a configurable, 5-stage pipelined, single-issue RISC processor, in which instructions run in a single clock cycle. It controls the dedicated crypto IP cores such that the embedded system may perform cryptographic schemes such as private key encryption, public key digital signature, mutual authentication, key management, etc.

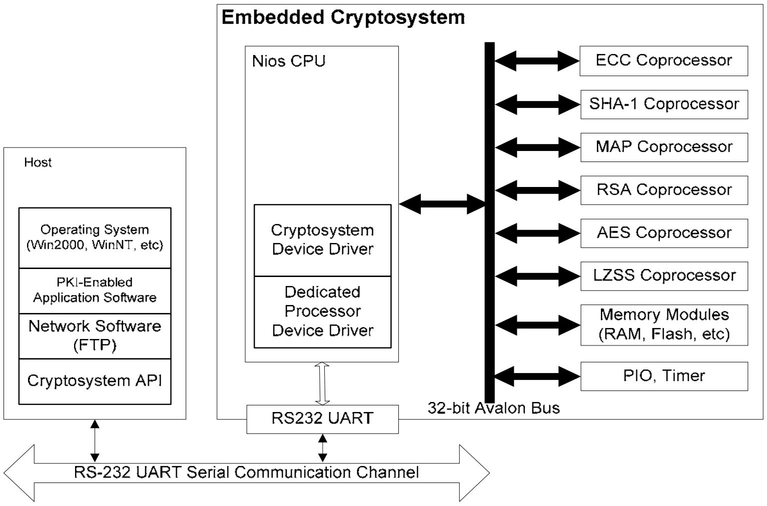

Embedded Cryptosystem
Nios CPU
Host
Operating System (Win2000, WinNT, etc)
PKI-Enabled Application Software
Network Software (FTP)
Cryptosystem API
Cryptosystem Device Driver
Dedicated Processor Device Driver
ECC Coprocessor
SHA-1 Coprocessor
MAP Coprocessor
RSA Coprocessor
AES Coprocessor
LZSS Coprocessor
Memory Modules (RAM, Flash, etc)
PIO, Timer
RS232 UART
32-bit Avalon Bus
RS-232 UART Serial Communication Channel

Fig. 1: Block diagram of the proposed crypto embedded system

The device drivers are embedded software executed on the Nios embedded processor. It is the main controller ensuring the correct execution of the IP cores, and acts as a bridge between the crypto coprocessors and the APIs on the host PC. The APIs are executed on the host PC. The APIs perform high-level functions such as input file reading and output file writing. It sends data to and receives output between the host PC and the subsystem hardware via a UART serial communication channel. These APIs facilitate software development by application developers.

For an easier comprehension of the design proposed in this paper, we begin, in the following section, with an overview of the secure e-document demonstration application prototype implemented in the proposed cryptosystem. In this way, the security schemes implemented are known at the outset, from which the functionality and designs of the various crypto co-processors can be explained more conveniently. The designs of these crypto co-processors are detailed out in Section 4.

## 3.0 APPLICATION: SECURE e-DOCUMENT TRANSFER

We have developed a real-time data security application, that is, an e-document system for the application of secure document transfer via Internet, which is an insecure communication medium. This demonstration application prototype is utilized to evaluate the functionality of the embedded cryptosystem and the reusability of the associated APIs and device drivers. In this application, sensitive documents, which are transferred electronically via FTP mechanism in a Local Area Network (LAN) environment, are made secure by encrypting and signing in real-time, using the proposed crypto hardware.

Fig. 2 shows GUI of the control and monitoring PC front-end for our e-document security application (written in Visual Basic). There are two processes involved, namely (a) sender file uploading, and (b) receiver file download. The block diagrams for the security schemes at the sender and receiver ends are given in Fig. 3(a) and 3(b) respectively. In the sending process, a document goes through operations depicted in the security scheme given in Fig. 3(a), that is:

1. PRNG (pseudo random number generation) module generates a session key for AES encryption.
2. Using this AES session key, the AES encryptor encrypts the document to produce ciphertext.
3. RSA encryptor encrypts AES session key using the receiver RSA public key.
4. SHA Hashing module generates a message digest of the document.
5. ECDSA signing module computes the corresponding digital signature using sender ECC private key.

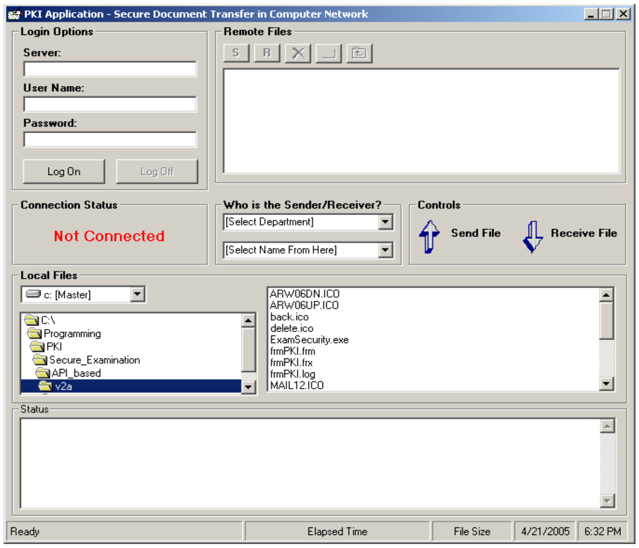


Fig. 2: The GUI of the e-document security system

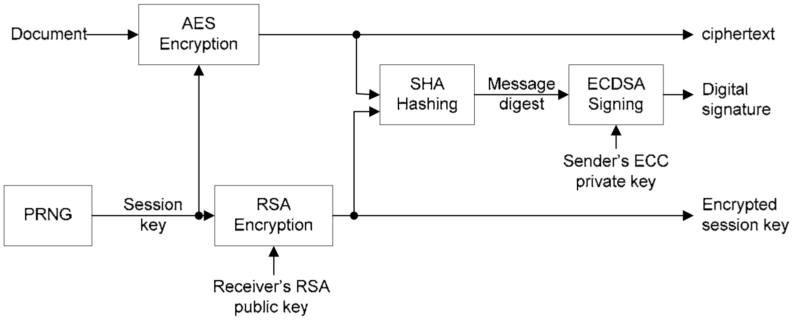


Fig. 3(a): Security scheme for Sender File Upload

In the receiving process, the now secured document undergoes the recovery process depicted in the security scheme given in Fig. 3(b) to recover the original data, that is:

1. ECDSA verifying module verifies the received digital signature, using sender ECC public key. If this signature verification fails, the receiving process stops here, else continue.
2. Using receiver RSA private key, the RSA Decryptor decrypts the encrypted session key to recover the AES session key.
3. With this key, the AES decryptor decrypts ciphertext to recover original document.

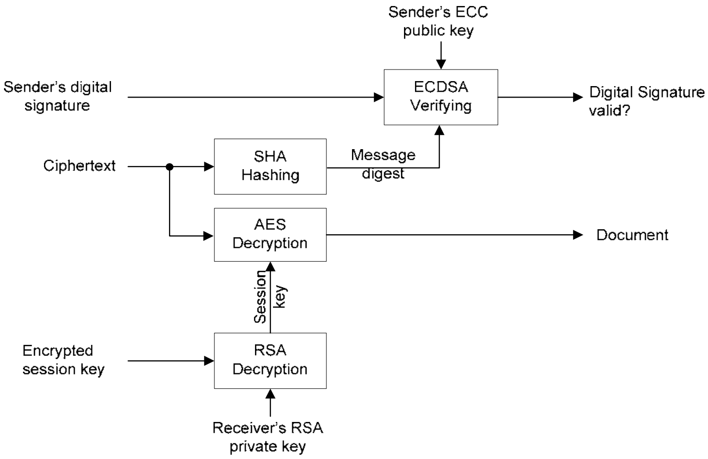
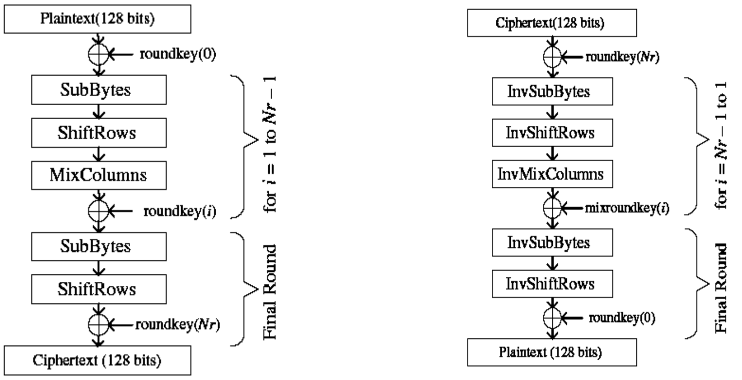


Fig. 3(b): Security scheme for Receiver File Download

## 4.0 THE CRYPTO CO-PROCESSORS

### 4.1 AES Processor for Symmetric Encryption

NIST (National Institute of Standards and Technology, US Government's official standard organization) has now specified AES-Rijndael in the document Federal Information Processing Standard (FIPS) 197, as the new standard for symmetric encryption. AES comes in three security strengths: 128-bits, 192 bits and 256 bits. In this paper, the architecture for 128-bit AES has been designed. The number of rounds depends on both of these parameters and is given in [6]. Therefore, the cipher in all configurations presented operates in $N_r = 10$ rounds.



(a) Encryption (b) Decryption

Fig. 4: Structure of AES algorithm

Fig. 4 shows the encryption and decryption structure of the AES algorithm applied in this work. In encryption, after the initial *roundkey* addition, $N_r$ rounds are performed. The operations are the same in the first $N_r$ -1 rounds, with a small difference in the final round. As shown in Fig. 4(a), each of the first $N_r$-1 rounds consists of four transformations: *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey*. The final round excludes the *MixColumns* transformation. The decryption algorithm uses a different ordering of the inverse forms of the transformations used in the encryption algorithms as shown in Fig. 4(b). The four transformation operations are summarized as follows:

- ***SubBytes***: This transformation is a non-linear byte substitution 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.
- ***ShiftRows***: In this transformation, the rows of the State shift cyclically to the left with different offsets. 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 s four-term polynomial over $GF(2^8)$ and multiplier by $a(x)$ modulo $x^4 + 1$, given by $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + 1$ for encryption and $a(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}\mathrm{x} + \{0E\}$ for decryption process.
- ***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 model given in Fig. 4(b) is modified from the original described in the NIST standard. It has been rearranged with some changes to the key generation unit to obtain a structure similar to the encryption model. This modification is based on the properties of the AES algorithm, which is:

- *InvShiftRows* transformation immediately followed by an *InvSubBytes* transformation is equivalent to *InvSubBytes* transformation immediately followed by an *InvShiftRows* transformation.
- *InvMixColumns* transformation is linear, which means:
  *InvMixColumns*(State XOR *roundkey*) = *InvMixColumns*(State) XOR *InvMixColumns*(*roundkey*)

Fig. 5 shows the hardware architecture of AES crypto processor core designed in this work. It combines encryption and decryption into one block, permitting the sharing of common modules. This crypto processor is designed based on 4-state pipeline so as to increase the maximum running frequency and timing performance. It consists of three main modules, which are KeyGen, Control Unit and Encryption/Decryption Unit. The KeyGen is designed to generate the *roundkey* for *AddRoundKey* transformation in every round. The data path for encryption/decryption is as follows:

$$Pt/Ct \rightarrow m2 \rightarrow ARK \rightarrow M\text{-}SB \rightarrow M\text{-}SR \rightarrow M\text{-}MC \rightarrow m1 \rightarrow m2 \rightarrow ARK \rightarrow Ct/Pt.$$

We implemented the *SubBytes* block (S-box) based on the calculation of multiplicative inverse and affine transform instead of using ROM-based lookup table to reduce the hardware logic count.

M-SB : Mixed SubBytes
M-SR : Mixed ShiftRow
M-MC : Mixed MixColumn
ARK : AddRoundKey
Pt/Ct : Plaintext/Ciphertext
Ct/Pt : Ciphertext/Plaintext
m1 : multiplexer 1
m2 : multiplexer 2

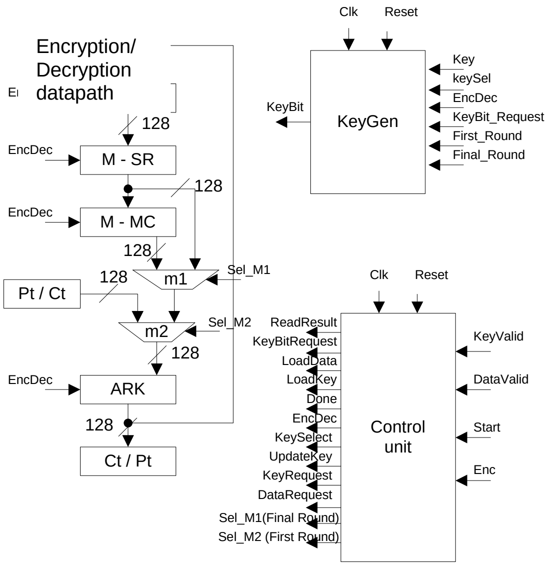


Fig. 5: Block diagram of the AES-128 core

## 4.2 RSA Processor for Public Key Cryptography

The RSA algorithm, invented in 1977 by Rivest, Shamir, and Adleman [7], is currently the most popular public key cryptosystem in use, particularly in high-end commercial software products that are typically employed in e-commerce and VPN servers. It can provide encryption and digital signatures. In hardware implementations, the RSA algorithm can be found in secure telephones, in Ethernet network cards, and smart cards. The main advantage of the algorithm is that it can provide both data confidentiality service (via public-key encryption) and data integrity, authentication and non-repudiation (via digital signatures) using the same key pair and under the same mathematical operation. Its hard problem is based on the large integer factorization problem.

RSA algorithm is the essence of simplicity [8]. To encrypt a message $X$ to its cipher text $P$, we perform $P = X^E$ mod $M$ using the public key ($E, M$). To restore the message, $X= P^D$ mod $M$ is performed, where ($D, M$) is the private key. For digital signature purpose, we use the private key in signing, $S = X^D$ mod $M$. To verify the signature, we use the public key to perform X = $S^E$ mod $M$. Fig. 6 below shows the RSA process. In RSA, whether encrypting, decrypting, signing, or verifying, the operation is basically a wide-operand modular exponentiation. The basic modular exponentiation equation, given by: $Y = X^E \ mod \ M$, essentially consists of thousands of modular multiplication, $A.B \ mod \ M,$ in $GF(p)$. Thus, the modular exponentiation is computation-intensive and requires a long computation time. A proven method to speed up its operation time is to utilize Montgomery modular multiplications, which do not have divisions.

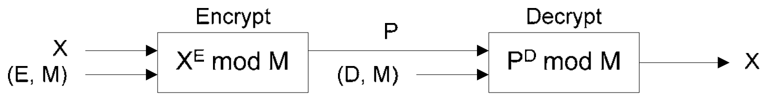


Fig. 6: RSA operation

The Modular Exponentiation algorithm, *ModExp( )*, applied in this work is given in Algorithm 1, Fig. 7 below. It utilizes the Montgomery modular multiplication, *MonMult( ),* with the algorithm given in Fig. 8. For hardware implementation, the multi-precision version of Montgomery multiplication is employed, in which the algorithm assumes that $M$ is an m-digit number in radix-r, and $R = r^m$. In our hardware design, radix-2 is chosen. More elaborate explanations on the derivations of these algorithms can be found in [9] and [10].

```
Algorithm-1:
ModExp($R^2$ mod M, X, E, M)

Compute $P = X^E \bmod M$,
$E = \sum_{i=0}^{n-1} e_i \cdot 2^i$ , $e_i$ € {0, 1}

1.  $P_0$ = MonMult(1, $R^2$ mod $M$, $M$)
    $Z_0$ = MonMult ($X$, $R^2$ mod $M$, $M$)
2.  For $i$ = 0 to $n$-1
2a.   $P_{temp}$ = MonMult($P_i$, $Z_i$, $M$)
      $Z_{i+1}$ = MonMult($Z_i$, $Z_i$, $M$)
2b.   if $e_i = 1$ then $P_{i+1} = P_{temp}$
          else $P_{i+1} = P_1$
3.  End For
4.  P = MonMult ($P_n$, 1, $M$)
```

Fig. 7: Modular Exponentiation algorithm

```
Algorithm-2: MonMult(A, B, M)

Compute $P = A.B.2^{-(m+2)} \bmod M$,
$M = \sum_{i=0}^{m-1} m_i \cdot 2^i$
$B = \sum_{i=0}^{m} b_i \cdot 2^i$ , $B < 2M$
$A = \sum_{i=0}^{m+2} a_i \cdot 2^i$ , $m_i$, $b_i$, $a_i$ € {0, 1},
        $a_{m+1} = a_{m+2} = 0, A < 2M$

1.  $P_0 = 0$
2.  For $i$ = 0 to $m$+2 Loop
2a.   $q_i = p_{i,0}$
2b.   $P_{i+1} = (P_i + q_i \cdot M)/2 + a_i \cdot B$
3.  End Loop
4.  Return $P = P_{m+3}$
```

Fig. 8: Montgomery Modular Multiply

We now present the design of the modular exponentiation module, which is the core operation in an RSA processor. This design can perform RSA operations with key length up to 1024-bit. The design is described completely in parameterizable VHDL code, such that it is scalable for increase in key length for future stronger key strength. Fig. 9 shows the architecture of RSA crypto processor core (*ModExp)*, which is designed based on Montgomery Modular Exponentiation algorithm given in Fig. 7. It consists of a Montgomery modular multiplier (*MonMult*), operand RAMs, multiplexer, 1-bit to 32-bit shift register (*MM2ME32*) and a state machine (*SM_ModExp*). All the modules are synchronously clocked. Five RAMs are needed to store the operands $R^2$ mod $M$, $X$, $E$, $M$ and a intermediate modular multiplication result $P_{i+1}$ in a Algorithm 1 step 2b. A 32-bit shift register, *MM2ME32*, is used to collect the modular exponentiation result bit stream in 32-bit packets. *SM_ModExp* is the controller that governs the signal's flow between modules.

The *MonMult* module performs the Montgomery modular multiplication based on algorithm shown in Fig. 8. The design is implemented as a one-dimensional two-way systolic array for high-speed operation. The systolic array consists of several processing elements (PE) and each PE computes successive values for a single bit position. The PEs are pipelined so that the data flow from the right cell to the left cell and each cell takes a clock cycle to process. Detailed architecture design and hardware mapping can be found in [10].

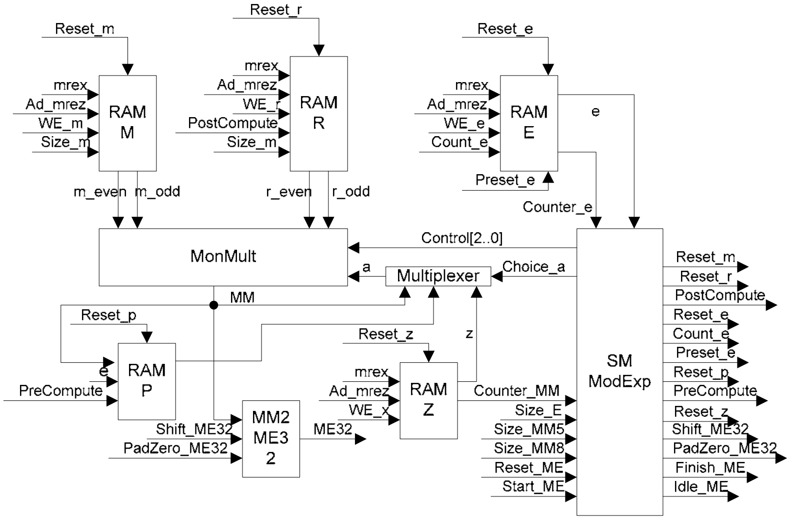


Fig. 9: *Modular Exponentiation* architecture for RSA

In RSA, each user has a pair of keys, i.e. public key *(E, M)* and private key *(D, M)*. *M* is the modulus, *E* is the public exponent and *D* is the private exponent. RSA key length refers to the bit length of the modulus. These keys are generated using the algorithm as shown in Fig. 10. *p* and *q* must be kept secret or deleted. (*gcd* stands for greatest common divisor. *D* is the multiplicative inverse of *E* modulo *M*.)

1. Generate 2 primes, $p$ and $q$ randomly.
2. Calculate $M = p.q$, and $\Phi(M) = (p-1).(q-1)$.
3. Generate E that fulfills both the conditions $1 < E < \Phi(\mathrm{M})$, and $gcd\ (\Phi(M)\ , E) = 1$.
4. Calculate $D = E^{-1}\ mod\ \Phi(M)$.

Fig. 10: RSA key pair generation algorithm

### 4.3 ECC Processor for Public Key Cryptography

N. Koblitz and V. Miller first suggested Elliptic Curve Cryptography (ECC) in 1985 [1]. The main strength of ECC rests on the concept of solving discrete logarithm problem (DLP) over the points on an elliptic curve. This provides for higher strength-per-bit than any other public key cryptosystem, implying that significantly smaller parameters can be used in ECC as compared to other competitive systems, but with equivalent levels of security. For example, the security level of a 160-bit ECC key is equivalent to 1024-bit RSA [11]. The benefits of having smaller key sizes include faster computations, and reductions in processing power, storage space and bandwidth. This makes ECC ideal for resource-constrained environments such as pagers, PDAs, cellular phones and smart cards.

Fig. 11(a) shows the arithmetic hierarchy of ECC, which indicates that ECC essentially requires the application of two types of mathematics: elliptic curve arithmetic and the underlying finite field arithmetic. ECC-based security protocols such as ECDSA (Elliptive Curve Digital Signature Algorithm), ECDH, and ECES are performed using ECC arithmetic functions at the top level of the ECC arithmetic hierarchy. An example is shown in Fig. 11(b), where ECC point multiplication and point add is performed in ECDSA security scheme. Elliptic curve arithmetic defines the algorithms to perform these point multiplying, point adding, point doubling, and other control/conversion functions. The finite field arithmetic provides basic finite field operations include field inversion, field addition, field multiplication and field squaring.

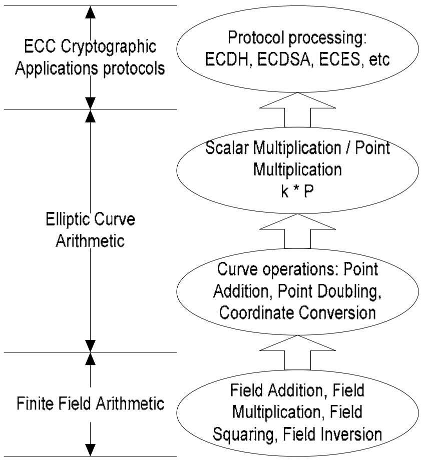

ECC Cryptographic Applications protocols
Protocol processing: ECDH, ECDSA, ECES, etc
Elliptic Curve Arithmetic
Scalar Multiplication / Point Multiplication k * P
Curve operations: Point Addition, Point Doubling, Coordinate Conversion
Finite Field Arithmetic
Field Addition, Field Multiplication, Field Squaring, Field Inversion

Fig. 11(a): The arithmetic hierarchy of ECC

**1.0 ECDSA SIGNATURE VERIFICATION ALGORITHM**

Input: ***G*** (Finite point in domain parameter),
***Q*** (public key of entity A),
***m*** (message),
***(r, s)*** Signature on message of entity A.

Output: Accept or reject signature from A.

1. Verify $r$, $s$ are integers in the interval [1, $n$-1].
2. Perform crypto hashing, $e = SHA\text{-}1\ (m)$.
3. Compute $w = s^{-1}$ **mod** $n$
4. Compute $u_1 = ew$ **mod** $n$ and $u_2 = rw$ **mod** $n$.
5. Perform <u>**ECC point multiply & point add**</u>: $X(x_2, y_2) = u_1G + u_2Q$.
6. If $X = O$, then reject the signature. Otherwise, compute $v = x_2$ **mod** $n$.
7. Accept the signature if and only if $v = r$.

Fig. 11(b) ECDSAsSignature verification

The current version of the proposed cryptosystem applies the ECC domain parameters as follows:

1. Type of underlying finite field: Binary field in GF($2^{163}$).
2. Field representation: Polynomial basis representation.
3. Type of elliptic curve: Over $F_2{}^m$, Koblitz Curve sect163k1 taken from [8].

4. Elliptic curve point representation: Point Multiplication in Projective Coordinates, and Point Addition in Affine Coordinates, supported by trinomial & pentonomial as recommended by IEEE, NIST.

The elliptic curve *point multiplication* is the core operation. In ECC, the fundamental crypto operation is this *point multiplication*, i.e. a point $P_a$ is (point) added to itself $k$ times, over an elliptive curve.

$$Q = kP \quad = P + P + \ldots + P \ (k \text{ times}) \qquad \ldots\ldots(1)$$

An elliptic curve over the finite field $GF(p)$ is defined as the set of points $(x, y)$ which satisfy the operation:

$$y^2 = x^3 + ax + b \qquad \ldots\ldots(2)$$

where $x, y, a$ and $b$ are elements of the field, and $4a^3 + 27b^2 \neq 0$. To encrypt, the data is represented as a point on the chosen curve over the finite field.

This work implements *binary field* $GF(2^{163})$, or so called binary finite field, because of its high efficiency achievable in hardware implementation. Due to the reason that normal basis representation is not the best choice for an elliptic curve processor with flexible finite field support, the polynomial basis representations have been chosen as the basis of binary field arithmetic in this work. The field elements are represented as polynomial basis with coefficient 0 or 1. The design is adapted towards supporting both trinomial and pentanomial as recommended by IEEE and NIST. The projective coordinate system is applied instead of affine coordinate system, and this is to eliminate field inversion in point addition and point doubling. Field inversion is the most complex and expensive operations in terms of processing time and hardware resource. Montgomery point multiplication algorithm in projective coordinate is chosen in this work because it prevents information leakage in the form of power signature differences and offers comparatively better performance. Fig. 12 gives a summary of Montgomery Point Multiplication in projective coordination.

**Algorithm-3: Montgomery Point Multiplication**

```
Input: $k = \sum_{i=0}^{l-1} k_i \cdot 2^i, k_i \in [0,1]$  (multiplier)
        $P(x, y)$ – a random point on an elliptic curve (multiplicand).
Output: $Q = kP = (x_Q, y_Q)$

1.  $(X_1, Z_1, X_2, Z_2)$ = Conv_affine_projective($x, y, b$)
2.  for i = $l$ – 2 down to 0 do
    2.1 if $k_i$ = 1 then
                    $(X_1, Z_1)$ = M_add($X_1, Z_1, X_2, Z_2, x$)
                    $(X_2, Z_2)$ = M_double($X_2, Z_2, b$)
        else
     $(X_2, Z_2)$ = M_add($X_2, Z_2, X_1, Z_1, x$)
     $(X_1, Z_1)$ = M_double ($X_1, Z_1, b$)
    end for
3.  $(x_Q, y_Q)$ = Conv_projective_affine($X_1, Z_1, X_2, Z_2, x, y$)
```

Fig. 12: Montgomery Point Multiplication (Projective Coordinate Version)

Fig. 13 shows the architecture of the control unit and the datapath module of the ECP core. The control unit utilizes a micro sequencer and an iteration counter. Meanwhile, the datapath module performs basic finite field arithmetic operations such as field addition, field multiplication and field squaring. It comprised a register file and an arithmetic unit incorporates a finite field multiplier, a parallel squarer and accumulator and a zero test circuit. For a more detailed description of the algorithm and design of the ECP, please refer to [12] and [13].

To strike a balance between parallel and bit-serial multiplication, this work implements Least Significant Digit First (LSD-First) multiplication to perform finite field multiplication in the ECP design due to its shorter critical path delays. Besides, parallel squaring with fixed irreducible polynomial support together with Itoh-Tsuji inversion are applied in this design for the purpose of simple and efficient hardware implementation of field inversion. The ECP design is described completely in parameterized VHDL code, such that the core is reconfigurable and reusable. In this parameterizable design, the processor can be configured to support any field size and polynomial, as recommend by NIST and Certicom [3]. For working environment with memory, bandwidth, or power dissipation constraint, user can vary the degree of parallelism in the multiplier to achieve desired performance-cost trade-off.

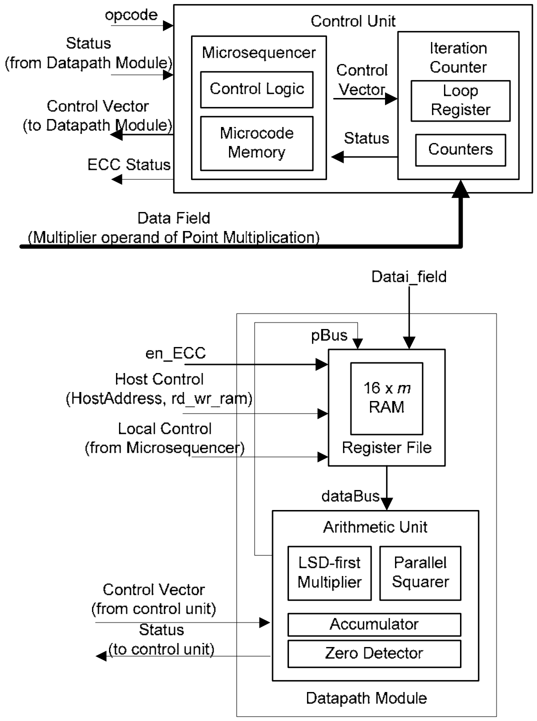


Fig. 13: Architecture of Control Unit and Datapath Module in ECP

### 4.4 SHA Processor for Crypto Hashing

A hash function maps binary strings of arbitrary length to binary strings of some fixed length, called message digest. Hash function with cryptographic properties is called cryptographic hash function. Cryptographic hash function is generally a one-way and collision resistant function [14]. It refers to the function that is relatively easy to compute, but significantly harder to reverse. Collision resistant means that it is computationally infeasible to find different messages with same message digest. The hash function serves to compress the large input data for public-key encryption/signing function.

Among the hashing functions available, the SHA-1 algorithm [15] is preferable. This is because of its balanced security (secure to all known attacks), speed (higher parallelism) and applicability. In SHA-1 algorithm, input message that is less than $2^{64}$ bits is hashed into a 160-bit fixed length message digest. The main operation of SHA-1 algorithm is given in Fig. 14.

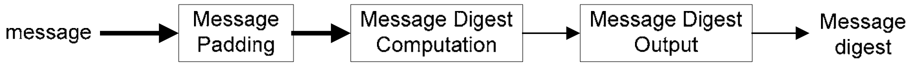


Fig. 14: SHA-1 Message digest computation

SHA-1 hash algorithm consists of three main stages, i.e. padding, message digest computation and message digest output.

- **Padding**: converts input message into block of 512-bit each.
- **Message digest computation**: Initializes message digest buffers, and then loops the SHA-1 compression function for all converted message blocks, such as message expansion, bit-wise logical function, constant multiplexing, etc.
- **Message digest output**: Concatenation of five chaining variables after the message digest computation of the final compression function loop.

Because they are not computationally intensive, the padding stage and message digest output stage are normally performed in software. However, the message computation stage is computation demanding, and therefore should be performed in hardware. In this work, the Bosselears's architecture [16] is modified to exploit parallelism and customization offered by hardware. The modification leads to a systolic array implementation of adder tree in iteration module.

Fig. 15 shows the block diagram of the *SHA1_engine* module that maps the SHA-1 compression function into hardware. The main unit is iteration-step-variable-update unit that initialize, stores and updates the iteration-step-variables. The message expansion unit reads and expands external input message word (*Mt*). The constant multiplexer supplies a constant word for each iteration step. The logical function unit converts three words (*B*, *C*, and *D*) into single word (*Ft*) based on logical functions. The feed forward unit stores and updates the chaining variables, which are finally sent out as the message digests (*MD*). For an elaborate description of the detail design of the proposed SHA-1 hardware core, please refer to [14].

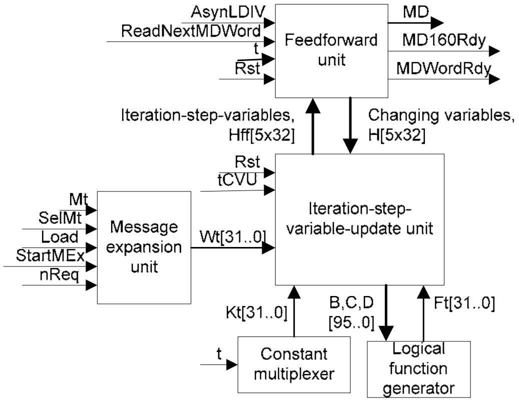


Fig. 15: Datapath unit of SHA1_engine

### 4.5 LZSS Processor for Data Compression

In data communications and data storage system, it is desirable to have faster transfer rates and greater storage capacity at lower costs. Data compression techniques address these demands by removing redundancy from the source data and thereby increase the density of transmitted or stored data. Since cryptographic applications require restored data to be identical to the original, lossless data compression is applied in this work. The design of the compression core proposed in this work is based on the combination of Lempel-Ziv-Storer-Szymanski (LZSS) compression algorithm and Huffman coding. In our design of the data compression processor core, the compression and decompression are performed in separate modules. With a systolic array hardware architecture mapping approach to part of the algorithm, the resulting IP core offers a data-independent throughput that can process a symbol on every clock cycle. The design is described completely in parameterized VHDL code such that it can provide a suitable compromise between the constraint of resource, speed and compression saving, and adaptable for any specific target application. For a more elaborate description of the design, please refer to [17].

Fig. 16(a) shows the block diagram of the Compression Unit of LZSS processor core. It consists of three hierarchical blocks, which are LZSS coder, fixed Huffman coder and data packer. All the modules are synchronously clocked. The LZSS coder performs the LZSS encoding to symbols of source data, which is obtained from the Compression Interface, and the fixed Huffman coder re-encodes the LZSS codeword to achieve a better compression ratio. Finally, the data packer packets the unary codes from the fixed Huffman coder into a fixed-length output packet and sends it to the Compression Interface. Fig. 16(b) shows the block diagram of the Decompression Unit of LZSS processor core.

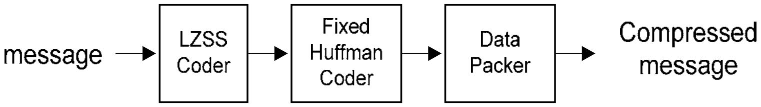


Fig. 16(a): Block diagram of compression unit

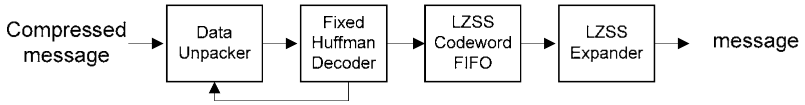


Fig. 16(b): Block diagram of decompression unit

### 4.6 MAP Processor for Wide-Operand Modular Arithmetic

This section presents the design of a Modular Arithmetic Processor (MAP) for the computation, in hardware, of the arithmetic operations defined over a Galois Field *GF(P)*. This processor is designed as a complementary core to enhance the overall timing performance of ECDSA subsystem. The wide-operand (or large integer) modular arithmetic operations were originally performed in embedded software, but its performance is inadequate for our purpose as it suffers from a very long computation time. The underlying modular arithmetic operation of ECDSA scheme requires modular multiplication, division/inversion and addition/reduction. Among these operations, the modular division is the most computation-intensive underlying operation in ECDSA scheme. Besides, division is by far the most costly operation in terms of speed and the speed of the overall processor will depend on a high-speed divider. Therefore, special attention is given to the computation of the modular division, and is now elaborated.

The modular division ($y = a/b \bmod M$) is then integrated with other operations, namely modular multiplication ($y = a*b \bmod M$) and modular addition ($y = a+b \bmod M$) into a complete modular arithmetic unit. This processor is also able to perform modular inversion ($y = 1/b \bmod M$) by letting *a=1 and* using modular division instruction. By letting $a=0$ in modular addition operation*,* it can implement modular reduction ($y = a \bmod M$). All of the other algorithms are carefully chosen so they can share the common hardware resources in the modular divider without any major modification on the hardware architecture. However, to implement the mentioned operations, it needs to be somewhat extended.

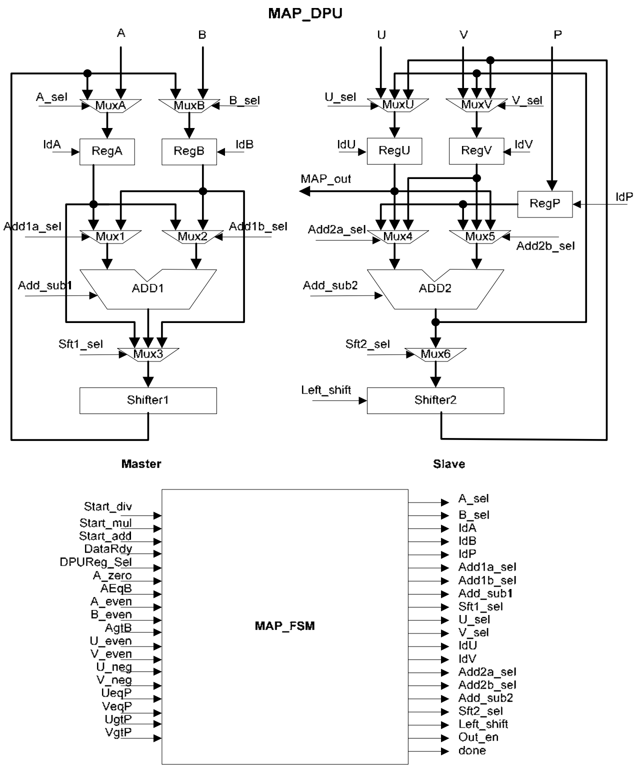


Fig. 17: Hardware architecture of MAP core

The proposed modular divider is designed based on a binary add-and shift algorithm to perform modular division presented by S.C Shantz in [18]. This provides an alternative to division by inversion followed by multiplication. This algorithm is chosen because the most critical operation is shifting and addition/subtraction operation only instead of hardware costly operation such as multiplication and exponentiation operation, resulting in simpler hardware architecture. For modular multiplication, we choose an interleaved algorithm [9], where addition alternates with modular subtraction. This algorithm is simple and straightforward, yet quite efficient. Since it uses standard number representation, it is relatively easy to implement in hardware. Fig. 17 shows the top-level architecture of the MAP core. It consists of *MAP_FSM* and *MAP_DPU*. The *MAP_FSM* works as the control unit to receive the condition signals from and send the control signals to *MAP_DPU* unit. The MAP design is described completely in parameterized VHDL code, such that the core is reconfigurable and reusable for different field sizes. For the prototype presented in this work, it is set to perform 163-bit modular arithmetic operation to fit into the implemented ECDSA scheme.

## 5.0 FPGA IMPLEMENTATION & PERFORMANCE EVALUATION

The design effort of the proposed embedded system involves hardware design, interface design and software development. Hence, it calls for hardware software co-design, in which both hardware and software are designed in tandem to provide overall system functionality. The hardware blocks are described in VHDL (VHSIC Hardware Description Language) using VHDLmg, a design entry tool developed in-house in UTM [19]. Modeling the processor in VHDL facilitates quick prototyping and modification of the target design while considering various possible trade-offs in different implementations of the crypto algorithms with different speed and area characteristics.

Each crypto core is synthesized, simulated, place and routing, and timing analyzed using Quartus II EDA software from Altera. The complete embedded cryptosystem is prototyped in a single Altera Stratix EP1S40F780C5 FPGA chip employing Altera SOPC Builder. Design verification is performed via timing simulations and the complete system is validated through the functional evaluation tests performed on the demonstration application prototype. These tests verified that all the required security services (data confidentiality, data integrity, non-repudiation and authentication) are achieved at an approximate speed of 2.2 kbyte /sec. Due to lack of space in this paper, we provide below only a sample of the design verification of one of key crypto core, and then we present the performance analysis and results of each processor core developed in this work, in turn. Fig. 18 shows a snapshot of the timing simulation output of the modulation exponentiation module in our RSA coprocessor core. The timing diagram shows the behavior the core at the end of operation, which completes with the reading the 32-bit result from RAM module in the core to Avalon Bus of the main Nios processor. This verifies this design.

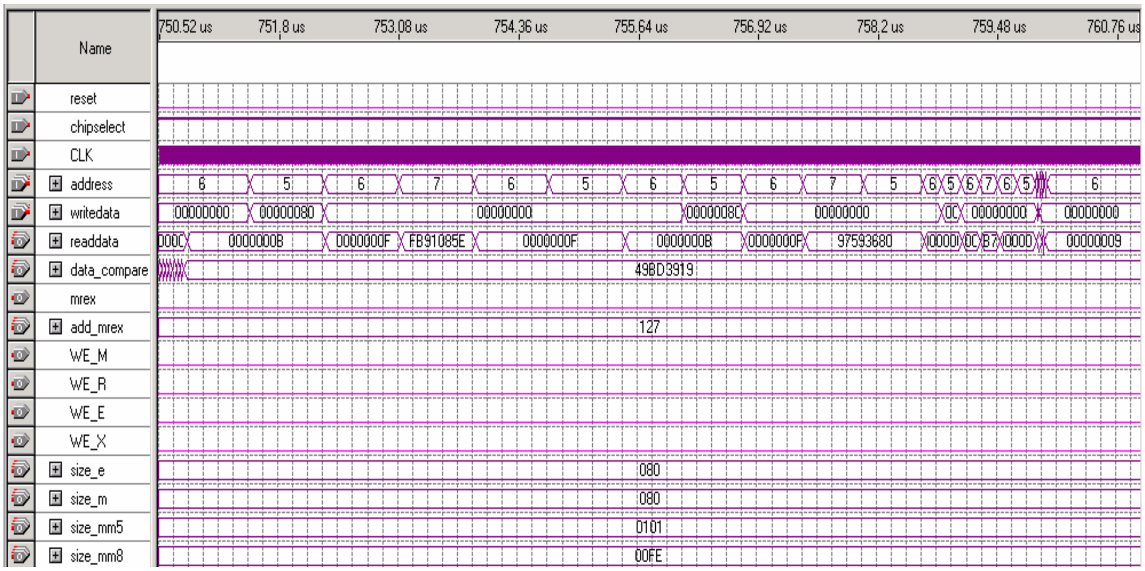


Fig. 18: Timing simulation of Modular Exponentiation core - at the completion of the operation

Table 1 shows the performance of AES crypto processor. This AES processor needs 43 clock cycles for single block of encryption process. However, for the decryption process, 86 clock cycles are needed for first block decryption process due to the key derivation process. From second block until last block, the clock cycles needed is the same as the encryption process, that is, 43 clock cycles.

Table 1: Performance of AES processor

| Features | AES Processor (128-bit) |
|---|---|
| Speed | 297 Mbit/s with 100 MHz clock rate |
| Area | 4584 LEs (Logic Elements) |
| Latency | 43 clock cycles (encryption) |
| | 86 clock cycles (for first 128 bit block in decryption) |

Table 2: Performance of RSA processor

| Features | RSA Processor (1024-bit) |
|---|---|
| RSA key length | 1024-bits |
| $f_{max}$ | 100.25 MHz |
| Area | 12881 LEs ( ≈ 200,00 gates) |
| Encryption (5-bits) with 66 MHz | 0.25ms (4000 op/s) |
| Decryption (1024-bits) with 66Mhz | 31.93ms (31 op/s) |

Table 2 shows the performance of RSA crypto processor for modulus length (key length) of 1024-bit, while Table 3 shows the area and timing performance of ECP processor. The ECP processor implements the binary field *GF*($2^{163}$) in polynomial basis representations as the basis of binary field arithmetic. It supports both trinomial and pentanomial as recommended by IEEE and NIST [20]. Montgomery point multiplication algorithm in projective coordinate is chosen. For this performance evaluation purposes, the elliptic curve system domain parameters over $F_2{}^m$ associated with a Koblitz Curve *sect163k1* [20] is implemented. The Digit Size is the number of bits of LSD multiplier operand processed in parallel, and it can be varied to achieve the trade-offs between speed, area, power consumption, and other performance metrics. Register usage in the table refers to the programmable register in each LEs. As can be seen from Table 3, processing time for each operation decreases as the digit size of the LSD multiplier increases. However, larger digit size also causes modest reduction of maximum operating frequency in each case.

Table 3: Area Performance of ECC processor

| Digit Size (bits) | $F_{max}$ (MHz) | Point Multiplication | | Point Addition | | Logic Elements (LEs) | Registers |
|---|---|---|---|---|---|---|---|
| | | Clock Cycle | op/sec | Clock Cycle | op/sec | | |
| 8 | 115 | 26445 | 4348 | 813 | 141451 | 2994 | 1095 |
| 16 | 110 | 16599 | 6626 | 705 | 156028 | 4323 | 1179 |
| 32 | 105 | 11685 | 8985 | 651 | 161290 | 5801 | 1194 |
| 64 | 95 | 8711 | 10905 | 617 | 153970 | 9792 | 1225 |

Table 4 shows the area and timing performance of SHA-1 crypto processor. The input is a single block of 512-bit message after padding function according to FIPS PUB 180-1 standard [15]. The output is a 160-bit message digest of the single block message after hashing function. The SHA-1 crypto processor is based on systolic architecture.

Table 4: Area and timing performance of SHA-1 processor

| Features | SHA Processor |
|---|---|
| $f_{max}$ | 100 Mhz |
| Speed | 0.83 x $10^6$ blocks/s (1 block = 512-bit ) |
| Latency | 120 clock cycles (for each 512-bit input block) |
| Area | 1194 LEs |

Table 5 shows the area and timing performance analysis of our LZSS data compression and decompression core. Since, in our design a symbol is processed every clock cycle, the formula for calculating our throughput is:

$$Throughput = f * SymbolWIDTH$$

where $f$ is the operation frequency and *SymbolWidth* is one of our parameterized values. In this case the *SymbolWidth* = 16. The compression rate achieved by our LZSS data compression processor is 2.35:1.

Table 5: Area and timing performance of LZSS processor

| Evaluation | | Compression Core | Decompression Core |
|---|---|---|---|
| Area | LE | 8888 | 1243 |
| | Memory Bit | 23136 | 26112 |
| Speed | $f_{max}$ | 124.84 MHz | 121.51 MHz |
| | Throughput | 1997.44 Mbps | 1944.16 Mbps |

Table 6 shows the area and timing performances of MAP arithmetic processor. This architecture is targeted for 163-bit *GF(p)* large integer modular arithmetic operations. The architecture is designed based on add-and-shift algorithm in butterfly structure.

Table 6: Area and timing performance of MAP processor

| **Modular Arithmetic Operation** | | **$f_{max}$ = 40 MHz** | |
|---|---|---|---|
| | **clock cycles** | **Elapsed time (ms)** | **ops/s** |
| Modular division | 806 | 0.02015 | 49627 |
| Modular Multiplication | 866 | 0.02165 | 46189 |
| Modular Addition | 40 | 0.001 | 1000000 |
| Modular Reduction | 40 | 0.001 | 1000000 |
| Area | | 5161 LEs | |

Table 7 shows the timing performance of ECDSA operations. For evaluation purposes, the input of signature signing and verification process is assumed to be a block of 512-bit message ready for hashing operation, while the output is a 163-bit ECDSA digital signature or digital signature verification result. The elliptic curve system is fixed to domain parameters over $F_2^m$ associated with a *Koblitz Curve sect163k1* as recommended in [20]. Table 7 shows that the ECDSA cryptosystem operates at system frequency of up to 40 MHz. This upper limit of the frequency is due to the bottleneck at the MAP processor which supports $f_{max}$ of 40 MHz. Therefore, to enhance the timing performance of overall ECDSA cryptosystem, the MAP processor should be upgraded by using more advanced algorithm or architecture such as systolic array.

Table 7: Timing performance of ECDSA operation

| **Operation** | **Running Frequency (40 MHz)** | | |
|---|---|---|---|
| | **Clock Cycles** | **Elapsed Time (ms)** | **Ops/s** |
| Key pair generation | 36846 | 0.92115 | 1085 |
| Signing | 23567 | 0.58918 | 1697 |
| Verifying | 42685 | 1.06713 | 937 |

## 6.0 CONCLUSIONS

In this paper, we have presented the design and implementation of a cryptohardware. The proposed embedded system successfully provides the complete suite of PKI-enabling functions, which include AES encryption, RSA encryption, ECC digital signature, SHA-1 hashing, and LZSS data compression. A wide-operand modular arithmetic module is also available to accelerate modular computations. The hardware design is completely described in VHDL, and is prototyped in a single reconfigurable microchip, Altera Stratix EP1S40F780C5 FPGA. The crypto core designs are parameterizable to allow for different implementations and applications, and dependent on the target hardware resources and required timing performance. A demonstration application prototype is developed to validate the proposed cryptohardware. The application is a secure e-document transfer via an insecure electronic communication channel. This application thoroughly tests and verifies the functionality of the system and reusability of the APIs designed. The high performance and flexibility achieved by the proposed embedded system makes it viable to be deployed in many security applications such as storage devices, embedded systems, hardware security module, network routers, security gateway, and the like.

## REFERENCES

[1] P. C. van Oorschot, A. J. Menezes, and S.A. Vanstone, "Handbook of Applied Cryptography", *CRC press Inc., Florida*, 1996.

[2] Understanding Public Key Infrastructure, RSA Data Security, 1999.

[3] The Elliptic Curve Cryptosystem, Certicom Corp., July 2000.

[4] A. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists", in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, August 2001, pp. 545-557.

[5] F. Crowe, A. Daly, T. Kerins, and W. Marnane, *Single-Chip FPGA Implementation of a Cryptographic Co-Processor*, Dept. of Electrical & Electronic Engineering, University College Cork, Ireland, 2004.

[6] J. Daemen and V. Rikmen, *The Design of Rijndael: AES – the Advanced Encryption Standard*, Springer-Verlag, Berlin, 2002.

[7] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems", in *Communications of the ACM,* February 1978, Vol. 21, No. 2, pp. 120-126.

[8] J. L. Massey, "An Introduction to Contemporary Cryptology", in *Proceedings of the IEEE,* 1988, Vol. 76, No. 5, pp. 533-549.

[9] P. L. Montgomery, "Modular Multiplication Without Trial Division", *Mathematics of Computation*, 1985, Vol. 44, pp. 512-521.

[10] M. Khalil, S. L. Tan, and S. Husin, "FPGA Implementation of RSA Public-Key Cryptographic Coprocessor", in *Proceedings of IEEE TENCON'2000*, Kuala Lumpur, September 2000, pp III-6.

[11] E. Mohammed, A. E. Emarah, and Kh. El-Shennawy, "Elliptic Curve Cryptosystems on Smart Cards", *Arab Academy for SCIENCE AND Technology*, 2001.

[12] M. Khalil and K. W. Lim, "Design of ECC cryptographic coprocessor for implementation in an FPGA-based Hardware", *Malaysian Science & Technology Congress (MSTC 2003)*, Kuala Lumpur, Malaysia, September 2003.

[13] M. Khalil and K. W. Lim, "Design of an Elliptic Curve Cryptography (ECC) Processor Core for Implementation in FPGA-based System-on-Chip (SoC) Cryptosystem", in *Proceedings of the 2003 Malaysian Science and Technology Congress (MSTC 2003),* Kuala Lumpur, Malaysia, September 23-25, 2003.

[14] M. Khalil, A. Z. Shameri, and W. S. Chong, " Pipeline Implementation of Secure Hash Algorithm SHA-1 for Cryptographic application in Network Security", in *National Conf. On Telecommunication Technology*, Johor Bahru, Malaysia, 20-21 November 2000.

[15] *Secure Hash Standard*, National Institute of Standards and Technology, 17 April 1995.

[16] A. Bosselaers, R. Govaerts, and J. Vandewalle, "SHA: A design for Parallel Architectures?", *Advances in Cryptology, Proc. Eurocrypt'97, LNCS 1233,* W. Fumy, Ed., Springer-Verlag, 1997, pp. 348-362.

[17] M. Khalil and K. M. Yeem, “FPGA Implementation of Lossless Universal Data Compression Hardware”, in *Malaysian Science & Technology Congress Symposium A*, Johor Bahru, Malaysia, 19-21 September 2002.

[18] S. C. Shantz, “From Euclid’s GCD to Montgomery Multiplication to the Great Divide”, *Technical Report TR-2001-95*, Sun Microsystems Laboratories, 2001.

[19] M. Khalil and K. H. Koay, “A VHDL Module Generator for Fast Prototyping of Multimedia ASICs”, in *Malaysian Journal of Computer Science*, Universiti Malaya, Malaysia, June 2000, Vol. 13, No.1, pp. 65-75.

[20] SEC2: Recommended Elliptic Curve Domain Parameters, Certicom Corp, 2000.

## BIOGRAPHY

Mohamed Khalil Hani obtained his PhD in Electrical & Computer Engineering from Washington State University, Pullman in 1992. Currently, he is a Professor of Digital Systems & Microelectronics at the VLSI-ECAD Research Laboratory in the Faculty of Electrical Engineering, Universiti Teknologi Malaysia, Skudai. His current research interests include Digital System, Embedded systems and System-on-Chip (SoC) Technology, VLSI & Microelectronics System Design, Hardware Cryptosystem Design for IT Security, Artificial Intelligence Techniques & Neurohardware for Pattern Recognition. He has published an extensive number of papers on these subjects in local and international conference proceedings and journals. He is a member of IEEE member since 1996 in the associated Circuits & System Society and the Computer Society.

Hau Yuan Wen obtained her Masters in Electrical Engineering from Universiti Teknologi Malaysia in 2005. Her specialization is in the field of advanced digital system design, embedded system and System-on-Chip (SoC). She has served as a research officer at the VLSI-ECAD Research Laboratory in the Faculty of Electrical Engineering, Universiti Teknologi Malaysia, Skudai.

Arul Paniandi obtained his Masters in Electrical Engineering from Universiti Teknologi Malaysia in 2006. His specialization is in the field of advanced digital system design, embedded system and System-on-Chip (SoC). He is currently employed as a design engineer in Altera Corp Sdn Bhd in Penang, working on IP core designs.