BUS ENCRYPTION AND AUTHENTICATION UNIT FOR SYMMETRIC SHARED MEMORY MULTIPROCESSOR SYTEM USING GCM-AES

By

VARUN JANNEPALLY

Bachelor of Technology in Electronics and

Communication Engineering

Jawaharlal Nehru Technological University

Hyderabad, AP, India

2006

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2008

BUS ENCRYPTION AND AUTHENTICATION UNIT

FOR SYMMETRIC SHARED MEMORY

MULTIPROCESSOR SYTEM USING GCM-AES

Thesis Approved:

Dr. Sohum Sohoni

Thesis Adviser

Dr. Louis G. Johnson

Dr. James E. Stine, Jr.

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

I sincerely thank my adviser Dr. Sohum Sohoni for giving me the opportunity to work with him. Dr. Sohoni has offered me tremendous support and encouragement throughout this study. It is an honor to have worked with him in the CAESAR Lab. I gratefully acknowledge the funding provided by the CAESAR Lab throughout the course of this study and making this work possible.

I would like to thank Dr. Louis G. Johnson and Dr. James E. Stine, Jr. for agreeing to serve as my committee members. I specially thank them for their time and cooperation.

It was a great experience working with my lab mates, especially David Fritz, Wira Mulia and Ying Xiong (Lawrence). I thank them for their support that they offered me whenever needed.

I thank my family from the bottom of my heart for their unconditional love and support. I finally dedicate this work to all my family members.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Background

In daily life, millions of financial transactions take place and large amounts of secure data gets communicated among numerous computers around the world. Information related to finance, defense, and other sensitive information is constantly processed by computers. To protect this sensitive information from attackers, traditionally computers rely on anti-virus software and firewalls to safeguard information. Hardware mechanisms provide higher security with a much lower overhead compared to software mechanisms. Several hardware security architecture models in uniprocessor and multiprocessor systems have been proposed. With the increasing role of multiprocessors and multicore architectures in e-commerce applications such as internet telephony, inventory management and database management applications which maintain medical records, credit history records etc. it is crucial to design efficient secure architectures and algorithms for these systems. This work proposes an enhancement to a secure architecture model for safeguarding the cache-to-cache communication in a shared memory multiprocessor system. The purpose of the scheme is to maintain confidentiality and the integrity of the bus data transfers with a minimum performance overhead.

1.2 Security Attacks

The types of attacks [1] which are possible on a system can be classified as

1. Physical attacks.
2. Software attacks.
3. Side channel attacks.

Physical attacks:

The manipulations of the physical components of a system to reveal some sensitive information are categorized as physical attacks. The following are some of the possible physical attacks:

a. Modchips installed in Microsoft Xbox can allow the console to play pirated games [2].

b. Opening, probing and reverse-engineering of the chips.

To protect the system from physical attacks, mechanisms that do not disclose information upon reverse engineering should be employed. For example, a device that checks for main memory corruption should be employed for devices which have direct memory access (DMA) support [1].

Software attacks:

The type of attacks which take advantage of the vulnerabilities in the application software or the operating systems, fall under this category. The most common types of software attacks are:

a. Buffer overflow attack: This attack overwrites the data stored in the system beyond the size of the buffer when the user inputs large amounts of data. This can cause the system to execute the user's choice of code. This is termed stack smashing [1].

b. Message blocking, reordering and spoofing are some of the common attacks on a bus or communication channel [2].
c. Injecting malicious code that infects the system and reveals information of specific interest can also be possible. The most common types which come under this category that infects numerous computers all around the world are computer viruses, trojans, and worms.

Side channel attacks:

These forms of attacks reveal sensitive information when physical actions of computations are analyzed. Timing attacks and power attacks come under this category [1].

Timing attacks:

The time taken for a computation depends upon the type of computation being performed. If a particular sequence of time patterns pertains to a set of operations that reveal some data, the corresponding information can be extracted by timing analysis.

Power attacks:

When computations are performed on a chip it consumes power. When the power traces are analyzed, hidden data can be obtained. There are examples where a single power trace can be analyzed to extract a DES key from commercial smart cards. The differential power analysis techniques study the correlations between data and the power consumption [1].

1.3 Hardware security mechanisms

To safeguard the system from some of the above types of attacks traditionally vendors provide security through software-only-solutions. These include the anti-virus software and firewalls. Despite their security features like encryption, inspection of network traffic and port monitoring, software solutions are still susceptible to attacks. This is the reason we often patch our Operating Systems and frequently update our anti-virus software and firewalls. Hardware security mechanisms are efficient and provide higher security. As such several researchers have proposed hardware security architecture models [2, 4, 5, 7, 16, 17, 19, 20, 32] for both uniprocessor and multiprocessor systems. These systems provide security deriving trust just from the processor. Everything else outside the processor chip main-memory, I/O (input-output) interface are not trusted. In these models even the operating system is not trusted. Since these models have a small trusted computing base the points of vulnerability are very few. These architectural additions for providing security made at the hardware level can be incorporated on to the chip or may be a separate co-processor [6] or even a separate core. Because of having dedicated hardware resources for providing security the performance of these systems is higher compared to that of their software counterparts. Some of the common security features [3] provided by these units are

1. Confidentiality
2. Integrity.
3. Authentication.

Confidentiality: A way of providing access to sensitive information only to authorized users and protecting from unauthorized group.

Integrity verification: A mechanism which helps a recipient to detect any tampering of the data on transmission and identifying a fake message.

Authentication: A mechanism which helps the recipient to verify the origin and also detect any intruder.

The hardware security models use encryption decryption units to provide confidentiality and a hash verification unit to provide integrity. The operation of these units is explained in detail in later sections. This work presents a secure architecture model for a symmetric shared memory multiprocessor (SMP) to safeguard the cache-to-cache transfers. This work proposes a hardware security mechanism, which employs Galois Counter Mode (GCM) of advanced encryption standard (AES) and modifies it to work in an SMP environment. The work focuses on why GCM is a better choice over cipher block chaining mode (CBC) which is used in current state of the art systems. It estimates the storage required by the additional hardware unit in both modes of operation. A full system SMP simulation quantifies the performance overhead introduced by the additional hardware unit in both schemes to safeguard the cache-to-cache transfers. The impact of increasing cache line sizes and the effect of varying throughput of the AES units in both the schemes is studied. Providing the same level of security a performance gain in the range of 4X-9X over the existing scheme is achieved using GCM.

1.4 Organization

The rest of the sections are organized as follows. Chapter 2 covers the related work in cryptography, and the hardware security architectural models in uniprocessor and multiprocessor systems. Chapter 3 covers the implementation of encryption and authentication unit for a symmetric shared memory multiprocessor system (SMP). To study the overhead of this unit we perform a full system simulation. The simulation methodology and the results of our study are discussed in Chapter 4 and Chapter 5 respectively. Chapter 6 lists the conclusions and the scope for future work.

CHAPTER II

RELATED WORK

2.1 Cryptography

Encryption is a process of mapping a plaintext message (P), using an encryption function E and a key K, to a ciphertext (C). It is denoted as E (P, K) = C. **Decryption** is the process of mapping the cipher text C back to the plaintext P using a decryption function D and key K. It is denoted as D (C, K) = P. The type of encryption system which uses the same keys in encryption and decryption processes is called **symmetric encryption system.** This is shown in Fig 2.1. Key distribution is a problem in symmetric encryption system and security lies in the confidentiality of the key [3].

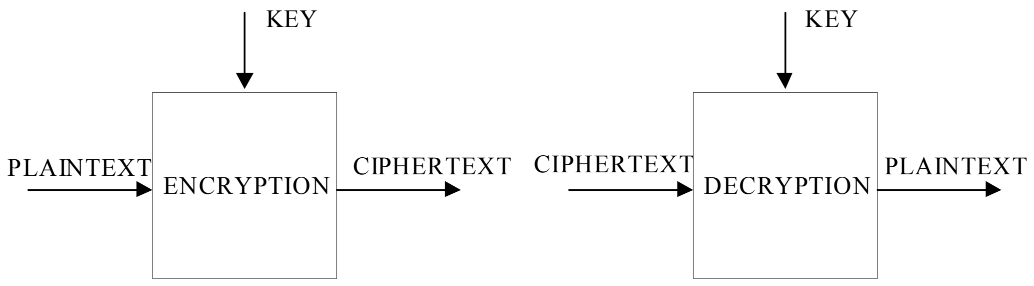


Figure 2.1: Encryption and Decryption

The type of encryption system which uses two different keys, one for encryption process and the other for decryption is called **public-key cryptosystem** or **asymmetric encryption system**. The decryption key is called the **private-key** and the key used for encryption is called **public-key**. In general, it is computationally infeasible to derive the private key from the public-key, hence the public-key can be disseminated to the parties that communicate with this type of system. Though key distribution is not a concern in public-key cryptosystems, computation time is a constraint.

The symmetric encryption algorithms can be categorized into **stream cipher** and **block cipher** algorithms. The block cipher algorithms work on blocks of data of equal size and the stream cipher algorithms work on a stream of bits. The commonly used symmetric block cipher encryption algorithms are data encryption standard **(DES)** algorithm [8] and advanced encryption standard **(AES)** algorithm [9]. The commonly used asymmetric algorithms are Diffie-Hellman [10] and Ron Rivest, Adi Shamir and Leonard Adleman (RSA) algorithm [11].

Hash functions:

A hash function is one that takes a string of any length and returns an output string of fixed length called **hash**, which is shorter than the input. Commonly used hash functions are message-digest cryptographic hash function (**MD5**) [12] and secure hash algorithm (**SHA**) [13]. Message authentication code (**MAC**) is generated using a one-way hash function and a key. In this scenario only the parties having the key can produce and verify the hash value. The schematic is shown in Fig 2.2. We can also modify the block cipher algorithms to generate MACs [3].

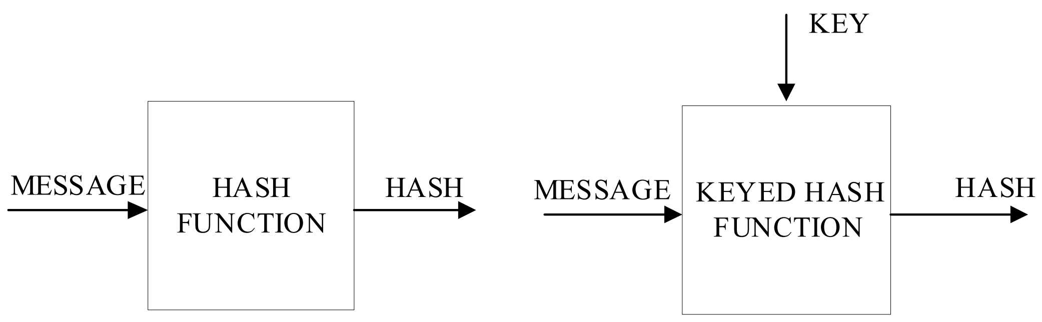


Figure 2.2: Hash Functions

2.2 Uniprocessor system security architecture models

Thekkath et al. [4] proposed Execute-only memory (XOM) for a uniprocessor system, one of the first schemes to propose that software solutions alone cannot handle security attacks. This scheme provides privacy and integrity against some of the physical attacks on memory or the system bus or even a compromised operating system. This model is implemented by making additions like XOM virtual machine monitor (XVMM) and tagging the registers, level 1 (L1), level 2 (L2) cache lines with XOM identifier tags. It also has a private memory on chip to store the keys for the corresponding XOM identifiers. When the data is sent off-chip, XVMM encrypts the data with the corresponding XOM identifier keys and it also generates a MAC. This provides privacy and also helps in verifying any tampering of the data when sent off-chip. It also protects the system against spoofing attacks [14]. XOM uses a simple and direct approach to provide these features, but has significant performance overhead as the security hardware units are on the critical path. The security architecture model of XOM is shown in Fig 2.3.

AEGIS [5] is another processor architecture that is secure against physical attacks. AEGIS uses physical random functions and a one-time pad (OTP) encryption scheme to provide privacy. It uses a cached hash tree [15] for integrity verification of off-chip memory. Shi et al. [16] proposed an efficient counter mode security architecture model which hides the decryption latency by predicting the sequence numbers and precomputing the pads. A pad is a value that can be pre-computed and is independent of the plaintext/ciphertext data. Yan et al. proposed a memory encryption/authentication scheme using GCM mode of operation [32]. They address the memory authentication latency problems posed by traditional hash algorithms such as MD5, SHA1. This scheme reduces memory authentication latency significantly by using GCM mode. A generalized security architectural model is shown in Fig 2.4. This figure shows an encryption/decryption unit which encrypts/decrypts the processor to memory data flow, a hash verification unit which computes the hash value of the data sent off-chip. It also shows the trusted boundary and the un-trusted sources [5].

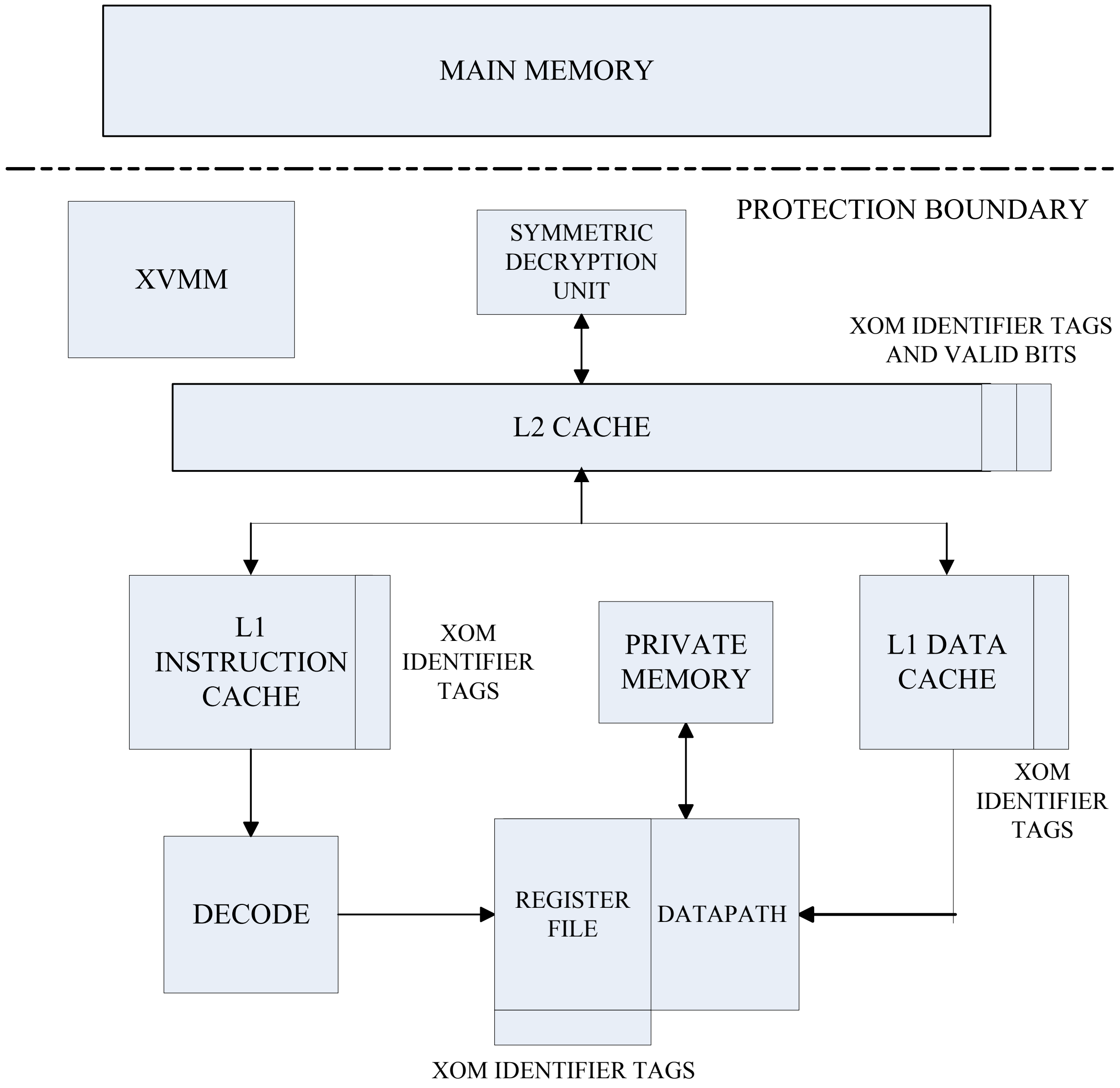


Figure 2.3: XOM Architecture

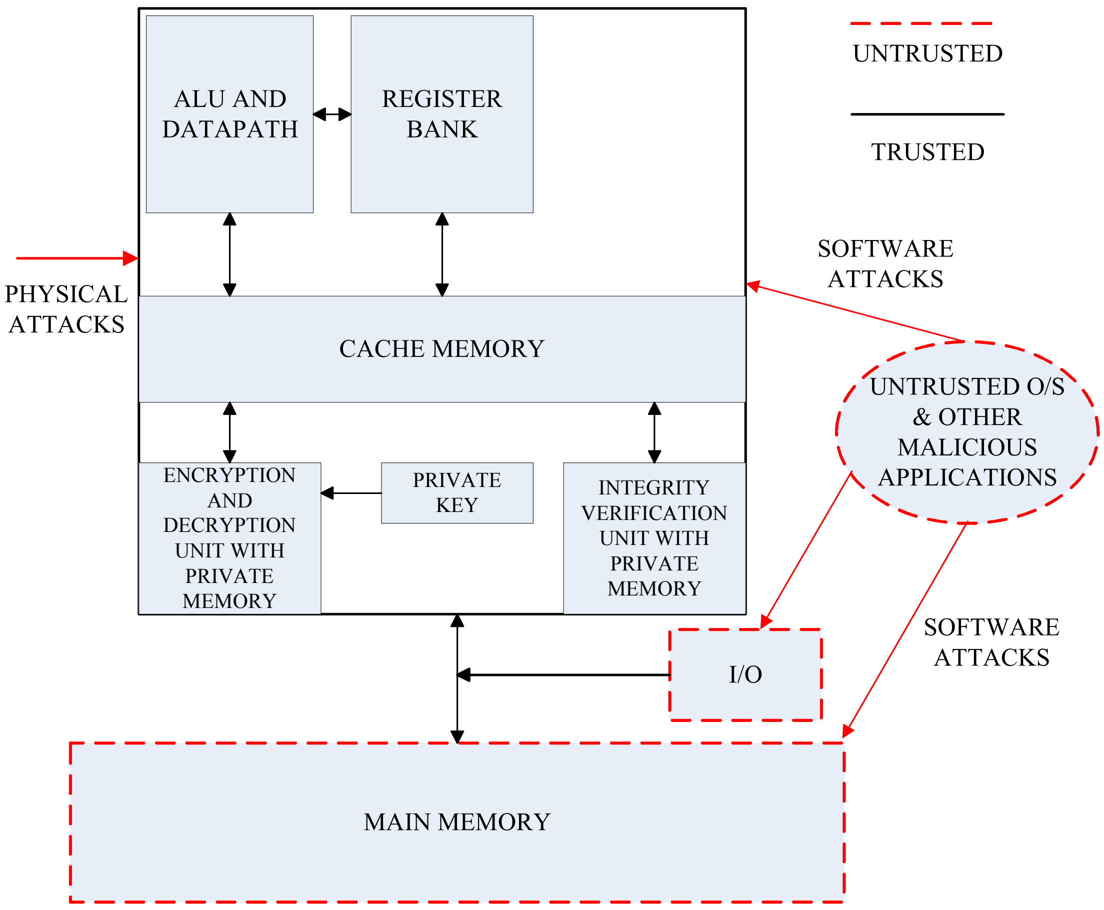

UNTRUSTED
TRUSTED
ALU AND DATAPATH
REGISTER BANK
PHYSICAL ATTACKS
SOFTWARE ATTACKS
CACHE MEMORY
UNTRUSTED O/S & OTHER MALICIOUS APPLICATIONS
ENCRYPTION AND DECRYPTION UNIT WITH PRIVATE MEMORY
PRIVATE KEY
INTEGRITY VERIFICATION UNIT WITH PRIVATE MEMORY
SOFTWARE ATTACKS
I/O
MAIN MEMORY

Figure 2.4: Generalized security architecture model

2.3 Multiprocessor system security architecture models

In a multiprocessor system apart from encrypting the data between the processors and memory we also need to encrypt the data communication in between the processors [2, 17]. To protect the bus communication from security attacks Zhang et al. [2] designed an encryption scheme, SENSS, for a symmetric shared memory multiprocessor (SMP) system, which is both secure and fast. This scheme encrypts/decrypts and also authenticates the data flow between processors. The proposed scheme uses the cipher block chaining mode of the advanced encryption standard (CBC-AES) symmetric block cipher [18] which is explained in detail in section 3.1. SENSS uses this cipher because of its capability in message authentication. It provides architectural additions like PID (processor ID) and GID (process ID) which earlier works [17] lack, and a secure hardware unit capable of providing bus encryption and authentication. Though CBC-AES provides authentication capability, it needs another invocation of the underlying block cipher to compute the authentication tag. CBC cannot be pipelined to encrypt many different blocks simultaneously.

Our study is an effort to increase the performance of SENSS in safeguarding the cache-to-cache data transfers. Our scheme employs the Galois/Counter Mode of Operation (GCM) [21], [22], a method which can use pipelined and parallelized implementations with minimum computation latency. The purpose of the scheme is to maintain confidentiality and the integrity of the bus data transfers for all types of bus attacks. The objectives of the scheme are to provide a lower overhead of encryption latency and message authentication even under heavy bus loads, while maintaining the same level of security provided by SENSS. Lee et al. [19] proposed interconnect independent security enhanced SMP (I^2SEMS), which investigates security for SMP with varying size of keystream pools and also the relationship between keystream hit rate and various cache coherence protocols. We assume a fixed keystream pool size, that is the memory which stores the pre-computed pads, and compare the performance losses of SENSS and our scheme with increasing cache line sizes and varying the throughput of the AES units. We estimate additional memory required to provide these features for both CBC and GCM schemes.

As security features developed for uniprocessor and SMP cannot be adapted directly for safeguarding distributed shared memory multiprocessor (DSM), Rogers et al. [20] proposed a security scheme for data protection in DSM, which is the first work to study privacy and integrity in DSM. Yang et al. [7] proposed SecCMP: a secure chip-multiprocessor (CMP) architecture which addresses the key protection and core authentication issues in multi-core systems.

CHAPTER III

IMPLEMENTING ENCRYPTION AND AUTHENTICATION UNIT

3.1 Modes of operation of advanced encryption standard algorithm (AES)

The modes of operation of an underlying block cipher commonly include cipher block chaining mode (CBC), electronic book mode (ECB) and counter mode (CTR) [24]. ECB mode is not secure because it reveals pattern information when processed offline. CBC mode offers better security than the ECB mode [24, 23]. In CBC mode, encryption of the block depends on the feedback of the previous block encipherment. CTR mode is equally secure with no dependencies among different blocks, allowing operations to be fully pipelined to achieve greater performance, but CTR mode lacks authentication capability [24]. It is important to have an authentication capability, apart from encryption, in the employed block cipher because any interruption or modification in the communication must be detected. Galois counter mode (GCM) mode [21, 22] is another mode of operation which can use pipelined and parallelized implementations with minimum computation latency. GCM mode combines the efficiency of CTR mode with the authentication capability of the CBC mode. The following sub sections discuss in detail the CBC and GCM modes of operation.

3.2 CBC mode of operation

In this mode the plaintext message is exclusive or'ed (XOR) with the previous block encipherment. This is illustrated in Fig 3.1. If the plaintext message is divided into '*i*' blocks and if E is the encryption cipher and K is the key. The ciphertext C_i is produced by passing the XOR of P_i and C_{i-1} (which is $(i-1)^{th}$ ciphertext) through the encryption function E.

A similar approach is followed on the decryption side. The ciphertext message C_i is passed through the decryption function D. This result is XOR'ed with the (*i*-1)th ciphertext to form the plaintext P_i. This is shown in Figure 3.1 [3].

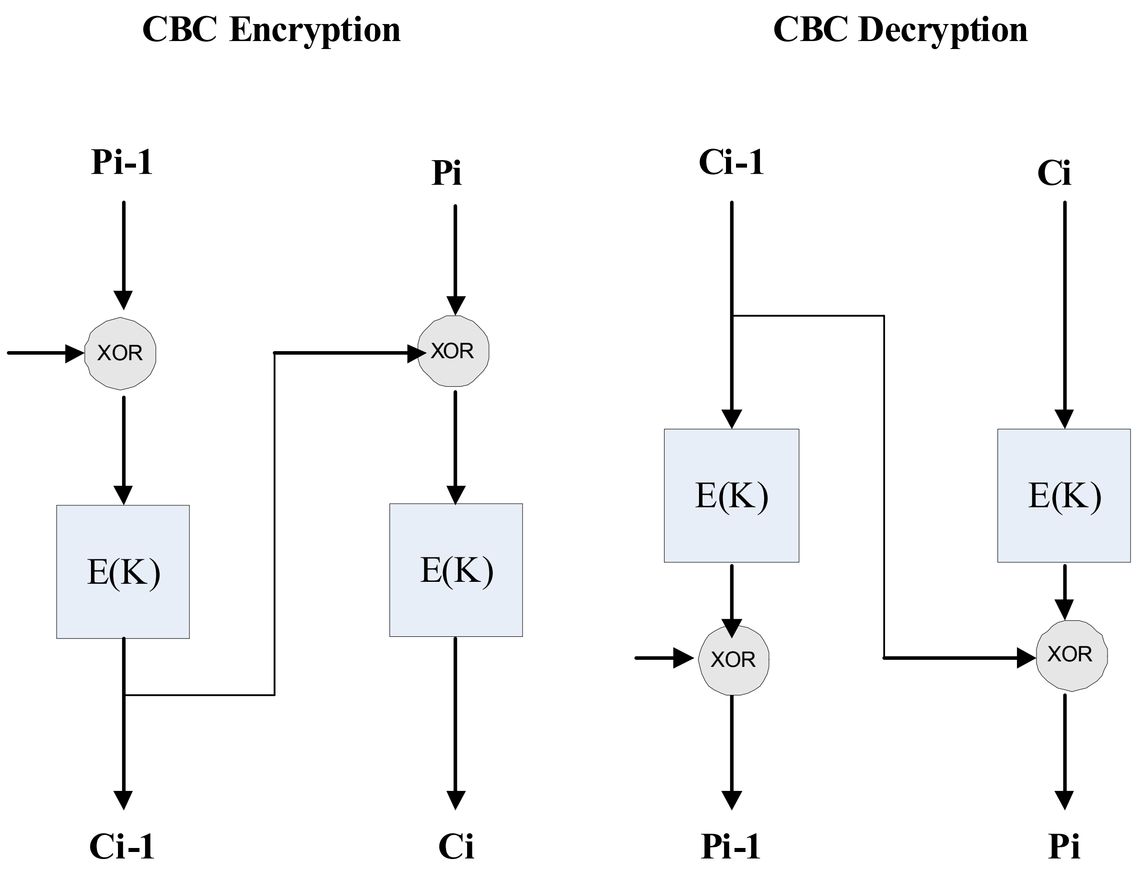


Figure 3.1: CBC Encryption and Decryption

CBC-MAC: In CBC scheme MAC of a block is generated by invoking the underlying block cipher as AES (block). The initialization vector for authentication is different from that used in encryption. The least significant ‘s’ bits are taken to form MAC. In CBC ‘*n*’ blocks can be authenticated at a time by chaining all the MACs of the previous blocks [2]. In CBC scheme we need another invocation of the underlying block cipher for authentication. In addition we need to maintain separate masks (results of previous XOR operations) for authentication and encryption [2].

3.3 GCM mode of operation

The operation of GCM mode includes two functions: authenticated encryption and authenticated decryption. Authenticated encryption takes the plaintext data, additional authentication data (AAD) and an initialization vector (IV) as inputs. The output of the authenticated encryption function is cipher text and an authentication tag. The schematic representation is shown in Fig 3.2. The authenticated decryption function takes the ciphertext, the initialization vector, additional authentication data and authentication tag as inputs. The output is the plaintext data only if the authentication tag which is computed is equal to the authentication tag provided, as shown in Fig 3.3.

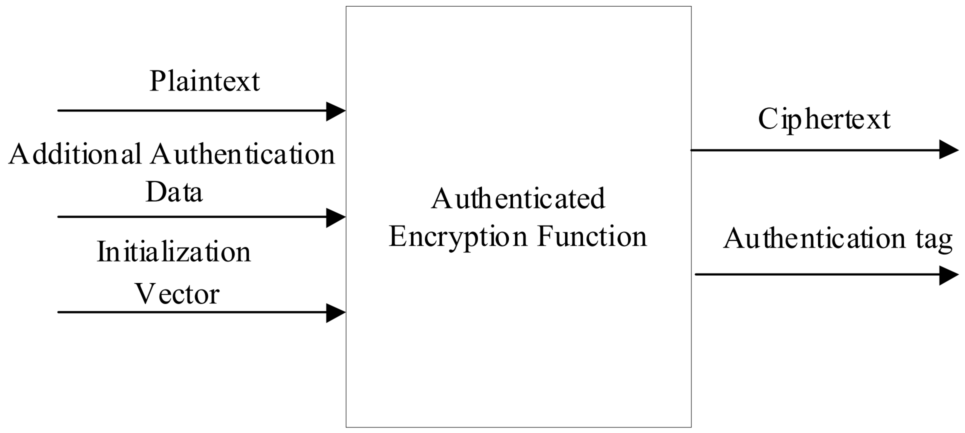

Plaintext
Additional Authentication Data
Initialization Vector
Authenticated Encryption Function
Ciphertext
Authentication tag

Figure 3.2 GCM Authenticated Encryption

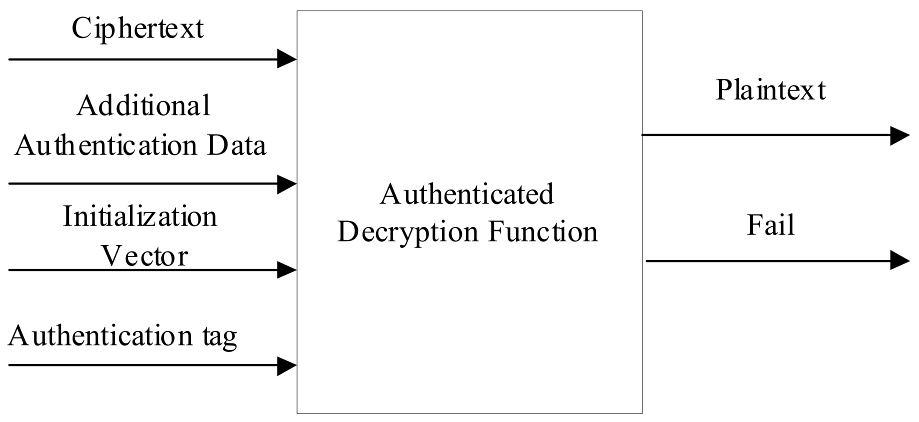

Ciphertext
Additional Authentication Data
Initialization Vector
Authentication tag
Authenticated Decryption Function
Plaintext
Fail

Figure 3.3 GCM Authenticated Decryption

GCM authenticated encryption:

Step 1: The IV of length 96 bits is taken and appended with strings of 0's and a 1 to make it 128 bits in length.

$$Y0 = IV \,||\, 0^{31}1.$$ (Where '||' denotes concatenation operation).

Step 2: This is passed through an increment function (*incr*).

$$Y_i = incr\ (\ Y_{\ (i-1)}\ )\ \text{for each i.}$$

Step 3: The plaintext message Pi is XOR'ed with E(K, Yi). Where E is the encryption function and K is the key.

$$C_i = P_i\ (XOR)\ E\ (\ K,\ Y_i).$$

Step 4: The authentication tag is computed using the function GHASH. The inputs to the GHASH function are the C_i, AAD. This is XOR'ed with E(K, Y). The most significant't' bits (MSB) of this result form the authentication tag. GHASH function is a series of multiplications in the galois field. The GHASH algorithm is defined in [22].

$$\text{Authentication tag} = MSB_t\ (GHASH\ (C_i,\ AAD)\ XOR\ E\ (K,\ Y)).$$

GCM authenticated decryption:

Step 1: Is same as the authenticated encryption step1.

Step 2: A new authentication tag is computed with the received ciphertext.

New Authentication tag = MSB_t (GHASH (C_i, AAD) XOR E (K, Y)).

Step 3: If the new authentication tag computed is equal to the authentication tag provided, then Yi is calculated.

Y_i = *incr* ($Y_{(i-1)}$) for each i.

Step 4: The plaintext data is produced by the XOR operation of C_i and E (K, Y_i).

P_i = C_i (XOR) E (K, Y_i)

The GCM hardware is shown in the figure below [21].

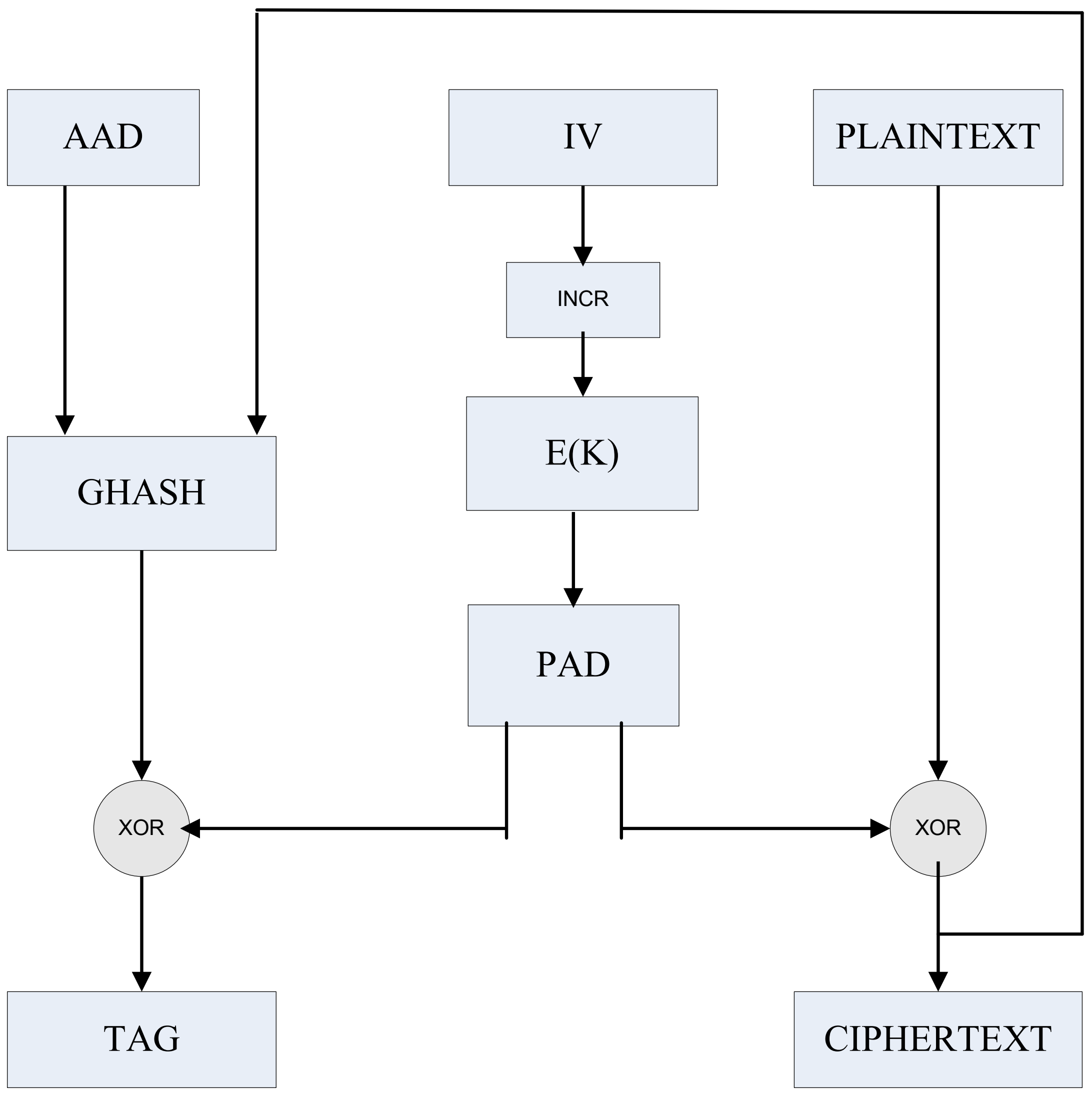


Figure 3.4: GCM Hardware

3.4 Processor communication model

The objective of the study is to design a security hardware unit capable of encrypting and authenticating the bus communication between the processors on a symmetric shared memory processor (SMP) network using GCM-AES. In a multiprocessor system apart from safeguarding the processor to memory data flow we also need to safeguard the processor-to-processor (cache-to-cache) data communication. The security hardware unit that encrypts/decrypts the processor to memory communication cannot be used to encrypt/decrypt processor to processor communication. This is because if we use the same pads to encrypt both types of communication, if an adversary tracks the transfers between the processors on the communication channel which use the same pad over a long period of time, the adversary can gain information of the plaintext by XOR'ing the transfers. For instance, let P be the pad and D the data. If the data D gets modified to D^{l} and if the transfers use the same pad P, information can be obtained by (P XOR D) (XOR) (P XOR D^{l}) [2]. Therefore two separate encryption/decryption units that use different pads should be used to encrypt the processor-to-memory transfers and processor-to-processor transfers. This work only considers the design of the security hardware unit that encrypts and authenticates processor-to-processor communication. The schematic of the communication model is shown in Figure 3.5.

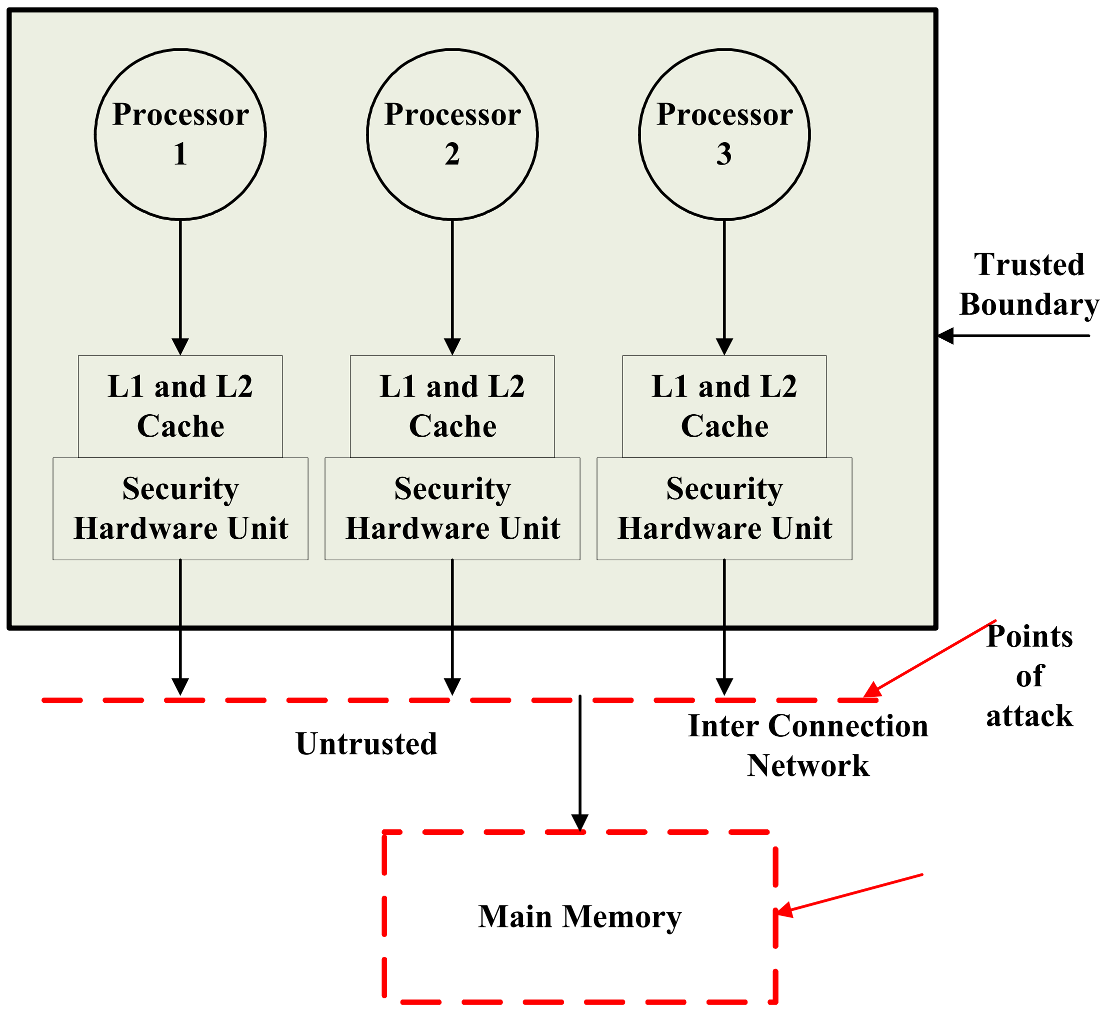


Figure: 3.5 Processor communication model

Every processor on the network is assigned a public-private key pair. The private key is hard burned on that particular processor node when manufactured and is not accessible. Apart from its private key each processor also has the public keys of all other processors stored on the processor node, the use is explained in the later sections. The security hardware unit maintains the process ID's of all the processes on that particular processor node. The applications which are to be run on the SMP system with support for hardware security are encrypted using a key K. In the next step the key K is encrypted with the processor's public key [2]. When the application is run for the first time, the processor decrypts the key K with its private key and next decrypts the application. This restricts the use only to the processor entitled to decrypt the application.

3.5 Operation of CBC-AES unit in an SMP

SENSS uses CBC mode to encrypt and authenticate transactions between the processors. The sequence of operations for encryption in CBC:

1. $X = P_i$ XOR C_{i-1}
2. $C_i =$ AES (X, key)
3. Send C_i

But in the above scenario step 2 consumes time because of the invocation of AES. The time this step consumes is equal to the latency of the underlying AES unit. SENSS modifies this as follows

1. $C_i = P_i$ XOR Mask $_{last}$
2. Send C_i
3. Mask= AES (C_i , key)

In the above scenario step 2 is moved to step 3. The mask is updated in the background. In this way the message (P_i) need not wait for the mask to be calculated. If we need to encrypt another message before the mask is ready for the present transaction, we have to wait for the mask update which is still a bottleneck. In SENSS for every process ID running on that node, 2-masks are maintained one mask for the sending sequence and the other mask for the receiving sequence for each process ID. We need to maintain separate masks for each process ID as masks of one process ID cannot be shared with the other. The encryption model is shown in Figure 3.6.

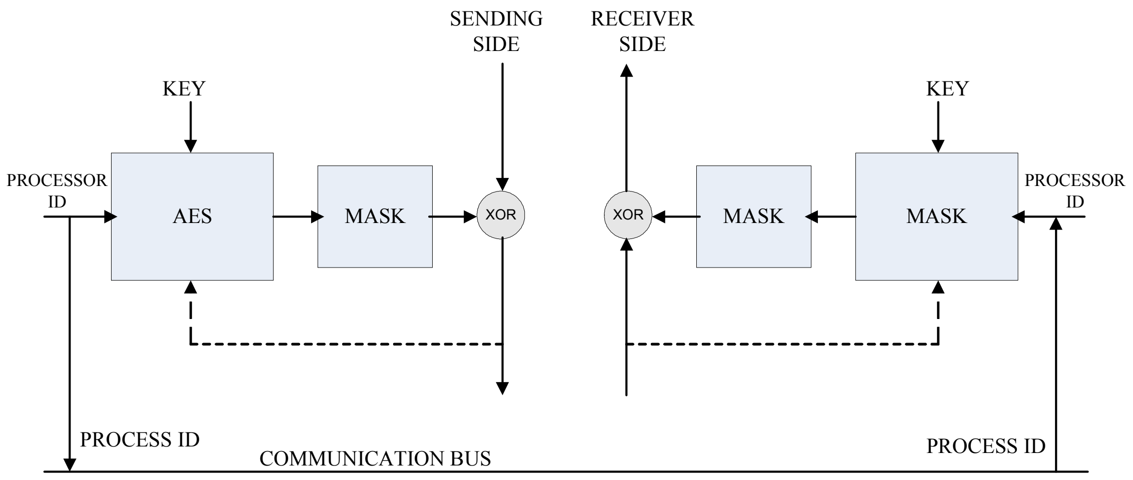


Figure 3.6 Bus Encryption Model

Apart from encryption it also needs to maintain separate masks for authentication because in CBC, authentication needs a separate invocation of the block cipher. This increases space requirement and also consumes more time. The performance of SENSS can be greatly improved by using GCM mode of operation which can encrypt as well as authenticate transaction within a single invocation of the block cipher. We also need not maintain separate masks for encryption, for each process ID as in CBC. This reduces storage requirement by a significant fraction. The storage requirement is discussed in section 3.8.

3.6 Operation of GCM-AES unit in an SMP

In a symmetric shared memory environment, where there are *n* processor nodes, the individual nodes communicate among themselves using a shared bus. The security hardware unit, which uses GCM-AES mode of operation, is embedded in all the nodes of the network. The security hardware unit maintains the process IDs of all the active processes on that particular processor. It also has the memory required for operation of the GCM-AES unit. Our adaptation of GCM to work in an SMP environment is explained below. Figure 3.7 shows the details of this implementation. We assign a single counter value for a particular processor communication pair. When communication is initiated for the first time every processor node shares its counter value with the other nodes by encrypting the counter value with the public key of the receiving processor. As the communication proceeds, the counter value is incremented for each send and receive request. The counter values for all the processors on the network are maintained by the security hardware unit. The initialization vector is formed using this counter value and the processor ID. This initialization vector is given as the input for the cipher block to compute the pad. Since the computation of the pad is independent of the plaintext data, pads can be pre-computed and stored in the security hardware unit's memory. When a request comes in, the processor first verifies the processor ID and process ID for the incoming request with the ones maintained by the security hardware unit. If they match it reads the associated counter value for the sending sequence for that processor ID. Next, it computes the pad, if not pre-computed. This pad is XOR'ed with the plaintext data which produces the ciphertext, as shown in Figure 3.7. The GCM-AES hardware keeps generating the pads as they are consumed, for the subsequent transfers. *As long as we have pre-computed pads in the memory of the security hardware unit, the encryption and decryption latency will only be a single cycle, i.e. for the XOR operation*. The performance of the scheme varies with the number of pre-computed pads.

In the next step the ciphertext and the process ID, which is used as the additional authentication data, are processed to generate the authentication tag within the same invocation of the cipher block. In CBC, we would need another invocation of the underlying cipher. The ciphertext and the authentication tag are transferred on to the bus by the security hardware unit. For the subsequent transfers, the counter value is increased and the associated pads are provided in the same way. The decryption part operates similarly and decrypts the cache line on the receiver node. It computes the tag and compares with the received tag and takes suitable action. The sequence of actions that take place in GCM authenticated encryption and decryption functions are shown in Fig 3.8 and Figure 3.9 respectively [33].

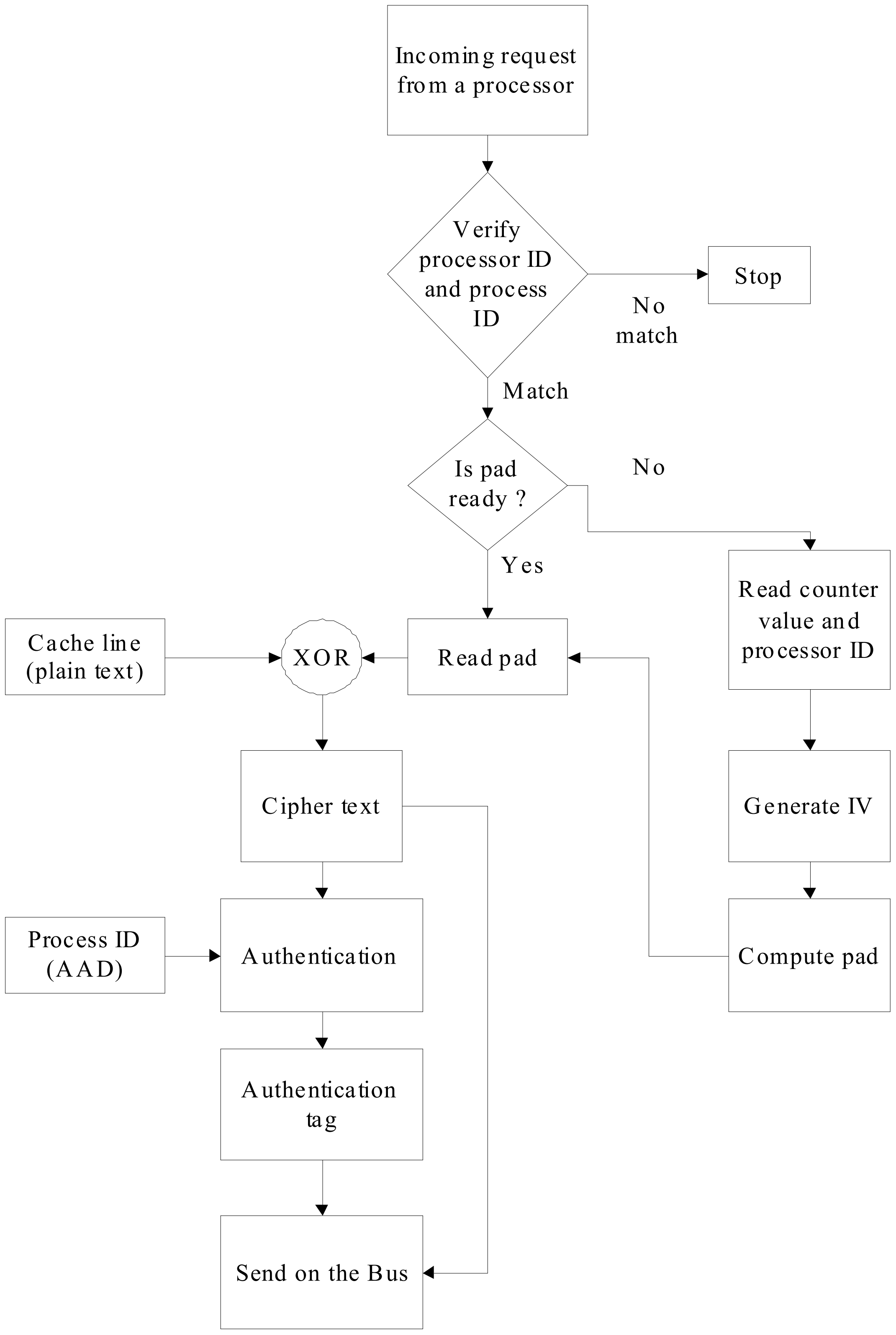


Figure 3.7 Implementing authenticated encryption in SMP

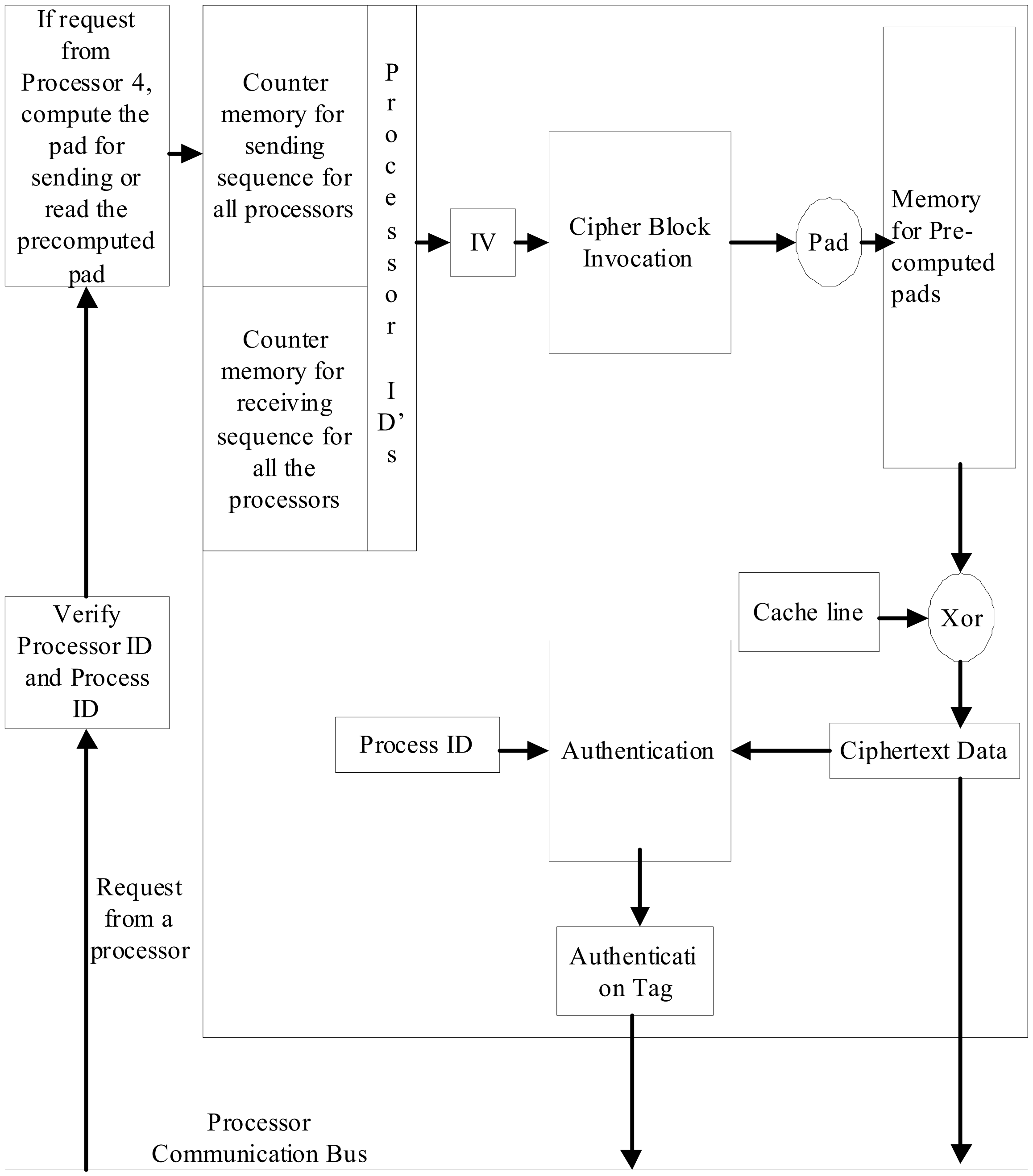

If request from Processor 4, compute the pad for sending or read the precomputed pad
Counter memory for sending sequence for all processors
Counter memory for receiving sequence for all the processors
Processor ID's
IV
Cipher Block Invocation
Pad
Memory for Pre-computed pads
Cache line
Xor
Verify Processor ID and Process ID
Process ID
Authentication
Ciphertext Data
Authentication Tag
Request from a processor
Processor Communication Bus

Figure 3.8 Security hardware unit model for encryption

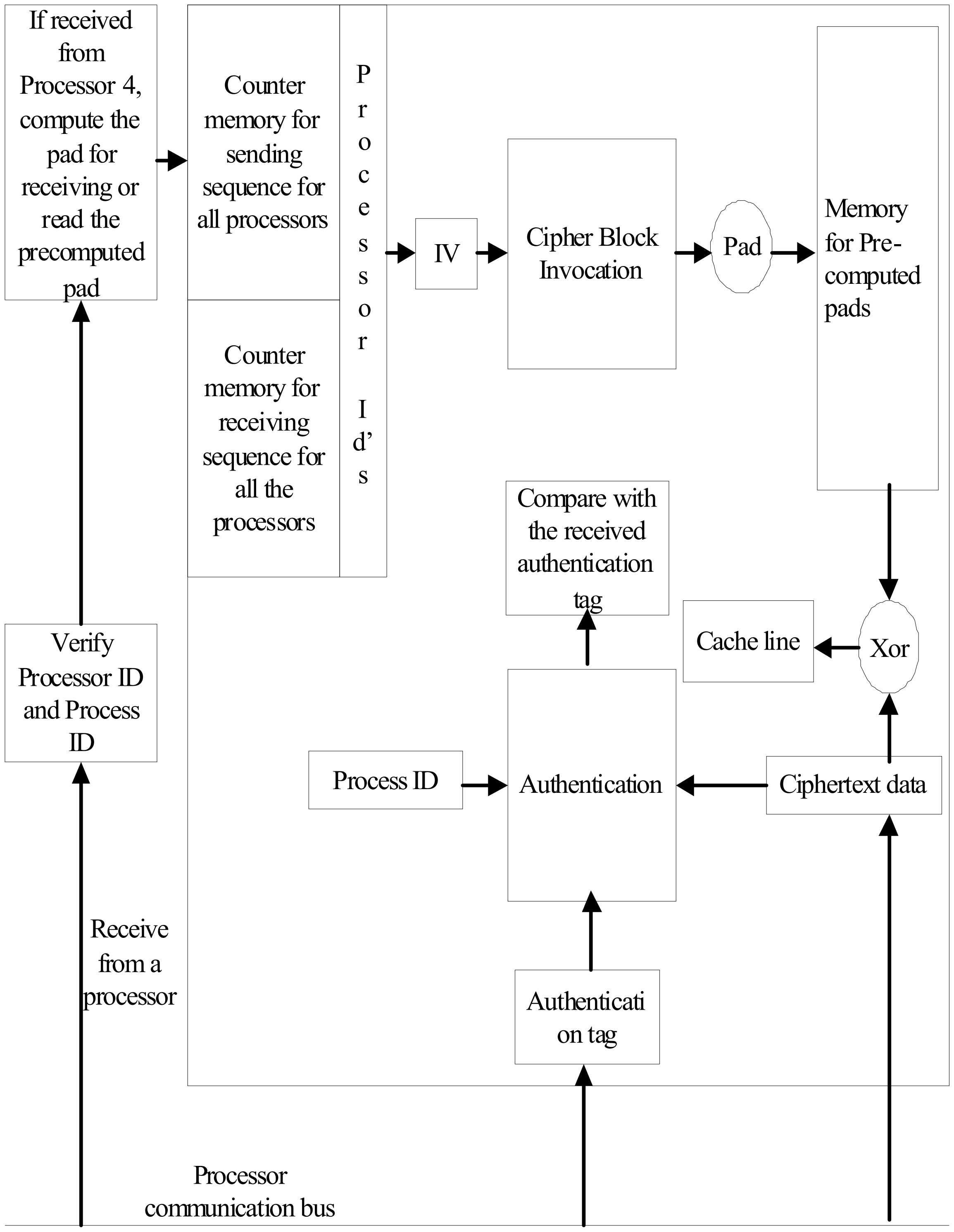

If received from Processor 4, compute the pad for receiving or read the precomputed pad
Counter memory for sending sequence for all processors
Counter memory for receiving sequence for all the processors
Processor Id's
IV
Cipher Block Invocation
Pad
Memory for Pre-computed pads
Compare with the received authentication tag
Cache line
Xor
Verify Processor ID and Process ID
Process ID
Authentication
Ciphertext data
Receive from a processor
Authentication tag
Processor communication bus

Figure 3.9 Security hardware unit model for decryption

3.7 Bus attacks prevention mechanism

The three types of attacks that are possible on a shared bus in an SMP environment are blocking messages, message reordering and message spoofing [1, 2]. These attacks can be prevented using GCM-AES. If an attacker blocks some message from reaching a node, the authentication tag which is pre-computed will not be equal to the tag computed by the security hardware unit. Thus dropped messages can be detected. When reordering of messages take place, again the authentication tag generated differs hence these types of attacks can be detected. Message spoofing is not possible because the attacker does not have access to the sequence numbers maintained by the processors security hardware unit, which is on chip and secure. Moreover, the process ID is included as additional authentication data, due to which only the processors entitled to run the process can decrypt data [33].

3.8 Storage estimate for security hardware unit in GCM and CBC schemes

The security hardware unit stores the public keys of all the other processors in the SMP environment. It stores these keys because when we initiate communication for the first time to share initial counter values with all the other processors we need to encrypt the initialization vectors with the public keys of the respective processors. When the processor receives the encrypted value it decrypts the same with its private key pair, which is unique and no other processor can decrypt it. It stores its private key to decrypt the counter values from the other processors. After the communication is initialized, each processor will have all the counter values it shares with the other processors.

If we have an 8 processor SMP environment, we need a storage space of

$$128 * 8$$

for public keys and 128 bits for the private key. To verify if a process is running on a particular node, we need 9 bits of storage. If we have 110000101 on row 32 this implies the process ID 32 is running on node 1, 2 and 8. The extra bit specifies whether the process is valid or not. If we have a maximum of 100 processes running, we need a total of

$$100 * 9 \text{ bits}$$

of storage. The IV is formed from the counter value, which is 96 bits long and the processor ID field, which is 3 bits long. Since the scheme maintains separate counter values for sending sequence and receiving sequence, for a total of 8 processors it takes

$$(96 + 3) * (7 * 2) \text{ bits}$$

Thus storage requirement, excluding the space for storing the pads, for an 8 processor network is

$$(128 * 9) + (100 * 9) + (99 * 14) = 3672 \text{ bits, or } 430 \text{ bytes}$$

For higher performance, the pads are pre-computed. The number of pads to be pre-computed depends on the bus speed and AES-GCM throughput. Based on our initial simulations we found that storing 5 pre-computed pads per processor pair in every processor node provides adequate performance. The pad size is equal to the size of the cache line. This storage is to be provided in addition to the above storage. If we consider a 64 byte cache line size, a total of

$$64 * 5 * 14 = 4480 \text{ bytes}$$

is required for pads storage. For a 128 byte cache line size, 8960 bytes of storage is required. In SENSS, for a 64 byte cache line size the storage provided in SENSS for a 2-mask system assuming 100 processes running on a processor is

$$12800 * 2 = 25600 \textit{ bytes}$$

for encryption as well as authentication masks. *The amount of storage required in our scheme is far less compared to SENSS. Also, our storage requirement for the pads is independent of the number of processes and thus our scheme scales better with increasing number of processes* [33].

CHAPTER IV

SIMULATION METHODOLOGY

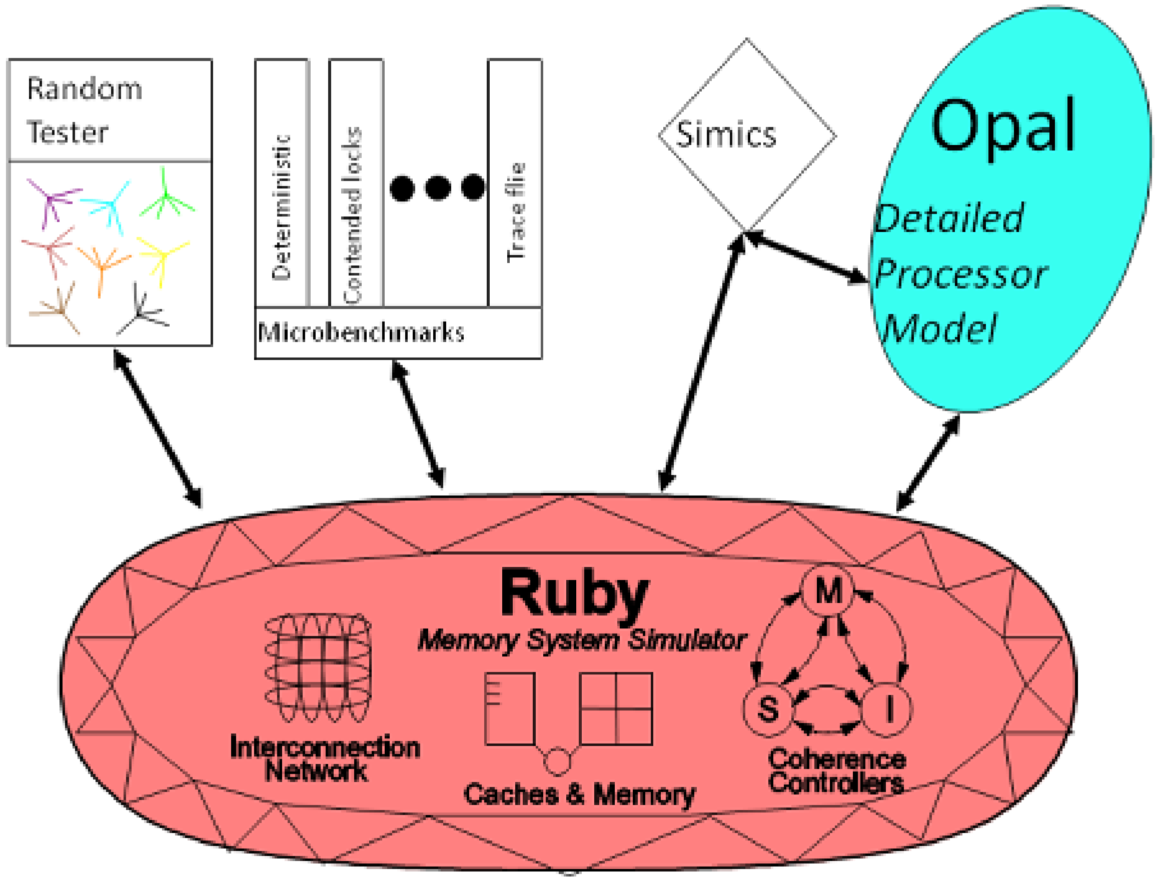
To simulate the cache-to-cache transactions between processors on a multiprocessor system, we simulate a full system SMP environment using both SIMICS [25], a full system simulator and GEMS [26], a memory system extender of SIMICS.

4.1 SIMICS a full system execution driven simulator

Simics is a full system execution driven simulator developed by Virtutech. It supports various architectures such as Alpha, ARM, IA-64, MIPS, Sparc, x86 and x86-64. It can simulate embedded systems, desktops, multiprocessor systems [29]. Virtutech also provides disk dumps and kernel images to simulate various targets. This work uses *cashew* target to simulate 2, 4, 8 – processor systems. *Cashew* target simulates Ultrasparc processor architecture running Aurora 2.0 (Fedora core 3). It uses Linux kernel 2.6.15 which includes SMP support [35].

4.2 GEMS (memory system simulator)

General execution driven multiprocessor simulator (GEMS) is a memory system extender for Simics. GEMS leverages the power of Simics to simulate multiprocessor systems [26, 27]. GEMS can be used to simulate uniprocessor, SMP systems, chip multiprocessor systems (CMP), multiple-CMP systems. GEMS provides RUBY memory system model which includes physical components such as caches, system interconnect, memory and coherence controllers. This is shown in Figure 4.1. CPU driver issues memory requests to ruby, which simulates the requests and callbacks the driver with the latency [26, 28]. Ruby provides several SMP and CMP protocols such as MOSI_SMP_bcast [34], MOSI_SMP_directory, MOESI_CMP_directory, MOESI_CMP_token, etc.



4.1 GEMS simulator model [28]

Opal is used for out-of-order processor model, it implements Sparc instruction set architecture. This work uses only RUBY and an in-order processor model. The steps involved in simulating an SMP system are:

1. Creating checkpoints for multiprocessor system with pre-compiled workload using Simics.
2. Building Ruby memory system module.
3. Configuring Ruby.
4. Loading Ruby.
5. Running simulation and collecting statistics.

This work modifies the Ruby memory system to include the functionality for the operation of the security hardware unit used for the encryption/decryption and authentication of the cache-to-cache communication on an SMP network. The security hardware unit is modeled to operate in both CBC and GCM modes. This functionality is achieved by modifying the cache controller logic. The performance overhead introduced by the additional hardware unit in both schemes to safeguard the cache-to-cache transfers is studied using SPLASH2 benchmarks [30, 31]. The impact of increasing cache line sizes and the effect of varying throughput of the AES units in both the schemes are studied. This work simulates a 2, 4 and 8 processor SMP system with separate L1 Instruction and Data-cache and an integrated L2 cache. We simulate L2 caches with two different line sizes of 64 and 128 bytes to show the impact of increasing line size. We also vary the throughput of the AES unit from 0.8 GBps to 3.2 GBps. The simulation parameters for 64 and 128-byte cache line sizes are given in Table 4.1 and Table 4.2 respectively.

Processor Clock Frequency	**1 GHZ**
L1-Instruction Cache Size	64 KB
L1-Data Cache Size	64 KB
L2-Cache Size	1 MB
Main Memory	1 GB
AES throughput	0.8 GBps-1.6 GBps
Cache Line Size	64 bytes
L1 Hit Latency	4 cycles
L2 Hit Latency	10 cycles

Table 4.1: Simulation Parameters for 64-byte cache line size

Processor Clock Frequency	**1 GHZ**
L1-Instruction Cache Size	128 KB
L1-Data Cache Size	128 KB
L2-Cache Size	2MB
Main Memory	4GB
AES throughput	1.6 GBps-3.2 GBps
Cache Line Size	128 bytes
L1 Hit Latency	4 cycles
L2 Hit Latency	10 cycles

Table 4.2: Simulation Parameters for 128-byte cache line size

4.3 GCM & CBC throughput schemes

The throughput in the simulations is varied from 0.8 GBPS to 3.2 GBPS.

This is derived as follows:

Throughput of the scheme = (Cache line size) * (Clock frequency) / AES latency.

CBC 0.8 GBPS scheme takes 320 cycles to encrypt a 64 byte cache line. CBC 1.6 GBPS scheme takes 160 cycles to encrypt a cache line size of 64 bytes. Here 320 cycles and 160 cycles is the latency of the AES unit. Similarly to encrypt a 128 byte cache line size GCM 1.6 GBPS scheme takes 320 cycles and GCM 3.2 GBPS scheme takes 160 cycles respectively.

4.4 Workload

This work uses FFT, LUNC, LUC, and RADIX multiprocessor applications from SPLASH2 [30, 31] benchmark suite. The overhead introduced by the security hardware unit for these benchmarks is studied. The function of each benchmark is tabulated in Table 4.3.

Benchmark	Function
FFT	A complex 1-dimensional Fast Fourier Transform.
LUNC	Factors a dense matrix into the product of lower triangular and upper triangular matrix. The data structure prevents blocks from being allocated contiguously.
LUC	Same function as LUNC, but allows blocks to be allocated contiguously.
RADIX	A program that performs an integer radix sorting.

Table 4.3 Workload

CHAPTER V

DISCUSSION OF RESULTS

5.1 Effect of throughput in CBC and GCM schemes

In GCM scheme we maintain 5 pre-computed pads per processor pair to send as well as to receive. On every send and receive transaction the associated pre-computed pad is read without any delay. If the pad is not pre-computed it computes the pad and performs an XOR operation with the plaintext/ciphertext. Results show that most of the time the pad is pre-computed unless there is a series of cache-to-cache transactions which consume all the pre-computed pads and the subsequent transactions have to wait for pad computation. This might not be the case in CBC where the pad computation of the current transaction depends on when the previous transaction has occurred. If there is a series of transactions the present transaction should wait for the mask update whereas in GCM scheme we can pre-compute the pads because there is no dependency between the previous and present transaction. The number of transactions which consume all the pads within the latency of AES unit is 2 in SENSS where the subsequent transaction might have to wait for the mask if it occurs while the previous mask update. In our scheme, since 5 pads are pre-computed, the series of transactions which consume all the pads within the latency of the AES unit is 5. Hence this scope for performance improvement is exploited using GCM. To compute the authentication tag in CBC, we need two invocations of the cipher block but in GCM the authentication tag can be computed with an additional few cycles of latency within a single invocation of the cipher block.

Figure 5.1 and Figure 5.2 show the total percentage performance loss of CBC and GCM schemes for 2, 4, and 8- processor systems with a cache line size of 64 bytes, 128 bytes respectively and AES unit throughput of 0.8 GBps, 1.6 GBps and 3.2 GBps. The other parameters as shown in Table 4.1 and Table 4.2 and kept unchanged throughout the simulation. The performance loss is calculated by taking the ratio of the difference of execution time in cycles with the security features to a baseline system without any security feature.

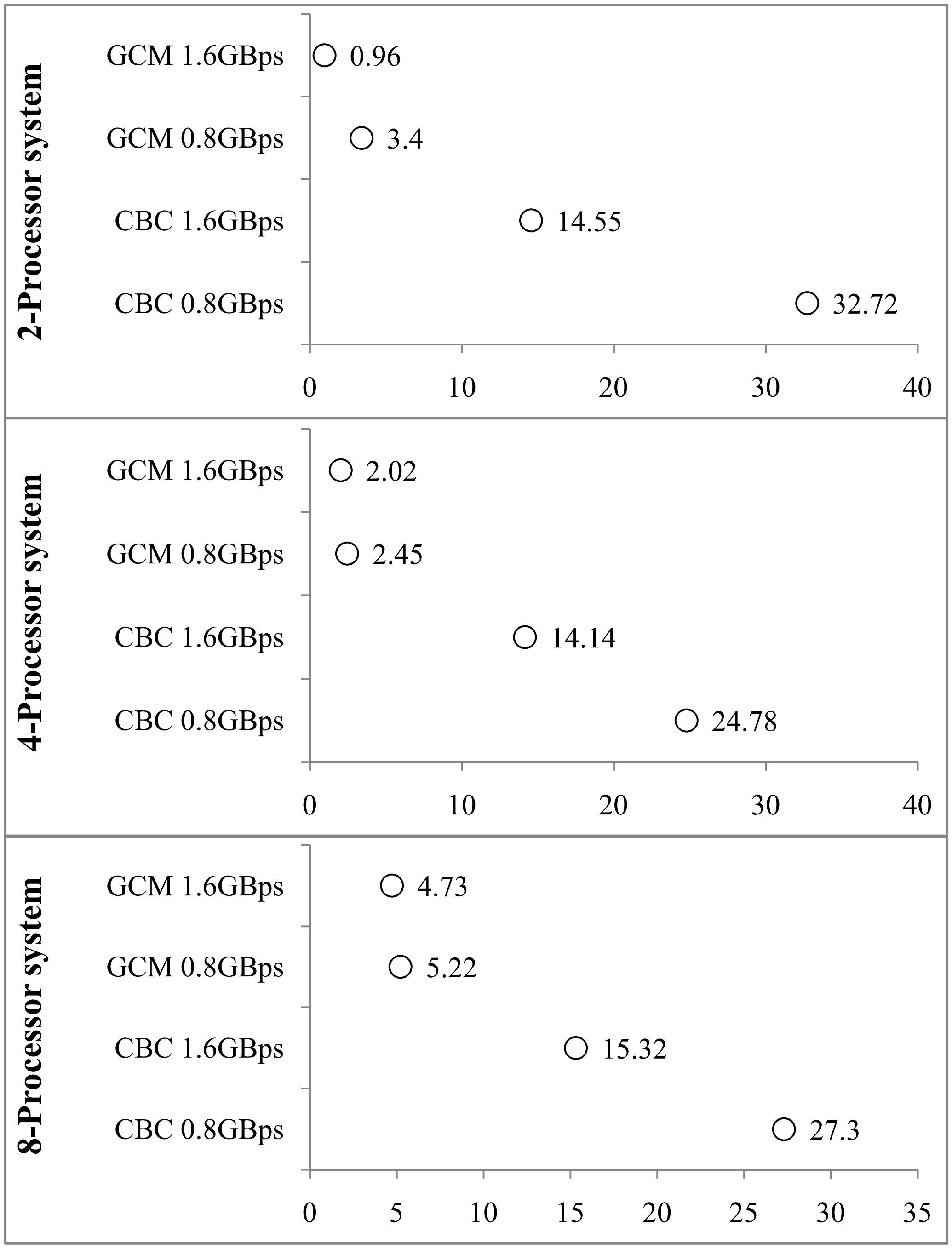


Figure 5.1 Total Performance loss (given by the ratio of the difference of the execution time in cycles with security features to a base line system without any security feature) for 64 byte cache line size for 2, 4, 8-Processor systems to provide encryption and authentication.

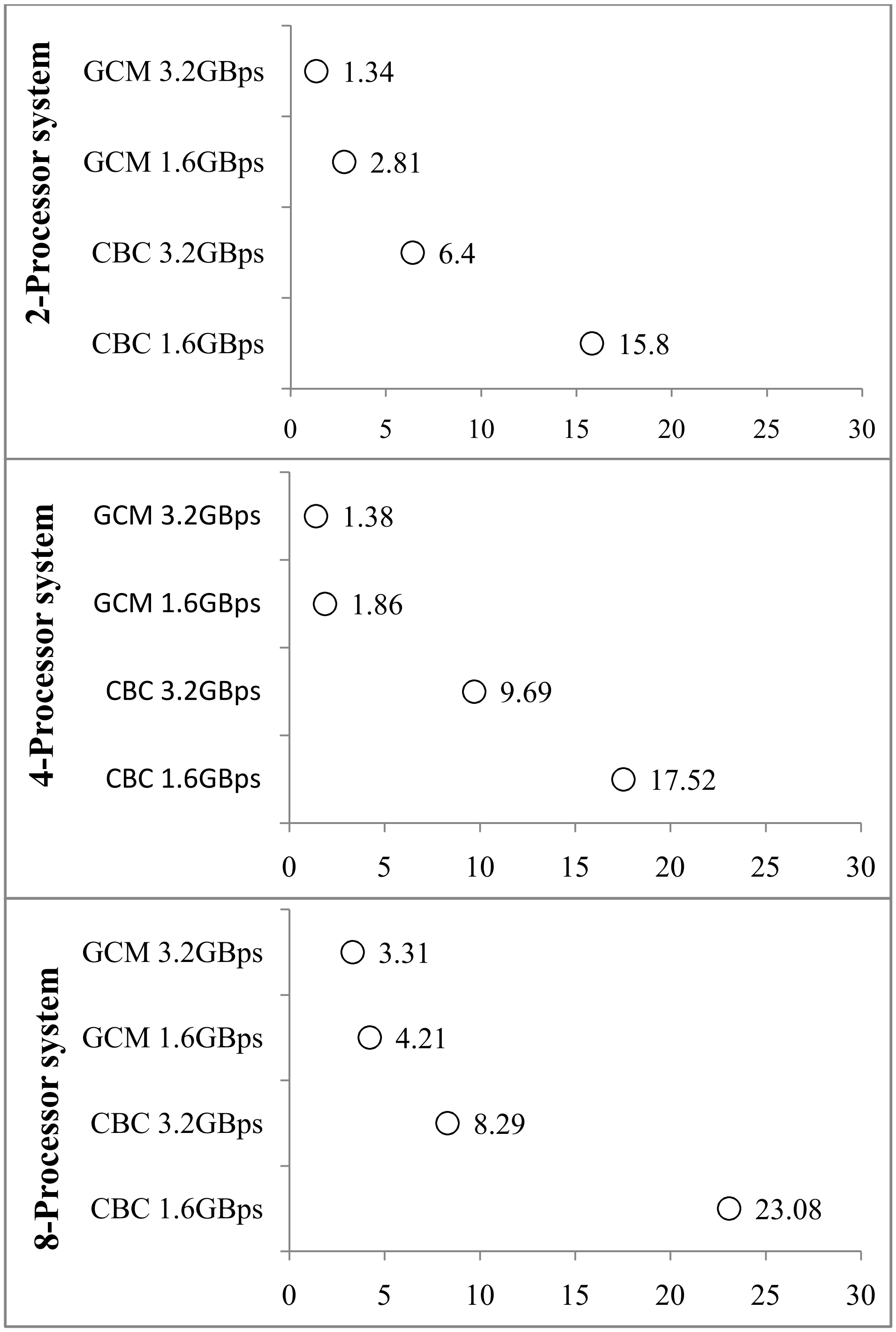


Figure 5.2 Total Performance loss (given by the ratio of the difference of the execution time in cycles with security features to a base line system without any security feature) for 128 byte cache line size for 2, 4, 8-Processor systems to provide encryption and authentication.

We can clearly see that the performance of GCM scheme is better compared to SENSS which uses CBC. The performance of GCM scheme is better even with low throughput of the AES unit. In CBC, we also need to accommodate another AES unit for authentication since the masks used for authentication are different from those of encryption. This increases space requirement, as discussed in Section 3.6. If we consider a system with 128 byte cache line size, the performance loss of CBC 1.6GBps scheme is high because any 2 back-to-back cache transfers within the current latency of the AES unit, that is 320 cycles, will consume 2 masks. The additional mask generation for the next waiting cache transfer takes 320 cycles. The performance of CBC 3.2 GBps is better because for 2 masks to be consumed, any 2 back-to-back cache transfers should take place within 160 cycles. The additional mask generation now takes 160 cycles. For every single cache-to-cache transaction, we need another invocation of AES to authenticate transactions. Authentication in CBC 3.2 GBps takes 160 cycles, whereas in CBC 1.6 GBps it takes 320 cycles. A similar discussion holds for 64 byte cache line size. If we consider a 64 byte cache line size system with GCM 0.8 GBps, the number of back-to-back cache transfers that consume all the pre-computed pads is 5, which is greater than in the CBC schemes. If all the pads are consumed, the next pad generation takes an additional 320 cycles. Similarly, in GCM 1.6 GBps the next pad generation takes 160 cycles. The authentication transaction interval for our scheme for both cache line sizes is 1. We can see that the performance does not vary drastically for authentication in GCM as we can compute the authentication tag within a single invocation of AES with an additional few cycles of latency, which is far less compared to the latency of two invocations of AES as in CBC. In this way, we can afford to authenticate every individual transaction.

We generated performance numbers for CBC with an authentication interval of 10 transactions. This is shown in Figure 5.3. A performance improvement is seen compared to single transaction authentication. As seen from Figure 5.3 in most cases GCM outperforms CBC. Therefore, in CBC the performance varies with the authentication transaction interval.

We can also see that the performance loss of GCM with reduced throughput is not very high compared with a higher throughput design. This is because all the pre-computed pads may not be consumed most of the times. In such a scenario, even pre-computing more pads would not result in a performance gain. In CBC scheme the throughput of the design plays a greater role [33].

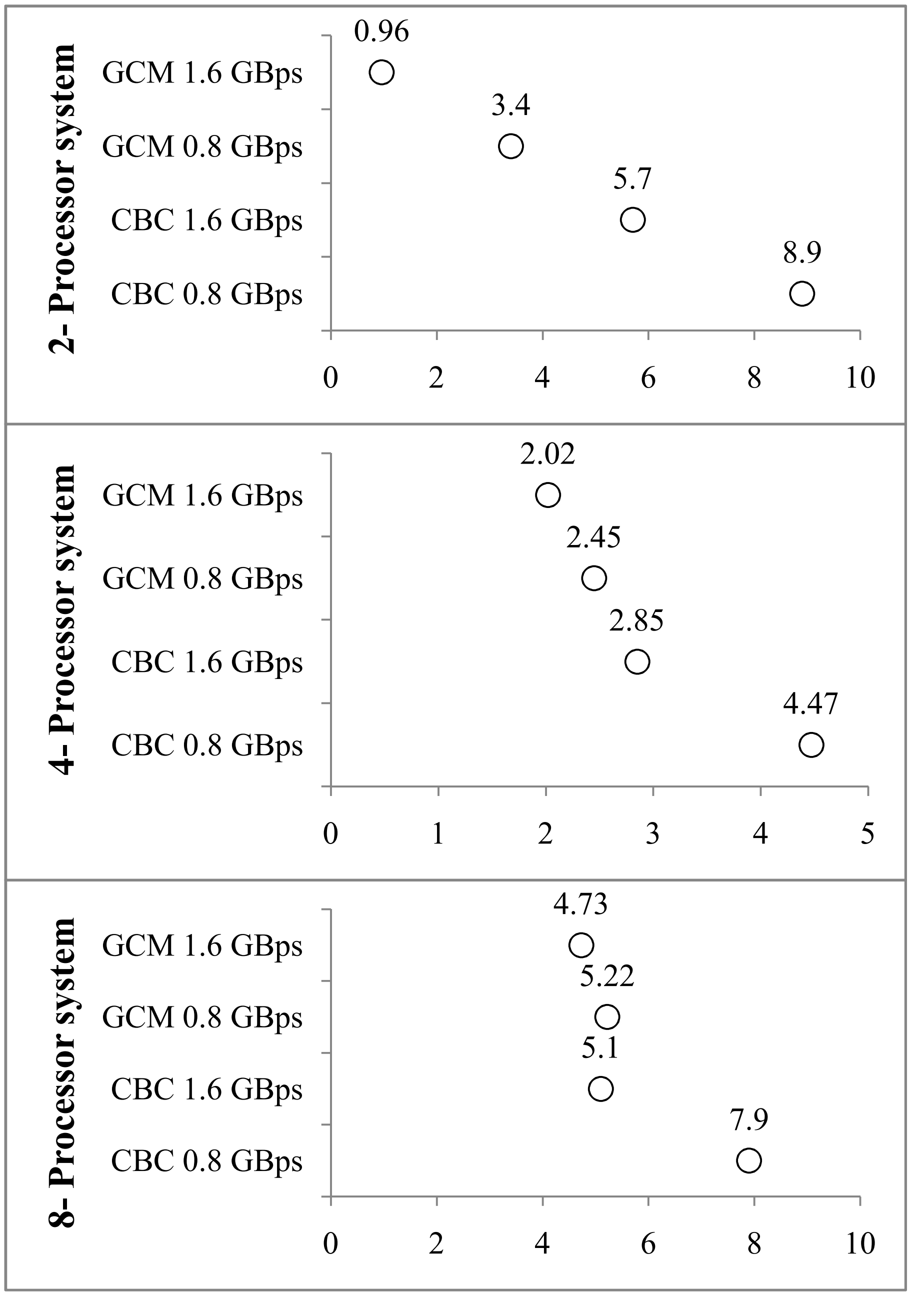


Figure 5.3 Total Performance loss (given by the ratio of the difference of the execution time in cycles with security features to a base line system without any security feature) for 64 byte cache line size for 2, 4, 8-Processor systems to provide encryption and authentication for 10 transaction interval for CBC scheme.

5.2 Effect of varying cache line size

On a full system scale, though the performance of both schemes are improved in 128 byte cache line size system compared to the 64 byte cache line size, because of the reduced number of back-to-back cache transfers, GCM performs much better when compared to the CBC scheme.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

This work proposes a security mechanism for a symmetric shared memory multiprocessor environment to safeguard the cache-to-cache data transfers. This work is closely related to SENSS [2]. This work focuses on improving the performance of the existing scheme by adapting the GCM-AES mode of operation as the underlying algorithm. The impact of increasing cache line sizes and the effect of varying throughput of the AES units in CBC and GCM modes is studied. Results show that a performance gain in the range of 4X-9X over the CBC scheme is achieved. This scheme consumes less space on chip while providing the same level of security as in SENSS. This scheme can be used to protect the SMP system from common bus attacks. It can detect any fake messages, reordered messages and blocked messages on the communication bus in an SMP environment. It protects the system from hardware security attacks such as modchip installations. It also secures the system against software attacks by providing confidentiality and authentication mechanisms.

Future work

This work considers the implementation of a security hardware unit to secure only the cache-to-cache communication. A model which also secures the processor to memory data flow can be included in an SMP system. The bus traffic increase for providing the security features can also be simulated for GCM scheme. A prototype of the security hardware unit can also be developed on a field programmable gate array (FPGA). This work can also be extended to safeguard the communication among CMP's in a multiple-CMP system.

REFERENCES

[1] Sean W. Smith, Trusted Computing Platforms: Design and Applications, Springer, New York, NY, USA, 2005, pp 21-34.

[2] Zhang, Y., Gao, L., Yang, J., Zhang, X., and Gupta, R. 2005. SENSS: Security Enhancement to Symmetric Shared Memory Multiprocessors. In *Proceedings of the 11th international Symposium on High-Performance Computer Architecture* (February 12 - 16, 2005). HPCA. IEEE Computer Society, Washington, DC, 352-362.

[3] Bruce Schneier, Applied Cryptography, John Wiley & Sons, US, 1996

[4] Thekkath, D. L., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J., and Horowitz, M. 2000. Architectural support for copy and tamper resistant software. *SIGARCH Comput. Archit. News* 28, 5 (Dec. 2000), 168-177.

[5] Suh, G. E., O'Donnell, C. W., Sachdev, I., and Devadas, S. 2005. Design and Implementation of the AEGIS Single-Chip Secure Processor Using Physical Random Functions. *SIGARCH Comput. Archit. News* 33, 2 (May. 2005), 25-36.

[6] http://www-03.ibm.com/security/cryptocards/pcicc/faqcopvalidity.shtml

[7] Yang, L. and Peng, L. 2006. SecCMP: a secure chip-multiprocessor architecture. In *Proceedings of the 1st Workshop on Architectural and System Support For Improving Software Dependability* (San Jose, California, October 21 - 21, 2006). ASID '06. ACM, New York, NY, 72-76.

[8] Data Encryption Standard Federal Information Processing Standards Publication 46-3, http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

[9] Elbirt, A.J.; Yip, W.; Chetwynd, B.; Paar, C., "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol.9, no.4, pp.545-557, Aug 2001.

[10] Diffie, W.; Hellman, M., "New directions in cryptography," *IEEE Transactions on Information Theory",* vol.22, no.6, pp. 644-654, Nov 1976.

[11] http://en.wikipedia.org/wiki/RSA

[12] R. Rivest, MIT Laboratory for Computer Science and RSA Data Security, Inc, The MD5 Message-Digest Algorithm, http://tools.ietf.org/rfc/rfc1321.txt.

[13] Secure Hash Standard, Federal Information Processing Standards Publication 180-1, http://www.itl.nist.gov/fipspubs/fip180-1.htm#FORE_SEC.

[14] http://home.eng.iastate.edu/~zzhang/courses/cpre585-f04/presentations/mikel-reggie.ppt

[15] Gassend, B., Suh, G. E., Clarke, D., van Dijk, M., and Devadas, S. 2003. Caches and Hash Trees for Efficient Memory Integrity Verification. In *Proceedings of the 9th international Symposium on High-Performance Computer Architecture* (February 08 - 12, 2003). HPCA. IEEE Computer Society, Washington, DC, 295.

[16] Shi, W., Lee, H. S., Ghosh, M., Lu, C., and Boldyreva, A. 2005. High Efficiency Counter Mode Security Architecture via Prediction and Precomputation. In *Proceedings of the 32nd Annual international Symposium on Computer Architecture* (June 04 - 08, 2005). International Symposium on Computer Architecture. IEEE Computer Society, Washington, DC, 14-24

[17] Shi, W., Lee, H. S., Ghosh, M., and Lu, C. 2004. Architectural Support for High Speed Protection of Memory Integrity and Confidentiality in Multiprocessor Systems. In *Proceedings of the 13th international Conference on Parallel Architectures and Compilation Techniques* (September 29 - October 03, 2004). PACT. IEEE Computer Society, Washington, DC, 123-134.

[18] http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation#Cipher-block_chaining_.28CBC.29.

[19] Lee, M., Ahn, M., and Kim, E. J. 2007. I2SEMS: Interconnects-Independent Security Enhanced Shared Memory Multiprocessor Systems. In *Proceedings of the 16th international Conference on Parallel Architecture and Compilation Techniques* (September 15 - 19, 2007). PACT. IEEE Computer Society, Washington, DC, 94-103.

[20] Rogers, B., Prvulovic, M., and Solihin, Y. 2006. Efficient data protection for distributed shared memory multiprocessors. In *Proceedings of the 15th international Conference on Parallel Architectures and Compilation Techniques* (Seattle, Washington, USA, September 16 - 20, 2006). PACT '06. ACM, New York, NY, 84-94.

[21] D. McGrew and J. Viega. The Galois/Counter Mode of Operation (GCM). Submission to NIST Modes of Operation Process, 2004.

[22] Morris Dworkin, National Institute of Standards and Technology Special Publication 800-38D Natl. Inst. Stand. Technol. Spec. Publ. 800-38D 37 pages (November 2007).

[23] Morris Dworkin, National Institute of Standards and Technology Special Publication 800-38A 2001 ED Natl. Inst. Stand. Technol. Spec. Publ. 800-38A 2001 ED, 66 pages (December 2001).

[24] Francisco, N.A. Saqib, Arturo, Kaya, *Cryptographic Algorithms on Reconfigurable Hardware*, Springer, New York, NY, USA, 2006, p. 254-256.

[25] Magnusson, P. S., Christensson, M., Eskilson, J., Forsgren, D., Hållberg, G., Högberg, J., Larsson, F., Moestedt, A., and Werner, B. 2002. Simics: A Full System Simulation Platform. *Computer* 35, 2 (Feb. 2002), 50-58.

[26] Martin, M. M., Sorin, D. J., Beckmann, B. M., Marty, M. R., Xu, M., Alameldeen, A. R., Moore, K. E., Hill, M. D., and Wood, D. A. 2005. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Comput. Archit. News* 33, 4 (Nov. 2005), 92-99.

[27] http://www.cs.wisc.edu/gems/

[28] http://www.cs.wisc.edu/gems/doc.html ISCA 2005 Tutorial slides.

[29] David Jakob Fritz, Out of context cache prefetching, Master's Thesis, Oklahoma State University, 2008.

[30] Woo, S. C., Ohara, M., Torrie, E., Singh, J. P., and Gupta, A. 1995. The SPLASH-2 programs: characterization and methodological considerations. *SIGARCH Comput. Archit. News* 23, 2 (May. 1995), 24-36.

[31] http://www.capsl.udel.edu/splash/, The Modified SPLASH-2 Home Page.

[32] Yan, C., Englender, D., Prvulovic, M., Rogers, B., and Solihin, Y. 2006. Improving Cost, Performance, and Security of Memory Encryption and Authentication. *SIGARCH Comput. Archit. News* 34, 2 (May. 2006), 179-190.

[33] Varun Jannepally, Sohum Sohoni, "Fast Encryption and Authentication for Cache-to-Cache Transfers using GCM-AES", to be published in the *Proceedings of the International Conference on Sensors, Security, Software and Intelligent Systems, Coimbatore, India, Jan, 2009.*

[34] MOSI broadcast cache coherence protocol, http://www.cs.wisc.edu/gems/doc/gems-wiki/moin.cgi/Protocols.

[35] https://www.simics.net/

VITA

VARUN JANNEPALLY

Candidate for the Degree of

Master of Science

Thesis: BUS ENCRYPTION AND AUTHENTICATION UNIT FOR SYMMETRIC SHARED MEMORY MULTIPROCESSOR SYTEM USING GCM-AES.

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Hyderabad, Andhra Pradesh, India on September 9, 1984

Education: Received the B.Tech. degree from Jawaharlal Nehru Technological University, AP, India in 2006, in Electronics and Communication Engineering; Completed the requirements for the Master of Science in Electrical Engineering at Oklahoma State University, Stillwater, Oklahoma in December, 2008.

Experience: Research Assistant at Computer Architecture Evaluation, Simulation Research Lab in the School of Electrical and Computer Engineering, Oklahoma State University from December 2006 to December 2008.

Name: Varun Jannepally Date of Degree: December, 2008

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: BUS ENCRYPTION AND AUTHENTICATION UNIT FOR SYMMETRIC SHARED MEMORY MULTIPROCESSOR SYSTEM USING GCM-AES

Pages in Study: 51 Candidate for the Degree of Master of Science

Major Field: Electrical Engineering

Scope and Method of Study: Hardware security mechanisms in uniprocessor and multiprocessor systems have been proposed to safeguard information more efficiently. This work presents a secure architecture model for a symmetric shared memory multiprocessor (SMP) to safeguard the cache-to-cache transfers. This work proposes a hardware security mechanism, which employs Galois Counter Mode (GCM) of advanced encryption standard (AES) and modifies it to work in an SMP environment. The work focuses on why GCM is a better choice over cipher block chaining mode (CBC) which is used in current state of the art systems. It estimates the storage required by the additional hardware unit in both modes of operation. A full system SMP simulation quantifies the performance overhead introduced by the additional hardware unit in both schemes to safeguard the cache-to-cache transfers. The impact of increasing cache line sizes and the effect of varying throughput of the AES units in both the schemes is studied.

Findings and Conclusions: Results show that a performance gain in the range of 4X-9X over the CBC scheme is achieved by using GCM mode of operation. The work shows that the throughput of the AES design has a greater impact on the performance of the CBC scheme. The performance loss is very high in CBC scheme with a lower throughput of the AES design compared to GCM. The performance in CBC scheme varies according to the authentication interval while authentication interval does not affect the GCM scheme, thus providing higher security. The presented work using GCM consumes less space on chip providing the same level of security as in the CBC scheme.

ADVISER'S APPROVAL: Dr. Sohum Sohoni