Fast Secure Communications in Shared Memory Multiprocessor Systems

Manhee Lee, Minseon Ahn, *Student Member*, *IEEE*, and Eun Jung Kim, *Member*, *IEEE*

Abstract—Protection and security are becoming essential requirements in commercial servers. To provide secure memory and cache-to-cache communications, we presented Interconnect-Independent Security Enhanced Shared Memory Multiprocessor System (I^2SEMS), mainly focusing on how to manage a global counter to encrypt, decrypt, and authenticate data messages with little performance overhead. However, I^2SEMS was vulnerable to replay attacks on data messages and integrity attacks on control and counter messages. This paper proposes three authentication schemes to remove those security vulnerabilities. First, we prevent replay attacks on data messages by inserting Request Counter (RC) into request messages. Second, we also use RC to detect integrity attacks on control messages. Third, we propose a new counter, referred to as GCC Counter (GC), to protect the global counter messages. We simulated our design with SPLASH-2 benchmarks on up to 16-processor shared memory multiprocessor systems by using Simics with Wisconsin multifacet General Execution-driven Multiprocessor Simulator (GEMS). Simulation results show that the overall performance slowdown is 4 percent on average with the highest keystream hit rate of 78 percent.

Index Terms—Multiprocessor systems, shared memory, interprocessor communications, data encryption, authentication.

1 Introduction

Recent malicious attacks to many research/commercial servers have made protection and security essential requirements in all computer systems. Especially, server systems operated by institutions dealing with highly confidential data such as banks, military, or government agents need absolute security because even one incident of information leakage could result in very serious problems. Therefore, those institutions should seek effective schemes to prevent/block any possible security attacks on their computer systems. Considering that most of recent server systems have multiple processors with shared memory, we believe that there should be appropriate protections on the memory and communications between processors. Moreover, since performance compromise should be the last resort in the server systems, any security schemes free of performance overhead are much desirable despite additional hardware costs.

There are two types of possible attacks: software and physical attacks. Software attacks exploit software vulnerabilities of the operating system (OS) and server applications such as web and database services. Physical attacks that are our focus capture or modify data from the system memory or system bus through external devices physically attached to the system. Although high-end servers are often assumed to be secure against such physical attacks, that assumption cannot be sustained in many situations; it is possible that inside personnel turn against their organization to steal data, and, moreover, there exist real snooping devices attachable to system buses [2], [3]. To mitigate the physical attacks, memory encryption and authentication have been widely investigated for uniprocessor systems [4], [5], [6], [7], [8], [9], [10]. Appendices A.I and A.II, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.131, have extensive literature reviews of uniprocessor and multiprocessor secure models, respectively. However, the uniprocessor security models are not sufficient to design secure multiprocessor systems due to the absence of cache-to-cache communication protection.

When multiple processors share data, a cache coherence protocol guarantees data consistency among caches. Without protecting cache-to-cache communication, memory protection would be useless. A few studies raised this question and provided fast encryption and authentication techniques that can be deployed on bus-based shared memory multiprocessors [11], [12]. One common feature in the studies is that a bus sequence number that counts every message on a shared bus is used as a counter to generate an encrypted counter, called *keystream*[1] [13]. Since all communications on the shared bus are broadcasted to all processors, they can predict the next bus sequence number to generate the same keystream in advance, thus turning a complex encryption/decryption operation into a simple XOR (eXclusive OR) operation. However, since these schemes are dependent on the shared bus, they cannot be easily extended to general multiprocessor systems having different underlying networks or distributed shared memory (DSM).

A recent study by Rogers et al. proposed a novel mechanism for protecting cache-to-cache communication in distributed shared memory multiprocessors as a remedy for this problem [14]. However, since their design focused on the directory-based cache coherence protocol used for point-to-point communication, their idea cannot be directly applied to

- *M. Lee is with the National Security Research Institute, Daejeon 305-600, South Korea. E-mail: manheelee@ensec.re.kr.*
- *M. Ahn and E.J. Kim are with the Department of Computer Science, Texas A&M University, College Station, TX 77843. E-mail: {msahn, ejkim}@cse.tamu.edu.*

Manuscript received 9 Dec. 2008; revised 14 Nov. 2009; accepted 10 Mar. 2011; published online 4 Apr. 2011.
Recommended for acceptance by J. Zhang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-12-0476.
Digital Object Identifier no. 10.1109/TPDS.2011.131.

1. Also known as one-time-pad (OTP) in previous literature.

the systems with other cache coherence protocols used for multicasting/broadcasting communication such as the token coherence protocol [15], [16]. This limitation makes it impossible to get additional performance improvements which could be up to 18 percent by using multicast network with directory-based cache coherence protocol [17], [18], [19].

To provide diverse multiprocessor architectures with a more general security model, in our previous study we proposed an interconnect independent and cache coherence protocol independent security model for shared memory multiprocessor systems, referred to as Interconnect-Independent Security Enhanced Shared Memory Multiprocessor System (I^2SEMS) [1]. I^2SEMS was designed to prevent various attacks including eavesdropping, injecting, and modifying messages. Its main idea was that a *global counter controller (GCC)* assigns counters upon a request from a processor and broadcasts the assignment to the other processors. When receiving the counter assignment, processors generate the counters' keystreams and store them in a *keystream queue* or a *keystream pool*, depending on whether the keystream will be used for encryption or decryption. To minimize the use of new counters, a *keystream cache* stores both counters and keystreams so that it can reuse them only when the reuse is safe; the same counter and its keystream can be used for the same data block securely. Simulation results showed that the overall performance slowdown is 4 percent on average, and the keystream pool hit rate is as high as 78 percent, meaning that 78 percent of incoming messages are decrypted without delay. To our best knowledge, I^2SEMS was the first attempt to support the protection of shared memory multiprocessor systems without any restrictions on communication types and cache coherence protocols. However, I^2SEMS was vulnerable to replay attacks on data messages and integrity attacks on control and counter messages because data message authentication was per-packet basis and there was no authentication method for control and counter messages.

In this paper, we focus on enhancing security strength of I^2SEMS by removing those security vulnerabilities. First, we propose Request Counter (RC) that is inserted into request messages to prevent replay attacks on data messages. By using RC to generate MAC for a reply message, any replay attacks among L2 caches and memory can be prevented. Second, since the previous study did not protect control messages, integrity attacks on control message like request messages modification can cause data inconsistency in L2 caches or memory. For this, we propose that control message carries MAC as well as RC, which will help detect any control message attacks before the attacks cause cache inconsistency. Third, we also propose to insert GCC Counter (GC) to authenticate counter messages, which prevents counter replay and modification attacks on counter messages.

The remainder of the paper is organized as follows. First, Section 2 introduces our threat model. In Section 3, we explain the architectural design of I^2SEMS briefly. Its security analysis is presented in Section 4, followed by the concluding remarks in Section 5.

2 Threat Model

Before enhancing security of a system, it is necessary to classify secure and insecure components within the system. Processing core, registers, caches, and control and data paths in a processor are considered to be secure. We assume that controllers for memory and I/O are secure, while other off-chip components including memory, memory bus, and I/O devices are insecure. We also assume that interconnection networks that connect multiple processors and memory banks are insecure.

Through these insecure components, the following physical attacks are possible. First, any wiretapping device attached to the memory bus or interconnection network can get information about memory transactions and processor-to-processor communications. Since the interconnection network cables are often exposed to the outside of the system to connect multiple nodes packaged in one or several racks, they are vulnerable to the eavesdropping attack. Even optical networks are vulnerable to the attack through a simple fiber optic splitter or coupler [20]. To make servers resistant to this attack, it is necessary to provide the confidentiality service where all communications from processors are encrypted. Second, assuming that attackers are much more determined and well equipped, they can further inject or modify messages to the systems. This attack includes injecting new data messages, modifying data in messages in transit, and replaying old messages. To prevent this attack, the authentication service needs to make sure that all data messages are genuine, not spoofed, modified, or replayed by illegal devices. Third, in some cases attackers may intend to undermine the availability of the systems by injecting garbage messages. These messages will be finally discarded since authentication information will not be correct, but they will have negative effects on the system performance because of possible congestion that would occur in the interconnection network or controllers for cache and memory. This is similar to the denial of service attack in the Internet. However, this attack does not seem much attractive to attackers not only because attackers get little benefit from the attack, but also because a sudden performance drop or a system crash would lead to a thorough search for attached devices. Therefore, the availability service is not our focus.

3 Architectural Design of I^2SEMS

Fig. 1 shows the architecture of I^2SEMS and its data flow.

The system-wide GCC is located in the memory controller.[2] In addition to a unified L2 cache, called System Cache, each processor has a keystream queue, a keystream cache, and a keystream pool. The GCC assigns counters upon a request from a processor and broadcasts the assignment to the other processors. Keystreams in the keystream queue and the keystream cache are used to encrypt outgoing messages, while those in the keystream pool to decrypt incoming messages. Explanations on data flow between components are in the followings.

2. Depending on the location of the memory controller, the GCC can reside off or on chip. When a processor has an off-chip memory controller like most Intel processors, a multiprocessor system using such processors will have the GCC in its off-chip memory controller usually located in the north bridge (or Memory Controller Hub). However, some recent processors such as IBM's POWER5 and AMD's Athlon 64 and Opteron processors have the on-chip memory controller for faster memory access. In this case, by enabling one processor's GCC, a single GCC will be located in an integrated on-chip memory controller.

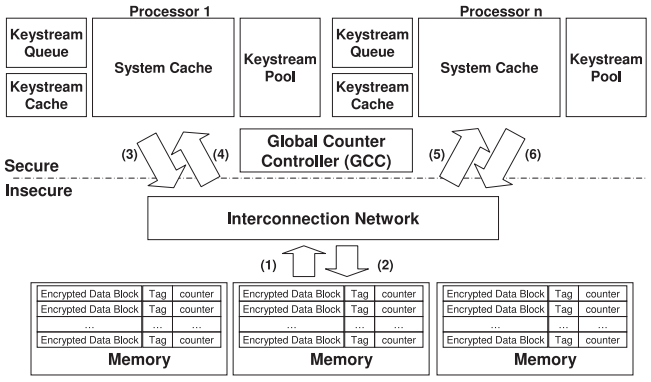

Fig. 1. I^{2}SEMS security model.

When a processor transmits a data block to other caches or memory as shown in Fig. 1 (3) and (6), it needs to encrypt the block. If the cache block is modified or exclusively owned by the processor, a new counter and its keystream are popped from the keystream queue to encrypt the cache block. In many cases, the cache status changes to *Owned*, which means that this processor is in charge of replying requests that other processors send to get this block's copy. In this status, the previously used counter and keystream can be reused because the data did not change. For this purpose, after using the new counter and its keystream, we store them in the keystream cache. Therefore, when a cache block that will be transmitted has the Owned status, the processor looks up the keystream cache. If not found, a new counter and its keystream need to be popped from the keystream queue. This operation is described in Algorithm 1 (3) and (6). We use Advanced Encryption Standard (AES) as a basic block cipher due to its security strength and recent research on fast hardware implementation [21].

When a processor receives a data block as shown in Fig. 1 (4) and (5), it tries to find a keystream in the keystream pool. Simultaneously it begins to generate p keystreams by using the pipelining ability of AES logic. If found in the keystream pool, referred to as a *keystream hit*, the counter's keystream is used to decrypt the arrived message. If not found, called a *keystream miss*, the keystream will be available in an AES latency because the keystream generation already started regardless of keystream hit or miss. All generated keystreams except the first one are stored in the keystream pool, hoping that subsequent counters may be used in the near future. p is called *prediction depth* and set to five as default. When the remaining number of keystreams in the keystream queue becomes less than *counter reserve* (CR), it sends a request to the GCC to get new counters. CR is the number of counters that the GCC assigns at a time. Upon arrival, the GCC assigns and sends new counters to the requesting processor. The GCC also broadcasts the assignment to the other processors for them to precompute the counters' keystreams. When a reply or a broadcast message from the GCC arrives at a processor in (4) and (5), the processor generates the CR number of keystreams and store them in the keystream queue or the keystream pool, respectively. This operation is summarized in Algorithm 1 (4) and (5). All other operations shown in Fig. 1 are also described in Algorithm 1.

```
(1)
(EncryptedDataBlock, counter) = Memory.read(address);
Send(address, EncryptedDataBlock, counter);

(2)
(address, EncryptedDataBlock, counter) = Receive();
Memory.write(address, EncryptedDataBlock, counter);

(3,6)
(DataBlock, counter, Keystream, status) =
   SystemCache.Read(address);
if (isNewKeystreamNecessary(status)){
        (counter, Keystream) = KeystreamQueue.pop();
        Send(address, DataBlock ^ Keystream, counter);
        KeystreamCache.store(counter, Keystream);}
else if (KeystreamCace.IsKeystreamPresent (counter)){
        (counter, Keystream) =
        KeystreamCache.read(counter);
        Send(address, DataBlock ^ Keystream, counter);}
else    {(counter, Keystream) = KeystreamQueue.pop();
        Send(address, DataBlock ^ Keystream, counter);
        KeystreamCache.store(counter, Keystream);}

(4,5)
if (Data block arrived){
        (address, EncryptedDataBlock, counter) =
        Receive();
        PARALLEL{ // Generate p Keystreams from
          counter to counter + p^1
          counter [0,...,p − 1] = {counter,...,
          counter + p − 1};
          Keystream[0,...,p − 1] = AES(counter[0,...,
          p − 1], secretkey);
          KeystreamPool.store(counter[1,...,p − 1],
          Keystream[1,...,p − 1])};
        PARALLEL {if (KeystreamPool.
        IsKeystreamPresent(counter)) {
          Keystream = KeystreamPool.read (counter);
          DataBlock = EncryptedDataBlock^
          Keystream;
          SystemCache.store(address, DataBlock);
          KeystreamPool.delete (counter);}
        else {//wait until for an AES latency
          DataBlock = EncryptedDataBlock^
          Keystream[0];
          SystemCache.store (address, DataBlock);}}}
else if (GCC reply arrived){
        (counter) = Receive();
        counter[0,...,CR − 1] = {counter-CR + 1,...,
        counter;}
        Keystream[0,...,CR − 1] = AES (counter[0,...,
        CR − 1], secretkey);
        KeystreamQueue.enqueue (counter[0,...,
        CR − 1], Keystream[0,...,CR − 1]);}
else if (GCC broadcast arrived){
        (counter) = Receive();
        counter[0,...,CR − 1] = {counter-CR +
        1,...,counter;}
```

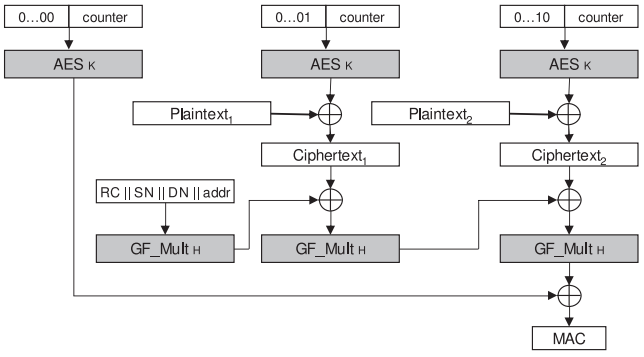

Fig. 2. Galois/Counter mode.

Keystream$[0,\ldots,\mathrm{CR}-1]=$ AES(counter$[0,\ldots,\mathrm{CR}-1]$, secretkey);
KeystreamPool.store(counter $[0,\ldots,\mathrm{CR}-1]$, Keystream$[0,\ldots,\mathrm{CR}-1]$);}

Algorithm 1. Pseudocode for I^{2}SEMS Security Model

4 Secure Communications of I^{2}SEMS

In this section, we explain how I^{2}SEMS provides secure communications. First, we discuss how data messages are protected in detail. Then, we describe methods to protect other messages such as GCC messages and control messages.

4.1 Protection on Data Messages

By architectural support of I^{2}SEMS, we can assume that a sender and its receiver share the same counter. Even if the counter is not correctly predicted in the receiving node, it will be available soon after the message arrives. Fig. 2 shows how to encrypt a data message carrying a 32-byte plaintext and generate MAC using GCM. Decryption has a similar design except that ciphertext is input while plaintext is output of XOR. AES_K is hardware implementation of AES algorithm with a secret key, K. Since the block size of AES is 128 bits, the 32-byte plaintext is divided into two subplaintexts, $\mathrm{Plaintext}_1$ and $\mathrm{Plaintext}_2$. Larger cache blocks will be divided into more number of subplaintexts. The two plaintexts are encrypted in parallel by XORing with two keystreams, making two ciphertexts, $\mathrm{Ciphertext}_1$ and $\mathrm{Ciphertext}_2$. The ciphertexts are concatenated to make a 32-byte encrypted payload. Additional operations for making MAC are three sets of $\mathrm{GF\_Mult}_H$ and XOR. $\mathrm{GF\_Mult}_H$ denotes Galois field (GF) multiplication in GF(2^{128}) using a hash key H, which is the encrypted zero counter, AES $(0^{128}, K)$, with a secret key K. By using a pipelined implementation method [22] that takes one clock cycle for one $\mathrm{GF\_Mult}_H$, the total additional delay will be six clock cycles, conservatively assuming XOR takes one clock cycle too. This authentication delay can be ignored as long as it is overlapped with other operations. In our study, this delay is overlapped with transmission delay at the sender side and cache access time at the receiver side. A sender needs a transmission delay to put a 32-byte-long cache block on wire. The 3.2 GB/sec network that we use for simulation takes 10 ns to transmit one block, so the authentication delay can be ignored. At the receiver, an arriving cache block needs to be written to the system cache. Since the generation of MAC requires not plaintext but ciphertext, it begins in parallel with the system cache access. Our simulation's system cache access time is 6 ns, so if there is an authentication fails, it can raise an alert before the next cache access starts. Therefore, in our study, we ignore the authentication delay.

4.1.1 Encryption of Data Message

Since AES is considered to be secure without any serious weakness until now, the plaintext will be kept secret. Note that in Fig. 2 one 64-bit global counter is used to compose three 128-bit counters to generate three different keystreams. In this example, I^{2}SEMS appends three different prefixes: $0^{62}00$, $0^{62}01$, and $0^{62}10$, by using the 62-bit common prefix. If the 64-bit counter is globally unique and all components including processors and the memory share the same 62-bit common prefix, every counter used as an input of AES is unique in the whole system. To prevent the global counter from starting with the same number, the common prefix will be increased by one and distributed to all processors and the memory controller at every system booting as described in [23].

The last concern of confidentiality of data messages is counter-wrap-around. If the 64-bit global counter wraps around, two different data blocks will use the same keystream. We estimate the expected wrap-around time by considering the maximum counter usage rate. Let's suppose a system has 2^6 processors using a 3.2 GB/sec network and a 32 byte-long message and counter is 64 bits long. When one half of the processors keep sending modified cache blocks at its full speed and the other half are receiving them, this system will consume 2^{32} keystreams per second, thus taking about 2^7 years to wrap around the counter in the worst case. Considering that I^{2}SEMS assigns new counters only when necessary, we expect the actual counter wrap around time will be far longer than the worst-case estimation. Therefore, the 64-bit counter will be big enough to support system lifetime of large multiprocessor systems. Even if a huge multiprocessor system runs for exceptionally long period of time enough to reach its theoretical wrap-around-time, the system can be simply rebooted. Therefore, I^{2}SEMS can protect data messages from eavesdropping attacks.

4.1.2 Authentication of Data Message

Since the security strength of GCM is the same as the strength of its block cipher [24], the authentication strength is greatly improved. Owing to this, any modification and spoofing attack will not be successful because the hacker does not know the secret key K. Additionally, the source node (SN) and destination node (DN) addresses are input of $\mathrm{GF\_Mult}_H$, so the redirection attack is not possible, either.

The last authentication concern is replay attacks on data messages. According to why a data message transfer is necessary, there are different replay attack scenarios as shown in Fig. 3. The first relay attack point is on data messages between two L2 caches, which are numbered as a_1, a_2, and a_3. When an L2 cache read miss occurs in C_1 and its authoritative copy is stored in other L2 cache, C_2, a read request a_1 is delivered to C_2. C_2 sends the requested block a_2 to C_1, and then C_1 sends an acknowledgment message a_3 to C_2. A possible replay attack is as follows. Suppose a hacker captures two messages, a_1 and a_2, for a specific address. Later, if C_1 evicts the cache block and needs it again, C_1 will send a

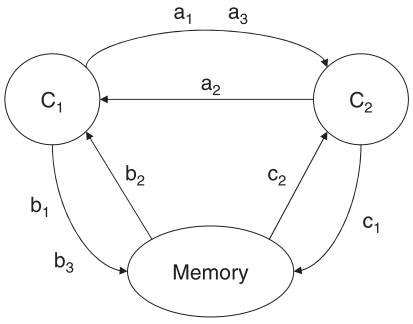

Fig. 3. Data messages transfer.

new request a_1' to C_2 with the same address. The hacker can drop a_1', replay a_2 to C_1, and drop a new acknowledgment a_3'. This attack will be successful because a_2 has a correct MAC with the same node addresses and memory address. To prevent this attack, we propose that every read request from L2 cache should need to carry a unique sequential number, referred to as $Request\ Counter$. Whenever a read request is sent out, the RC is increased and attached to the request message. In the above example, a_1 will carry RC_{C_1}. C_2 includes RC_{C_1} in generation of its MAC. This is why the first input of $\mathrm{GF\_Mult}_H$ is $RC_{C_1}\|SN\|DN\|addr$. Let's look at the replay attack again. a_1' will carry a new RC'_{C_1} that is different from RC_{C_1} used in a_1. The authentication of replayed a_2 will not be successful because C_1 expects an MAC generated with $RC'_{C_1}\|SN\|DN\|addr$ but the replayed a_2 carries an MAC generated with $RC_{C_1}\|SN\|DN\|addr$. Since C_1 keeps increasing RC_{C_1} per memory request, no two RC_{C_1} will be the same within its wrap-around-time. To estimate the wrap-around-time of RC, we need to fix the number of nodes and the size of memory. Suppose there are 2^6 processors and the memory size is 2^8 GB. Since $|SN|$ is 6 bits, $|addr|$ is 38 bits, and the input width of $\mathrm{GF\_Mult}_H$ is 128 bits, $|RC|$ can be 78 bits. Even if RC's width is 59 bits and a cache keeps sending 32-bits messages using a new RC in 3.2 GB/sec network, its wrap-around-time is approximately 2^7 years like that of the global counter. Thus, the first 19 bits can be used as a prefix that is incremented by one at the booting stage as we did for the global counter. After 2^{19} rebootings, we can see the same RC from the same cache, which will be almost impossible to happen in real systems. Therefore, our scheme can prevent replay attacks on data message between caches.

The second relay attack point is on data messages from the memory to L2 cache, which are numbered as b_1, b_2, and b_3. When C_1 wants to access the memory for read, a memory request b_1 is delivered to the memory. The memory sends the requested block b_2, and then C_1 sends an acknowledgment message b_3 to the memory. Since communication steps are exactly the same as those between two caches, replay attack scenario and its prevention are also the same.

The third replay attack point is on data messages from L2 cache to the memory. The transfer happens when a modified block is evicted from the cache and written back to the memory. This replay attack is different from the two previous attacks because there will no read request such as a_1 and b_1. C_2 directly sends a write-back request c_1 with data in Fig. 3. A possible replay attack is as follows. Suppose a hacker captured c_1 for a specific address. Then, the hacker can recover the old data anytime by replaying c_1. Since c_2' will also be dropped, the attack will go unnoticed by cache coherence protocols. To prevent this replay attack, we insert the RC in the request as well as use RC to make an MAC. That is, C_2 uses $RC_{C_2}\|SN\|DN\|addr$ for its MAC, and c_1 carries RC_{C_2}. The memory should remember each cache's last RC so that it can raise a security alert when an arriving RC is smaller than or equal to the last RC. Therefore, I^{2}SEMS prevents the replay attacks on data messages.

For the complete protection of memory, we additionally adopt Merkle tree or a recently proposed Bonsai Merkle tree [9], [25]. Even though it is out of our research scope, Merkle tree can prevent memory authentication attacks like memory block replacements. Note that since I^{2}SEMS prevents replay attacks, we can ease a hard requirement of Merkle tree authentication that a root authentication code needs to be stored in processor/cache that is assumed to be secure. In our design, as long as the secure memory controller can store the root authentication code, making sure that the memory contents are not modified or replaced through physical attacks, I^{2}SEMS can guarantee secure communications between caches and memory. This will remove possible overhead for updating the root authentication codes in multiple processor systems at every memory writing.

Until now, we implicitly assumed that the underlying general interconnect provides in-order delivery, so we simply compared the most recent RC and a newly arrived RC. When there are skipped RCs in arrived packets, it implies that some packets were lost. However, when a network uses an adaptive routing, two packets with the same source and destination may take different paths depending on network congestion and thus they can arrive out of order (OOO). In this case, it is necessary to store OOO packets' RCs in a buffer until all the middle packets arrive at the destination. We need to consider two issues to apply the scheme: the size of buffer and replay attack detection. First, according to [26] that claims to be the first to investigate the degree of OOO delivery in interconnection networks, OOO delivery happens only when network is saturated and a 4 MB buffer can rearrange OOO packets to ensure in-order delivery in a 128-node system interconnected with InfiniBand. Even though their network is not the same as our general interconnection networks, we think it will be enough to get a rough estimate. If applied to our design, the buffer's size will be 128 KB because only 64-bit RC, not 256 bytes InfiniBand data packet, is stored in the buffer. When a series of continuous packets arrive in order after a skipped counter, only the starting and ending RCs of the continuous packets can be stored with a tag indicating that all the middle RCs arrived in order. Since this simple compression scheme will further reduce its space requirement, we expect the buffer size will be much smaller than 128 KB. Second, replayed attack can be detected in OOO network by use of the buffer. When a replayed packet arrives at the destination, there will be three cases. First, the original packet is already checked in, meaning its preceding packets and the original packets arrived already. Thus, the replayed packet's RC is smaller than or equal to the largest RC that arrived in order so far. By putting the largest RC in the buffer as a head RC, the buffer can detect this replay attack instantly. Second, when the original packet stays in the buffer, the replayed packet will be detected easily because its RC is already marked as arrived in the buffer. The last case is that the replayed packet arrives earlier than

the original packet. Since the replayed packet is exactly the same as the original one, there is no security problem to use the replayed packet as the original one. Later when the original one finally arrives, a replay attack will be detected. Therefore, our replay attack detection scheme also works for OOO network with a reasonable size of RC buffer.

4.2 Protection on Control Messages

Control messages like invalidation or write requests need appropriate protection for the following reason. In Fig. 3, suppose that C_2 is an exclusive owner of a cache block of an address. A hacker spoofs a write request with C_1 as the source node and C_2 as the destination node. C_2 would invalidate its block by understanding that C_1 wants to have an ownership to overwrite the block. This attack results in cache incoherence because the cache coherence protocol considers C_2 as the owner of the address, but C_2 already invalidated the block. For this, we propose to authenticate control messages by using local counters. Each cache has its own 64-bit counter and increases it by one at every control message. This local counter instead of the global counter is used as an input of AES_K in Fig. 2. Since there is no data to encrypt, $\mathrm{Plaintext}_1$ and $\mathrm{Plaintext}_2$ will be simply 0^{128}. Instead, two ciphertexts will be used only for generation of MAC. Still, the input of $\mathrm{GF\_Mult}_H$ is $RC\|SN\|DN\|addr$, and the message needs to carry RC. Like the global counter, the local counter will be initialized at every booting stage to prevent every local counter from starting with the same number. Note that we do not consider performance implication of authentication of control messages. That is because we can presume this will not cause significant performance overhead by using the combination of local counter prediction and speculative authentication. As we did in the global counter, each cache can predict the next local counter and precompute its keystream so that it can instantly generate incoming control message's MAC. According to [14], there is a temporal locality in cache coherence communications, meaning that a cache communicates with a relatively small number of caches at a time. Therefore, the prediction hit rate is expected to be very high. Furthermore, in case of a local counter miss, which could incur an authentication delay, we use authentication speculation. In other words, when a keystream to authenticate a control message is not available, instead of waiting for the generation of the keystream, the receiving cache simply checks RC first, and then processes the control message as requested. Considering that the authentication will be successful when there is no security attack, the authentication of control messages will not affect the overall performance.

4.3 Protection on Counter Messages

Since the whole security mechanism depends on the counters assigned by the GCC, the protection of the GCC and its counter distribution is critical. Therefore, the GCC itself needs to be tamper-resistant so that its operation and secret keys will be protected. So, we need to provide appropriate protection on the counter messages to remove any possibility of modifying, forging, or replaying attacks.

First of all, counters carried on counter messages need authentication only, not encryption, because even if the counter is available to a hacker he cannot generate or predict any keystreams without knowing a secret key in use. For authenticating counter messages, we propose a new counter, $GCC\ Counter$, similar to the local counter in protection of control messages. When a keystream queue needs more global counters, it sends a counter request with an RC to prevent a possible replay attack on counter request and reply messages. Upon a counter request, the GCC first makes a reply message to the requester. The input of AES_K uses GC, $\mathrm{Plaintext}_1$ is a new global counter, $\mathrm{Plaintext}_2$ is zero, and the input of the input of $\mathrm{GF\_Mult}_H$ is $RC\|GCC\|DN$. Then, the GCC makes a broadcast message to all keystream pools. The input of AES_K uses $GC+1$, plaintexts are the same, and the input of $\mathrm{GF\_Mult}_H$ is $GCC\|*$ where $*$ is a special character for the broadcasting address. The GCC increases GC by two for next counter requests. A replay attack on counter broadcast messages will not be interesting to a hacker since it will end up with broadcasting an old counter that can be simply dropped by comparing the arriving global counter and the most recent one. OOO delivery interconnects may allow a broadcast message to arrive earlier than its reply message or its previous broadcast messages. In the former case, the cache keeps waiting for a reply message by discarding the broadcast message. In the latter case, if there is skipped global counters in the broadcasted messages, keystream pools assume that OOO delivery or packet loss happens, so it can precompute all the global counters' keystreams including the skipped counters. With regard to performance of authentication of GCC messages, keystream queues and keystream pools authenticate most GCC messages with little delay by correctly predicting GC because the GCC keeps broadcasting GC to all processors.

5 Conclusions

In this paper, we first briefly introduced the architecture of I^{2}SEMS that guarantees confidentiality and integrity of shared memory and cache-to-cache communication in multiprocessor systems. New additional hardware components including GCC, keystream queue, keystream cache, and keystream pool work together to minimize encryption and decryption delay. The performance overhead of I^{2}SEMS is 4 percent on average and the keystream pool hit rate is as high as 78 percent where keystream pool hit rate is the ratio of instantly decrypted and authenticated messages out of all incoming messages. For detailed performance analysis with various configurations using various cache coherence protocols and different keystream pool sizes, please see Appendix B, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.131.

Then, this paper mainly focused on the security aspect of I^{2}SEMS which was missed in our previous study. The followings are important conclusions of this work: First, I^{2}SEMS successfully protects data messages from eavesdropping attacks by encrypting all data messages with the 64-bit globally unique counter. The counter is not likely to wrap around during system lifetime of fairly large multiprocessor systems under normal usage. For example, in a 2^6 processors system, the system should run 2^7 years without a rebooting before the global counter wraps around.

Second, the previously proposed authentication method that calculates MAC per packet can prevent any modification and spoofing attack, but the per-packet authentication was vulnerable to replay attacks. To solve this problem, we proposed that any read request from L2 cache needs to

carry a Request Counter. By using RC to generate MAC, I²SEMS detects replay attacks on data messages between L2 caches as well as between L2 cache and memory. Furthermore, with a reasonable size of RC buffer, this scheme is also able to detect replay attacks in out-of-order delivery interconnect network where it is hard to differentiate between replayed packets and OOO packets.

Third, the previous design of I²SEMS depended on cache coherence protocols for detecting attacks on control messages because they will cause cache inconsistency, and cache coherence protocols will figure it out afterward. To overcome this problem, this paper presented a new method to detect control message attacks before the attacks cause the cache inconsistency; control message also carries an RC, and its MAC is generated using a local counter, not a global counter. Different from data messages, control messages are not shared by multiple processors or maintained for later use, so the local counter is proper to authenticate control messages.

Lastly, we elaborated on counter message protection. Without proper protection schemes, counter messages created by GCC can be modified, forged, and replayed, possibly resulting in early global counter wrap-around or disruption of security operation of each processor. To prevent this, we proposed GCC Counter to protect counter messages. By using GC to generate MAC of counter messages, I²SEMS can prevent authentication attacks on counter reply and counter broadcasting messages even in OOO interconnect network.

We are currently examining a number of possible expansions to this work. First, we were unable to analyze the realistic web-based or database servers. In-depth experiments with those server applications should fortify the I²SEMS design. Next, we would like to expand our research to much larger DSM with/without reconfigurable interconnects [27] and to new multiprocessor architectures such as Chip Multiprocessor (CMP) systems.

ACKNOWLEDGMENTS

A preliminary version of this paper was presented at the 16th International Conference on Parallel Architectures and Compilation Techniques (PACT), September 2007 [1]. This work was partially supported by US National Science Foundation (NSF) CCF-0541360 and CCF-0541384.

REFERENCES

[1] M. Lee, M. Ahn, and E.J. Kim, "I²SEMS: Interconnects-Independent Security Enhanced Shared Memory Multiprocessor Systems," *Proc. 16th Int'l Conf. Parallel Architecture and Compilation Techniques (PACT '07),* pp. 94-103, 2007.

[2] A. "bunnie" Huang, "The Trusted PC: Skin-Deep Security," *Computer,* vol. 35, no. 10, pp. 103-105, Oct. 2002.

[3] *Hacking the Xbox: An Introduction to Reverse Engineering.* No Starch Press, 2003.

[4] B. Gassend, G.E. Suh, D. Clarke, M. van Dijk, and S. Devadas, "Caches and Hash Trees for Efficient Memory Integrity Verification," *Proc. Ninth Int'l Symp. High-Performance Computer Architecture,* pp. 295-306, 2003.

[5] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural Support for Copy and Tamper Resistant Software," *Proc. Ninth Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX),* pp. 168-177, 2000.

[6] W. Shi, H.S. Lee, M. Ghosh, C. Lu, and A. Boldyreva, "High Efficiency Counter Mode Security Architecture via Prediction and Precomputation," *Proc. 30th Ann. Int'l Symp. Computer Architecture,* pp. 14-24, 2005.

[7] G.E. Suh, C.W. O'Donnell, and S. Devadas, "Aegis: A Single-Chip Secure Processor," *IEEE Design and Test of Computers,* vol. 24, no. 6, pp. 570-580, Nov./Dec. 2007.

[8] G.E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "Efficient Memory Integrity Verification and Encryption for Secure Processors," *Proc. 36th Ann. IEEE/ACM Int'l Symp. Microarchitecture,* pp. 339-350, 2003.

[9] C. Yan, B. Rogers, D. Englender, Y. Solihin, and M. Prvulovic, "Improving Cost, Performance, and Security of Memory Encryption and Authentication," *Proc. 33rd Ann. Int'l Symp. Computer Architecture,* pp. 179-190, 2006.

[10] J. Yang, Y. Zhang, and L. Gao, "Fast Secure Processor for Inhibiting Software Piracy and Tampering," *Proc. 36th Ann. IEEE/ACM Int'l Symp. Microarchitecture,* pp. 351-360, 2003.

[11] W. Shi, H.-H. S. Lee, M. Ghosh, and C. Lu, "Architectural Support for High Speed Protection of Memory Integrity and Confidentiality in Multiprocessor Systems," *Proc. 13th Int'l Conf. Parallel Architecture and Compilation Techniques,* pp. 123-134, 2004.

[12] Y. Zhang, L. Gao, J. Yang, X. Zhang, and R. Gupta, "SENSS: Security Enhancement to Symmetric Shared Memory Multiprocessors," *Proc. 11th Int'l Symp. High Performance Computer Architecture,* pp. 352-362, 2005.

[13] H. Lipmaa, P. Rogaway, and D. Wagner, "CTR-Mode Encryption," *Proc. NIST Workshop Modes of Operation,* 2000.

[14] B. Rogers, M. Prvulovic, and Y. Solihin, "Efficient Data Protection for Distributed Shared Memory Multiprocessors," *Proc. 15th Int'l Conf. Parallel Architecture and Compilation Techniques (PACT '06)* pp. 84-94, 2006.

[15] E.E. Bilir, R.M. Dickson, Y. Hu, M. Plakal, D.J. Sorin, M.D. Hill, and D.A. Wood, "Multicast Snooping: A New Coherence Method Using a Multicast Address Network," *Proc. 26th Ann. Int'l Symp. Computer Architecture (ISCA '99),* pp. 294-304, 1999.

[16] M.M. Martin, M.D. Hill, and D.A. Wood, "Token Coherence: A New Framework for Shared-Memory Multiprocessors," *IEEE Micro,* vol. 23, no. 6, pp. 108-116, Nov./Dec. 2003.

[17] H.-C. Hsiao and C.-T. King, "An Application-Driven Study of Multicast Communication for Write Invalidation," *The J. Supercomputing,* vol. 18, no. 3, pp. 279-304, 2001.

[18] M.P. Malumbres, J. Duato, and J. Torrellas, "An Efficient Implementation of Tree-Based Multicast Routing for Distributed Shared-Memory Multiprocessors," *Proc. Eighth IEEE Symp. Parallel and Distributed Processing (SPDP '96),* pp. 186-189, 1996.

[19] D.K. Panda, S. Singal, and P. Prabhakaran, "Multidestination Message Passing Mechanism Conforming to Base Wormhole Routing Scheme," *Proc. First Int'l Workshop Parallel Computer Routing and Comm. (PCRCW '94),* pp. 131-145, 1994.

[20] K. Shaneman and S. Gray, "Optical Network Security: Technical Analysis of Fiber Tapping Mechanisms and Methods for Detection and Prevention," *Proc. IEEE Military Comm. Conf. (MILCOM '04),* vol. 2, pp. 711-716, 2004.

[21] Nat'l Inst. of Science and Technology, "Advanced Encryption Standard (AES)," FIPS 197, 2001, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf; Accessed Apr. 2008.

[22] B. Yang, S. Mishra, and R. Karri, "A High Speed Architecture for Galois/Counter Mode of Operation (GCM)," *Cryptology ePrint Archive, Report 2005/146,* 2005.

[23] R.B. Lee, P.C.S. Kwan, J.P. McGregor, J. Dwoskin, and Z. Wang, "Architecture for Protecting Critical Secrets in Microprocessors," *Proc. 32nd Ann. Int'l Symp. Computer Architecture (ISCA '05),* pp. 2-13, 2005.

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

[25] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin, "Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly," *Proc. 40th Ann. IEEE/ACM Int'l Symp. Microarchitecture,* pp. 183-196, 2007.

[26] J.C. Martinez, J. Flich, A. Robles, P. Lopez, J. Duato, and M. Koibuchi, "In-Order Packet Delivery in Interconnection Networks Using Adaptive Routing," *Proc. 19th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '05),* p. 101, 2005.

[27] W. Heirman, J. Dambre, I. Artundo, C. Debaes, H. Thienpont, D. Stroobandt, and J. Van Campenhout, "Predicting Reconfigurable Interconnect Performance in Distributed Shared-Memory Systems," *Integration the VLSI J.,* vol. 40, no. 4, pp. 382-393, 2007.

Manhee Lee received the BE and ME degrees in Computer Engineering Department from Kyungpook National University, Korea in 1995 and 1997, respectively, and the PhD degree from the Department of Computer Science and Engineering, Texas A&M University, in 2008. He is currently a senior researcher at the National Security Research Institute in Korea. His interests include computer architecture, parallel/distributed systems, secure computing architecture, secure cluster computing, Internet security, and computer networks.

Minseon Ahn received the BS degree in electric engineering and the MS degree in computer science from Korea Advanced Institute of Science and Technology in 1995 and 1997, respectively. After several work experiences in Korea, he has been working toward the PhD degree in the Department of Computer Science and Engineering in Texas A&M University since 2004. His current research interests consist of on-chip interconnection networks including Nanophotonics. He is a student member of the IEEE.

Eun Jung Kim received the BS degree in computer science from Korea Advanced Institute of Science and Technology, Korea, in 1989, the MS degree in computer science from Pohang University of Science and Technology, Korea, in 1994, and the PhD degree in computer science and engineering from the Pennsylvania State University in 2003. She is an associate professor in the Department of Computer Science and Engineering in Texas A&M University. From 1994 to 1997, she worked as a member of Technical Staff in Korea Telecom Research and Development Group. Her research interests include Computer Architecture, Parallel/Distributed Systems, Low Power Design, Secure Computing, Performance Evaluation, and Fault-Tolerant Computing. She is a member of the IEEE.