Single-Level Integrity and Confidentiality Protection for Distributed Shared Memory Multiprocessors *

Brian Rogers[†], Chenyu Yan[‡], Siddhartha Chhabra[†], Milos Prvulovic[‡], Yan Solihin[†]

[†]Dept. of Electrical and Computer Engineering
North Carolina State University
{bmrogers, schhabr, solihin}@ece.ncsu.edu

[‡]College of Computing
Georgia Institute of Technology
{cyan, milos}@cc.gatech.edu

Abstract

Multiprocessor computer systems are currently widely used in commercial settings to run critical applications. These applications often operate on sensitive data such as customer records, credit card numbers, and financial data. As a result, these systems are the frequent targets of attacks because of the potentially significant gain an attacker could obtain from stealing or tampering with such data. This provides strong motivation to protect the confidentiality *and* integrity *of data in commercial multiprocessor systems through architectural support. Architectural support is able to protect against software-based attacks, and is* necessary *to protect against hardware-based attacks. In this work, we propose architectural mechanisms to ensure data confidentiality and integrity in Distributed Shared Memory multiprocessors which utilize a point-to-point based interconnection network. Our approach improves upon previous work in this area, mainly in the fact that our approach reduces performance overheads by significantly reducing the amount of cryptographic operations required. Evaluation results show that our approach can protect data confidentiality and integrity in a 16-processor DSM system with an average overhead of 1.6% and a maximum of only 7% across all SPLASH-2 applications.*

1 Introduction

Computer users are becoming increasingly aware of and concerned for the security of their computer systems. While many security software packages are available, they are ineffective in protecting against an emerging class of security attacks that involve directly tampering with the physical operation of the system. Such *hardware attacks* have been demonstrated to be feasible and relatively easy to perform in breaking the Digital Rights Management features in game consoles [5, 6], and in breaking the security features of a version of a secure processor DS5002FP [10]. These hardware attacks may involve inserting a device on the communication path between the processor and other chips to *passively* snoop [5, 6] or to *actively* modify [10] data communication.

Realizing these threats, researchers have proposed *secure processor* architectures [2, 3, 4, 12, 13, 14, 15, 19, 20, 21, 22, 23, 25, 27, 29, 30]. The microprocessor industry has also offered secure processors for commercial use, including the IBM SecureBlue [7], and Dallas Semiconductor DS5002FP [16]. A secure processor assumes that data on a processor chip is secure, while off-chip data can be observed or modified by an attacker. Since snooping or manipulating on-chip transistors/wires is much more difficult than snooping or manipulating off-chip components/interconnections, combined with the fact that the chip can be protected through the use of various manufacturing techniques such as special coating [16], the processor chip provides a reasonable security boundary. With this security boundary, in order to protect the secrecy and integrity of data stored off-chip, data must be encrypted before being moved off-chip, and then decrypted and authenticated when brought back on-chip.

Prior studies have proposed secure processor designs for single-chip uniprocessor systems [3, 4, 13, 14, 15, 20, 21, 22, 23, 25, 27, 29, 30], and bus-based Symmetric Multiprocessors (SMPs) [2, 21, 30]. However, secure designs suitable for Distributed Shared Memory (DSM) multiprocessor systems have not been thoroughly investigated, with only a few designs proposed [12, 19]. This is unfortunate because multiprocessor computer systems are currently widely used in commercial settings. These multiprocessors often run applications that operate on sensitive data, such as customer records, credit cards numbers, financial information, etc. Obtaining or modifying such data can be very lucrative to attackers, so this data should be protected from unauthorized access. Also, secure DSMs are increasingly needed as *utility* or *on-demand* computing proliferates. In utility computing, companies "lease" computation and storage resources of a large-scale, powerful system (e.g. the HP Superdome [18]) to customers who need such resources on a temporary basis or who want to offload their IT operations. These large-scale DSM systems are not under the control of the customers who are using them, so naturally customers may want the secrecy and integrity of their code and data ensured. Indeed, industry analysts have reported that security concerns have partly restricted adoption of the utility computing model [1].

Certainly, utility computing providers would take steps to avoid unwanted physical tampering to their DSM system through enforcing physical security, such as by restricting physical access to the DSM to a few "trusted" employees and contractors. However, one of the key principles of security is that relying only on one layer of security is *risky* and is often *insufficient*. The risk of security attacks by select employees or parties that are trusted with physical access to the machine should not be underestimated. We can take an example from the case of Automated Teller Machines (ATMs) which also employ a certain level of physical security. However this is often insufficient, as is evident in a report by Global ATM Security Alliance (GASA) which states that more than 80% of computer-based bank-related frauds involve employees [11]. In the case of DSM systems used for utility computing, the large amounts of sensitive data in

*This work is supported in part by the National Science Foundation through grants CCF-0347425, CCF-0447783, and CCF-0541080.

978-1-4244-2070-4/08/$25.00 ©2008 IEEE

these systems create a financial incentive for attackers to perform corporate espionage or other malicious intents.

Additionally, physical or hardware attacks on DSM systems may be performed more easily than on uniprocessor systems because there are interconnect wires that are more exposed. For example, to snoop processor-to-memory communication in a uniprocessor system, attackers must tamper with the motherboard of the system. While in a DSM system, attackers can snoop the interconnect wires that are exposed at the back of the server racks using snooping devices similar in principle to a keyboard logger, without much disruption to the system. This lack of disruption is important for attackers since attacks can be performed quickly and without leaving traces. The possibility of hardware attacks may prompt customers to demand that DSM utility computing systems be equipped with secure hardware features that make them resistant even to hardware attacks. Utility computing providers that offer these features have an important competitive advantage compared to those who do not. Hence, we believe that data security in DSM systems will become an increasingly important issue in the future.

To the best of our knowledge, only two protection schemes have been proposed to address the data protection problem in multiprocessor systems with non-bus based interconnects [12, 19]. The main drawbacks of the scheme proposed in [12] are that very large on-chip storage overheads are required (e.g. a 512KB cache structure). It is also vulnerable to replay attacks against data integrity, especially if attackers can drop messages. The scheme proposed in [19], as with our approach, is designed specifically for DSM systems. This scheme differentiates between processor-to-memory communication and processor-to-processor communication across the interconnect, protecting each with a separate security mechanism. For example, a remote data request may result in one processor fetching, decrypting, and authenticating data from its local memory using processor-to-memory protection mechanisms. Then this processor will encrypt, sign, and communicate the data to the requesting processor which will decrypt and authenticate the data again, all using processor-to-processor protection mechanisms.

Because inter-node communication involves two separate security mechanisms, we refer to this approach as a *two-level* approach. Such a two-level approach results in a number of inefficiencies. First, the latency-hiding techniques of both protection mechanisms must *simultaneously succeed* for the cryptographic latencies to be completely hidden. This may be difficult to achieve in practice, and thus cryptographic latencies may be exposed frequently. Second, this results in a large number of cryptographic operations, which can increase cryptographic latencies because of *contention* due to excessive utilization of the hardware cryptographic engines. Finally, authentication-related operations (e.g. MAC generations and verifications) are directly in the *critical path* of remote data requests because the authentication by one mechanism should be completed before passing data to the next mechanism. Overall, this type of approach is inherently performance inefficient. However, because fundamentally processor-to-processor communication seems best suited to a communication based protection scheme (e.g. associating a MAC value with each message), while processor-memory communication seems best suited to a storage based protection scheme (e.g. a Merkle Tree covering memory), it is difficult to provide a single, unified scheme that is able to protect all types of data communication in a DSM system efficiently.

Contributions. In this work we propose a new and efficient memory encryption and authentication solution for protecting the confidentiality and integrity of data in a DSM system. Our solution requires modest on-chip storage overheads and removes the inefficiencies of a two-level protection scheme by using a single mechanism to protect both processor-memory and processor-to-processor communication. We refer to our approach as *single-level* memory encryption and authentication. The efficiency of our single-level approach comes from a significant reduction of cryptographic operations on remote requests. A single mechanism is used to encrypt and sign data when it is sent off-chip by a processor (either to memory or a remote processor), and to decrypt and authenticate data when it is brought on-chip for use by a requesting processor (either from memory or a remote processor). This approach not only reduces the amount of cryptographic work involved, but also reduces the possibility of contention-related latencies at the cryptographic engines. In addition, only one latency-hiding mechanism is involved, which has a higher chance of succeeding compared to the simultaneous successes of multiple latency-hiding mechanisms of a two-level approach. Finally, we avoid transitions between two security mechanisms, allowing authentication of data on a remote request to be moved *off the critical path*.

Overall, we find that our techniques can provide secure data encryption and authentication in a DSM system with very low overheads. Passive or eavesdropping attacks are avoided by encrypting off-chip data communication, while active attacks involving message modification, injection, and deletion, are detected as failed authentication or coherence protocol errors. Our simulation results across all SPLASH-2 [26] benchmarks show that the average overhead of our mechanisms on a 16-processor DSM system is less than 1.6% with a maximum of 7%, relative to an identical but unprotected system. Compared to a two-level approach which has an average overhead of 5.3% and worst-case overhead of 16.3%, our approach shows a reduction in overheads by a factor of $3.3\times$ on average, and by a factor of $2.3\times$ in the worst case. Since DSMs are often used to run performance-critical applications, this is important because our single-level scheme reduces performance overheads of even problem applications to an acceptable level, giving customers confidence that their applications would perform well.

The remainder of this paper is organized as follows. Related work is discussed in Section 2. We discuss our assumptions and attack model in Section 3. Section 4 introduces our single-level memory encryption and authentication scheme for DSM systems. Section 5 details our evaluation setup. Section 6 presents our evaluation results and insights. Finally, we conclude with Section 7.

2 Related Work

Researchers have proposed various designs of *secure processor* architectures [2, 3, 4, 12, 13, 14, 15, 19, 20, 21, 22, 23, 25, 27, 29, 30]. The microprocessor industry has also offered secure processors for commercial use, including the IBM SecureBlue [7], and Dallas Semiconductor DS5002FP [16]. The motivation behind secure processors is to provide a private and tamper-resistant execution environment. It is typically assumed that on-chip data is secure, but off-chip data can be subjected to eavesdropping and direct modifications. Therefore, off-chip data communication and storage is protected by encryption and Message Authentication Code (MAC)-based integrity verification. The various designs differ in the *encryption modes* used and the *latency-hiding techniques* employed.

XOM uses a *direct encryption* approach where a block cipher such as AES is used to directly encrypt and decrypt data [4, 14, 15]. Direct encryption has an advantage of being simple and straightforward to implement. However, because on a cache miss a block must be fetched on chip before it can be decrypted, the latency of off-chip data fetches is directly increased by cryptographic operation laten-

cies. There is also a security concern with direct encryption in that the statistical distribution of ciphertexts matches that of the plaintexts. More recent studies have proposed *counter-mode encryption* in which a data block is not directly encrypted [22, 25, 27, 29]. Rather, a pseudo-random pad is obtained and XORed with the data block. The pseudo-random pad is constructed by encrypting a *seed*, which is typically composed of a per-block counter and the block's address. The ability of counter-mode encryption to hide the decryption latency of an off-chip data fetch depends on whether the pad of a block can be generated while the block is being fetched on chip. Since pad generation can only begin when a data block's counter is available on chip, various counter management strategies have been proposed, which include counter caching [25, 27, 29], and counter value prediction [22].

For integrity verification, XOM stores MAC values along with off-chip data, and verifies the integrity of the data by recomputing its MAC and comparing it with the stored MAC. A stronger integrity verification scheme utilizes a *Merkle Tree* of MACs [3] to additionally protect against data replay attacks. A Merkle Tree is formed over the data blocks in memory, with a tree node in one level protecting multiple nodes in the lower level. The tree root is always kept on-chip to ensure its safety against tampering. Data integrity can be ensured by verifying MACs up the tree to the secure root.

Researchers have also proposed secure multiprocessors. Constructing a secure multiprocessor system by piecing together secure processors alone does not give sufficient protection because communication between processors is not automatically protected. The main challenge is that communicating processors must share necessary encryption information, so that a data block encrypted by one processor can be decrypted by another processor. In a bus-based Symmetric Multi-Processor (SMP) system, the shared bus provides an ideal medium for sharing encryption information because all processors can observe the same bus. For example, a global bus counter can be used to encrypt data transfers between processors [21], or data transmitted on the bus itself can be used to encrypt new data blocks through Cipher Block Chaining [30]. Additionally, [2] proposes a technique to authenticate a shared bus.

Distributed Shared Memory (DSM) multiprocessors use a point-to-point interconnect, so there is no shared communication medium that all processors can monitor. Hence, the mechanisms proposed for SMP protection cannot be extended easily for protecting DSM systems. To the best of our knowledge, only two secure architectures have been proposed which can be used for non-bus based interconnects [12, 19]. Most closely related to our work is the design in [19] which is also a protection scheme for DSM systems. The main drawback of this approach is that it relies on a *two-level* encryption and authentication approach, which can lead to a variety of inefficiencies. The scheme in this paper utilizes a novel *single-level* approach that achieves lower execution time overheads and avoids the complexity of supporting two security protection schemes compared to a two-level approach.

In [12], an approach is proposed to protect communication in multiprocessors across an arbitrary interconnection network and coherence protocol. Instead of maintaining per-processor-pair counters for communication between processors as in [19], this approach uses a global counter controller to distribute different counter ranges to processors in the system. To minimize the cryptographic delays caused by communication between processors, a keystream cache is used at the sender to move the AES latency off the critical path. The pads which are pre-computed and stored in the keystream cache can be used to encrypt blocks sent to other processors immediately. At the receiver side, a keystream pool is maintained to store the pads for all potential senders. When an encrypted data block arrives at the receiver, the data can be used shortly afterwards if the pads used for encryption can be found in the keystream pool. The main drawbacks of this approach are that significant on-chip storage overheads are required for the various structures (e.g. a 512KB keystream pool). Also, data communications are vulnerable to replay attacks, especially if attackers can drop messages in the system.

3 Attack Model and Assumptions

Before we present our solution for ensuring data confidentiality and integrity in DSM systems (Section 4), we discuss the types of attacks that we assume possible in a DSM system. Our attack model is the same as one assumed in [19], and we will briefly overview it in this section.

Like other work on secure processors [2, 3, 4, 7, 12, 14, 15, 19, 20, 21, 22, 23, 25, 27, 29, 30], our main assumption is that data located on a processor chip is secure, while data located off-chip is vulnerable to attack. In a DSM multiprocessor, the main off-chip structures that are vulnerable to attack are the interconnection network which connects the processors, and the local main memory of each processor (memory bus and DRAM). Such an assumption is also used in commercial secure processors such as IBM SecureBlue [7] and Dallas Semiconductor DS5002FP [16].

In addition, we assume that attackers can target major off-chip structures such as the local memory bus and DRAM, and the interconnection network. The types of attacks that may be attempted on these structures include *passive* attacks (eavesdropping) and *active* attacks (altering data in memory or messages on the interconnect and local buses). Through active attacks, attackers may modify headers or the data payload of a coherence message, as well as replay an old version of a message. We refer to the latter attack as a *message replay* attack. Attackers can also act as a *man-in-the-middle*, dropping and injecting messages in the interconnect.

We assume that attackers are primarily interested in discovering secret data, and hence our goal in protecting against passive attacks is to always ensure privacy by encrypting all data communicated off-chip between processors and their local memories or between different processors.

We assume that *traces of attacks*, which we define as detectable anomalous behavior such as cryptographic errors (failed authentication), invalid coherence messages, or system crashes, are sufficient deterrents to active attacks. The rationale for this assumption is that since DSM systems are typically large and expensive, they are likely protected with physical security that includes restricted access to a few select people. Therefore, attackers are likely malicious employees or contractors with access to the system. We have discussed that this is a real possibility as hinted by a case study in Automated Teller Machines [11]. Such attackers will likely prefer performing an active attack that produces no *traces* that can alert other users and allow an investigation to correlate the traces to the attackers. Consequently, it is sufficient for our security mechanism to ensure that all active attacks produce traces that are detectable. We also assume that a secure mechanism is in place to log these traces and alert users when a log is created.

We also assume that *precise authentication*, which refers to delaying load instruction retirement (or even load data use) until data authentication is completed [21, 27], is not needed. The rationale for this assumption is that a hardware attack takes time to perform, so detection within a reasonable time window (say, thousands to millions of cycles) serves as a sufficiently powerful deterrent to attackers. In our design, detection of an attack occurs within the la-

tency of a message round trip, which is on the order of hundreds to thousands of cycles.

A secure DSM system with memory encryption support must have mechanisms in place to provide secure booting of the machine and secure key setup. This is an important issue that is beyond the scope of this paper. In this paper, we simply assume that there is a mechanism in place to guarantee secure booting and key set up on the machine.

Finally, we do not assume any specific configuration of the interconnect network such as message delivery ordering or routing strategies. However, we assume that the DSM employs a home-based coherence write invalidate protocol, and we specifically illustrate our mechanisms with the MESI protocol in mind.

4 Data Protection for DSM Systems

4.1 Counter-Mode Encryption/Authentication

Popular cryptographic functions such as 3DES/AES encryption and SHA-1/MD-5 authentication can be used in a direct encryption or *electronic codebook* (ECB) *mode*, where these functions are applied directly to the plaintext of the data to produce its ciphertext and a MAC (Message Authentication Code). One drawback of ECB in the context of memory encryption is that decryption and authentication of the ciphertext cannot begin until it is fetched on-chip. Consequently, cryptographic latencies are added directly to the critical path of off-chip data fetches. Therefore, in this paper, we focus our solution around counter-mode encryption and authentication.

Counter-mode encryption assigns a counter for each data block that is used for encrypting/decrypting the block. While there are various counter mode encryption and authentication algorithms, one promising algorithm is Galois/Counter Mode (GCM) [17, 28]. GCM combines encryption and authentication operations very efficiently, and has been demonstrated to be as secure as the underlying AES encryption algorithm [17, 28]. In encryption, a unique *encryption seed* is input to the AES engine to obtain an *encryption pad*, which is XORed with the plaintext to obtain the ciphertext. To guarantee security, a *necessary* condition in counter-mode encryption is that each encryption pad value can only be used for encryption once, so consequently the encryption seed value must also be used only once. To encrypt a cache block, typically the encryption seed can be composed of the concatenation of *block address* (to ensure spatial uniqueness), a *per-block counter* that is incremented after each use (to ensure temporal uniqueness), and an *encryption initial vector* (EIV – to ensure that pads used for encryption are different than ones for other uses).

To generate a MAC for a block in GCM, an *authentication seed* is input to the AES encryption unit to obtain an *authentication pad*. The authentication seed must also be unique for each use, so similar to encryption seed, it can be composed by concatenating an *authentication initial vector* (AIV), block address, and a per-block counter. The ciphertext together with *additionally authenticated data* (AAD) are input to a function called GHASH. The AAD contains data that needs to be authenticated but not encrypted. The GHASH function consists of a short chain of bitwise XORs and Galois Field multiplications. Finally, we can clip the MAC to the desired length based on security requirements by taking its most significant bits (MSBs). It is worth noting that a MAC in GCM is generated using a secret key. Consequently, given a ciphertext, an attacker cannot generate a valid MAC for it.

Performance-wise, most of the cryptographic latency comes from generating encryption and authentication pads. Once these pads are available, a ciphertext can be obtained through a bitwise XOR, and a MAC can be obtained through the GHASH and XOR operations. All operations past pad generation only require a few cycles to complete. As a result, we primarily need to focus on hiding the latency of pad generation.

4.2 Challenges in Providing Secure DSM Architectures

Secure processor architectures, which include counter-mode encryption and Merkle Tree authentication, already have the capability of providing secrecy and integrity to data in the local memory. However, data transmitted between processors are not automatically protected from attacks. The main challenges to providing such protection are: (1) how to allow data encrypted by one processor to be decrypted by another processor, and (2) how to protect data that is communicated among different processors.

The first problem, in counter-mode encryption, essentially implies that communicating processors must share the *same* counter that is used for encrypting a data block. However, counter sharing necessitates communicating its value across multiple processors, implying that counters will be subjected to alteration attempts by attackers. In counter-mode encryption, it is well-known that undetected counter modifications can be used by attackers to break the encryption through reuse of encryption pads. Hence, we have to deal with an open problem of how to provide *secure counter communication*. In the past, researchers avoided the problem of protecting counter communication by relying on information that is naturally shared by multiple processors without needing communication. In bus-based Symmetric Multi-Processor (SMP) system, the shared bus naturally provides a shared medium for all processors to use for processor-processor encryption [21, 30]. Note that because the global bus counter or bus data is used for encrypting processor-processor communication (rather than the per-block counters used for processor-memory encryption), such an approach relies on two separate security mechanisms: one to handle processor-processor communication, and another to handle processor-memory communication. We refer to this approach as a *two-level* memory encryption and authentication.

The same principle of two-level memory encryption and authentication to avoid secure counter communication is applied in Distributed Shared Memory (DSM) multiprocessors by Rogers et al. [19]. Since DSMs use a point-to-point interconnect and lack a shared communication medium that all processors can monitor, their scheme relies on maintaining shared communication counters between each processor pair that are incremented by each processor independently after encrypting or decrypting a block communicated between the two. This way counters do not need to be communicated between processors in the common case. In a two-level approach, since a remote read request involves two security mechanisms, there are two major performance drawbacks. First, in order for the full cryptographic latencies of a remote read request to be hidden, all latency-hiding techniques associated with each cryptographic operation must simultaneously succeed. This is illustrated in Figure 1(a).

The critical path of the two-level approach includes decryption and Merkle Tree authentication after local memory fetch (Circle 1), re-encryption of the requested block and its MAC generation using the processor-processor mechanism (Circle 2), and finally decryption and MAC verification at the requestor side (Circle 3). Each of these cryptographic operations incurs a latency that needs to be hidden by latency-hiding techniques such as counter caching, counter prediction, and pad pre-generation. Each technique is imperfect (the counter cache may miss, counter prediction may mis-predict, while pads may be pre-generated too late), and this contributes to

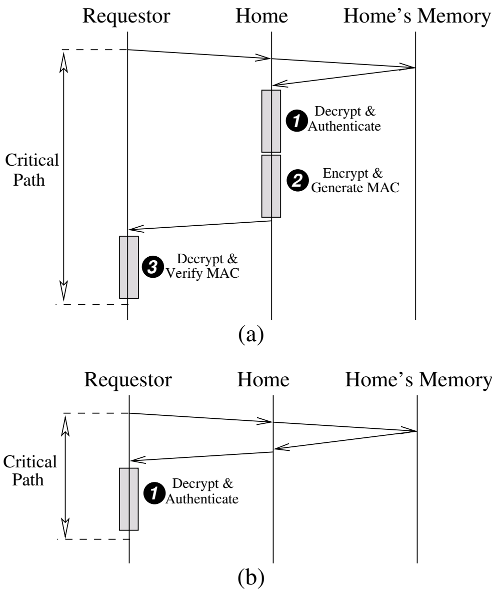

Figure 1. Critical Path of a remote data request for a Two-Level encryption scheme (a) and Single-Level encryption scheme (b).

the inability to fully hide cryptographic latencies. For example, if individual techniques operate independently and the success rate of each technique ranges from 70% to 80%, full latency-hiding only occurs between 34% to 51% of the time (i.e. because $0.7^3 = 0.343$ and $0.8^3 = 0.512$), which means that a significant fraction of the time only parts of cryptographic latencies are hidden. Note that this is in addition to non-hidable latency such as the GHASH function in GCM. Another major drawback is the large amount of cryptographic work which increases occupancy and contention at the cryptographic engine, which could further exacerbate cryptographic latencies. Overall, a two-level approach is bound to suffer from relatively high execution time overheads and high complexity due to employing two security mechanisms.

We note that the drawbacks of the two-level approach can be avoided if we employ a unified encryption and MAC generation scheme for both processor-memory and processor-processor communication, as illustrated in Figure 1(b). On a remote request, the home node fetches the ciphertext of a data block, and immediately sends it off to the requestor. Only the requestor performs cryptographic operations to decrypt the block and verify its MAC. This cuts down the total cryptographic latencies in the critical path by *two thirds*. More importantly, because only one latency-hiding mechanism must succeed in order to hide the full cryptographic latency, a single-level approach achieves lower and more stable overheads than a two-level approach, as we will show in Section 6.

4.3 Single Level Memory Encryption

In a single-level encryption scheme, the per-block encryption counter that was used to encrypt a data block in one node (e.g., the home) is needed to decrypt it in another node (the requestor). Logically, one may think that this counter can be cached at both nodes and can automatically be kept coherent by the coherence protocol. However, it is problematic to keep counters coherent when coherence messages that ensure counter coherence are themselves subject to security attacks, e.g. attackers can replay an old counter message to force pad reuse. We can protect counter coherence messages using a separate security mechanism, but this increases complexity and the difficulty of hiding latencies even more.

To avoid the complexity of relying on the cache coherence protocol to keep counters securely coherent, we adopt an alternative approach in which a *counter of a block can only be cached at the home node and at the owner node (if any)*. For a data block in a clean state, only the home can cache the counter value. For a data block in the modified state cached at an owner, the owner eventually needs to encrypt the block and send it off to another processor (due to intervention or write-back), and hence it is allowed to cache the counter value temporarily until the block is replaced or becomes clean. When a processor asks for an exclusive state for a data block through an upgrade or read-exclusive request, the home processor increments the counter value of the block, and replies with the new counter value to the requestor. Certainly, this counter value must be protected from tampering, so we protect it using the same mechanism used to protect data communication (to be discussed in Section 4.4). We will now discuss the latency-hiding aspect of this approach, and leave the security aspect until later in Section 4.4.

Hiding Decryption Latency at the Requestor. In our single-level memory encryption, typically the only node that performs a cryptographic operation is the requestor of a data block, which must decrypt the block before it can use it. The key to successful latency-hiding at the requestor is that the requestor must have the counter to pre-generate the appropriate decryption pad before the data arrives. Since the requestor is not allowed to cache the counter, the home node that caches the counter must supply the counter early. Fortunately, with simple coherence protocol modifications, this is achievable. Figure 2 shows how various scenarios are handled. The first scenario is when the requested data is in the home node's local memory (Figure 2(a)). In this case, the home looks up its local counter cache to obtain the block's counter (CTR). If it finds the counter, it forwards the counter immediately to the requestor, and in parallel begins to fetch the data block ciphertext (CTEXT) from its local memory. The counter value arrives at the requestor one memory-latency before the data, allowing the requestor to generate the decryption pad ahead of receiving the data ciphertext.

Figure 2(b) shows another scenario in which the data block requested is also in the home processor's cache (in plaintext form because it is on chip). Before forwarding the data block to the requestor, the home processor must first encrypt the data block. In parallel, it sends the counter value to the requestor. The requestor, upon receiving the counter, immediately pre-generates the pad needed to decrypt the ciphertext of the data block that is still yet to arrive. In this case, the one pad generation latency at the home node is exposed, but the pad generation latency at the requestor is hidden. This is still better than the two-level approach which in the worst case can suffer up to three pad generation latencies. Furthermore, the scenario in Figure 2(b) may be rare because typically different processors work on different data, so a remote read rarely finds the data in the *clean* state at the home processor's cache (if the state of the data is dirty, it is handled in the next scenario).

Figure 2(c) shows a scenario in which the data is owned by another processor. In this case, the home node will forward the request to the owner, and in parallel send the counter value to the requestor. When the data block owner receives the request, it will first encrypt the data block and send it to both the home processor and the requestor. The requestor still receives the counter before it receives the data, and can overlap pad generation with the communication latency. However, one pad generation latency at the owner is exposed. Unfortunately, compared to the scenario in Figure 2(b), this case quite frequently occurs when different processors share data

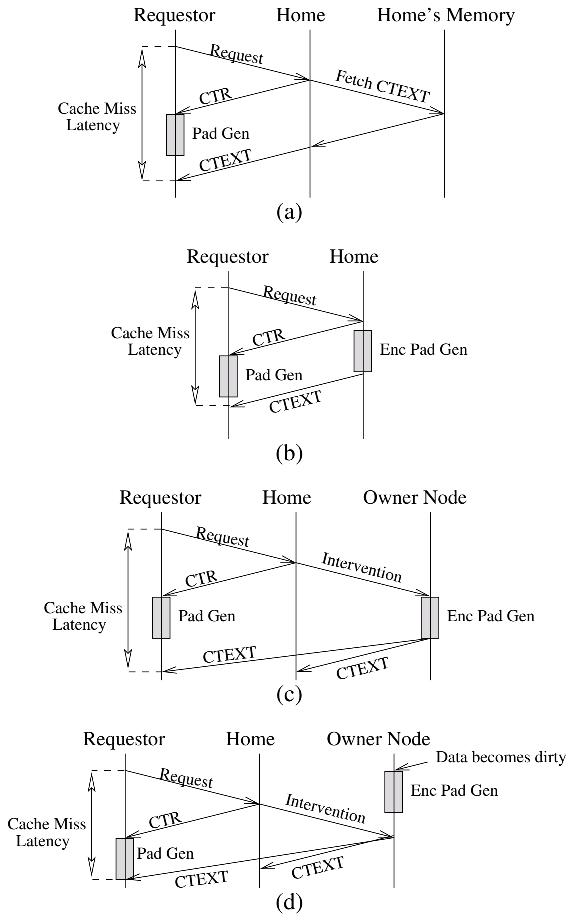

Figure 2. Coherence protocol modifications for hiding cryptographic latencies of remote data request in our single-level counter mode memory encryption when data is supplied by the home node's local memory (a), home node's cache (b), remote owner (c), and remote owner with an owned block pad buffer (d).

that is frequently written (both true and false sharing). Hence, we seek ways to optimize this scenario next.

Hiding Encryption Latency at the Sender/Owner. In Figure 2(c), the encryption pad generation at the owner's side is in the critical path. In Figure 2(d), we illustrate an optimization to move the latency off the critical path. To achieve that, when a data block becomes dirty or modified, immediately we trigger its encryption (and authentication) pad generation. These pads are then stored in a small buffer called the *Owned-Block Pad Buffer*. We only pre-generate pads for modified blocks because these are the only blocks that have a possibility to be intervened by another processor. When an owner receives an intervention for a data block, if it finds pads for the block in the owned-block pad buffer, it can immediately encrypt and generate the MAC for the block in a few cycles, and send the block off to the requestor.

One important question is how large the owned-block pad buffer needs to be. We observe that frequently communicated blocks typically reside in any one cache for only a short time since they receive interventions quite frequently (e.g. shared locks), and that blocks that have been in the modified state for a long time are unlikely to be intervened frequently. Therefore, the owned-block pad buffer only needs to store pads for the blocks which are most recently upgraded to a modified state. We find that a 32-entry owned-block pad buffer is sufficiently large to ensure that in most cases, pre-generated pads are available for intervened data blocks.

Local Counter Management and Counter Prediction. Note that Figure 2 assumes that the home finds the counter in its local counter cache for each request. If this is not the case, the home must first fetch the counter from the local memory before the counter can be sent to the requestor. The frequency of this case can be made very rare with a good counter caching policy at the home. Since a home node only caches counters of blocks that are in its local memory, the counter cache size can be kept constant regardless of the number of processors in the DSM. In fact, when a cache block is shared by multiple processors, the first processor that requests the data block from its home node will cause the home node to cache the counter of the block, effectively prefetching the counter for subsequent processors that want to fetch the block. In addition, since block counters can be small (e.g. 8 bits), a counter cache block can hold many counters from many cache blocks. Hence, the common case is that the home finds a counter of a requested block in its local counter cache.

Furthermore, the requestor can employ a *counter prediction* mechanism such that even if a home node cannot find the counter in its local counter cache, the requestor can still pre-generate pads ahead of time. For example, each processor can keep track of the last counter value of remote blocks that it has recently evicted. If the last value is zero, it is highly likely that it corresponds to a read-only block for which the counter value will remain zero, hence we can start pad generation assuming that the counter value is zero. In addition, tracking such zero-valued counters is very space efficient since we only need a single bit per counter. We use a *mask buffer* to store these bits, with a bit value of "1" indicating a particular counter was last seen as zero. We find that a 32 entry mask buffer is able to provide effective counter prediction of zero-valued counters. When a node sends a data request to a remote node, it can send along the predicted counter value that it used for pregenerating its pad. If the predicted value of the counter is correct, the home node can skip sending the counter value to the requestor, hence we have an additional benefit of conserving network bandwidth. We note that there are many other ways for predicting counter values and it is beyond the scope of the paper to search for the best counter prediction scheme exhaustively. However, we find that even this simple counter prediction scheme can achieve high accuracy of 76% (Section 6).

Summary. Overall, we have shown that cryptographic latency-hiding can be achieved through simple coherence protocol modifications that allow the home node to forward counter values, pad pre-generation at owner nodes for modified blocks that they cache, and relatively simple counter prediction at the requestor. With these mechanisms in place, pad generation latency is exposed in only a few cases, and even in these cases, only one pad generation latency is exposed.

Small architecture components are added to each processor as shown in Figure 3. A secure processor architecture for uniprocessor systems already contains a cryptographic engine and local counter cache. Over them, we add a 32-entry owned-block pad buffer to hold pre-generated encryption and authentication pads, and a 32-entry mask buffer to store the bit masks indicating zero-valued counters for counter prediction.

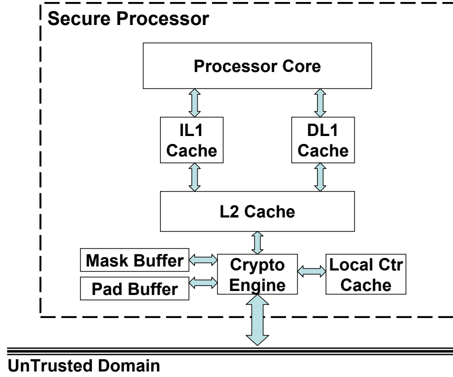

Figure 3. Architecture modifications to a node

4.4 Single-Level Memory Authentication

In this section, we will discuss the security aspects of our scheme. Since data is always communicated as ciphertext, passive attacks are protected against. We now discuss our mechanisms to ensure that active attacks are detected. Memory authentication in DSM systems requires protection not only from attacks on data loaded from a processor's local memory, but also from attacks on data communicated between processors across the system interconnect. Protection of data loaded from off-chip memory has been proposed in a prior study [3] in the form of Merkle tree authentication for uniprocessor systems. However, as with counter-mode encryption, it is not an easy task to extend Merkle tree based protection to protect data communicated between processors. It may seem possible to cover the entire shared memory with a single Merkle tree which is distributed across all processors. However, keeping the Merkle tree nodes coherent is similar to the problem of keeping block counters coherent, as discussed in Section 4.3.

To avoid the significant complexity of supporting a coherent global Merkle Tree, we adopt an alternative approach. Each node maintains its own *local Merkle Tree* that covers only the node's local memory, and a lightweight message authentication protocol provides authentication for all data communicated in a DSM system. In a traditional *early* authentication protocol such as that used in [19] and illustrated in Figure 4(a), protection is provided as follows. To send a message X over the interconnect from node A to node B, the message authentication code (MAC) of the message is first computed at the sender A as $MAC = H_K(X)$, where $H_K(.)$ is a cryptographic keyed hash function with private key K. Upon receiving the message, B recomputes $H_K(X)$ and compares it against the received MAC. If either X or the MAC has been altered, the recomputed MAC will mismatch at B. B then sends an acknowledgement message back to A to indicate the receipt of the message. Attackers cannot generate a correct MAC for a given value of X or for an arbitrarily chosen value since they do not know the secret key K. However, this approach may be vulnerable to replay attacks where the value of X and its MAC are replaced by old, stale values that an attacker has recorded, unless another protection mechanism is used to prevent such attacks. In addition, the MAC computation is in the critical path of communicating X from A to B.

Our approach uses a different *delayed* protocol as illustrated in Figure 4(b). First, node A sends X to node B (we refer to it as the *forward edge*). Node B then computes the MAC for the message value it receives and sends it back to node A piggybacked to the standard acknowledgement message (we refer to it as the *backward edge*). Node A knows the original value of X so it can recompute $H_K(X)$ and match it against the received MAC. If either X or the MAC have been tampered with, the recomputed MAC will mismatch at A. Note that both protocols incur the same cryptographic work, but our approach incurs no delay in the latency-critical path of communicating X from node A to node B. If the communication already involves such a *loop* (both forward and backward edges are naturally present), no extra bandwidth is consumed over the traditional early protocol. In addition, our approach supports protection against replay attacks as we will describe below.

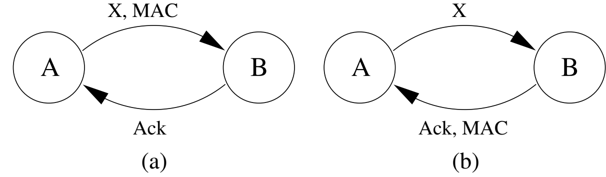

Figure 4. High-Level illustration of an early (a) versus our delayed (b) authentication protocol.

While attackers cannot inject a valid message or compute a valid MAC given a message, attackers may attempt to reroute a message destined for one node to another node. To detect such attempt, the MAC computed by node B should include its own processor ID, i.e. $MAC = H_K(X, ID_B)$. Meanwhile, node A can remember the message value X and its intended destination ID_B in its structure which tracks outstanding transaction, so it can compute the MAC that it expects to receive from node B. If the received MAC mismatches, then it has discovered a message tampering or rerouting attempt by the attacker. Similarly, attackers may attempt to replay an old message. As a general mechanism to detect such attacks, the sender node A may piggyback a unique timestamp $ID_A||TS$ to each message. TS can be a monotonically increasing *message counter* that each node increments when it sends a forward-edge message. Thus the combination of a processor's ID and current message counter forms a unique timestamp for each message. Node A can also store the timestamp value in the outstanding transaction table. Node B must compute the MAC that includes the timestamp it receives, i.e. $MAC = H_K(X, ID_B, ID_A||TS)$, and send the MAC to A in the backward edge. Since no two messages share the same timestamp value, the computed MAC is always unique. Overall, our protocol is able to detect communication attacks such as message tampering, message reroute, and message replay without adding delay to the critical path of communication.

In our secure DSM coherence protocol, we can identify five main types of communication: *request* (sent by a requestor to the home), *data response* (data sent by the home or owner as a response to a request), *non-data response* (acknowledgement or negative acknowledgment as a response to a request), *write-back* (data sent to the home by an owner), and *counter* (per-block counters sent to requestors to enable decryption latency hiding). In principle, it is straightforward to protect them all by (1) requiring that all of them to have loops (adding backward edges when necessary), and (2) applying our delayed authentication protocol to protect them.

Based on this classification of messages, we identify two protection levels that seem to be reasonable. The first is simply *full* protection in which all types of communication are protected using delayed authentication. Full protection's security level is the strongest since all messages in the system are cryptographically protected. Another option is to provide *data-counter-only* protection in which only communications that include data and counter values are protected (i.e. data response, write-back, and counter messages), but others are not (i.e. request, and non-data response

messages). We note that requests, invalidations, and other non-data messages are directly involved in maintaining the correctness of coherence protocol implementations and hence their tampering will result in anomalous coherence protocol operations. For example, a non-home node could receive a read request or an invalidation could be received by a node that does not cache the block. So it may be an attractive option to use data-counter-only protection and rely on coherence protocol anomaly detection to further deter attackers.

While the above discussion of our delayed authentication protocol provides a framework for us to provide protection, another interesting issue is with regard to optimizations of different cases in terms of efficiency or flexibility. For example, a node requesting data will receive both a counter value and clean data from the data provider. As such, for both communication types, the requestor can send a single backward edge message for validation, rather than two separate backward edge messages. In the following subsections, we provide a more concrete discussion of how to apply the delayed authentication protocol to protect DSM systems.

4.4.1. Authenticating Data Response and Counter Messages. The first type of protection we discuss is that for data response and counter messages. As we mentioned, since these messages are always paired, we can merge the protection of each with a single MAC on the backward edge message. Figure 5 shows the steps of our process to authenticate data communicated to a requesting processor, where in this case the data is supplied by the data's home node. This figure also notes the roles of various messages and message components as they relate to our delayed authentication approach described in the previous section. This scenario matches that in Figure 2(a), except that now we augment its security through message authentication. When the home node receives the data request, it first sends the block counter as the forward message of counter communication (Circle 1b). At the same time it fetches both the data ciphertext (CTEXT) and also the block's MAC – which is the lowest level MAC in the node's local Merkle tree and is generated as described in Section 4.1 (Circle 1a). Then the home sends the forward message for the data response which consists of the ciphertext, its MAC, and a *message counter* (MCTR) (Circle 2). The MCTR is the per-processor, on-chip message counter value as described in the previous section. The MCTR and processor ID will form the timestamp used to prevent replay attacks. Each processor's message counter is stored in an on-chip, non-volatile register so that its value remains even across system resets. Also, each processor will need to buffer the MCTR values for outstanding messages so that it can be used to verify the response MAC that is included in the backward edge acknowledgement message. This is needed so that processors can have more than one outstanding transaction at a time. Also, note that we choose to send the block's MAC value in this message. We describe the reason for this below, but note that this MAC is not in the critical path of sending the data response.

Upon receiving the data response message, the requestor decrypts the data for use by the processor. Then it verifies the MAC of the ciphertext, which ensures that the ciphertext, block counter, and MAC combination are valid, since the counter is a component of MAC generation using GCM (Circle 3). Thus, up to this point, the only option for an attacker is to attempt a replay attack by providing stale, but corresponding block counter, ciphertext, and MAC values. However, our delayed authentication protocol still has one more step, the backward edge MAC, which ensures that both the counter and data response messages are free from tampering, including replay. The MAC computed (Circle 4) for the backward edge message (MSGMAC as shown in the figure) is computed based on the components shown in Equation 1.

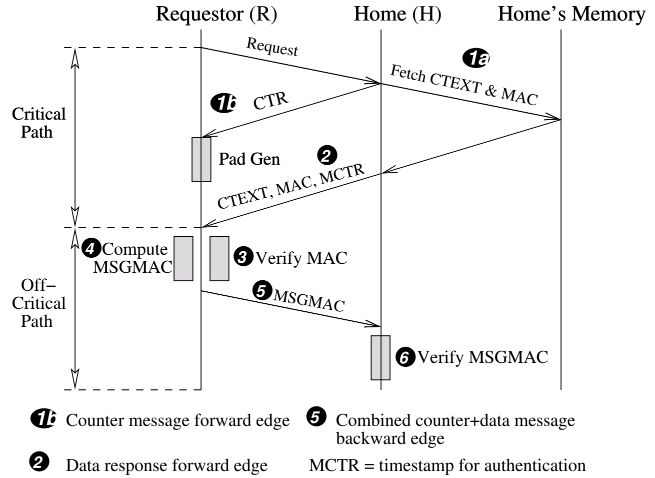

Figure 5. Steps for authenticating data sent to a requesting processor across the interconnect.

$$MSGMAC \Leftarrow H_K(MAC||PID_R||PID_H||MCTR) \quad (1)$$

The MSGMAC is computed by the requestor following the general process described in the previous section for backward edge MACs. Again, the combination of the home's PID (PID_H) and the home's message counter ($MCTR$) form the timestamp to make each MSGMAC computation unique, and the requestor's PID (PID_R) ensures that messages cannot be rerouted. The final component is the data block's MAC. Since a replay attack on data response message must roll-back the block ciphertext, block counter, and block MAC in unison, verifying that the correct block MAC was received by the requestor allows the verifying processor to ensure that such a replay attack did not occur.

Finally, the MSGMAC is piggybacked to the traditional acknowledgement message (Circle 5), and the home node can then verify MSGMAC (Circle 5) by recomputing it based on the components that it *knows* the requestor should have used. If any of the components of the counter or data response messages have been modified, the MSGMAC verification will fail. Additionally, an attacker cannot generate a correct MSGMAC because: (1) they do not have the secret key of the keyed-MAC function, (2) the timestamp included in the MSGMAC ensures that MSGMAC values do not repeat, and (3) the home node always knows the components that it is verifying in the MSGMAC.

We would like to point out that we have made the design choice to include the block's MAC value in the data response message and have this verified by the requestor for the following reason. This prevents an attacker from being able to forge any arbitrary value that the requesting processor will receive and use before the tampering can be detected at the home node through the MSGMAC. However, if this issue is not a concern, it would be straightforward to eliminate the block's MAC from the data response message and have the MSGMAC computation include both the block ciphertext and block counter instead of the MAC value. Finally, we note that the procedure of authenticating a counter and data response message is similar if the data is provided by a remote processor which currently owns the data. The main difference is that in this case, the

backward edge message and MSGMAC are communicated between the requesting and owning processors. Additionally, the owning processor may also be required to flush the data to the home node using the writeback protocol discussed in the next subsection.

4.4.2. Authenticating Data Writeback Messages. For authenticating writeback messages from a node owning a data block to the home node we apply another optimization compared to the basic delayed protocol described in Section 4.4, which allows us to eliminate the backward edge message. The key observation is that due to the way our encryption scheme manages the per-block counters, both a node owning a data block and the data's home node know the correct, current block counter value. Therefore, the block counter value can be used as the timestamp in the MAC computation for writeback messages. The owner of the data block can first generate the block ciphertext and MAC value using GCM and the per-block counter, and then send the ciphertext and MAC in the writeback message to the home node. Then the home node can use its known block counter value to recompute and verify the MAC of the received ciphertext.

Attackers cannot alter the ciphertext or MAC without triggering a failed MAC verification at the home node. Additionally they cannot replay an old ciphertext and MAC pair. The reason is that this would require also replaying the block counter value since it is included in the MAC computation, but this is infeasible since both the owning node and the home node know the correct counter value. The advantage of including the MAC value on the forward edge for writeback messages is that it still prevents replay attacks, and writeback messages are typically not latency-critical. This also relieves the owning processor from having to send its current message counter value to the home node with the writeback message and from having to buffer the MAC value that it would expect to see on the backward acknowledgement edge from the home node.

4.4.3. Authenticating Request and Other Non-Data Coherence Messages. The last types of messages that may require authentication are coherence messages which do not contain data or counters such as request, invalidation, and intervention messages. For these types of messages, we can apply the delayed authentication protocol in a straightforward manner. In full protection, along with each forward edge request, invalidation, or intervention message, the initiating node will also send its message counter value. Then the node receiving the message can compute the response MAC based on the timestamp of the message counter and initiator processor ID as well as the content of the received message. This MAC is sent along with the backward edge reply, ACK, NACK, etc. message to be verified by the initiating node.

Protection of these messages may be important in protecting certain types of attacks. For example, if an invalidation message is dropped by an attacker, then the node meant to receive that message may continue to use stale data from its cache, resulting in an *indirect replay attack*. By cryptographically protecting these messages, we can prevent such attacks because the initiator of the message will expect to see a valid MAC of the message returned to it, and the attacker cannot generate this valid MAC.

4.5 Security Analysis

In the previous sections we have discussed how our authentication mechanisms protect the various types of messages in our secure DSM system, and how this protection can prevent various attacks. Now we will reiterate the security of our delayed authentication protocol for DSM systems by identifying some of the major attacks that may be attempted and how our schemes can prevent these attacks. First and foremost, attackers may attempt to tamper with ciphertext, block counter, or MAC values in the system in hope of discovering some plaintext information or altering the execution of an application. However, since all data and block counter communication is protected by the data's cryptographically secure MAC value, attackers cannot tamper with any of these components without triggering a failed MAC verification. In addition if an attacker tries to replay all three components in unison to a requesting processor, this will be detected when the node that supplied the data verifies the response MAC included in the backward edge message. Since the response MAC includes information from the original message plus a unique timestamp, attackers also cannot forge this response MAC.

Attackers may also attempt to manipulate the system in other ways than just by modifying data or counter values. For example, attackers may attempt to drop messages. However, doing this will simply result in a timeout and retry if the initiating node does not receive a backward edge acknowledgement message. Additionally, attackers cannot forge such messages since they cannot generate the valid MAC that should accompany backward edge acknowledgement messages. Attackers may also attempt to reroute a message destined for one processor to another, however this is prevented because the response MAC to authenticate the message contains the destination node's processor ID.

Finally, we would like to reiterate that it may not be strictly necessary to protect all types of messages cryptographically with MAC values (*full* protection as discussed in Section 4.4). We believe that messages containing data, counter, or MAC values will certainly require this type of protection since this information is not directly used to maintain the coherence protocol. However, other types of messages and other message information such as invalidation and request messages and the type of a message typically have a direct effect on the state of the coherence protocol. Therefore tampering with these messages may be better protected by having a mechanism to report an abnormal amount of coherence protocol anomalies, as discussed in [19]. Nonetheless, our DSM authentication mechanisms allow for cryptographic protection of such messages if required.

5 Experimental Setup

We model our DSM system using a detailed execution-driven simulator based on SESC [8], an open source simulation environment. The simulated DSM system consists of 16 2GHz, 3-way out-of-order issue processors as the default. Each processor has separate data and instruction L1 caches that are both 16KB, 2-way with 64B lines and a 2 cycle hit latency. The L2 cache of each processor is 256KB, 8-way associative, with a 64B block size and 10 cycle round-trip hit latency. The reason for choosing a relatively small L2 cache for the evaluation is to induce higher miss rates that would stress our protection mechanisms more. For our protection mechanism, we model our single-level encryption scheme (Section 4.3 and the *data-counter-only* authentication scheme (Section 4.4). The memory system uses round-robin page allocation among the processors with 4KB pages. Each processor uses a 1 GHz, 4-Byte wide, split transaction memory bus to access the main memory with a 200 cycle uncontended round-trip latency. The processors are connected by a hypercube network with fixed-path routing. Each link has a bandwidth of 2 GB/s and the hop delay is 50ns modeled after the SGI Altix [24]. A MESI cache coherence protocol is implemented using a full bit vector with a home-based directory using reply forwarding.

The hardware for our protection scheme includes a default 32KB, 8-way counter cache to store the block counters of frequently accessed blocks in a processor's local memory. This cache is the same as in uniprocessor memory encryption schemes. For the owned-block pad buffer, we use a 32-entry FIFO buffer having a total size of 1 KB. For counter prediction, we add a small 32-entry mask buffer for storing a bit vector of which data blocks last had a counter value of zero. The total size of the bit vector is approximately 512 Bytes because each entry is 16-bytes in size. We assume a 2-cycle latency for accessing the pad buffer and mask buffer. The AES encryption engine is pipelined with an 80 cycle latency and 5 cycle occupancy. On a 2 GHz processor, this is comparable to the 37ns implementation shown in [9].

To evaluate our scheme, we use all 12 applications from the SPLASH-2 benchmark suite [26]. We use the standard input sets, and simulate all applications from start to completion with no fast forwarding or sampling. Table 1 shows the global L2 miss rate (number of L2 misses divided by all memory reference instructions), local L2 cache miss rate (number of L2 misses divided by L2 accesses), and percentage of L2 misses satisfied by a home node's local memory, of each application running on the simulated DSM without any security protection.

Table 1. Our applications and their memory access statistics.

App	Glb L2 MR %	Local L2 MR %	% Home Reqs
Barnes	0.05%	3.3%	64%
Cholesky	0.24%	45.9%	88%
FFT	0.83%	63.0%	99%
FMM	0.04%	18.9%	60%
LU	0.05%	24.4%	80%
Ocean	1.05%	27.4%	75%
Radiosity	0.07%	28.4%	66%
Radix	0.65%	22.3%	96%
Raytrace	0.39%	22.3%	91%
Volrend	0.26%	34.6%	89%
Water-n2	0.04%	6.4%	50%
Water-sp	0.03%	12.1%	88%

6 Evaluation

6.1 Overhead of DSM Data Protection

In Figure 6 we show the execution time overhead our single-level DSM data protection scheme, normalized to a DSM system with no support for data encryption or authentication. For comparison, we also show the overhead of the two-level, CACHED scheme from the previous work on data protection for DSM systems [19]. We compare against this scheme since it is the only one which is similar to ours in that it meets the design criteria of small on-chip storage overheads and the ability to scale to arbitrarily large DSM system sizes (in terms of number of processors in the DSM). However, we note that this scheme on its own cannot prevent all data replay attacks, so we augmented it with the ability to detect replay attacks using our delayed authentication technique discussed in Section 4.4. Thus, the two schemes are also comparable in terms of security strength.

From this figure, it is clear that while both schemes are similar in terms of small hardware support, the ability to support large systems, and security, our single-level scheme provides significantly better performance than the CACHED two-level scheme. The average overhead across all applications of our scheme is 1.6% while the overhead of CACHED is 5.3%, representing a reduction of overheads by a factor of $3.3\times$. In addition, there are several applications

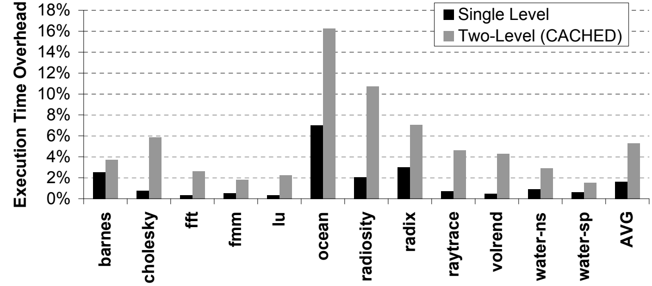

Figure 6. The execution time overhead of our single-level DSM data protection scheme versus previously proposed two-level protection using the CACHED scheme with 4-entry communication counter table [19].

which suffer from fairly significant overheads under the CACHED scheme, for example *ocean* and *radiosity* at 16.3% and 10.7% respectively. Our scheme reduces these overheads to 7.0% for *ocean* and 2.1% for *radiosity*. Equally important is the number of cases in which the execution time overheads are practically negligible. With CACHED, there are nine benchmarks that are slowed down by more than 2%, while for our proposed single-level scheme there are only three benchmarks slowed down by more than 2%. Since DSM systems are typically very pricey and they are often used to run critical applications, it is likely that performance overheads such as those seen for the worst-case applications with a two-level scheme are not tolerable. Additionally, the performance of our single-level scheme is more stable than that of the two-level scheme. With a standard deviation of 1.9% in execution time overhead, our single-level scheme provides more confidence to users that their applications will perform acceptably well than the two-level scheme with a standard deviation of 4.3%.

The central reason for the high performance overheads of the two-level scheme shown in Figure 6 is that cryptographic latencies may be exposed at multiple points in the critical path of a data fetch from a remote processor. More specifically, there are three points in this critical path where cryptographic delays may occur as shown in Figure 1 ((1) when a memory block requested by a remote processor is fetched on-chip by the block's home processor and decrypted, (2) when the block is encrypted again to be sent to the requesting processor, and (3) when the requesting processor receives and decrypts the block on-chip). Our data confirms this observation: on average, cryptographic latencies are at least partially exposed at point (1) 9% of the time, at point (2) 29% of the time, and at point (3) 46% of the time. This means that roughly only $(1-0.09) \times (1-0.29) \times (1-0.46) = 35\%$ of the time full cryptographic latencies are hidden in the CACHED scheme (versus 82% of the time for our scheme – we will discuss the result in Figure 7 later). This shows that two-level schemes are inherently inefficient because there are too many points on the critical path where delays can be introduced.

Now that we have examined the performance of our single-level DSM data protection scheme, we will take a closer look at the reasons for its low performance overhead. Figure 7 shows the percentage of off-chip data requests for which the decryption pads are fully generated (pad hit), partially generated (pad half-miss), or not generated (pad miss) when the requested data arrives on-chip. If a pad is fully generated, the decryption latency is totally hidden, while if it is partially generated then the latency is partially hidden. From this figure, we can see that for most applications and on average the full,

critical-path decryption latency is hidden 80% of the time or more. Additionally, the percentage of requests which suffer from the full decryption latency is less than 1% in almost all applications. On average, 82.4% of the time pad generation latency is fully hidden, 17.1% of the time it is partially hidden, while only 0.5% of the time the latency is fully exposed.

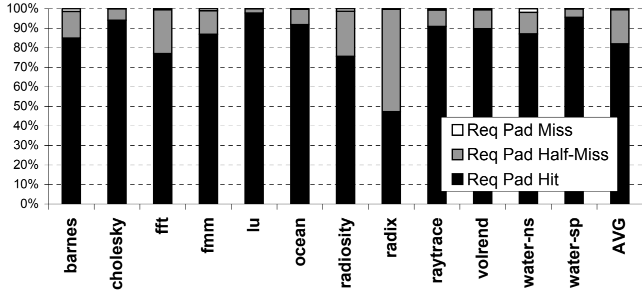

Figure 7. Fraction of off-chip data requests that experience pad hit (full latency- hiding), half-miss (partial latency-hiding), and miss (no latency-hiding).

To further explain the performance of our scheme, we present Figures 8 and 9. In Figure 8, we show the percentage of off-chip data requests for which we correctly predict the counter value for the data block. In Figure 9 we show how frequently the home node of a data block finds the requested block's counter cached in its local counter cache. This percentage is the total height of the two bars, and it corresponds to the percentage of data requests in which the home node can reply with the data's counter early to hide the decryption delay at the requestor. However, with our counter prediction scheme, if the counter has been predicted correctly by the requestor, the home node does not need to reply with a separate counter message. The gray portion of the bars shows how frequently this event occurs.

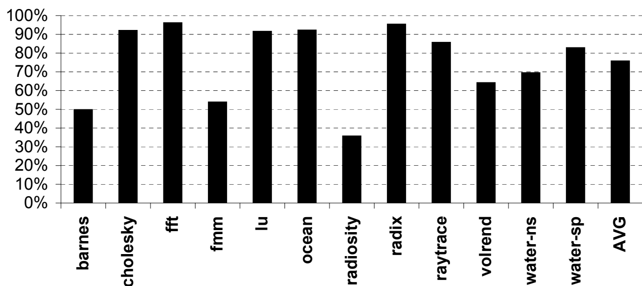

Figure 8. Percentage of requests for which the counter value is correctly predicted.

Figure 8 shows that despite its simplicity, our counter prediction scheme is very successful at correctly predicting counter values. The correct prediction rate is 76% on average, and over 90% for 5 applications. This high prediction rate benefits our scheme in two ways. First, if counters are predicted correctly, then it is *very* likely that the decryption pad is fully pre-generated before it is needed, thus fully hiding the decryption latency. Also, as shown in Figure 9, we can eliminate a large number of separate counter messages from the home processor to the requestor. This reduces the pressure placed on interconnect bandwidth, because a separate counter message requires more overhead than simply including the counter value with the data of the reply message. Figure 9 also

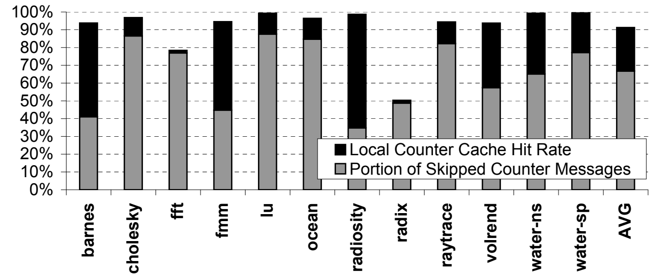

Figure 9. Local counter cache hit rate and portion of counter messages skipped due to correct prediction.

shows that, when counter prediction fails, most of the time we can still hide the decryption latency by forwarding the correct counter. This figure shows that a block's counter value can either be predicted or sent early over 90% of the time in most cases, and 91% of the time on average.

The final comparison we make between our single-level scheme with the previously proposed two-level scheme is on the AES unit bandwidth utilization shown in Figure 10. This figure shows that due to the reduced amount of cryptographic work, for most applications we observe a large decrease in AES utilization with our single-level scheme. For all but two of the applications, we use the AES unit 30% less than in the two-level scheme, and for some applications this savings is closer to 40%. This result shows that we provide secure data encryption and authentication in a DSM system with fewer cryptographic operations required compared to a two-level scheme.

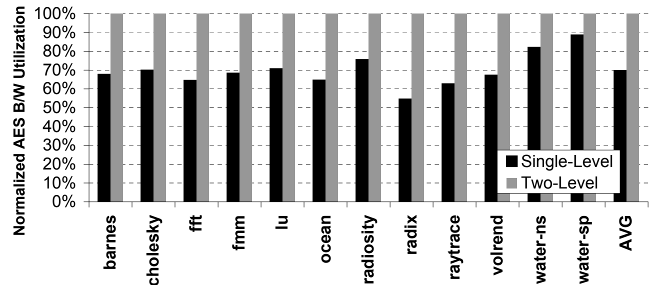

Figure 10. The normalized AES bandwidth usage of single-level vs. two-level DSM data protection.

6.2 Sensitivity Analysis

In Figure 11, we show how our single-level DSM data protection scheme performs as the number of processors in the DSM system increases. Again, the performance is shown as execution time normalized to a DSM system with equivalent configuration but with no support for data protection.

It is clear from this figure that the overheads of our protection scheme remain low as the number of processors increases. In fact, in most cases and on average the overhead decreases with respect to the system size. On average, the overhead goes from 1.6% on 16-processor DSM to 1.0% for both 32-processor and 64-processor DSM system. The worst-case overhead also improves significantly from 7.0% on 16 processors down to 4.9% on 32 processors and 2.4% on 32 processors. With a larger DSM size, there are more remote data requests because data is scattered around more nodes. However, at the same time communication latencies increase with

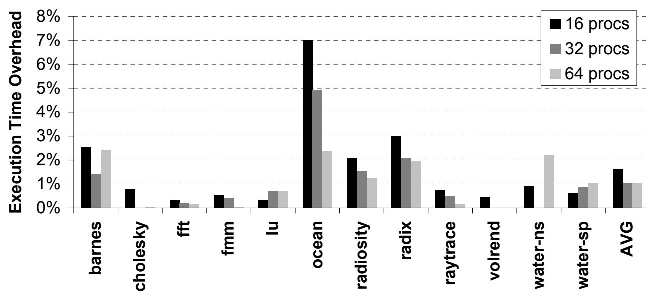

Figure 11. Execution time overhead of our single-level DSM data protection scheme across system size.

the number of processors since data must travel more hops across the interconnection network on average, making the impact of cryptographic latencies less significant relative to the total remote data fetch latencies. The figure shows that in general, the increase in inter-node communication is the more important factor, resulting in execution time overheads of just 1% as the DSM size increases.

7 Conclusions

In this paper, we have proposed a single-level data encryption and authentication scheme to protect the confidentiality and integrity of data in Distributed Shared Memory multiprocessors that use a point-to-point interconnect. Our scheme reduces the amount of cryptographic work by a factor of three compared to a previously proposed two-level approach, and reduces the average execution time overhead by a factor of 3.3× (from 5.3% to 1.6%). In addition, our single-level scheme reduces the performance penalty of applications that suffer from intolerable overheads with a two-level scheme down to a much more acceptable level. The overheads due to our single-level scheme are also much more stable than those seen with a two-level scheme. Our approach requires only relatively minor modifications to secure processors used for uniprocessor systems, and can scale to large numbers of processors.

References

[1] D. Bartholomew. On Demand Computing – IT On Tap? *http://www.industryweek.com/ReadArticle.aspx?ArticleID=10303&SectionID=4*, June 2005.

[2] D. Clarke, G. E. Suh, B. Gassend, M. van Dijk, and S. Devadas. Checking the Integrity of Memory in a Snooping-Based Symmetric Multiprocessor (SMP) System. In *MIT CSAIL CSG-TR-470*, July 2004.

[3] B. Gassend, G. Suh, D. Clarke, M. Dijk, and S. Devadas. Caches and Hash Trees for Efficient Memory Integrity Verification. In *Proc of the 9th International Symposium on High Performance Computer Architecture (HPCA-9)*, 2003.

[4] T. Gilmont, J.-D. Legat, and J.-J. Quisquater. Enhancing the Security in the Memory Management Unit. In *Proc. of the 25th EuroMicro Conference*, 1999.

[5] A. Huang. *Hacking the Xbox: An Introduction to Reverse Engineering*. No Starch Press, San Francisco, CA, 2003.

[6] A. B. Huang. The Trusted PC: Skin-Deep Security. *IEEE Computer*, 35(10):103–105, 2002.

[7] IBM. IBM Extends Enhanced Data Security to Consumer Electronics Products. *http://domino.research.ibm.com/comm/pr.nsf/pages/news.20060410_security.html*, April 2006.

[8] J. Renau et al. SESC. *http://sesc.sourceforge.net*, 2004.

[9] T. Kgil, L. Falk, and T. Mudge. ChipLock: Support for Secure Microarchitectures. In *Proceedings of the Workshop on Architectural Support for Security and Anti-Virus (WASSA)*, Oct. 2004.

[10] M. G. Kuhn. Cipher Instruction Search Attack on the Bus-Encryption Security Microcontroller DS5002FP. *IEEE Transactions on Computers*, Oct(10), 1998.

[11] M. Lee. Global ATM Security Alliance focuses on insider fraud. *ATMMarketplace, http://www.atmmarketplace.com/article.php?id=7154*, May 2006.

[12] M. Lee, M. Ahn, and E. Kim. I2SEMS: Interconnects-Independent Security Enhanced Shared Memory Multiprocessor Systems. In *Proc. of the International Conference on Parallel Architectures and Compilation Techniques*, 2007.

[13] R. B. Lee, P. C. Kwan, J. P. McGregor, J. Dwoskin, and Z. Wang. Architecture for Protecting Critical Secrets in Microporcessors. In *Proc. of the International Symposium on Computer Architecture*, 2005.

[14] D. Lie, J. Mitchell, C. Thekkath, and M. Horowitz. Specifying and Verifying Hardware for Tamper-Resistant Software. In *IEEE Symposium on Security and Privacy*, 2003.

[15] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. MItchell, and M. Horowitz. Architectural Support for Copy and Tamper Resistant Software. In *Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[16] Maxim/Dallas Semiconductor. DS5002FP Secure Microprocessor Chip. *http://www.maxim-ic.com/quick_view2.cfm/qv_pk/2949*, 2007 (last modification).

[17] D. A. McGrew and J. Viega. The Galois/Counter Mode of Operation (GCM). http://csrc.nist.gov/Crypto Toolkit/modes/ proposed-modes/gcm/, 2004.

[18] T. Olavsrud. HP Issues Battle Cry in High-End Unix Server Market. *ServerWatch, http://www.serverwatch.com/news/article.php/139-9451*, 2000.

[19] B. Rogers, Y. Solihin, and M. Prvulovic. Efficient data protection for distributed shared memory multiprocessors. In *International Conference on Parallel Architectures and Compilation Techniques*, 2006.

[20] W. Shi and H.-H. Lee. Authentication Control Point and Its Implications for Secure Processor Design. In *Proc. of the 39th Annual International Symposium on Microarchitecture*, 2006.

[21] W. Shi, H.-H. Lee, M. Ghosh, and C. Lu. Architectural Support for High Speed Protection of Memory Integrity and Confidentiality in Multiprocessor Systems. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pages 123–134, September 2004.

[22] W. Shi, H.-H. Lee, M. Ghosh, C. Lu, and A. Boldyreva. High Efficiency Counter Mode Security Architecture via Prediction and Precomputation. In *Proceedings of the 32nd International Symposium on Computer Architecture*, June 2005.

[23] W. Shi, H.-H. Lee, C. Lu, and M. Ghosh. Towards the Issues in Architectural Support for Protection of Software Execution. In *Proceedings of the Workshop on Architectureal Support for Security and Anti-virus*, pages 1–10, October 2004.

[24] Silicon Graphics, Inc. SGI Altix 3000 Data Sheet. *http://www.sgi.com/products/servers/altix*, 2004.

[25] G. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. Efficient Memory Integrity Verification and Encryption for Secure Processor. In *Proc. of the 36th Annual International Symposium on Microarchitecture*, 2003.

[26] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The splash-2 programs: characterization and methodological considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture*, pages 24–36, 1995.

[27] C. Yan, B. Rogers, D. Englender, Y. Solihin, and M. Prvulovic. Improving cost, performance, and security of memory encryption and authentication. In *Proc. of the International Symposium on Computer Architecture*, 2006.

[28] B. Yang, S. Mishra, and R. Karri. A high speed architecture for galois/counter mode of operation (gcm). In *Cryptology ePrint Archive: Report 2005/146*, 2005.

[29] J. Yang, Y. Zhang, and L. Gao. Fast Secure Processor for Inhibiting Software Piracy and Tampering. In *Proc. of the 36th Annual International Symposium on Microarchitecture*, 2003.

[30] Y. Zhang, L. Gao, J. Yang, X. Zhang, and R. Gupta. SENSS: Security Enhancement to Symmetric Shared Memory Multiprocessors. In *International Symposium on High-Performance Computer Architecture*, February 2005.