ECE 5770 : Resilient Computer Systems
Surveying Efficient Low-Cost Memory Protection Techniques

Udit Gupta
School of Electrical and Computer Engineering, Cornell University, Ithaca, NY
ug28@cornell.edu

1. Problem Statement and Motivation

Recently, there has been an increasing effort in developing trusted computing platforms augmented with specialized hardware modules that provide security features such as authentication and decryption/encryption. One such security feature that remains a focus in trusted computing research is protecting off-chip memory. Protecting off-chip memory includes integrity verification and maintaining confidentiality of private data. Challenges in designing memory protection mechanisms include providing memory authentication and encryption primitives at a low cost without compromising security. Current work also focuses on memory protection schemes for general purpose and high performance computing systems. The domain of memory protection in the mobile and embedded computing domains is relatively less studied and poses interesting challenges. Mobile and embedded devices often operate under strict power and area constraints. Providing secure memory protection while following the prescribed power and area constraints as well as maintaining performance of the computing devices is an ongoing challenge.

As mobile and embedded computing chips incorporate an increasing number of transistors, they inevitably become more heterogeneous - encompassing multi-core processors, graphics processing units (GPUs) and hardware accelerators. Providing memory authentication and encryption for such heterogeneous, shared memory systems is another challenge as current work mainly focuses on the uni-processor architectures.

This survey aims to explore state-of-the-art architectures and techniques developed by research in the area of efficient memory integrity verification and encryption. We limit this survey to mainly exploring hardware based memory protection schemes as past research as shown that implementing the encryption and integrity verification schemes solely in software incurs large processing overhead. This survey does however, discuss some memory protection schemes that couple secure OS-kernel features and hardware modules for providing additional memory protection.

1.1 Threat Model

This survey assumes the following threat model, unless stated otherwise, in the following discussions:

1. Attackers are capable of constructing `spoofing` devices or hardware. Generally a `spoofing` attack replaces an existing memory block or hardware unit with a fake one installed by the attacker.
2. Attackers are capable of `splicing` or `relocation` attacks. `Splicing/relocation` attacks involve replacing a memory block at address `A` with a memory block at a different address `B`. This can also be achieved by changing the memory address being accessed.
3. Attackers are capable of `snooping` any interconnection fabrics and buses connecting the processors and off-chip memory. This involves both software and hardware based snooping attacks.
4. Attackers are capable of `replay` attacks. Replay attacks also include intelligent replay attacks where attacks are made after monitoring the communication between the processor and off-chip memory. We assume that any communication patterns/protocols may be exploited in this threat model. In general, `replay` attacks involve writing a memory block at address `A` with an older version of the memory block at the same address `A`.
5. Attackers are capable of isolating the off-chip memory chip and reading and modifying the contents of the memory. This is especially pronounced in systems where non-volatile memory technologies, such as phase change memory (PCM), are used since memory persists even after the system is powered off. When using non-volatile memory technologies, limited write durability and should also be considered.

Generally, the thread model assumes that the interconnect fabric and buses are not secure and attackers have access and the capability to tamper with these hardware systems. The security measures employed aim to prevent or detect attacks following this threat model. Attacks targeting availability of hardware modules such as the processor or off-chip memory are outside of the scope of this survey. We believe that this

is a valid threat model as it has been used in previous studies [3].

The following sections are organized as follows: Section 2 reviews the related work in the research community, Section 3 discusses the strengths and weakness of the current work and Section 4 concludes with the future directions for efficient, low-cost memory protection techniques.

2. Related Work

We organize the related work surrounding efficient, low-cost memory protection techniques in two separate sections : **memory encryption** and **memory integrity verification**.

2.1 Memory Encryption

The goal of memory encryption is to protect the confidentiality of sensitive information from unauthorized subjects. Sensitive information includes but is not limited to: secret keys, secret code (e.g. proprietary boot code) and private data. The `National Institute of Standards and Technology` (NIST) actively regulates the encryption standards. Recently, two symmetric key cryptographic algorithms are being increasingly used in trusted hardware computing : Triple Data Encryption Standard (3DES) and Advanced Encryption Standard/Rjindael (AES). As AES provides systems with a higher minimum security strength, it is more widely used in recent hardware accelerated trusted computing studies as compared to 3DES. For instance, Intel has developed extensions to it's instruction set architecture (ISA) to enable fast encryption and decryption using AES. The studies in this survey mainly focus on using AES to develop efficient, low-cost encryption and decryption modules.

2.1.1 AES Counter Mode Encryption

AES supports three general modes of operation: Electronic Codebook (ECB), Cipher Block Chaining (CBC) and Counter (CTR). Since the AES-ECB mode preserves patterns from the plaintext to the ciphertext, it does not provide adequate confidentiality protection. Both AES-CBC and AES-CTR eliminate this information leak by chaining the encryption between individual AES encryption blocks (16-bits). This however precludes users from encrypting and decrypting blocks in parallel (as supported by AES-ECB). As discussed in [7], decrypting the ciphertext stored in memory can drastically increase the load use delay latency of processing systems. The added latency is a result of the serial memory fetch then decrypt memory block operation. To hide this latency Suh et. al. use a `one-time pad` (OTP) based encryption / decryption technique. The OTP is generated by encrypting the plaintext under a a secret key using AES-CTR with the following equation:

$$\texttt{plaintext} = \texttt{FV}||\texttt{Addr}||\texttt{TS}||\texttt{i}$$

In the equation above, `FV` is a fixed vector, `Addr` is the memory address being accessed, `TS` is a timestamp stored along with the encrypted data and `i` is the block number. The use of a monotonically incrementing counter, `TS`, ensures the uniqueness of the `OTP` - proven to be cryptographically secure. Generating the `OTP` using this method enables the system to overlap the AES-CTR decryption latency with the memory fetch / accessing latency [7]. To mitigate the fetch latency, Suh et. al. also mentions the `TS` values can be cached latency to better hide the AES-CTR decryption latency. This method reduces the load-use-delay latency to the latency of performing an $\oplus$ between the generated `OTP` and encrypted data.

2.1.2 AEGIS

Furthering the AES-CTR based decryption [7], AEGIS [8] adds low-level OS-kernel support to provide additional confidentiality features. That is, AEGIS is a single-chip secure processor that leverages OS-kernel modes to provide additional encryption (ME mode) and integrity-verification (IV mode) features. The OS also manages four distinct protected regions in virtual memory:

1. Read-only (static) verified memory
2. Read-write (static) verified memory
3. Read-only (dynamic) private memory
4. Read-write (dynamic) private memory

These modes help isolate the virtual memory space of each process to ensure that secret keys are kept confidential across multiple users and between users and the supervisor. AEGIS also employs physically random functions (also known as physically unclonable functions) to generate secret keys as they are less prone to physical hardware attacks than the alternatives such as using non-volatile memory (e.g. EEPROM) and fuses.

2.1.3 Improving Encrypted Memory Performance

The previous two studies [7] [8] illustrate the fundamental AES-CTR generated `OTP` encryption scheme and OS kernel based memory isolation schemes. Though these studies provide the basic building blocks for memory encryption, they do not address the challenge of minimizing power, performance and durability overhead when writing encrypted segments to memory. Traditionally, plaintext `writes` to memory only change 12 % of destination block's bits [11]. According to the `Avalance Effect` [9] changing a single plaintext bit causes 50 % of the ciphertext's bits to change. For instance, changing the most significant bit in the plaintext should cascade the change to every encryption block in the AES-CTR chain. This is a desired property when using cryptographic encryption algorithms; changing a single plaintext bit should cause the ciphertext to appear completely "random". In comparison to writing plaintext data, encrypted data causes 4x as many bits to flip in memory.

Many modern memory systems use more efficient write schemes such as `Data Comparison Write` (DCR) [12] and `Flip-n-write` [1] (FNW) to take advantage of the fewer overridden plaintext bits. Techniques such as DCR and FNW however, only work when writing plaintext data

to memory. For instance, FNW bounds the number of bit flips to a maximum of half the number of bits per line. Such optimizations limit the number of modified bits to only 10 % to 15 % on average. This greatly improves write performance and durability of limited-endurance memory such as non-volatile memory (e.g. PCM). Unfortunately, these optimizations are not amenable to encrypted data when 50 % of the bits change on average - increasing power, reducing durability and reducing write bandwidth. Dual Counter Encryption (DUECE) [11] is one solution to write-efficient encryption.

DUECE makes the observation that only a few words are changed during each write operation so only the changed words need to be re-encrypted. DUECE maintains a leading (LCTR) and trailing (TCTR) counter for each cache line as well as an encryption time slot : `Epoch`. An `Epoch` is defined as four writes to the cache. Within the `Epoch` interval, modified words are encrypted using the LCTR. At the `Epoch` boundary, the entire line is re-encrypted under a new counter, `LCTR = TCTR`. In order to reduce the storage space requried by LCTR and TCTR, the TCTR is computed by masking a pre-determined number of least-significant bits of the LCTR. The pre-determined number of bits make up a `wordsize`. The `Epoch` and `wordsize` must be carefully chosen to optimize the bit flips, full encryption overhead, storage overhead and fineness of tracking modified words in cachelines. DUECE performs well when programs write to cachelines in sparse and regular patterns. DUECE reduces the number of bit flips for encrypted memory writes by a factor of 2 (from 50 % to 24 % bit flips) [11].

DUECE also has the option of enabling a DynDUECE operating mode. DynDUECE allows the write scheme to dynamically switch from DUECE to FNW if the program modifies all the words in a specific cache line. In the case of denser memory writes, FNW is able to reduce bit flips of encrypted memory by 14 % [11]. The dynamic switching between DUECE and FNW also only incur a 1 bit overhead per cacheline.

Finally, DUECE employs a novel `Horizontal Wear Leveling` technique to evenly distribute the writes across words in a cacheline. In programs with regular memory access patterns, it is often the case that some words in a cache line are accessed more frequently than others. This causes differential hardware wear leveling on the memory units - which is more pronounced in non-volatile memory architectures such as PCM. To mitigate wear leveling DUECE employs an efficient horizontal wear leveling technique that uses algebraic functions to calculate rotation levels. Overall, DUECE is able to drastically improve write throughput as well as overall system performance while reducing the energy delay product (EDP).

2.1.4 Galois Counter Mode Based Encryption

In addition to using AES-CTR mode encryption, some studies have investigated the use of Galois/Counter Mode (GCM) encryption: a highly parallel and efficient block cipher mode that supports authenticated encryption [2]. GCM is similar to AES-CTR mode in that it can be used to generate `OTPs`. Yan et. al developed a GCM based encryption method that uses a split counter comprising of a major and minor counter [10]. The major counter is common over an encryption page and a minor counter is maintained for each block. The minor counter requires an additional 8 bits for each 64 bits. This study will be discussed more in the following section.

2.2 Integrity Verification

The goal of memory integrity verification is to protect the integrity of memory and detect corrupted memory. Generally, the protection mechanism must be able to detect when memory is maliciously changed. That is, when a processor reads from memory it should get the last version of the data that was written. Over the years a few main cryptographic integrity verification algorithms have been developed including : Hashes and Message Authentication Code (MAC). The first strategy in providing integrity verification involves storing either a hash or MAC generated tag with each memory block of the off-chip memory. The hash or MAC is computed and stored during write operations and verified during read operations. The hash (128 bit) based approach incurs a 25 % memory overhead and the MAC (64 bit nonce) incurs a 12.5 % memory overhead for standard 512-bit cache line off-chip memory authentication [3]. The previously discussed authentication techniques provide memory authentication mechanisms at a high cost in terms of off-chip memory overhead. The following sections focus on discussing `Integrity Trees` which aim to reduce the memory overhead required to provide integrity verification for off-chip memories.

2.2.1 Integrity Trees

The first approach to using integrity trees for off-chip memory authentication is Merkle trees [5]. A Merkle tree is a tree of hashes as shown in Figure 1a.

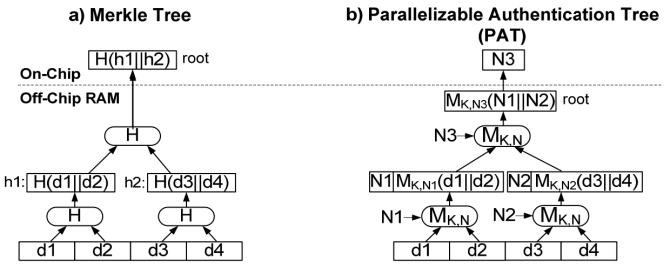

Figure 1: H: Hash Tree. M: MAC function. d_y : data block y. N_z: Nonce z. K : Secret Key.[3]

The `integrity check` or `verification` operation performed during a memory read operation can be parallelized for a Merkle Tree however, updating the tree is a

sequential process that must be performed from the leaves up to the root. Qualitatively, the Merkle tree reduces the `integrity check` from a growing linearly with respect to the number of memory blocks to growing logarithmically. There has been work in reducing the architectural overhead of `integrity check` operations including optimizing the arity of the hash tree, the granularity of the authentication check blocks and maintaining a cache for the hash tree to improve the "hash fetch" rate. In general these techniques tend to be complicated and require significant hardware enhancements. For instance, aggregating the hash and data caches into a single monolithic cache can increase miss rates and cache pollution - degrading the overall system performance.

As seen in Figure 1b, the Parallelizable Authentication Tree (PAT) [4] is an alternative to Merkle trees. Unlike Merkle trees, PATs overcomes the issue of non-parallelizable integrity tree update operation during memory write operations by using MACs for providing authentication instead of hash functions [3]. The parallelizability of the integrity tree update does come at a memory storage cost - PAT tree generally requires $\frac{3}{2}$ the amount of memory that Merkle trees require.

2.2.2 Log Hash Tables

As discussed earlier, Merkle trees and PATs maintain hashes or MAC tags for authenticated memory blocks. Suh et. al. adopt an alternative approach using a log based hash table to provide memory integrity. In stead of maintaining a hash for each memory block, a log of memory operations is generated to maintain a snapshot of the system [7]. A separate read and write log is maintained with minimal runtime overhead so the processor can verify the integrity of *sequence* of operations later in time [7]. This is considered `lazy` memory integrity management as the `integrity check` is not performed for every read operation but rather less frequently - the intervals of performing `integrity check` operations is much longer. That is, the system would detect a corrupted memory much after the physically attack took place and maliciously tampered with the memory. The hash log is shown to be more performance efficient while incurring a lower memory space overhead than the cached hash tree approach discussed earlier.

2.2.3 AEGIS

As discussed in the memory encryption section, AEGIS is a single-chip secure processor that provides both memory encryption and integrity verification mechanisms. AEGIS leverages Physically Random Functions (PRF), sometimes called Physically Unclonable Functions (PUFs), to provide off-chip memory protection. AEGIS also uses the OS-kernel modes to provide additional integrity verification (IV) features. AEGIS allows programs to overlap memory encryption (ME) and IV stages together, or keep them separate, in order to maximize flexibility and provide tamper-resistant off-chip memory protection [7].

2.2.4 Galois/Counter Mode Integrity Verification

In addition to using hash and MAC based authentication schemes, some studies investigate the use cryptographic authenticated encryption algorithm such as GCM to provide memory integrity verification [10]. GCM relies on GMAC / GHASH function to provide authentication capabilities. GMAC / GHASH functions are less computationally intensive and incur lower latency overheads compared to hash algorithms such as MD-5, SHA-1 and SHA-2. Like the AES-CTR mode and GCM based encryption mechanisms, the memory fetch latency can be used to hide the GMAC / GHASH functions. The GMAC / GHASH functions generate `OTP`s that are $\oplus$'ed with the plaintext data to make the tags that will be checked on memory read operations. By hiding the `OTP` generation latency with memory fetching, the GMAC / GHASH functions incur very little latency overhead to the overall system. In order to further improve performance, Yan et. al. also parallelized the Merkle tree `integrity check` operation [10].

2.2.5 Multi-Core Integrity Verification

Where the previous studies and techniques focused on single-core tamper-resistant memory protection, Shi et. al. investigate memory encryption and integrity verification techniques for multi-core systems [6]. Shi et. al discuss how Merkle tree (and similarly PATs and log hash tree) approaches alone do not work for multiprocessor systems. Additional hardware or software support must be provided to correctly implement memory encryption and integrity verification. For instance, processes running on separate cores must not be allowed to access and modify the same memory integrity trees otherwise the false appearance of malicious memory corruption could be detected by the `integrity check` operations. Shi et. al. instead use a single global MAC tree managed by the memory controller and rely on a secure OS kernel to manage private session keys across cores for specific user and supervisor processes. The session keys are used to generate unique secret keys used in the MAC based memory integrity verification.

3. Strengths and Weaknesses of Existing Works

A number of studies have been discussed for both memory encryption and integrity verification. This section will compare the discussed techniques for encryption and integrity verification separately.

3.1 Memory Encryption

The following table summarizes the studies discussed for memory encryption mechanisms [7, 8, 10, 11] for parameters such as encryption algorithms used, memory isolation capabilities, hardware overhead, power and durability.

Table 1. Comparing Memory Encryption Mechanisms

	AES-CTR	AEGIS	DUECE	GCM
Encryption Alg.	AES-CTR	AES-CTR	AES-CTR	GCM
Mem. Isolation	No	Yes	No	No
Counters	1	1	1	1
AES-Engines	1	1	2	1
Avg. Bit Flips	50%	50%	24%	50%
Durability	1x	1x	2x	1x

The AES-CTR based method [7] acts as the baseline approach for comparing memory encryption mechanisms as it provides the minimum required security features without tremendous hardware enhancements. AEGIS [8] leverages the same cryptographic encryption technique as the AES-CTR [7] study, however it also provides memory isolation between users and supervisor. AEGIS is the only study to use secure OS-kernel level techniques to provide additional memory confidentiality protection.

DUECE [11] is the only study to optimize writing encrypted data to memory in terms of performance, power and memory durability. DUECE reduces the average bit flips for programs with sparse, regular memory access patterns to 24% percent from the nominal 50% of encrypted data as determined by the `Avalance` effect [9]. The reduction greatly improves write performance, write bandwidth, power consumption and durability. DUECE is also the only study to leverage *wear leveling* mechanisms to improve memory durability by a factor of 2x - especially for non-volatile memory systems such as PCM. Though DUECE uses both the LCTR and TCTR, since it only stores one counter in memory, since the TCTR is derived from the LCTR, we say that it only has 1 counter worth of memory overhead. DUECE however does require two AES-Engines to efficiently decrypt the ciphertext on read operations [11]. Finally, even though the GCM approach may in reality incur a lower memory overhead compared to the DUECE approach, we omit comparing specific bitwidths for counters as these are largely system and design dependent and obfuscate the general trade-offs of each approach.

In general, the DUECE based memory encryption technique is the most optimized for performance, power, and durability however there may be room for improvement to reduce the number of required AES-Engines and provide additional virtual memory isolation techniques.

3.2 Integrity Verification

The following table summarizes the studies discussed and their general trade-offs in regards to integrity verification of off-chip memories [4] [5] [6] [7] [8] [10].

For memory integrity verification, the Merkle trees [5] serve the as the baseline protection technique since the latter improve upon Merkle trees in different ways. We do not compare memory overhead for each technique as the studies do not employ similar systems or use comparable metrics for clear overhead comparisons.

Table 2. Comparing Integrity Verification Mechanisms

	Parallel	Mem. Isolation	MultiCore	Alg.
Merkle	No	No	No	Hash
PAT	Yes	No	No	MAC
Log Hash	No	No	No	Hash
AEGIS	No	Yes	No	Hash
GCM	No	No	No	GMAC
MultiCore	No	No	Yes	Hash

As seen in Table 2, each technique exhibits it's own advantages and disadvantages. PATs are the only fully parallelizable authentication trees covered in this study. Generally speaking however, PAT has a greater memory overhead than Merkle trees - by a factor of $\frac{3}{2}$ [3]. Similarly, AEGIS is the only system that offers memory isolation between users and the supervisor and Yan et. al.'s authentication mechanism is the only one that supports multicores. AEGIS requires secure OS-kernel support to provide memory isolation between users and the supervisor. The multicore approach [6] adds additional hardware complexity to mediate multiple read/write operations and keeping a consistent memory state. The Log Hash [7] approach is more efficient for programs where `integrity check` operation calls are infrequent. Finally the GCM approach [10] leverages a more efficient cryptographic authentication algorithm - `GMAC/GHASH` than the other studies. Comparable to hiding the decryption latency on read operations by using AES-CTR generated `OTPs`, `GMAC/GHASH` can also hide the `integrity check` operation by pre-generating `OTPs` which are used to authenticate the data. This greatly improves the latency of authenticating memory.

In general, there is no comprehensive memory authentication technique like DUECE for memory encryption that provides a holistic solution to integrity verification.

4. Future Directions

This survey aims to summarize the current work in efficient low-cost memory protection - integrity verification and decryption/encryption. This survey is a brief overview of the field and thus does not cover all techniques and approaches to memory protection. In stead, this survey uses representative studies to broadly illuminate the domain. From the studies it seems as though more thorough work has been conducted in memory encryption - namely the DUECE approach. Integrity verification still needs more system level development - optimizing for power, performance, and area overhead of homogeneous uni-core and multicore processors and well as heterogeneous systems. It is clear that simple uni-core processor protection schemes do not directly apply to multicore systems. Much of the current research also does not optimize a unified encryption and integrity verification systems. Studies usually focus on one of these memory protection domains which raises the question if there is a less

explored design space for unified memory protection mechanisms. Also, though some studies consider the performance and memory durability, this is also less explored than performance benchmarks for various encryption and integrity verification schemes.

Some avenues of future work include exploring heterogeneous architecture memory protection mechanisms, memory protection mechanisms with power and area as first-order constraints - focusing on the domain of embedded or `Internet of Things` domains, or heterogeneous architectures. There may also be merit in exploring more GCM based encryption and integrity verification schemes as the cryptographic authenticated encryption algorithm provides efficient derivatives of both security features.

References

[1] S. Cho and H. Lee. Filp-n-write: A simple deterministic technique to improve pram write performance, energy and endurance. *Int'l Symp. on Microarchitecture (MICRO)*, 2009.

[2] M. Dworkin. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. *NIST Special Publication*, 2007.

[3] R. Elbaz, D. Champagne, C. Gebotys, B. R. Lee, N. Potlapally, and L. Torres. Hardware Mechanisms for Memory Authentication: A Survey of Existing Techniques and Engines. *Transactions on Computational Science IV, Lecture Notes in Computer Science (LNCS)*, 2009.

[4] E. W. Hall and S. C. Jutla. Parallelizable Authentication Tree. *Springer-Verlag Berlin Heidelberg*, 2006.

[5] R. Merkle. Secrecy, authentication and public key systems. *PhD thesis, Department of Electrical Engineering Stanford*, 1979.

[6] W. Shi, S. H.-H. Lee, M. Ghosh, and C. Lu. Architectural Support for High Speed Protection of Memory Integrity and Confidentiality in Multiprocessor Systems. *Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2004.

[7] E. G. Suh, D. Clarke, B. Gassend, v. M. Dijk, and S. Devadas. Efficient Memory Integrity Verification and Encryption for Secure Processors. *Int'l Symp. on Microarchitecture (MICRO)*, 2003.

[8] G. E. Suh, W. C. O'Donnell, and S. Devadas. AEGIS: A single-chip secure processor. *Information Security Technical Report*, 2005.

[9] A. Webster and T. S. On the design of s-boxes. *Advances in Cyptology CRYPTO*, 1986.

[10] C. Yan, B. Rogers, D. Englender, Y. Solihin, and M. Prvulovic. Improving Cost, Performance, and Security of Memory Encryption and Authentication. *Int'l Symp. on Computer Architecture (ISCA)*, 2006.

[11] V. Young, J. P. Nair, and K. M. Qureshi. DEUCE: Write-Efficient Encryption for Non-Volatile Memories. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015.

[12] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. *Int'l Symp. on Computer Architecture (ISCA)*, 2009.