InvisiMem: Smart Memory for Trusted Computing

Shaizeen Aga and Satish Narayanasamy
University of Michigan, Ann Arbor
{shaizeen, nsatish}@umich.edu

Abstract

A practically feasible low-overhead secure hardware design that provides strong defenses against memory bus side channels remains elusive. This paper observes that smart memory, memory with compute capability and a packetized interface, can dramatically simplify this problem. InvisiMem expands the trust base to include the logic layer in the smart memory to implement cryptographic primitives, which allows the secure host processor to send encrypted addresses over the untrusted memory bus. This eliminates the need for expensive address obfuscation techniques based on Oblivious RAM (ORAM). In addition, smart memory enables simple and efficient solutions for ensuring freshness using authenticated encryption, and for mitigating memory bus timing channel using constant heart-beat packets. We demonstrate that InvisiMem designs are one to two orders of magnitude lower in performance, space, energy, and memory bandwidth overhead, compared to ORAM-based solutions.

1. Introduction

Cloud computing allows clients to outsource their computation to untrusted cloud service providers. Many organizations (health, finance, defense, etc.) hesitate to use third-party cloud services due to privacy concerns [69, 21]. Privacy concerns exist even when these organizations privately own and operate their data-centers [7, 6], as they can neither trust all the software that run on these systems, nor all the employees that maintain them.

Ensuring privacy of code and data while executing software on a computer physically owned and maintained by an untrusted party is a challenging problem, as we must assume a powerful adversary. An adversary may not only be able to exploit security vulnerabilities in the cloud, including its operating system (OS) and middleware, but some (e.g., malicious employees) may have super user privileges to them. Furthermore, a malicious insider may even have physical access to the data-centers, making them vulnerable to physical attacks, such as probing the memory bus [2, 68] or cold reboots [34].

A common solution is to reduce the attack surface by minimizing the trusted computing base (TCB) to a secure processor [9, 31, 64] and a small portion of the client's application. Intel Software Guard Extensions (SGX) [47, 16] provides hardware primitives for this purpose. An SGX-enabled secure processor seeks to isolate code and data of private *enclave* functions in an application from the rest of the system, including its own public functions, system software, and hardware peripherals.

A significant challenge in secure processors such as SGX [47, 16] is in providing defenses against memory bus side channel [22]. While the secure processor is trusted, the memory bus and the memory are not. A secure processor guarantees *confidentiality* by encrypting data before sending it to memory. In addition, by storing hash message authentication code (HMAC) and version number along with data in memory, a secure processor checks *data integrity* and freshness on reads.

Memory address, however, is sent in plain-text on the bus, as required by the DRAM's DDR interface. Just by observing the memory addresses, an adversary can infer an execution's control flow, and thereby infer sensitive program inputs [70, 76] and cryptographic keys [76]. Also, the time and number of memory requests/responses can leak sensitive information [27].

Memory address side channel attack can be mitigated using Oblivious RAM (ORAM) [30]. ORAM is a cryptographic construction that obfuscates memory accesses to make them indistinguishable from a random access pattern. A secure processor implements ORAM by issuing several memory accesses for every normal access. In spite of significant recent advancements [52, 29, 44, 74], even with significant custom hardware support [29], an ORAM-enabled secure processor can increase memory access latency by over 20X, which can result in a performance overhead of about 4X. ORAM also has security limitations in that it does not help guarantee data integrity or guard against memory timing channel [42].

In this paper, we present InvisiMem, a low overhead secure processor that provides ORAM equivalent guarantees for the address side channel, ensures data integrity and mitigates memory timing channel. InvisiMem is based on our observation that smart memories (memories with compute capability) with packetized interface (as opposed to DDR interface) can be taken advantage of to design an ultra-low overhead secure processor. Recent advancements in 3D integration technology such as the Hybrid Memory Cube (HMC) [11] make it possible to stack DRAM layers on top of logic layers, and connect them using Through Silicon Via (TSV). Unlike a memory bus that is exposed to an adversary, the TSVs pass completely through a silicon wafer, and therefore it is almost impossible to probe them without destroying the 3D package. Thus, there is no need for expensive mitigation solutions to protect the communication over the TSVs.

InvisiMem executes the private enclave functions in a SGX-like secure high-performance host processor connected to the smart memory through a conventional memory bus using a packetized interface. The logic in smart memory is included in TCB and used to implement cryptographic functions, while memory layers remain outside TCB. This trusted computational capability in memory allows the host processor to send the whole request packet, including the address, in an encrypted form. We observe, however, that encrypting addresses alone is not sufficient to provide ORAM guarantees. First, the packets used for reading and writing data should appear indistinguishable to an adversary snooping the memory bus, because otherwise the type of a memory access would be leaked. Second, on a read to a location, if the memory simply returns previously stored encrypted data at that location, an adversary can relate it to the last write or previous reads to this location. Solutions to address these problems are discussed.

Smart memory can also help guarantee freshness guarantee efficiently. Freshness requires that an adversary should not be able to rollback the state of a memory block by recording and replaying older packets. To defeat such replay attacks, a conventional secure processor maintains current versions of memory blocks on-chip and verifies that a read response returns the latest version [71, 29]. Compute capability in memory allows InvisiMem to use authenticated encryption and establish a secure channel of communication between the secure host processor and memory. This solution obviates the need for maintaining version numbers in the secure host processor.

Finally, smart memory also enables an efficient solution for solving one type of memory timing channel: memory access times seen on the memory bus. InvisiMem sends a constant rate heart-beat packets between the secure host processor and smart memory in both directions. When there is an actual request or response packet to be sent, the next available heart-beat packet's slot is utilized. Unlike the DDR interface, compute capability in smart memory allows responses to be sent at a constant rate. It helps hide variations in access times to different memory locations. Also, it allows dummy requests to be ignored, saving energy. Our solution to this type of memory timing channel also allows us to support a system with multiple memory modules, as it allows us to hide module access patterns and in turn any useful information about address access pattern.

Encrypting and decrypting packets constitute the majority of the performance overhead in InvisiMem. Specifically, computing OTPs (one-time pads) using AES incur the highest latency. We take these operations out of the critical path of a memory access by *precomputing OTPs* before a request/response is sent or received. We also investigate various design options for efficiently storing and retrieving meta-data (timestamps for encrypting memory blocks and authentication tags for integrity check) in HMC. Our design is optimized to meet the constraints and exploit new opportunities (e.g., vault-level parallelism) in HMC.

The logic layer in 3D smart memory has sufficient area and thermal power budget (nearly 55W [25]) to include even a low-power processor core. Executing private enclave functions in this core can hide all the communication between the core and memory, and thereby further reduce the overhead of a secure processor. The trade-off, however, is that its compute capabilities may not match of a high-performance host processor. We study this using a variant of our design, named InvisiMem_near.

We show that InvisiMem_far incurs about 20.74% performance overhead, 35.49% energy overhead and 37.5% memory space overhead compared to a processor similar to Intel Xeon. This is a significant improvement compared to ORAM based solutions that incur one to two orders of magnitude performance, memory bandwidth, and space overhead [29, 42, 52, 44, 74]. The trade-off, however, is that we assume smart memory, and expand the trusted computing base to include the logic layer in smart memory. Using the public key of the smart memory (established using standard public key infrastructure (PKI) similar to conventional secure processors), a secure host processor can easily establish secure and authenticated communication channel with the smart memory's logic to bring it into the TCB. Given that smart memories are becoming increasingly common in consumer products [12, 10, 17], we believe it is a practically viable design option to build trusted clouds.

2. Motivation and Background

In this section we briefly describe hardware support for secure containers (enclaves), which we assume in our work. We also present a threat model and discuss prior ORAM-based defenses. Finally, we provide a brief background on 3D stacked smart memory, which we use in our system.

2.1. Enclaves for Isolation

Intel SGX [22, 16, 47] is the latest hardware support for building trusted computing systems. It provides capability for isolating the execution of a private enclave function from the rest of the system, including the public functions of the application, system software, and other hardware peripherals.

An enclave is a secure container that contains private data and the code that operates on it. An application is responsible for specifying enclaves and invoking them. When an enclave is invoked through special CPU instructions, the untrusted system software loads the enclave contents to the portion of the protected memory allocated for the enclave's execution. The secure processor computes the enclave's measurement hash over initial data and code, which the remote client uses for software attestation. Thereafter, the enclave is executed in a protected mode, where the hardware checks ensure that every memory access to protected memory is from its enclave.

Channel	Leak/Vulnerability	Freecursive ORAM [29]	Ghostrider [42]	InvisiMem_far	InvisiMem_near
Passive Memory Bus Probe	Data	Data encryption	Data encryption	Data encryption	Eliminate memory bus channel
	Address	ORAM	ORAM	Whole packet encryption + Double data and timestamp encryption	
	Access type (R/W)	ORAM	ORAM	Same packet size for read and write	
	Trace length	with [27], yes	Deterministic execution	constant rate requests/responses	
	Access time	with [27], yes	Deterministic execution	constant rate requests/responses	
Active Memory Bus Probe	Data	Data encryption	Data encryption	Enclave checks + Authenticated Encryption (HMAC)	Enclave checks + Authenticated Encryption (HMAC)
	Data corruption	HMAC	no		
	Replay attack	HMAC + Merkel tree	no		
	Write set	ORAM	ORAM		
Cold Boot	Data	Data encryption	Data encryption	Data encryption	Data encryption
System software	Execution time	no	Deterministic execution	no	no

Table 1: Comparison of InvisiMem to ORAM-based defenses. Smart memory enables more efficient and simpler solutions.

2.2. Threat Model

We assume a secure processor with support for isolated execution of private enclave functions (such as Intel SGX [16, 47]). We assume the logic layer in smart memory is secure, and also that the adversary cannot observe the communication between the layers in smart memory. Using this trusted base, we seek to ensure integrity, confidentiality and freshness of private enclave functions and its data.

We assume a powerful adversary that can compromise the operating system and use OS privileges to compromise the confidentiality and integrity of user applications. This adversary also has physical access to the computers running client computations. Thus, the adversary can probe the off-chip memory bus to observe and modify the communication between the secure processor and the memory, including the event times. We assume that DRAM die is untrusted, as the adversary may have the capability to scan DRAM contents through cold (re)boot attacks [34] or corrupt state using Row-Hammer attacks [38].

We assume that the execution of a private enclave function and its data in the processor (registers, caches, on-chip interconnect, performance counters) is secure and isolated from other computation. Several prior studies have discussed solutions for ensuring this property in a multi-core processor with shared hardware structures [23, 40, 43, 67, 58, 66, 19]. We also assume prior solutions for mitigating page-fault side-channel [23, 61] in enclaved systems. Power [39, 13], thermal [51], program execution time [73] side-channels, and information leaks via communication patterns over the network [57, 48], have been addressed in prior work and are outside the scope of this paper.

2.3. Memory Bus Side Channel and Cold Boot Defenses

Table 1 compares various leaks through memory bus side channel and cold boot attack that secure processors must protect. It describes the solutions used in two of the most recent ORAM-based work, Freecursive ORAM [29] and Ghostrider [42].

In most trusted computing systems such as SGX, all the hardware components outside the secure processor are untrusted, including the memory and the memory bus. To ensure confidentiality, they use randomized encryption to encrypt the data before writing to memory, and decrypt it when it is read back. This protects sensitive data from leaking directly through active and passive memory bus probe and cold boot attacks. However, an adversary can still observe addresses and access types (read or write) by passively probing the bus.

To protect confidentiality of addresses (also, access types and write sets), prior solutions employ expensive ORAM [30] solutions. To obfuscate the address pattern, depending on the memory size, an ORAM access may require one to two orders of more memory accesses compared to a normal DRAM access. Recent hardware innovations such as Freecursive ORAM have made significant improvements to bring down the performance cost to about 4x [29], though at a significant increase in hardware complexity, on-chip (84KB [29] to 512KB [42]) and off-chip (more than 2x) space overhead.

ORAM does not protect memory access time and total number of memory accesses from leaking. To protect this information from leaking, Ghostrider [42] guaranteed a stronger property called memory-trace obliviousness (MTO). To guarantee MTO, it ensures that the number and type of instructions executed, and their execution time, are independent of all sensitive inputs to a program. This requires a deterministic compiler and hardware solution that prohibits almost all commonly used optimizations (caches, instruction re-ordering, speculation, etc.). Also, the input program needs to obey non-trivial constrains (e.g., loop guards should be independent of sensitive input). That said, Ghostrider [42] provides a strong MTO guarantee that prevents even program execution time from leaking through memory bus.

Ensuring data integrity requires additional support. An adversary can overwrite data in memory through active memory probes and violate data integrity. A replay attack is also possible, where an adversary manages to rollback the state of a memory block by replaying an older write message. To provide data integrity, secure processors typically create and store hash message authentication code (HMAC) along with data in memory. On a read, HMAC can be used to detect data integrity, but cannot cannot prevent data corruption. Replay attacks can be prevented by including a version number (e.g., timestamp) during the HMAC tag creation. The processor must securely track the current version of the memory state using on-chip storage [71, 29], and use it to check if a read is returning the latest version. But these data integrity checks incur additional

performance and space overhead, and complexity.

By employing 3D smart memory, and increasing the trusted computing base to include its logic layer, we can reduce the complexity of these problems, and realize low overhead security solutions. We seek to guarantee memory-trace obliviousness (MTO) property, except not protecting program execution time from leaking.

2.4. Smart Memory

3D integration has led to the rise of 3D-DRAM devices such as the Hybrid Memory Cube (HMC) [11]. A typical 3D-DRAM consists of several layers of DRAM dies stacked on top of each other, with a logic layer at the bottom, all internally connected using Through Silicon Vias (TSVs) [11]. The layers are partitioned vertically into vaults. Each vault consists of several DRAM banks. Vaults can be accessed in parallel. HMC device is connected to the host processor through a conventional memory bus (SERDES link). Unlike traditional DRAM's DDR interface with low-level commands, HMC device is exposed through a more flexible packet interface.

Recent HMC device has a capacity of 4GB and can provide maximum memory bandwidth of 160GB/s [1]. While the logic in current devices contain only essential circuits for interfacing with the vaults (memory controllers), it has sufficient area and thermal power budget of 55W [25] to include fairly sophisticated computational units, such as a low-power processor and/or cryptographic units.

In 2.5D stacking, the memory and the processor can be efficiently interconnected through metal layers within a silicon interposer [24]. These metal tracks are etched using the same processes as the tracks on the silicon chips, and as a result they are orders of magnitude smaller than the tracks on a conventional memory bus. It is therefore reasonable to assume that an adversary will be unable to tap the communication between the processor and memory in 2.5D system, providing similar security properties as 3D. Also, since logic is not stacked at the bottom of the memory layers, the thermal power budget would allow it to support a high-performance core.

3. InvisiMem Design

InvisiMem builds upon enclave support similar to Intel SGX or Sanctum [23] for isolation. Since DRAM memory layers are untrusted due to possibility of cold-boot attacks (Section 2.2), InvisiMem stores encrypted data in memory using randomized encryption, similar to SGX.

We first discuss InvisiMem_far, which executes the enclave functions within a secure high performance host processor (Figure 1). Later, we discuss a more optimized variant of this design, InvisiMem_near, which executes entire enclave functions within smart memory's logic.

We start by discussing the advantages of smart memory, and then explain how it helps design low-overhead defenses against address and memory bus timing side channels, and

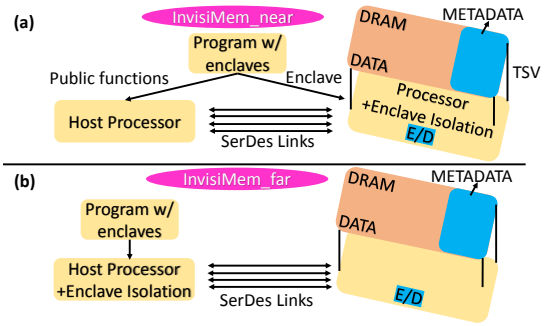

Figure 1: a) InvisiMem_near b) InvisiMem_far

guarantee freshness. We also discuss performance optimization using OTP pre-computation, and storing meta-data efficiently in 3D memory.

3.1. Advantages of Smart Memory

Compute capability in memory allows whole packets to be encrypted and decrypted. Also, it makes it possible to generate dummy responses, and discard dummy requests.

In traditional DRAM systems, on-chip memory controller issues low-level DDR standard compliant commands to interact with the off-chip DRAM modules. In contrast, a smart memory has a packetized interface. The logic layer in smart memory decodes command packets from processor and internally routes them to the memory controller associated with every vault in 3D. The memory controller then communicates with the DRAM memory in its vault through DDR commands. Smart memory's packetized design allows us to seamlessly extend its packet processing logic with security functionality, without modifying the DDR standard, which is harder.

Unlike a memory bus, the TSVs that connect the logic layer and the DRAM memory pass entirely through silicon. It is almost impossible for an adversary to launch a physical attack by probing the TSVs without destroying the 3D package.

3.2. Protecting Memory Address and Type

In InvisiMem, secure processor encrypts and sends the whole packet, including data, address, access type (read or write) using randomized counter-mode encryption. This is possible only because smart memory is capable of decrypting addresses. Randomized encryption makes it hard for an adversary to correlate messages that carry the same address. However, encrypting address alone is not enough to ensure ORAM properties.

First, an adversary can correlate a read to a location with an earlier write to the same location by simply comparing the encrypted data (or timestamp used to encrypt it). To solve this problem, while responding to a read request, the smart memory double encrypts an already encrypted data and its timestamp stored in memory, before sending a response.

Second, the communication between the processor and the memory in an insecure design are noticebly different for reads and writes. A read request does not carry data, while a write request does. Similarly, while the read response carries data, a

write response does not. Thus, an adversary could easily infer whether an access is a read or a write. We eliminate this leak by ensuring that both read and write request/response packets are of the same size. This requires adding a dummy block to read request and write response packets.

These solutions are sufficient to provide guarantees equivalent to ORAM. However, they are not sufficient to prevent the number of memory accesses and their access times from leaking (ORAM leaks these too). Furthermore, response times may vary depending on the memory location accessed. We address these timing channel problems in Section 3.4.

3.3. Guaranteeing Data Integrity and Freshness

Our threat model assumes that DRAM layers are untrusted and therefore stored data can be corrupted (e.g. Row-hammer attacks [38]). An adversary may also corrupt data communication on the bus through active probing. Creating and storing a hash message authentication code (HMAC) with data on a write, and checking the code on a read can solve these issues.

Guaranteeing freshness, however, requires more extensive support in conventional hardware. In replay attack, an adversary manages to rollback the state of a memory block by replaying an older data. Replay attacks can be prevented by including a version number during the HMAC tag creation. The processor must securely track the current version of the memory state [71, 29], and use it to check if a read is returning the latest version. But these data integrity checks incur additional performance and space overhead, and complexity.

InvisiMem_far uses authenticated encryption [46] to guarantee freshness. The sender (processor or smart memory) generates and sends an authentication tag using GCM [46] over the encrypted packet sent to the receiver. This tag is used by the receiver to check if the packet is from the trusted sender, which ensures data integrity over the untrusted memory bus.

Since the encrypted data is generated from an one-time pad, every message needs to be unique, otherwise the authentication will not succeed. This prevents an adversary from replaying an older message, guaranteeing freshness. Unlike prior designs [71, 29], this protocol avoids significant hardware state needed to to track and check the versions of memory blocks in the secure processor.

The secure logic in memory performs the integrity checks only for accesses to protected memory range reserved for enclaves. It relies on the secure host processor to perform the necessary enclave checks to ensure that the accesses to enclave locations are from the enclave function that owns them.

3.4. Mitigating Memory Bus Timing Channel

Memory access times observed on the memory bus can leak sensitive information about paths taken in a program's execution [27]. Memory response time to a request from the secure processor can also reveal access patterns. For example, reads with row-buffer locality will have significantly lower response latency than reads to different rows.

To solve both these leaks, the host processor and the memory send packets at a constant rate in both directions. In the absence of a real packet, a dummy packet is sent, which is then ignored by the receiver. When there is a real packet to send, the sender transmits it at the next available slot. This design trivially eliminates the two leaks noted above. Smart memory's capability to generate and discard dummy packets makes this design feasible. In a conventional DRAM based system, since only the processor has the ability to send requests at a chosen rate, it does not help hide the leaks due to variations in response times.

We also experimented with a dynamic scheme that adjusted the packet rate according to application's memory access characteristics [27], but we did not find any significant performance or energy benefit in the context of a smart memory based design (Section 5.4). One reason is that SerDes links in packetized interface consume non-trivial idle energy as they send null packets at a constant rate for synchronization [11]. Therefore, choosing a lower rate for a compute intensive programs saves little link energy. Also, the total power to trasmit packets and encrypting/decrypting them does not constitute a significant enough fraction of the total system power, even while operating at a packet rate that is high enough for the most memory intensive programs we studied.

Given that a dynamic scheme's security guarantee is also weaker than a static rate, we chose the latter. Instead of a constant packet rate for applications, using profiling, we could select a rate for each application, without sacrificing security properties. Though we did not find a significant benefit for this approach, it may be useful for very memory intensive applications that need a higher packet rate than what we chose.

3.5. Multiple Memory Modules

While we consider a commodity system with a secure host processor and smart memory, researchers have considered more powerful systems with multiple 3D memory modules connected in a network [37]. A secure processor can use the solutions described in this paper to setup secure channel with each of the memory modules. An adversary can gain some information about an address accessed simply by observing which module gets accessed. Fortunately, our timing channel solution (Section 3.4) is adequate to address this problem.

3.6. Performance: OTP Pre-computation

One-time pad (OTP) generation, which uses an AES encryption, is the most time consuming portion of GCM [46] that we use for authenticated encryption. We take it off the critical path of a request or a response by pre-computing it.

An OTP is generated from a timestamp counter and a private key. A timestamp counter's state is shared between the processor and the smart memory. We enable sharing by initializing the respective timestamp counters in both processor and memory at the start of a program's execution to the same value. Thereafter, the sender and the receiver synchronously

increment the counter on sending or receiving a packet. This solution is feasible as the communication network is point-to-point between the processor and memory, and is lossless.

Thus, the sender or the receiver can pre-compute an OTP even before a packet is ready to be encrypted or decrypted. The only case where this is not possible is when decrypting a read response as the timestamp stored with the data must be first recovered to determine the OTP and then decrypt the data using it. See Section 4.3 and Figure 3 for more details.

3.7. Space: Meta-Data in 3D Memory

We consider two design alternatives for storing meta-data (timestamp and tag) with their data. In the fragmented design, data of a cache block is stored along with its meta-data in memory. This design has relatively lower complexity as the memory controller can fetch both data and its meta-data using a single request. However, storing meta-data with data consumes two cache blocks worth of space, even though meta-data is smaller than a cache data block.

In the non-fragmented design, we store data and meta-data at non-contiguous locations. This allows meta-data of multiple data blocks to be compactly packed together with less wastage of memory. However, this requires two requests per data access. We reduce any potential performance overhead due to the serialization of these requests by exploiting vault-level parallelism (Section 2.4). To achieve this, we map addresses to physical locations such that data and its meta-data are always stored in different vaults. Furthermore, as adjacent data blocks may be accessed in close succession, our mapping ensures that data and meta-data of spatially adjacent data blocks in the address space are stored in different vaults. See Section 4.5 for details. With these performance optimizations, non-fragmented design incurs only a negligible performance overhead compared to the fragmented design, but has significantly better space utilization (91.66% compared to 68.75%).

3.8. Near InvisiMem

As noted in Section 2.4, the logic layer integrated with memory could support low-power (3D) or even high-performance cores (2.5D). InvisiMem_near exploits this opportunity by assuming that the secure host processor is within smart memory's logic layer. Since the TSVs are secure, it obviates the need for protecting communication between the core and memory. However, we conservatively exclude the DRAM memory layers from TCB (Section 2.2). Therefore, data is still stored in encrypted form in memory, and its integrity is checked by storing HMAC tags with data and checking them on read.

The memory controller in the secure processor bounces any access from an external device, including the host processor, by checking if it falls within the range of protected memory region dedicated for enclaves. Apart from these measures and support for enclave checks, all of which are already supported in commodity secure processors such as Intel SGX, InvisiMem_near requires little else support to provide the guarantees we seek. Note that this design does not need additional support to guarantee freshness or timing channel.

4. Implementation

This section describes hardware support for cryptographic primitives, and details how OTP pre-computation helps reduce the latency of encryption/decryption in a read/write operation, and how meta-data is stored in 3D memory efficiently.

4.1. Hardware Support for Cryptographic Primitives

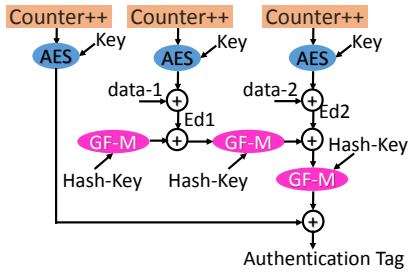

Figure 2: Authenticated Encryption using Galois Field Multiplication (GF-M).

4.1.1. Authenticated Encryption We use Galois/Counter operation mode (GCM) [46] with AES for authenticated encryption (Figure 2). GCM operates on 128-bit blocks. Therefore, a single cache block (64 bytes) is broken into 4 blocks of plain-text. One Time Pads (OTP) is generated by using AES encryption on a counter along with a 128-bit encryption key. OTP is then XORed with a plain-text to generate its cipher-text. The counter used to generated OTP is incremented for every block that is processed to provide randomized encryption [53]. For authentication, GCM employs a GHASH function [46], which creates hash of a message ciphertext using a secret 128-bit hash key (H) derived from the encryption key. The output is an authentication tag, which is regenerated at the receiver to verify data integrity.

4.1.2. Metadata: Timestamps and Keys InvisiMem_far uses three symmetric keys: address key (*Ka*), data key (*Kd*), and data double encryption key (*Kd-de*). The data double encryption key is used to double encrypt encrypted data and timestamp to defend against correlation attack (Section 3.2).

We use three timestamps. 128-bit address timestamp (ATS) is used for generating address OTP. Address key (*Ka*) and ATS are used to encrypt packet header and tail, which includes the address, command, etc. For brevity, we simply refer to these in terms of encrypting/decrypting addresses in this section.

Smaller 64-bit data timestamp (DTS1) is used for encrypting 64-byte cache block data as follows. The cache block is broken into four 128-bit chunks. Timestamp for each chunk is produced by concatenating 64-bit timestamp (DTS1) with a 62-bit fixed vector (FV) and a two bit chunk-id representing the position of the chunk in its block. Since the timestamp for a cache block has to be stored along with data in mem-

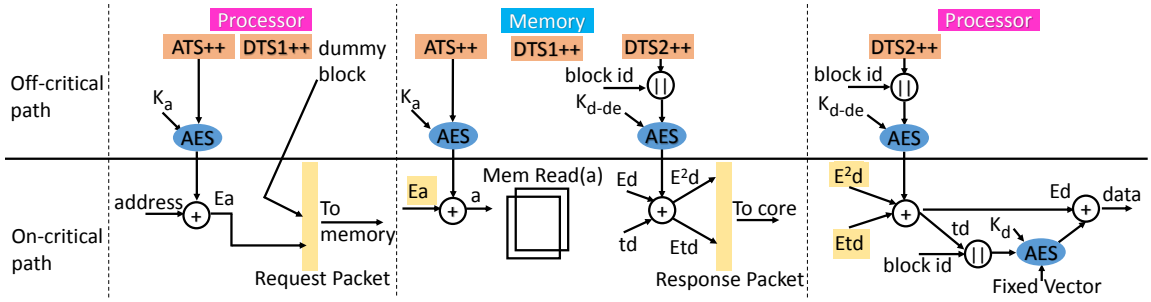

Figure 3: InvisiMem_far Security Protocol for Read. `td`**: timestamp stored with data.**

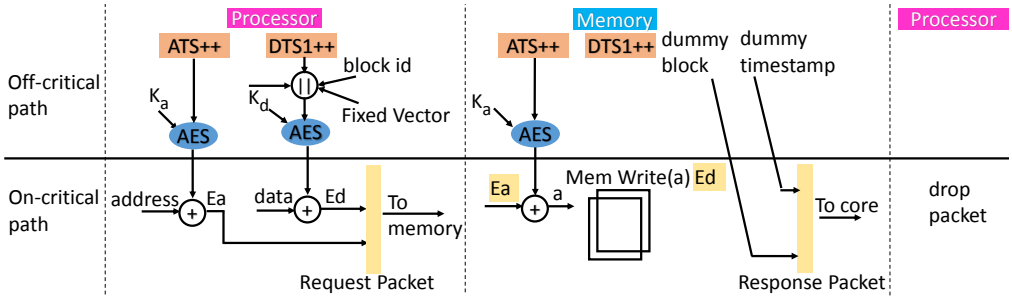

Figure 4: InvisiMem_far Security Protocol for Write.

ory, using a smaller 64-bit timestamp helps save space. For double-encryption of data and its timestamp, we use a 125-bit timestamp DTS2 concatenated with 3-bit chunk-id while generating the OTP. This timestamp is never stored in memory.

4.1.3. Augmented memory controller Smart memory has memory controller/s (MMC) in the logic layer which communicate with the integrated memory controller (PMC) in the processor. We augment both PMC and MMC to perform authenticated encryption (Figure 5 (a)). This requires registers for timestamps and keys mentioned in Section 4.1.2. PMC and MMC have three AES and XOR units each. For authentication, both MMC and PMC have four Galois Field multipliers (GF-M) [56] each.

4.2. Secure Initialization and Key Exchange

Client uses remote attestation to ensure she is communicating with a trusted hardware hosting trusted software, before transferring encrypted data and code. Attestation typically involves authentication of the host processor by the remote client, which relies on the manufacturer provisioning and certifying each secure processor with a unique public-private key pair (Public Key Infrastructure or PKI). Remote attestation remains unchanged under InvisiMem. Both standard secure attestation implementations [8] and specific implementations (Intel SGX) exist and can be used in our system. The remote user ensures that the remote host secure processor is trustworthy and ships encrypted code and data to it. The trusted host processor then locally sets up a secure and authenticated communication channel with the smart memory.

Setting up such a secure communication channel with memory requires secure sharing of keys and timestamps described in Section 4.1.2. We assume that the smart memory module establishes a certified private-public key combination as is the case currently with secure processors. At the program start, the processor uses the memory's public key to encrypt and share keys and timestamps with it. This requires support for asymmetric encryption on both processor and memory side. The protected memory is also wiped out before a reboot.

4.3. InvisiMem_far Security Protocol

Figures 3 and 4 depict the steps involved in InvisiMem_far on a read and write respectively. We classify all the actions into either "off-critical path" or "on-critical path" of a read or a write access. For simplicity, we only depict the encryption part of GCM and not authentication tag generation which can be partly overlapped with encryption/decryption. We also ignore our timing channel solution, which simply requires that once a packet is ready it is sent at the next available slot.

In Figure 3, PMC encrypts an address for read request. Using ATS, we can pre-compute the OTP required for address encryption, leaving only an XOR operation on the critical path. Request packet for a read includes the encrypted address, incremented DTS1 and dummy encrypted data block. The latter two pieces added to request packet are needed to make it impossible for an attacker to differentiate a read request from a write request. On receipt of a request, MMC decrypts the address; again with a pre-computed OTP and issues a read to DRAM. On receiving a response from DRAM, MMC encrypts data and its associated timestamp using pre-computed OTP generated from DTS2 (double encryption) and sends it to PMC. Double encryption is done to guard against correlation attack by observing encrypted data or timestamp. On receipt of response, PMC first decrypts data and timestamp using OTP pre-computed from DTS2. Then uses the decrypted timestamp

to again decrypt the data, which is the expensive step in our protocol.

For a write (Figure 4), PMC encrypts data and address using pre-computed OTPs. MMC decrypts address using pre-computed OTPs. Thus, only XOR operations are on the critical path.

For authentication, a Galois Field multiplication and an XOR operation are also needed per ciphertext (Section 4.1.3). Read/write requests/responses require address and either data or dummy data tag generation on both PMC and MMC side. Dummy data tag generation can be avoided on receiving side. For a read response, MMC first checks the tag read from DRAM and generates another tag on double encrypted data which is transmitted. We can overlap some of these operations with data (and address) encryption/decryption at both PMC and MMC.

4.4. InvisiMem_near Security Protocol

The protocol for near-memory secure processor involves data encryption and authentication tag generation on a write. A read requires data decryption, and authentication tag generation and check. While write request will add authentication delay to critical path, for a read response it can be overlapped with data decryption.

4.5. Storing Meta-Data in 3D Stacked Memory

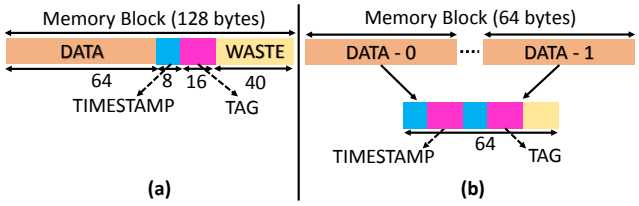

Figure 5: (a) Fragmented Design (b) Non-fragmented Design

Meta-data for a cache block consists of 64-bit timestamp value (DTS1) and an integrity tag. Section 3.7 described two designs for storing this meta-data: fragmented and non-fragmented.

Figure 5 (a) depicts fragmented layout. Storing data and its metadata together requires 88 bytes (64 data, 8 timestamp, 16 tag). HMC Specification [11] mandates that memory block sizes can be 32/64/128/256 bytes. Therefore, in the fragmented layout, 88-byte data block and its meta-data consumes 128-bytes, resulting in effective memory utilization of 68.75%.

Figure 5(b) depicts non-fragmented layout, where meta-data and data are not stored together. A 64 byte block can store meta-data for two data blocks (2 timestamps and 2 tags). This leads to a far better memory utilization of 91.66%. To exploit vault-level parallelism, our data mapper places data block and its meta-data in different vaults, so that they can be accessed in parallel. We also take care that meta-data of spatially adjacent data blocks are mapped to different meta-data blocks. Memory waits for both data and metadata before responding to a request from PMC.

While non-fragmented layout saves space, both designs transfer same amount of data (128 bytes) over the TSVs to service a memory request.

Configuration	4 core CMP, commit width 4, 72 entry LQ, 42 entry SQ
Processor	Near Memory: 2.5 GHz out-of-order core Far Memory: 4 GHz out-of-order core
L1-I/D Cache	32KB, 8-way, 4 cycle access
L2 Cache	inclusive, private, 256KB, 8-way,11 cycle access
L3 Cache	inclusive, shared, 8MB, 16-way, 40 cycles
Interconnect	Split Bus, 6 cycles, arbitrate latency: 1 cycle
DRAM	4GB, 2 channels, tCL = tRCD = tRP = 13.75ns, tCk=1.25ns
3D Memory	4GB, 32 vaults [11], 128 TSV's per vault @2Gb/s [36]
Off-chip links	4 full-duplex SerDes links, 16 lanes per link per direction

Table 2: Processor and Memory Model.

Benchmark	LLC_MPKI	IPC	Benchmark	LLC_MPKI	IPC
povray	0.06	0.94	perlbench (perl)	1.42	1.20
gamess	0.1	1.33	gcc	1.49	0.65
namd	0.13	1.13	cactusADM (cactus)	3.58	0.71
hmmer	0.26	1.08	zeusmp	4.02	0.84
calculix	0.31	1.17	bwaves	10.32	0.69
gobmk	0.34	0.93	leslie3d (leslie)	17.53	0.38
h264ref	0.43	1.21	GemsFDTD (Gems)	20.25	0.27
gromacs	0.46	0.76	milc	20.58	0.45
sjeng	0.47	0.86	soplex	25.93	0.27
tonto	0.54	1.01	libquantum (libq)	33.06	0.32
bzip2	0.55	0.75	mcf	40.67	0.15

Table 3: Benchmarks with LLC_MPKI and IPC for DRAM_hp.

5. Evaluation

5.1. Methodology

We study 22 benchmarks from SPEC 2006 [35] suite with reference inputs. We use the Simpoint [59] methodology with interval size of 100 million instructions to choose representative execution samples. Table 3 reports the LLC misses per kilo instructions (MPKI) and IPC values for DRAM_hp.

Processor Model: We modeled our processor designs (Table 2) using MARSSx86 [49], a full system cycle accurate simulator. Processor in InvisiMem_far is similar to Intel Quad Core i7-4790K Processor [4]. Invisimem_near places secure processor in the logic die of smart memory. Eckert et al. [25] investigated power dissipation possible in the logic layer of smart memory under various cooling solutions. They conclude that with an active heat sink, the power dissipated can be as high as 55W without affecting memory die temperatures adversely. Hence, we model Invisimem_near as Intel i7-3770T [3] at 2.5GHz and 45W.

Latency of Cryptographic Primitives: We synthesized a pipelined AES core from OpenCores [5] at 45nm and scaled it using ITRS projections to model the AES latency in our system. The Galois Field multiplication used in authenticated encryption (Figure 2) is a purely combinational circuit [56] that operates in single cycle [56, 71].

Power Model: We model processor power using McPAT [41]. For every 128-bit block, we model AES energy to be 302 pJ [45]. For baseline DRAM, we model access

Design	Read-Req	Read-Resp	Write-Req	Write-Resp
Baseline	16	80	80	16
InvisiMem_far	112	120	112	120

Table 4: Request and response packet sizes for InvisiMem_far (in bytes).

energy to be 65 pJ/bit [36]. A recent industry prototype reports 10.48pJ/bit for HMC access of which 43% is attributed to SerDes circuits [36, 50], rest is for access to DRAM and logic layer. We model 0.47W for DRAM static power based on [50].

Smart Memory Model: We use DRAMSIM2 [55] to model 4GB of DRAM memory for baseline (DRAM_hp). We modify DRAMSIM2 to model a 4GB 3D-stacked memory with 32 vaults and 128 TSVs (through silicon vias) per vault [11]. We assume the same DRAM device parameters (Table 2) for both traditional DRAM and 3D-stacked memory. However, we assume a DRAM clock in line with TSV signaling rate for smart memory.

Table 4 shows the request and response packet sizes in baseline and InvisiMem_far. We model these sizes according to HMC specification [11]. Each packet has an 8-byte header and an 8-byte tail, which carry useful meta-data like command, address etc. Thus, for a read request and write response, 16 bytes are communicated between processor and memory in the baseline. For write request and read response, 80 bytes (64 bytes of data, 8 bytes head, 8 bytes tail) are communicated. In our design, read and write requests transmit 112 byte packets (16 bytes for header and tail, 64 bytes for data, 16 bytes for header/tail tag, and 16 bytes for data tag). Read and write response packets send an additional 8 bytes for the timestamp of the data.

5.2. Unsecure Smart Memory Performance and Energy

Figure 6 shows the performance overhead with respect to unsecure baseline DRAM_hp of the various InvisiMem_far designs with increasing security guarantees. We plot the benchmarks in the increasing order of their LLC miss rates. The 3D_far design represents an unsecure high power processor connected to smart memory. Not surprisingly, high bandwidth 3D smart memory helps improve performance of memory intensive programs. `GemsFDTD`, in particular, sees significant gain of 31.41%. On average, replacing traditional DRAM with smart memory delivers an average performance improvement of 4.02%.

Figure 8 shows the energy overhead of 3D_far design (average 10.28%). While the DRAM energy is brought down by smart memory, the static power expended by the SerDes links are the primary cause of this overhead. The SerDes links expend high static power in absence of real packets, as they transmit null packets [11]. Prior work has also observed that SerDes link power is a significant fraction of the total HMC power [14, 50].

5.3. Far InvisiMem

We present in this section, the performance overhead of InvisiMem_far to guarantee security properties equivalent to ORAM, data integrity, freshness, and also avoid leaking memory access times. In Figure 6, the 3D_far+DE configuration shows the overhead of adding data encryption (DE) to the 3D_far design (non-fragmented memory design). We model this design so as to be able to separate out data encryption overhead (incurred in current secure processors such as Intel SGX) from the address encryption overhead. Adding data encryption incurs modest overhead for most benchmarks with an average overhead of 2.58% and highest overhead of 16.69% for `libquantum`.

To tease out the overhead of providing ORAM guarantees from the other guarantees we provide, we model InvisiMem_far (no DI) configuration which provides only ORAM guarantee, but not data integrity or freshness or hide memory access timing. In this design we encrypt both address and data, but authenticate only the address. Adding these guarantees increases the overhead from 2.58% (3D_far+DE) to 5.55%. This negligible overhead is in a sharp contrast to ORAM based designs which incur an order of magnitude higher performance overhead (Section 2). Performance overhead for `libquantum` increases from 16.69% to 22.56%.

The InvisiMem_far configuration depicts the design which has ORAM, data integrity and freshness guarantees, but no defense against the timing channel. InvisiMem_far design, incurs an average overhead of 10.81%; with the highest overhead being for bwaves of 52.65%. This is a significant improvement over prior ORAM based data integrity solutions [29], which also require additional hardware support for tracking and checking version numbers of memory blocks. The InvisiMem_far configuration which does not leak the timing of memory events is depicted as InvisiMem_far+Timing (Section 5.4). This design increases the average overhead from 10.81% to 20.74%.

Figure 8 shows the energy overheads of InvisiMem_far configurations with and without timing channel defense. Without timing channel defense, InvisiMem_far configuration increases the energy overhead of 3D_far design from 10.28% to a mere 17.62%. With timing channel protection the overhead increase to 35.49%. Note that this overhead is a significant improvement over prior schemes which incur one to two orders of magnitude of both performance loss, bandwidth, and commensurate energy overhead.

5.4. Static Packet Rate for Timing Channel

As we discussed in Section 3.4, we choose a static request and response rate in our system to address timing channel leaks. We provide empirical evidence to support this choice. Figure 9 depicts energy delay squared (ED^2) overhead of various static packet rates with respect to Invisimem_far without timing channel protection. We show results for two least and most

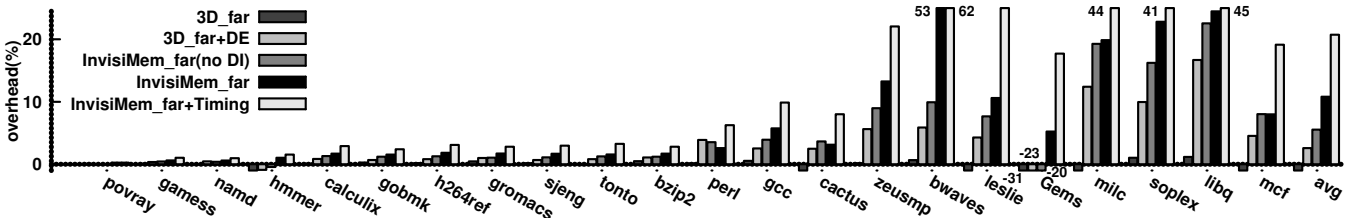

Figure 6: Performance overhead of far-memory processor unsecure and secure designs w.r.t DRAM_hp

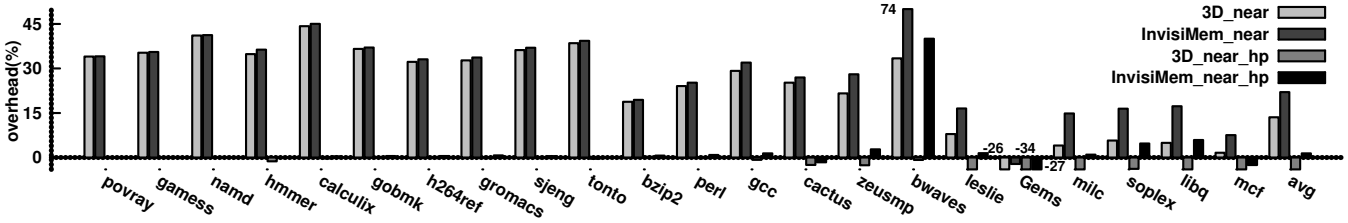

Figure 7: Performance overhead of near-memory designs w.r.t DRAM_hp

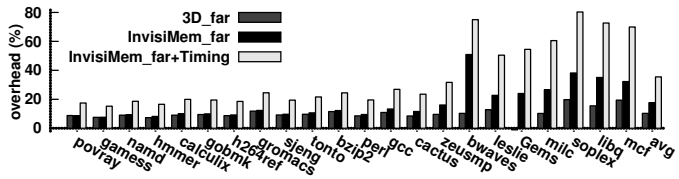

Figure 8: Energy overhead w.r.t DRAM_hp

Guarantee	BS	Encryption	ORAM	Integrity
Freecursive ORAM [29]	64	0.5GB	4.3GB (position+data)	1GB hash
Ghostrider [42]	4096	16MB	4.1GB (position+data)	absent
InvisiMem_far	64	0.5GB	0B	1GB hash

Table 5: Memory overhead of security guarantees (4GB data), BS: block size (bytes)

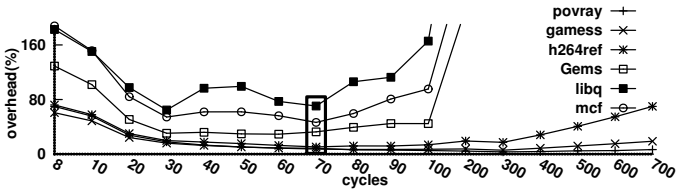

Figure 9: ED^2 overhead w.r.t InvisiMem_far without timing channel defense for various static memory access rates. X-axis represents an interval between two packets.

memory intensive benchmarks, and also two benchmarks one with the highest and another with the lowest IPC from our suite. We see the lowest (ED^2) overhead roughly at 70-cycles for all these diverse programs. The reason is that energy consumed by cryptographic units to process dummy packets stops being a significant fraction of system power as packet interval increases beyond about 30-70 cycles. As SerDes link constantly send null packets even when they are idle, there is not much to be gained by increasing the packet interval beyond this sweet point. Performance overhead starts to degrade significantly at about 100 cycles for memory intensive programs and later at about 400-500 cycles for compute intensive programs.

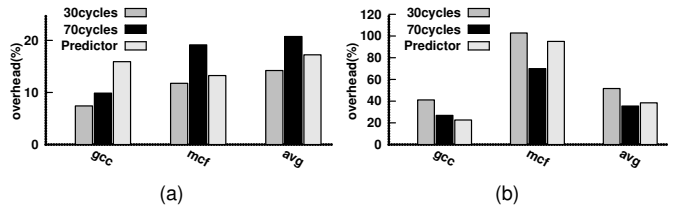

Figure 10: Comparison to dynamic scheme.

We also implemented a dynamic predictor [27] with rates (30, 60, 120, 240). Figure 10 compares this predictor to our static scheme with two packet intervals 30 (ED^2 of 157.20%) and 70 cycles (ED^2 of 154.51%). We show low (`gcc`) and high (`mcf`) LLC_MPKI rate benchmarks, and also average for all the programs. As these results show there is not a significant potential for performance and energy improvements using a dynamic scheme. Considering it has a weaker security guarantee, we chose a static rate.

5.5. Near InvisiMem

Figure 7 shows the performance overhead of two near-memory processor designs (low and high power) we model with and without security guarantees. A low power processor stacked with memory layers is depicted as 3D_near. Compared to high-performance far processor, it has an average overhead of 13.56%. Compute intensive benchmarks exhibit and overhead (44.23% for `calculix`), whereas we see performance gains for memory intensive benchmarks (`GemsFDTD`, 26.43%). InvisiMem_near which encrypts data and authenticates it on reads adds about 11% overhead to 3D_near design. To model the scenario where it may be feasible to connect a high performance core to memory through secure silicon interposer, we depict 3D_near_hp and InvisiMem_near_hp. With such a design, the average overhead of providing data encryption and integrity is a mere 1.41%.

5.6. Memory Space Overhead

Table 5 depicts the space overheads of two recent proposals Freecursive-ORAM [29] and Ghostrider [42] against InvisiMem_far for different security guarantees. Ghostrider, unlike other designs, accesses memory at larger granularity (4KB vs 64B) and trades off metadata overhead for increased memory bandwidth utilization.

InvisiMem_far similar to Freecursive-ORAM incurs 0.5GB memory space overhead for storing timestamps for data encryption. Ghostrider incurs lower overhead due to larger memory block size. InvisiMem_far incurs no memory space overhead to provide ORAM guarantees. Both prior proposals employ path ORAM algorithm [62] to guarantee ORAM. As such, they incur 100% memory space overhead to store dummy data and require megabytes (few to hundreds) of memory to store the randomized position of each memory block. Finally, for data integrity, while Ghostrider does not provide it, both our design and Freecursive-ORAM will incur similar overhead to store hash value per memory block.

5.7. Fragmented Vs Non-Fragmented

Section 4.5 discussed two ways in which a memory block and its metadata (timestamp, tag) can be stored and retrieved from memory. Fragmented design wastes memory space in comparison to non-fragmented design. However, while the former issues single DRAM request, the latter breaks every memory request into requests for data and metadata. Owing to vault level parallelism and our data mapping policy (Section 4.5) we see that the average overhead of InvisiMem_far only increases from 7.24% in fragmented design to 10.81% in non-fragmented design. Given its better memory utilization and negligible performance difference, we chose the memory layout of non-fragmented design for all our experiments.

6. Related Work

InvisiMem is the first work that uses smart memories to provide a low-overhead solution for memory address and timing side channel, data integrity, and freshness.

6.1. 3D Stacking for Security

While prior projects [15, 33] exploit 3D stacking for improving performance and energy, only two prior proposals[[65, 32]] harness 3D-Stacking to provide security guarantees. [65] models a control plane integrated with a conventional processor in 3D stack which provides security functionalities like monitoring activities of the processor to prevent cache-side channel attacks. In [32], the authors leverage logic near 3D memory to efficiently implement Bonsai Merkel Tree [54] for integrity checks. In our work, we harness memory isolation guarantees provided by Intel SGX and smart memory to provide simplified data integrity using authenticated encryption between processor and memory. Both prior works did not observe the benefit of smart memories to address memory bus side channel, freshness, or timing channel.

6.2. Secure Hardware

There have been many work on secure hardware [64, 9, 31, 63, 18]. Starting with execute-only memory (XOM) architecture [64], these proposals chiefly aim to provide isolation (protect sensitive application from other applications and untrusted system software) and software attestation. Intel Software Guard Extensions (SGX) [47, 16] is the latest offering to provide isolated execution, which reduces the amount of software included in software attestation to the application and few privileged containers. It also provides encryption, data integrity and freshness guarantees. Alternate proposals which provide SGX like security guarantees and/or add to it [20, 26, 23] have also been proposed. None of these proposals address memory address bus and timing side channels.

We discussed solutions [42, 52, 28, 44, 74, 76, 75, 27] that provide defenses against memory bus side channel and data integrity in Section 2. These solutions incur order of magnitude more memory accesses and hence result in huge performance overheads. We show in this work that with smart memory; which is capable of performing computation, memory bus side channel can be solved with low overheads.

Prior works have also considered sending memory requests from the processor at a static [28] or dynamic [27] rate. Both rely on ORAM algorithm to generate indistinguishable real and dummy requests. Besides not requiring ORAM, smart memory with packetized interface has several advantages in mitigating timing side channel. First, the memory can generate constant rate responses, which helps hide variations in response times. Second, dummy requests can be discarded in memory, saving energy. Third, unlike prior schemes [27], there is no requirement that there be only one pending memory request at any time.

6.2.1. Optimizing Memory Encryption Prior works propose several optimizations to reduce memory encryption overheads which can be used in our design. In [60], the authors predict encryption counters for speculative OTP pre-computation. We use non-speculative OTP pre-computation by employing synchronized counters at processor and memory. In [71, 72], the authors cache encryption counters which can also further reduce our overheads.

7. Conclusion

We present InvisiMem, which harnesses smart memory with compute capability and packetized interface to simplify solutions for providing data encryption, integrity, freshness and memory bus timing channel. By including logic layer of smart memory in the trusted computing base we demonstrate how each of the above guarantees can be obtained at an order of magnitude less performance, space, energy and memory bandwidth overhead, when compared to current ORAM-based solutions.

References

[1] Hybrid memory cube. "https://www.micron.com/products/hybrid-memory-cube".

[2] I2c bus monitor. "http://www.jupiteri.com/".

[3] Intel® core™ i7-3770t processor (8m cache, up to 3.70 ghz). "http://ark.intel.com/products/65525/Intel-Core-i7-3770T-Processor-8M-Cache-up-to-3_70-GHz".

[4] Intel® core™ i7-4790k processor (8m cache, up to 4.40 ghz). "http://ark.intel.com/products/80807/Intel-Core-i7-4790K-Processor-8M-Cache-up-to-4_40-GHz".

[5] Opencores. "http://opencores.org/".

[6] Photos of an nsa "upgrade" factory show cisco router getting implant. "http://arstechnica.com/tech-policy/2014/05/photos-of-an-nsa-upgrade-factory-show-cisco-router-getting-implant/".

[7] Report: Apple designing its own servers to avoid snooping. "http://arstechnica.com/information-technology/2016/03/report-apple-designing-its-own-servers-to-avoid-snooping/".

[8] Trusted computing group. "http://www.trustedcomputinggroup.org".

[9] Trusted computing group. tpm main specification. "http://www.trustedcomputinggroup.org/resources/tpm_main_specification,2003.".

[10] Nvidia gpu technology conference: Keynote. Technical report, March 2013, 2013.

[11] Hybrid memory cube specification 2.0, 2014.

[12] Advanced micro devices, inc. amd ushers in a new era of pc gaming with radeontm r9 and r7 300 series graphics line-up including world's first graphics family with revolutionary hbm technology. Press Release from http://www.amd.com, 2015.

[13] Onur Aciicmez, Jean-Pierre Seifert, and Cetin Kaya Koc. Micro-architectural cryptanalysis. *IEEE Security and Privacy*.

[14] J. Ahn, S. Yoo, and K. Choi. Dynamic power management of off-chip links for hybrid memory cubes. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2014.

[15] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture.

[16] Ittai Anati, Shay Gueron, Simon P Johnson, and Vincent R Scarlata. Innovative technology for cpu based attestation and sealing. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13.

[17] B. Black. Die stacking is happening. In *In International Symposium on Microarchitecture*, 2013.

[18] D. Champagne and R. B. Lee. Scalable architectural support for trusted software. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*.

[19] J. Chen and G. Venkataramani. Cc-hunter: Uncovering covert timing channels on shared processor hardware. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

[20] Siddhartha Chhabra, Brian Rogers, Yan Solihin, and Milos Prvulovic. Secureme: A hardware-software approach to full system security. In *Proceedings of the International Conference on Supercomputing*, ICS '11.

[21] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. Controlling data in the cloud: Outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, 2009.

[22] Victor Costan and Srinivas Devadas. Intel sgx explained. Cryptology ePrint Archive, Report 2016/086, 2016.

[23] Victor Costan, Ilia Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. Cryptology ePrint Archive, Report 2015/564, 2015.

[24] Yangdong Deng and Wojciech P. Maly. Interconnect characteristics of 2.5-d system integration scheme. In *Proceedings of the 2001 International Symposium on Physical Design*, ISPD '01.

[25] Yasuko Eckert, Nuwan Jayasena, and Gabriel Loh. Thermal feasibility of die-stacked processing in memory. In *Workshop on Near-Data Processing*, 2014.

[26] Dmitry Evtyushkin, Jesse Elwell, Meltem Ozsoy, Dmitry Ponomarev, Nael Abu Ghazaleh, and Ryan Riley. Iso-x: A flexible architecture for hardware-managed isolated execution. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47.

[27] C. W. Fletcher, Ling Ren, Xiangyao Yu, M. van Dijk, O. Khan, and S. Devadas. Suppressing the oblivious ram timing channel while making information leakage and program efficiency trade-offs. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, 2014.

[28] Christopher W. Fletcher, Marten van Dijk, and Srinivas Devadas. A secure processor architecture for encrypted computation on untrusted programs. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing*, STC '12.

[29] Christopher W. Fletcher, Ling Ren, Albert Kwon, Marten van Dijk, and Srinivas Devadas. Freecursive oram: [nearly] free recursion and integrity verification for position-based oblivious ram. ASPLOS '15, 2015.

[30] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*.

[31] David Grawrock. Dynamics of a trusted platform: A building block approach. Intel Press, 2009.

[32] Akhila Gundu, Ali Shafiee Ardestani, Manjunath Shevgoor, and Rajeev Balasubramonian. A case for near data security. In *Workshop on Near-Data Processing*, 2014.

[33] Zvika Guz, Manu Awasthi, Vijay Balakrishnan, Mrinmoy Ghosh, Anahita Shayesteh, and Tameesh Suri. Real-time analytics as the killer application for processing-in-memory. In *Workshop on Near Data Computing*, 2014.

[34] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold-boot attacks on encryption keys.

[35] John L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*.

[36] J. Jeddeloh and B. Keeth. Hybrid memory cube new dram architecture increases density and performance. In *VLSI Technology (VLSIT), 2012 Symposium on*.

[37] Gwangsun Kim, John Kim, Jung Ho Ahn, and Jaeha Kim. Memory-centric system interconnect design with hybrid memory cubes. PACT '13.

[38] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14.

[39] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 2011.

[40] Jingfei Kong, Onur Aciicmez, Jean-Pierre Seifert, and Huiyang Zhou. Deconstructing new cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the 2Nd ACM Workshop on Computer Security Architectures*, CSAW '08, 2008.

[41] Sheng Li, Jung Ho Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, and N.P. Jouppi. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, 2009.

[42] Chang Liu, Austin Harris, Martin Maas, Michael Hicks, Mohit Tiwari, and Elaine Shi. Ghostrider: A hardware-software system for memory trace oblivious computation. ASPLOS '15, 2015.

[43] Fangfei Liu and Ruby B. Lee. Random fill cache architecture. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, 2014.

[44] Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovic, John Kubiatowicz, and Dawn Song. Phantom: Practical oblivious computation in a secure processor. CCS '13, 2013.

[45] S. Mathew, F. Sheikh, A. Agarwal, M. Kounavis, S. Hsu, H. Kaul, M. Anders, and R. Krishnamurthy. 53gbps native gf(24)2 composite-field aes-encrypt/decrypt accelerator for content-protection in 45nm high-performance microprocessors. In *2010 Symposium on VLSI Circuits*, 2010.

[46] David A. McGrew and John Viega. The galois/counter mode of operation (gcm). "http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf".

[47] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13.

[48] Olga Ohrimenko, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Markulf Kohlweiss, and Divya Sharma. Observing and preventing leakage in mapreduce. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, 2015.

[49] Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. MARSSx86: A Full System Simulator for x86 CPUs. In *Design Automation Conference 2011 (DAC'11)*, 2011.

[50] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li. Ndc: Analyzing the impact of 3d-stacked memory+logic devices on mapreduce workloads. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, 2014.

[51] J-J. Quisquater and D. Samyde. Side channel cryptanalysis. In *Workshop on the Security of Communications on the Internet (SECI)*, 2002.

[52] Ling Ren, Xiangyao Yu, Christopher W. Fletcher, Marten van Dijk, and Srinivas Devadas. Design space exploration and optimization of path oblivious ram in secure processors. ISCA '13, 2013.

[53] Ronald L. Rivest and Alan T. Sherman. *Advances in Cryptology: Proceedings of Crypto 82*, chapter Randomized Encryption Techniques. 1983.

[54] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. Using address independent seed encryption and bonsai merkle trees to make secure processors os- and performance-friendly. MICRO 40, 2007.

[55] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. Dramsim2: A cycle accurate memory system simulator. *IEEE Comput. Archit. Lett.*

[56] A. Satoh. High-speed parallel hardware architecture for galois counter mode. In *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, 2007.

[57] Felix Schuster, Manuel Costa, Cedric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. Vc3: Trustworthy data analytics in the cloud. Technical Report MSR-TR-2014-39, February 2014.

[58] Ali Shafiee, Akhila Gundu, Manjunath Shevgoor, Rajeev Balasubramonian, and Mohit Tiwari. Avoiding information leakage in the memory controller with fixed service policies. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, 2015.

[59] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, 2002.

[60] Weidong Shi, Hsien-Hsin S. Lee, Mrinmoy Ghosh, Chenghuai Lu, and Alexandra Boldyreva. High efficiency counter mode security architecture via prediction and precomputation. In *Proceedings of the 32Nd Annual International Symposium on Computer Architecture*, ISCA '05.

[61] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. Preventing your faults from telling your secrets: Defenses against pigeonhole attacks. *CoRR*, 2015.

[62] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: An extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13.

[63] G. Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. Aegis: Architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the 17th Annual International Conference on Supercomputing*, ICS '03.

[64] David Lie Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. Architectural support for copy and tamper resistant software. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS IX, 2000.

[65] Jonathan Valamehr, Mohit Tiwari, Timothy Sherwood, Ryan Kastner, Ted Huffmire, Cynthia Irvine, and Timothy Levin. Hardware assistance for trustworthy systems through 3-d integration. ACSAC '10, 2010.

[66] Y. Wang, A. Ferraiuolo, and G. E. Suh. Timing channel protection for a shared memory controller. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014.

[67] Zhenghong Wang and Ruby B. Lee. New cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, 2007.

[68] L. Whetsel. An ieee 1149.1 based logic/signature analyzer in a chip. In *Test Conference, 1991, Proceedings., International*, Oct 1991.

[69] L. Whitney. Microsoft urges laws to boost trust in the cloud. "http://news.cnet.com/8301-1009_3-10437844-83.html.".

[70] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *In Proceedings of the 36th IEEE Symposium on Security and Privacy (Oakland)*, 2015.

[71] Chenyu Yan, Daniel Englender, Milos Prvulovic, Brian Rogers, and Yan Solihin. Improving cost, performance, and security of memory encryption and authentication. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*.

[72] Jun Yang, Youtao Zhang, and Lan Gao. Fast secure processor for inhibiting software piracy and tampering. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, 2003.

[73] Danfeng Zhang, Aslan Askarov, and Andrew C. Myers. Predictive mitigation of timing channels in interactive systems. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, 2011.

[74] Xian Zhang, Guangyu Sun, Chao Zhang, Weiqi Zhang, Yun Liang, Tao Wang, Yiran Chen, and Jia Di. Fork path: Improving efficiency of oram by removing redundant memory accesses. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48.

[75] Xiaotong Zhuang, Tao Zhang, Hsien-Hsin S. Lee, and Santosh Pande. Hardware assisted control flow obfuscation for embedded processors. In *Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '04.

[76] Xiaotong Zhuang, Tao Zhang, and Santosh Pande. Hide: An infrastructure for efficiently protecting information leakage on the address bus. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XI, 2004.