InvisiMem: Smart Memory Defenses for Memory Bus Side Channel

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

ABSTRACT

A practically feasible low-overhead hardware design that provides strong defenses against memory bus side channel 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 aid in addressing several memory bus side channel vulnerabilities efficiently. This allows the secure host processor to send encrypted addresses over the untrusted memory bus, and thereby eliminates the need for expensive address obfuscation techniques based on Oblivious RAM (ORAM). In addition, smart memory enables efficient solutions for ensuring freshness without using expensive Merkle trees, and mitigates memory bus timing channel using constant heart-beat packets. We demonstrate that InvisiMem designs have one to two orders of magnitude of lower overheads for performance, space, energy, and memory bandwidth, compared to prior solutions.

CCS CONCEPTS

• **Security and privacy** → **Hardware-based security protocols**; • **Hardware** → *3D integrated circuits*;

KEYWORDS

Security, die stacking, memory

ACM Reference format:
Shaizeen Aga and Satish Narayanasamy. 2017. InvisiMem: Smart Memory Defenses for Memory Bus Side Channel. In *Proceedings of ISCA '17, Toronto, ON, Canada, June 24-28, 2017,* 13 pages.
https://doi.org/10.1145/3079856.3080232

1 INTRODUCTION

Cloud computing allows clients to outsource their computations to untrusted cloud service providers. Ensuring privacy of code and data on a computer physically owned and maintained by an untrusted party is challenging, as we must assume a powerful adversary. 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, 56].

ISCA '17, June 24-28, 2017, Toronto, ON, Canada

ACM ISBN 978-1-4503-4892-8/17/06…$15.00
https://doi.org/10.1145/3079856.3080232

A common solution is to reduce the attack surface by minimizing the trusted computing base (TCB) to a secure processor [1, 35] and a small portion of the client's application. Intel Software Guard Extensions (SGX) [9, 39] provides hardware primitives for this purpose. An SGX-enabled 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.

While the secure processor is trusted, the memory bus and the memory are not. Defending against memory bus side channel requires solutions to at least three problems: data and address confidentiality, data integrity and freshness, and timing channel. Solutions to these problems are expensive. For example, best known solution for address confidentiality is Oblivious RAM (ORAM) [24], which increases memory bandwidth consumption by ~100X.

In this paper, we present InvisiMem, a low-overhead secure processor that provides ORAM equivalent guarantees for the address side channel, ensures data integrity, freshness, and mitigates memory timing channel. InvisiMem is based on our observation that smart memories (memories with compute capability) with packetized interface (as opposed to the 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) [6] make it possible to stack DRAM layers on top of logic layers, and connect them using Through Silicon Vias (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 via a memory bus using a packetized interface (Figure 1 (a)). The logic in smart memory is included in TCB and is used to implement cryptographic functions, while memory layers remain outside TCB.

Data and Address Confidentiality: A secure processor guarantees data confidentiality by encrypting data before sending it to memory. Memory address, however, is sent in plain-text on the bus, as required by the DRAM's DDR interface. By observing the memory addresses, an adversary can infer sensitive program inputs [57] and cryptographic keys [62]. Prior solutions tackle this leak using Oblivious RAM (ORAM) [24]; a cryptographic construction that obfuscates memory accesses to make them indistinguishable from a random access pattern. To do so, it issues several memory accesses for every normal access. In spite of significant recent advancements [21, 61] such ORAM-based solutions increase memory access latency considerably (~20X) and incur huge performance

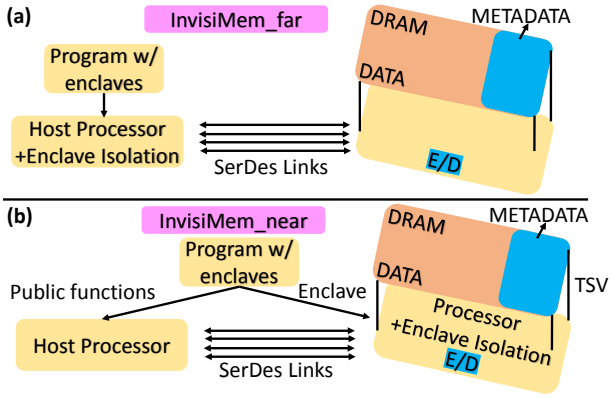


Figure 1: Smart memory based secure designs. a) InvisiMem_far b) InvisiMem_near

overheads(~4X). ORAM also has security limitations in that it does not help guard against memory timing channel [36].

InvisiMem's trusted compute capability in memory allows the processor to send the whole request packet, including the address, in an encrypted form. We observe, however, that *address encryption alone is insufficient to provide ORAM guarantees*. First, read requests or responses should appear indistinguishable from their write counterparts to an adversary snooping the memory bus to avoid leaking the memory access type. Second, on a read to a location, if the memory simply returns stored encrypted data at that location, an adversary can correlate it to the last write or previous reads to the same location. Solutions to address these problems are discussed.

Freshness without Merkle trees: Guaranteeing data freshness requires that an adversary should not be able to rollback the state of a memory block by recording and replaying older packets (either by manipulating values in memory or while being transmitted over the bus). To defeat such replay attacks, a conventional secure processor maintains current versions of memory blocks using Merkle trees [45] and verifies that a read response returns the latest version. Merkle trees impose severe memory space and bandwidth requirements [26].

We observe that smart memory can provide freshness without requiring the Merkle tree construct. To do that we set up a secure communication channel between processor and memory using authenticated encryption. This is possible thanks to the compute capability of memory. Using authenticated encryption, we can prevent data manipulation in memory or over the memory bus as both parties can ensure that requests/responses are originating from the authenticated counterpart and that they are fresh.

Timing Channel: Smart memory also enables an efficient solution for solving one type of memory timing channel: memory access and response times seen on the memory bus. InvisiMem sends constant rate heart-beat packets between the processor and smart memory in both directions. When an actual request or response is available, the next heart-beat packet's slot is used or else a dummy packet is transmitted. This is viable for systems with smart memory (HMC-like) for several reasons. First, the interface employed in such systems sends synchronization packets periodically even when there is no communication [4]. Hence, the energy overhead of turning them into heart-beat packets is relatively low. Second, unlike traditional memory systems, smart memory allows dummy requests to be ignored, lowering their energy overhead. Third, compute capability in smart memory allows responses from memory to be sent at a constant rate (unlike traditional memory systems). This helps hide variations in access times to different locations.

The above solution naturally supports a system with multiple memory modules connected to a processor. Sending constant rate heart-beat packets between every pair of communicating nodes hides all access patterns, and thereby also hides addresses accessed. We also discuss some precautions needed to support such systems.

Optimizations: 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 designs for efficiently storing and retrieving meta-data (for encryption and integrity checks) which exploit smart memory characteristics like vault-level parallelism.

With these optimizations, InvisiMem incurs 14.21% performance overhead, 53.03% energy overhead and 37.5% memory space overhead compared to an Intel Xeon-like processor.

The logic layer in smart memory has sufficient area and power budget (nearly 55W [18]) for a low-power processor. Executing enclaves in this core can hide all the communication between the core and memory, and thereby eliminate memory bus side channel. The trade-off, however, is that its compute capability may not match that of a high-performance core. We study this using a variant of our design, named InvisiMem_near (Figure 1 (b)).

Remote attestation and Key Management: InvisiMem expands TCB to include logic layer stacked with memory. Note that DRAM layers in memory are still outside TCB. Almost all secure processors such as Intel SGX, including all ORAM based designs, rely on public key infrastructure (PKI). Similarly, we propose that smart memory vendors also support PKI for trusted logic in memory. Using the public key of the smart memory, we show that a secure host processor can easily establish a secure communication channel with the smart memory's logic.

This paper makes the following contributions:

- This is the first paper which uses logic in 3D stacked smart memory and its packetized interface to efficiently address memory bus side channel vulnerabilities.
- We show that symmetric encryption of addresses alone is insufficient to provide ORAM-equivalent guarantees, and present efficient solutions to problems identified.
- We guarantee freshness using authenticated communication channel between processor and memory, obviating the need for more expensive solutions like Merkel trees [45].
- We solve memory bus timing channel elegantly by sending heart-beat messages at a constant rate in both directions. Given the synchronization messages needed in smart memory systems with packetized interface, the incremental cost of turning them into heart-beat messages is low.
- We customize meta-data storage and retrieval using smart memory characteristics such as vault-level parallelism.

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 defenses for memory bus side channel. Finally, we provide a brief background on 3D stacked memory, which we use in our system.

2.1 Enclaves for Isolation

Intel SGX [9, 39] is the latest hardware support for building trusted computing systems. It provides capability for isolating the execution of an enclave 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.

2.2 Threat Model

We assume a secure processor that supports isolated execution of enclaves (e.g. Intel SGX [9, 39]). We assume that an adversary cannot observe the communication between the layers in smart memory and that it's logic layer is secure.

We assume a powerful adversary that can compromise the operating system and use OS privileges to compromise the confidentiality and integrity of applications. This adversary also has physical access to the computers running client computations. Thus, he 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 [27] or corrupt state using Row-Hammer attacks [32].

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 [12, 15, 49, 55]. We also assume prior solutions for mitigating page-fault side-channel [15, 52] in enclaved systems. Power [33], thermal [43], program execution time [60] side-channels, and leaks via communication patterns over the network [40, 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 recent ORAM-based work, Freecursive ORAM [21] and Ghostrider [36]. While more recent ORAM-based work exists [61], we consider Freecursive ORAM [21] as it proposes a ORAM-based defense optimized for providing data integrity and freshness guarantees as well.

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 memory bus probes 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 [24] construct. 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 [21], though at a significant increase in hardware complexity, on-chip (80KB) and off-chip (more than 2X) space overhead.

ORAM does not prevent memory access time and total number of memory accesses from leaking. This is provided by the memory-trace obliviousness (MTO) guarantee provided by Ghostrider [36] which ensures that the program execution (instruction trace, time) is independent of program sensitive inputs. This requires a deterministic compiler, hardware which prohibits most commonly used optimizations (caches, instruction re-ordering etc) and also imposes non-trivial constraints on the program (e.g. sensitive input-independent loop guards). As a result, it incurs nearly 6X performance overhead. compared to a baseline with a single-issue, in-order processor.

An adversary can corrupt data in memory 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 violation. Replay attacks are thwarted by including a version number during the HMAC creation. The processor tracks the current version of the memory state using on-chip storage [21, 58], and uses it to ensure that a read returns the latest version. Both these guarantees incur additional performance and space overhead, and complexity.

With 3D smart memory, and by 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) along with data integrity and freshness guarantees.***

2.4 Smart Memory

3D integration has led to the rise of 3D-DRAM devices such as the Hybrid Memory Cube (HMC) [6]. 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) [4]. The layers are partitioned vertically into vaults (each with several DRAM banks) which can be accessed in parallel.

Channel	Leak/Vulnerability	Freecursive ORAM [21]	Ghostrider [36]	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/write	
	Trace length	with [22], yes	Deterministic execution	constant rate requests/responses	
	Access time	with [22], 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 + access count + position data checks	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.

HMC device is connected to the processor via SERDES links. Unlike traditional DRAM's DDR interface with low-level commands, HMC device is exposed via a more flexible packet interface.

Recent HMC device has a capacity of 2GB and can provide maximum memory bandwidth of 160GB/s [6]. While the logic layer in current devices contains circuits for interfacing with the vaults (memory controllers), it has sufficient area and thermal power budget (55W [18]) 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 interconnected through metal layers within a silicon interposer [16]. These metal tracks are etched using the same processes as the tracks on the silicon chips, and hence 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 [15] for isolation. As memory layers in smart memory are untrusted (cold-boot attacks, Section 2.2), we store encrypted data in memory using randomized encryption, similar to SGX.

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

We start by discussing smart memory advantages which help design low-overhead defenses for memory bus side channel and also lead to an efficient solution to guarantee freshness. We also discuss performance optimizations that we employ and efficient storage of meta-data in smart memory.

3.1 Advantages of Smart Memory

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

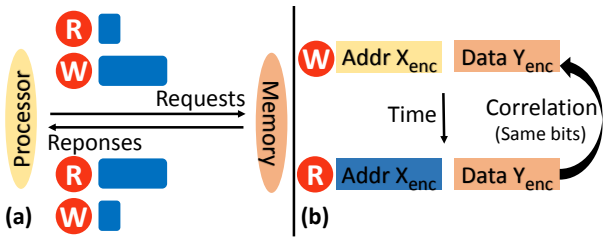


Figure 2: Symmetric encryption of addresses is not enough. (a) Distinguishing reads from writes (b) Correlation attack

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. 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 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) as depicted in Figure 2 (b). To solve

this problem, while responding to a read request, the smart memory double encrypts an already encrypted data and its timestamp, before sending a response.

Second, the communication between the processor and the memory in an insecure design is noticeably different for reads and writes (Figure 2 (a)). A read request and a write response do not carry data, while a write request and a read response do. Thus, an attacker could infer whether an access is a read or a write. We eliminate this leak by ensuring equal packet sizes for both read and write request/responses by adding a dummy block to read request and write response.

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 [32]). 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 a 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 creation. The processor must securely track the current version of the memory state [21, 58], 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.

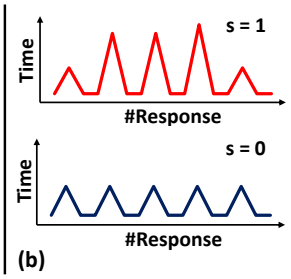
InvisiMem_far uses authenticated encryption [38] to guarantee freshness. Authenticated encryption ensures integrity and freshness of data sent over the untrusted memory bus. Using authenticated encryption, the sender (processor or memory) generates and sends an authentication tag over the encrypted packet sent to the receiver. The receiver uses this tag to check if the packet is the latest message from the trusted sender.

Authenticated encryption [38] uses an one-time pad (encryption of monotonically increasing counter) to generate encrypted data over which the authentication tag is then generated. As such, on a message replay, receiver's regenerated tag (using latest one-time pad) will not match the received tag (replayed) causing authentication failure. Note that when memory responds, it generates this tag over double encrypted data and timestamp (encrypted) (Section 3.2). Unlike prior designs [58], we avoid significant hardware state and memory space needed to track and check the versions of memory blocks.

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 that owns them.

```
for(i=0; i<N; i++){
  if(s)
    array[i % rand()]++;
  else
    array[i]++;
}
```

(a)



(b)

Figure 3: Time taken to respond by memory can leak sensitive inputs.

3.4 Mitigating Memory Bus Timing Channel

Memory access times observed on the memory bus can leak the program paths taken in an execution [22]. Memory response times to requests can also leak sensitive information. For example, reads with row-buffer locality will have significantly lower response latency than reads to different rows (Figure 3).

To solve both these leaks, the processor and the memory send heart-beat packets at a constant rate to each other. When there is a real packet to send, the sender transmits it at the next available slot. In the absence of a real packet, a dummy packet is sent, which is ignored by the receiver. This design trivially eliminates the two leaks noted above. Smart memory's capability to generate packets at a constant rate makes this design feasible. In a conventional memory system, as only the processor can send requests at a chosen rate, variations in response times noted above (Figure 3) are hard to mask.

We also experimented with a dynamic scheme that adjusted the packet rate according to application's memory access characteristics [22], but we did not find any significant performance or energy benefit in the context of a smart memory based design (Section 5.4).

We believe this is partly due to unique characteristics of smart memories. Smart memory is different from traditional memory systems modeled in prior work [22] in two aspects. First, smart memory can ignore dummy requests. Second, idle energy expended in smart memory is very high compared to a traditional DDR interface, as SerDes links in packetized interface require null packets to be sent at a constant rate for synchronization [4]. Link energy to transmit a null packet is about 75% of energy to send an actual packet [31].

Further, the energy expended in encrypting/decrypting packets does not constitute a significant fraction of the total system energy, even while operating at a packet rate that is high enough for the most memory intensive programs we studied.

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

While outside the scope of this work, an attacker capable of measuring power side channel can distinguish real and dummy requests in InvisiMem, as the smart memory ignores dummy requests. If this is a concern, instead of ignoring a dummy request, we can issue an access to a random location.

3.5 Performance: OTP Pre-computation

One-time pad (OTP) generation, which uses an AES encryption, is the most time consuming portion of GCM [38] which 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.

Synchronous timestamp counters avoid sending the timestamps along with each packet. More importantly, 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 compute the OTP and then decrypt the data using it. See Section 4.2 and Figure 5 for more details.

Synchronous timestamp counters are feasible as the communication network is point-to-point between the processor and memory, and is generally lossless. If a communication link is unreliable, then timestamp counters can lose synchronization when a packet is lost. Unreliable networks typically deal with lost messages by tagging packets with sequence numbers, and re-synchronize when a packet loss is detected. Similar techniques can be used to track lost packets in our system.

3.6 Space: Meta-Data in Smart 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 (note however that in a multiple module system, data and it's associated metadata are present in a single module). 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 4.4 for details. With these performance optimizations, non-fragmented design incurs only a negligible performance overhead compared to the fragmented design, but has better space utilization (91.66% compared to 68.75%).

3.7 Remote Attestation and Key Exchange

We now discuss a simplified client-processor remote attestation protocol [1, 9, 14], and how we could adapt it to set up a secure communication channel between the processor and memory in InvisiMem.

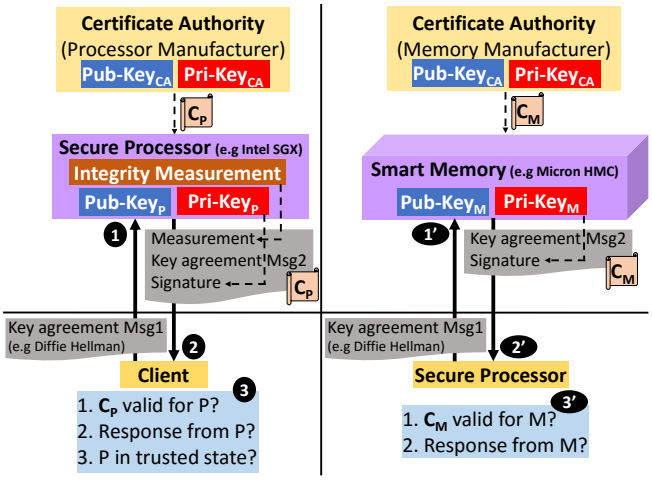


Figure 4: Existing client-host remote attestation and key exchange (left). Smart memory authentication and key exchange under InvisiMem (right).

Remote attestation is a process by which a remote host proves to a client that it is running trusted software on trusted hardware. On successful conclusion of remote attestation, the client shares its sensitive data (e.g., private keys) with the host before commencing computation.

A secure processor manufacturer endows each secure processor with a unique public-private key pair. It also serves as a certificate authority that provides a *certificate* that binds the processor's identity to its public key. In addition, a secure processor has support for integrity measurement (a hash of code, data, and system state).

The client aims to attest a remote processor and setup a shared session key to communicate sensitive data with it. To do that, it sends a key agreement message (❶) to the remote host [17]. The processor uses this to generate a response key agreement message. This along with its integrity measurement is signed with the processor's private key. The signed message is sent along with the processor's certificate issued by its manufacturer (❷) to the client. The client (❸), first verifies that the certificate is valid using manufacturer's public key. Then, using the processor's public key in the certificate, it verifies that the received message is indeed from the processor. It further checks if the measurement value is as expected. If it is, then the client uses the key agreement message received to compute the shared session key for further secure communication.

In InvisiMem, the secure processor and the client use the conventional remote attestation protocol described above. Secure processor uses a similar protocol to authenticate and exchange keys with its secure memory. The only difference is that the secure memory does not need integrity measurement support. To support this, just like processor manufacturers, we propose that memory manufacturers endow smart memory modules with public-private key pair, and serve as its certification authorities.

To support the above protocol, we assume that smart memory's logic can support asymmetric encryption. Given its area and thermal

budgets (Section 2.4), it should be able to support asymmetric encryption which is implemented even in smart-cards [29] with much lesser resources.

3.7.1 Security Considerations

Remote attestation protocol discussed above is immune to man-in-the middle (MITM) attacks. The second check in the last step of the protocol described above (❸) ensures that the response received is indeed from the trusted entity. Private keys stored in secure processor and secure memory are tamper-proof. But to manage scenarios where vulnerabilities are discovered after deployment, certificate authorities can maintain certificate revocation lists. Alternatively, certificates can be associated with expiration dates. None of these problems and solutions are unique to processor-memory authentication and key exchange described above, as they exist in client-processor remote attestation as well.

Secure distribution of public keys remains a challenging problem [14]. Any secure processor, including ORAM-based solutions, face this challenge. InvisiMem additionally requires public key distribution for secure memory as well. Today, processor manufacturers such as Intel serve as certificate authorities (CAs). Being reliant on processor and memory manufacturers to establish trust could place undue power in their hands and stifle innovation. We can get around this by designating a trusted third-party to act as a CA.

3.8 Key and Timestamp Management

Storage: Keys (data and address) and timestamps we employ are stored in special registers at memory controllers at both side. To ensure process isolation, each process has a different key. But we need only as many special registers for keys as there are processor cores, as the number of active processes cannot be more than that. A timestamp can be shared amongst processes. Security is guaranteed as long as a given timestamp is never reused for the same key, which is ensured by incrementing it each time it is used. Note that techniques that tackle timestamp overflow [58] can be adapted in our system.

Multiple Memory Modules: Systems where a secure processor is connected to multiple memory modules are possible [31]. Therein, the processor can setup a secure channel with each memory module. This will require a timestamp per memory module. Furthermore, to ensure that the same timestamp-key pair is not reused, we statically partition timestamp ranges amongst memory modules.

A security vulnerability in such a system is that an adversary can gain some information about an address accessed simply by observing which module is accessed. Fortunately, our timing channel solution (Section 3.4) addresses this problem.

3.9 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 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 prevent memory timing channel leaks (Section 3.4).

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

4.1.1 Authenticated Encryption

We use Galois/Counter operation mode (GCM) [38] with AES for authenticated encryption. GCM operates on 128-bit blocks. Therefore, a single cache block (64 bytes) is broken into 4 blocks of plain-text. One Time Pad (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 [44]. For authentication, GCM employs a GHASH function [38], 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 (*Ka*), data (*Kd*), and data double encryption (*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 blocks. Timestamp for each block is produced by concatenating 64-bit timestamp (DTS1) with a 62-bit fixed vector (FV) and a two bit block-id representing it's relative position in the cache block. Since the timestamp for a cache block has to be stored along with data in memory, 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

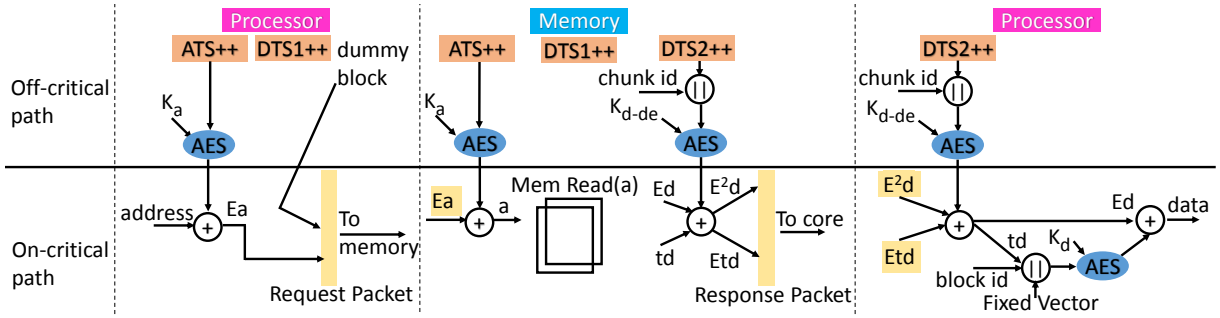


Figure 5: InvisiMem_far Security Protocol for Read. `td`: timestamp stored with data.

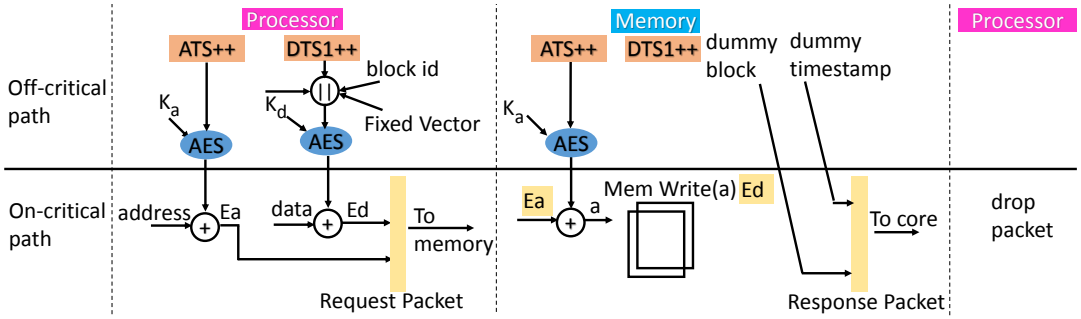


Figure 6: InvisiMem_far Security Protocol for Write.

authenticated encryption (Figure 7 (a)). This requires registers for timestamps and keys mentioned in Section 4.1.2. PMC and MMC have three AES and four Galois Field multipliers (GF-M) [47] each.

4.2 InvisiMem_far Security Protocol

Figures 5 and 6 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 5, 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 and dummy encrypted data block. The latter is added to the request packet 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 6), PMC encrypts data and address; while MMC decrypts address both 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 4.1.1). 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 for transmission. We can overlap some of these operations with data (and address) encryption/decryption.

4.3 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 authentication delay is added to critical path for a write request, a read response overlaps it with data decryption.

4.4 Storing Meta-Data in Smart Memory

Meta-data for a cache block consists of 64-bit timestamp value (DTS1) and an integrity tag. Section 3.6 described two designs for storing this meta-data: fragmented and non-fragmented.

Figure 7 (a) depicts fragmented layout. Storing data and its meta-data together requires 88 bytes (64 data, 8 timestamp, 16 tag). HMC Specification [4] mandates that memory block sizes can be 32/64/128/256 bytes. Therefore, in the fragmented layout, 88-byte

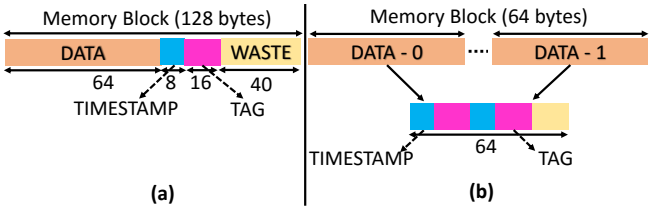


Figure 7: (a) Fragmented Design (b) Non-fragmented Design

Configuration	4 cores, 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 [4], 128 TSV's per vault @2Gb/s [30]
Off-chip links	4 SerDes links, 16 lanes per link, direction [4]

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: LLC_MPKI and IPC for DRAM_hp.

data block and its meta-data consumes 128-bytes, resulting in effective memory utilization of 68.75%.

Figure 7(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.

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 (in bytes).

5 EVALUATION

5.1 Methodology

We study 22 benchmarks from SPEC 2006 [28] suite with reference inputs. We use the Simpoint [50] 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 (unsecure baseline without smart memory).

Processor Model: We modeled our processor designs (Table 2) using MARSSx86 [41], a full system cycle accurate simulator. Processor in InvisiMem_far is similar to Intel Quad Core i7-4790K processor [5]. Invisimem_near places secure processor in the logic die of smart memory. Eckert et al. [18] 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 [7] at 45nm and scaled it using ITRS projections to model its latency in our system. The Galois Field multiplication (authenticated encryption) is a combinational circuit that operates in single cycle [47, 58].

Power Model: We model processor power using McPAT [34] and AES energy to be 302 pJ [37] per 128-bit block. For baseline DRAM, we model access energy to be 65 pJ/bit [30]. A recent industry prototype [30] reports 10.48pJ/bit for HMC access of which 43% is attributed to SerDes circuits [30, 42], rest is for DRAM access and logic layer. We model 1.42W for DRAM static power.

Smart Memory Model: We use DRAMSIM2 [46] 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 [4]. 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 [30].

Table 4 shows packet sizes communicated in baseline [4] and InvisiMem_far. Each packet has 8-bytes of header and tail, which carry useful meta-data like command, address etc. Hence, read request or write response is 16 bytes in size. Write requests and read responses carry 64 bytes of data as well. In our design read/write requests also transmit authentication tags (16 bytes for packet header/tail, 16 bytes for data). Responses additionally carry data timestamp (8 bytes).

5.2 Unsecure Smart Memory Performance and Energy

Figure 8 shows the performance overhead with respect to unsecure baseline DRAM_hp of various designs modeled with increasing security guarantees. We plot the benchmarks in the increasing order of their LLC miss rates. The 3D_far design represents an unsecure

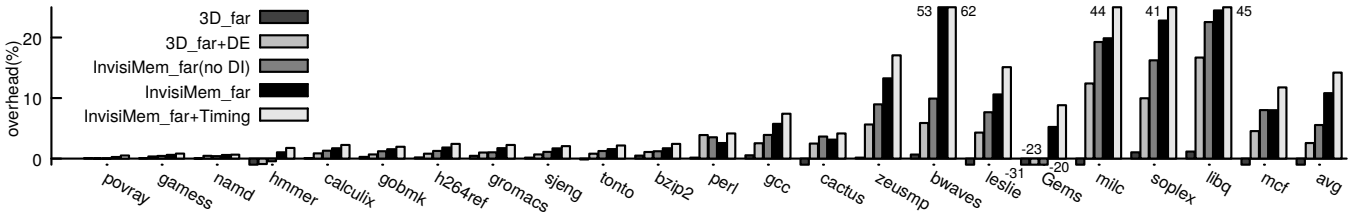


Figure 8: Performance overhead of far-memory processor unsecure and secure designs w.r.t DRAM_hp.

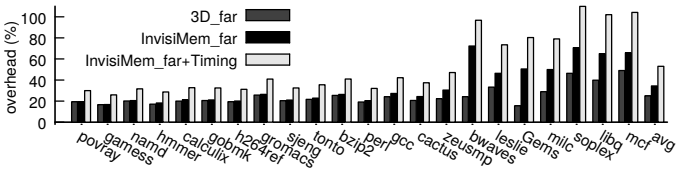


Figure 9: Energy overhead w.r.t DRAM_hp.

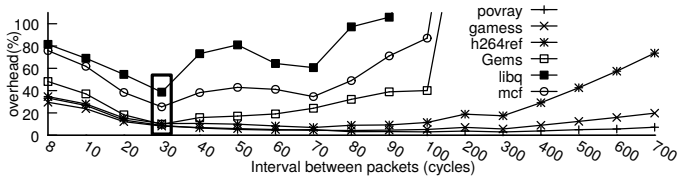


Figure 10: ED^2 overhead w.r.t InvisiMem_far without timing channel defense for various static memory access rates.

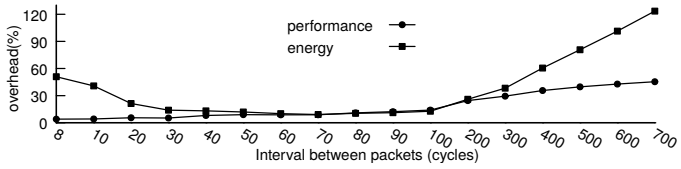


Figure 11: Overheads w.r.t InvisiMem_far without timing channel defense for various static memory access rates.

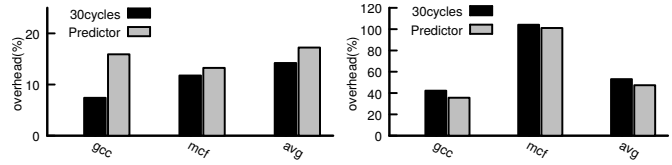


Figure 12: Comparison to dynamic scheme.

high power processor connected to smart memory. High bandwidth smart memory helps improve performance of memory intensive programs (GemsFDTD sees gain of 31.41%). On average, smart memory delivers performance improvement of 4.02%.

Figure 9 shows the energy overhead of 3D_far design (average 24.98%). While the DRAM energy is lower for smart memory, the static power expended by the SerDes links is the chief cause of this overhead. High static power is caused as SerDes links transmit null packets when idle [4]. Prior work also observes that SerDes link power is a significant fraction of the total HMC power [8, 23, 42].

5.3 Far InvisiMem

This section discusses performance and energy overheads of our InvisiMem_far designs to guarantee security properties equivalent to ORAM, data integrity, freshness, and also avoid leaking timing of memory events.

3D_far+DE configuration in Figure 8 adds data encryption (DE) to 3D_far design. This model helps us tease out data encryption overheads (incurred in secure processors like Intel SGX) from the address encryption overheads. Adding data encryption incurs modest overhead of 2.58% on average.

To tease out the overhead of providing ORAM guarantees from the other security guarantees, we model InvisiMem_far (no DI) configuration which provides only ORAM guarantee. In this design we encrypt both address and data, but authenticate only address. This increases the overhead from 2.58% (3D_far+DE) to mere 5.55%.

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% (highest overhead for bwaves of 52.65%). This is a significant improvement over prior ORAM-based solutions [21], 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 14.21%.

Figure 9 shows the energy overheads of InvisiMem_far. Without timing channel defense, InvisiMem_far increases the energy overhead of 3D_far design from 24.98% to 34.38%; with it the overhead is 53.03%. This is a sharp contrast to prior works which incur one to two orders of performance loss, bandwidth increase, and commensurate energy overhead.

5.4 Static Packet Rate for Timing Channel

As discussed in Section 3.4, we choose a static request and response rate to address timing channel leaks. We provide here the empirical evidence to support this choice. Figure 10 depicts energy delay squared (ED^2) overhead of various static packet rates with respect to Invisimem_far without timing channel protection. To depict spectrum of behaviors, we pick two least and most memory intensive benchmarks, and two benchmarks with highest and lowest IPC.

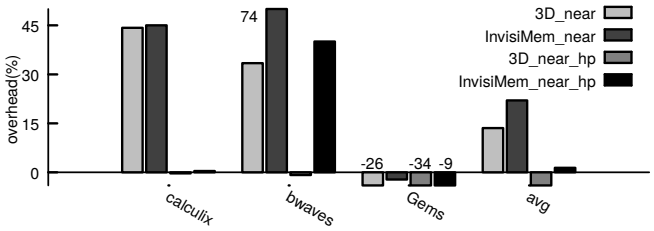


Figure 13: Performance overhead of near-memory designs w.r.t DRAM_hp.

Guarantee	Encryption	ORAM	Integrity
Freecursive ORAM [21]	256MB	5.8GB	2GB
InvisiMem_far	512MB	0B	1GB

Table 5: Memory space overheads.

We see that the lowest (ED^2) overhead occurs roughly at 30-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 links 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. This combined effect is depicted in Figure 11 (averaged across six benchmarks under study) wherein energy overheads first start to drop before showing a negative trend.

We also implemented a dynamic predictor [22] (ED^2 overhead of 165.58%) with rates (30, 60, 120, 240). Figure 12 compares this predictor to our static scheme with 30 cycles (ED^2 overhead of 159.73%). 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 improvements or energy savings with a dynamic scheme. Considering it has a weaker security guarantee, we chose a static rate.

5.5 Near InvisiMem

Figure 13 shows the performance overhead of two near-memory processor designs we model, with and without security guarantees (We depict outlier benchmarks only for space considerations). 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 overheads (44.23% for `calculix`), whereas memory-intensive benchmarks see performance gains (`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. For such a design, the average overhead of providing data encryption and integrity is a mere 1.41%.

5.6 Memory Space Overhead

Table 5 lists the space overheads of a recent ORAM-based proposal Freecursive-ORAM [21] and InvisiMem_far for various security guarantees. For both designs we report space overheads for 4GB of real data with 64 bytes block size.

InvisiMem_far incurs more space overhead to store encryption timestamps as these are per cache block. In contrast, in Freecursive-ORAM, these timestamps are per bucket which comprises of multiple cache blocks. However, this reduction in space overhead has concomitant performance, energy and bandwidth cost as buckets have to be read and written in their entirety. InvisiMem_far incurs no space overhead for ORAM guarantees. Freecursive-ORAM, on the other hand, incurs close to 100% space overhead to store dummy data blocks and other metadata needed to implement the ORAM algorithm. Finally, for data integrity, Freecursive-ORAM has higher overheads as it needs tags for dummy cache blocks as well.

5.7 Fragmented Vs Non-Fragmented

Section 4.4 discussed two ways in which a memory block and its metadata (timestamp/tag) can be stored and retrieved from memory. Non-fragmented design has better memory utilization than fragmented design. However, it also breaks every memory request into requests for data and metadata. Owing to vault-level parallelism and our data mapping policy (Section 4.4) 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 memory based solution for memory bus side channel.

6.1 3D Stacking for Security

Only two prior proposals [26, 54] harness 3D-stacking to provide security guarantees. In [54] a control plane is integrated with a conventional processor in a 3D stack which provides security functionalities like monitoring activities of the processor to prevent cache-side channel attacks. In [26], the authors leverage smart memory logic to efficiently implement Bonsai Merkle Tree [45]. Our work obviates need for Merkle trees by using memory isolation provided by Intel SGX and employing authenticated encryption between processor and memory. Both prior works did not harness smart memories to solve address side channel or timing channel.

Concurrent to our work, ObfusMem [10] also uses smart memory to provide ORAM-equivalent guarantee. InvisiMem provides a stronger memory-trace obliviousness (MTO) guarantee [36] by hiding memory access and response times using constant rate messages.

6.2 Secure Hardware

Several secure hardware proposals [1, 11, 25, 35, 53] exist which chiefly aim to provide isolation (protect sensitive application from other applications and untrusted system software) and software attestation. The latest proposal: Intel SGX [9, 39], provides isolated execution, and reduces the trusted computing base to the application and few privileged containers. It also provides encryption, data integrity and freshness guarantees. There are other proposals with similar or more security guarantees like SGX [13, 15, 19]. None of these proposals address memory bus side channel.

We discussed solutions [21, 36, 61] that provide defenses against memory bus side channel in Section 2. They incur order of magnitude more memory accesses and result in huge performance overheads. We show in this work that with smart memory, memory bus side channel can be solved with low overheads. Prior works have also considered sending memory requests from the processor at a static [20] or dynamic [22] rate. Both rely on ORAM algorithm to generate indistinguishable real and dummy requests. Our timing solution does not rely on ORAM-algorithm. By employing smart memory it can generate constant response rate to hide response time variations. It can avoid processing dummy requests in memory, saving energy. Also, unlike prior schemes [22], our solution is not limited to only one pending memory request at any time.

Prior works [49, 55] address information leakage when an untrusted program shares the memory system with a trusted application. Wang et al. [55] extended memory controller to allocate fixed time quantum for each thread when they can issue memory accesses. It does not hide when a thread issues requests or receives responses within its time quantum from an adversary who has physical access to memory bus. Shafiee et al. [49] proposed to issue a real or dummy request every Q cycles for each thread, and architected a deterministic memory that guaranteed a response before the interval ends. Deterministic memory forgoes optimizations such as row-buffer hits. Unlike InvisiMem, it does not hide type of memory access (read or write). Also, InvisiMem leverages packetized interface of smart memory to support constant rate responses without requiring any changes to DRAM design, and is more efficient.

Optimizing Memory Encryption: Prior works propose several optimizations to reduce memory encryption overheads which can be used in our design. In [51], the authors predict encryption counters for speculative OTP pre-computation. In [58, 59], encryption counters are cached which can also further reduce our overheads.

7 CONCLUSION

We present InvisiMem, which harnesses smart memory with compute capability to simplify solutions for providing address and data confidentiality, data integrity, freshness, and also address 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 lower overheads for performance, space, energy and memory bandwidth, when compared to prior solutions that relied on Oblivious RAM and Merkle trees.

8 ACKNOWLEDGMENTS

We thank Mohit Tiwari and anonymous reviewers for their comments which helped improve this paper. We also thank Sriram Rajamani for the valuable discussions. This work was supported in part by the NSF under the CAREER-1149773 and SHF-1527301 awards and by C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

REFERENCES

[1] 2003. Trusted Computing Group. Tpm main specification. Retrieved April 1, 2016 from "http://www.trustedcomputinggroup.org/resources/tpm_main_specification, 2003.". (2003).
[2] 2006. I2C bus monitor. Retrieved April 1, 2016 from "http://www.jupiteri.com/". (2006).
[3] 2012. Intel Core i7-3770T. Retrieved April 1, 2016 from "http://ark.intel.com/products/65525/Intel-Core-i7-3770T-Processor-8M-Cache-up-to-3_70-GHz". (2012).
[4] 2014. Hybrid memory cube specification 2.0. (2014).
[5] 2014. Intel Core i7-4790K. Retrieved April 1, 2016 from "http://ark.intel.com/products/80807/Intel-Core-i7-4790K-Processor-8M-Cache-up-to-4_40-GHz". (2014).
[6] 2016. HMC. Retrieved April 1, 2017 from "https://www.micron.com/products/hybrid-memory-cube". (2016).
[7] 2016. OpenCores. "http://opencores.org/". (2016).
[8] Junwhan Ahn, Sungjoo Yoo, and Kiyoung Choi. 2014. Dynamic Power Management of Off-Chip Links for Hybrid Memory Cubes. In *Proceedings of the 51st Annual Design Automation Conference (DAC '14)*. 139:1–139:6. https://doi.org/10.1145/2593069.2593128
[9] Ittai Anati, Shay Gueron, Simon P Johnson, and Vincent R Scarlata. 2013. 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)*.
[10] Amro Awad, Yipeng Wang, Deborah Shands, and Yan Solihin. 2017. ObfusMem: A Low-Overhead Access Obfuscation for Trusted Memories. In *2017 44th Annual International Symposium on Computer Architecture (ISCA)*. https://doi.org/10.1145/3079856.3080230
[11] David Champagne and Ruby Lee. 2010. Scalable architectural support for trusted software. In *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. 1–12. https://doi.org/10.1109/HPCA.2010.5416657
[12] Jie Chen and Guru Venkataramani. 2014. CC-Hunter: Uncovering Covert Timing Channels on Shared Processor Hardware. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 216–228. https://doi.org/10.1109/MICRO.2014.42
[13] Siddhartha Chhabra, Brian Rogers, Yan Solihin, and Milos Prvulovic. 2011. SecureME: A Hardware-software Approach to Full System Security. In *Proceedings of the International Conference on Supercomputing (ICS '11)*. 108–119. https://doi.org/10.1145/1995896.1995914
[14] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. Retrieved April 1, 2016 from Cryptology ePrint Archive, Report 2016/086. (2016).
[15] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, 857–874. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan
[16] Yangdong Deng and Wojciech P. Maly. 2001. Interconnect Characteristics of 2.5-D System Integration Scheme. In *Proceedings of the 2001 International Symposium on Physical Design (ISPD '01)*. 171–175. https://doi.org/10.1145/369691.369763
[17] Whitfield Diffie and Martin Hellman. 1976. New directions in cryptography. *IEEE Transactions on Information Theory* (1976), 644–654. https://doi.org/10.1109/TIT.1976.1055638
[18] Yasuko Eckert, Nuwan Jayasena, and Gabriel Loh. 2014. Thermal Feasibility of Die-Stacked Processing in Memory. In *Workshop on Near-Data Processing*.
[19] Dmitry Evtyushkin, Jesse Elwell, Meltem Ozsoy, Dmitry Ponomarev, Nael Abu Ghazaleh, and Ryan Riley. 2014. Iso-X: A Flexible Architecture for Hardware-Managed Isolated Execution. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 190–202. https://doi.org/10.1109/MICRO.2014.25
[20] Christopher W. Fletcher, Marten van Dijk, and Srinivas Devadas. 2012. A Secure Processor Architecture for Encrypted Computation on Untrusted Programs. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing (STC '12)*. 3–8. https://doi.org/10.1145/2382536.2382540
[21] Christopher W. Fletcher, Ling Ren, Albert Kwon, Marten van Dijk, and Srinivas Devadas. 2015. Freecursive ORAM: [Nearly] Free Recursion and Integrity Verification for Position-based Oblivious RAM. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '15)*. 103–116. https://doi.org/10.1145/2694344.2694353
[22] Christopher W. Fletcher, Ling Ren, Xiangyao Yu, Marten Van Dijk, Omer Khan, and Srinivas Devadas. 2014. Suppressing the Oblivious RAM timing channel while making information leakage and program efficiency trade-offs. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 213–224. https://doi.org/10.1109/HPCA.2014.6835932
[23] Maya Gokhale, Scott Lloyd, and Chris Macaraeg. 2015. Hybrid Memory Cube Performance Characterization on Data-centric Workloads. In *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*. 7:1–7:8. https://doi.org/10.1145/2833179.2833184
[24] Oded Goldreich and Rafail Ostrovsky. 1996. Software Protection and Simulation on Oblivious RAMs. *J. ACM* (1996), 431–473. https://doi.org/10.1145/233551.233553
[25] David Grawrock. 2009. Dynamics of a Trusted Platform: A building block approach. Retrieved April 1, 2016 from Intel Press, 2009. (2009).

[26] Akhila Gundu, Ali Shafiee Ardestani, Manjunath Shevgoor, and Rajeev Balasubramonian. 2014. A Case for Near Data Security. In *Workshop on Near-Data Processing*.

[27] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. 2009. Lest We Remember: Cold-boot Attacks on Encryption Keys. *Commun. ACM* (2009), 91–98. https://doi.org/10.1145/1506409.1506429

[28] John L. Henning. 2006. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comput. Archit. News* (2006), 1–17. https://doi.org/10.1145/1186736.1186737

[29] Zhen Huang and Shuguo Li. 2011. A Low Power RSA Design for Smartcard. In *Proceedings of the 2011 Third International Workshop on Education Technology and Computer Science - Volume 01 (ETCS '11)*. 31–34. https://doi.org/10.1109/ETCS.2011.16

[30] Joe Jeddeloh and Brent Keeth. 2012. Hybrid memory cube new DRAM architecture increases density and performance. In *2012 Symposium on VLSI Technology (VLSIT)*. 87–88. https://doi.org/10.1109/VLSIT.2012.6242474

[31] Gwangsun Kim, John Kim, Jung Ho Ahn, and Jaeha Kim. 2013. Memory-centric system interconnect design with Hybrid Memory Cubes. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*. 145–155. https://doi.org/10.1109/PACT.2013.6618812

[32] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 361–372. https://doi.org/10.1109/ISCA.2014.6853210

[33] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. 2011. Introduction to differential power analysis. *Journal of Cryptographic Engineering* (2011), 5–27. https://doi.org/10.1007/s13389-011-0006-y

[34] Sheng Li, Jung Ho Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, and N.P. Jouppi. 2009. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 42)*. 469–480. https://doi.org/10.1145/1669112.1669172

[35] David Lie, Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. 2000. Architectural Support for Copy and Tamper Resistant Software. *SIGPLAN Not.*, 168–177. https://doi.org/10.1145/356989.357005

[36] Chang Liu, Austin Harris, Martin Maas, Michael Hicks, Mohit Tiwari, and Elaine Shi. 2015. GhostRider: A Hardware-Software System for Memory Trace Oblivious Computation. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '15)*. 87–101. https://doi.org/10.1145/2694344.2694385

[37] Sanu Mathew, Farhana Sheikh, Amit Agarwal, Mike Kounavis, Steven Hsu, Himanshu Kaul, Mark Anders, and Ram Krishnamurthy. 2011. 53 Gbps Native *rmGF*2^{42} Composite-Field AES-Encrypt/Decrypt Accelerator for Content-Protection in 45 nm High-Performance Microprocessors. *IEEE Journal of Solid-State Circuits*, 767–776. https://doi.org/10.1109/JSSC.2011.2108131

[38] David A. McGrew and John Viega. 2004. The Galois/Counter Mode of Operation (GCM). Retrieved April 1, 2016 from "http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf". (2004).

[39] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. 2013. 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)*. 10:1–10:1. https://doi.org/10.1145/2487726.2488368

[40] Olga Ohrimenko, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Markulf Kohlweiss, and Divya Sharma. 2015. Observing and Preventing Leakage in MapReduce. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. 1570–1581. https://doi.org/10.1145/2810103.2813695

[41] Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. 2011. MARSS: A Full System Simulator for Multicore x86 CPUs. In *Proceedings of the 48th Design Automation Conference (DAC '11)*. 1050–1055. https://doi.org/10.1145/2024724.2024954

[42] Seth H. Pugsley, Jeffrey Jestes, Huihui Zhang, Rajeev Balasubramonian, Vijayalakshmi Srinivasan, Alper Buyuktosunoglu, Al Davis, and Feifei Li. 2014. 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*.

[43] Jean-Jacques Quisquater and David Samyde. 2002. Side Channel Cryptanalysis. In *Workshop on the Security of Communications on the Internet (SECI)*.

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

[45] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. 2007. Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 40)*. 183–196. https://doi.org/10.1109/MICRO.2007.44

[46] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. 2011. DRAMSim2: A Cycle Accurate Memory System Simulator. *IEEE Computer Architecture Letters* (2011), 16–19. https://doi.org/10.1109/L-CA.2011.4

[47] Akashi Satoh. 2007. High-Speed Parallel Hardware Architecture for Galois Counter Mode. In *2007 IEEE International Symposium on Circuits and Systems*. 1863–1866. https://doi.org/10.1109/ISCAS.2007.378278

[48] Felix Schuster, Manuel Costa, Cedric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2014. *VC3: Trustworthy Data Analytics in the Cloud*. Technical Report MSR-TR-2014-39. http://research.microsoft.com/apps/pubs/default.aspx?id=210786

[49] Ali Shafiee, Akhila Gundu, Manjunath Shevgoor, Rajeev Balasubramonian, and Mohit Tiwari. 2015. Avoiding Information Leakage in the Memory Controller with Fixed Service Policies. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48)*. 89–101. https://doi.org/10.1145/2830772.2830795

[50] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically Characterizing Large Scale Program Behavior. *SIGOPS Oper. Syst. Rev.*, 45–57. https://doi.org/10.1145/635508.605403

[51] Weidong Shi, Hsien-Hsin S. Lee, Mrinmoy Ghosh, Chenghuai Lu, and Alexandra Boldyreva. 2005. High efficiency counter mode security architecture via prediction and precomputation. In *32nd International Symposium on Computer Architecture (ISCA'05)*. 14–24. https://doi.org/10.1109/ISCA.2005.30

[52] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. 2015. Preventing Your Faults From Telling Your Secrets: Defenses Against Pigeonhole Attacks. *CoRR* (2015). http://arxiv.org/abs/1506.04832

[53] G. Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. 2003. AEGIS: Architecture for Tamper-evident and Tamper-resistant Processing. In *Proceedings of the 17th Annual International Conference on Supercomputing (ICS '03)*. 160–171. https://doi.org/10.1145/782814.782838

[54] Jonathan Valamehr, Mohit Tiwari, Timothy Sherwood, Ryan Kastner, Ted Huffmire, Cynthia Irvine, and Timothy Levin. 2010. Hardware Assistance for Trustworthy Systems Through 3-D Integration. In *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC '10)*. 199–210. https://doi.org/10.1145/1920261.1920292

[55] Yao Wang, Andrew Ferraiuolo, and G. Edward Suh. 2014. Timing channel protection for a shared memory controller. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 225–236. https://doi.org/10.1109/HPCA.2014.6835934

[56] Lee Whetsel. 1991. AN IEEE 1149.1 BASED LOGIC/SIGNATURE ANALYZER IN A CHIP. In *1991, Proceedings. International Test Conference*. 869–. https://doi.org/10.1109/TEST.1991.519753

[57] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *2015 IEEE Symposium on Security and Privacy*. 640–656. https://doi.org/10.1109/SP.2015.45

[58] Chenyu Yan, Daniel Englender, Milos Prvulovic, Brian Rogers, and Yan Solihin. 2006. Improving Cost, Performance, and Security of Memory Encryption and Authentication. In *33rd International Symposium on Computer Architecture (ISCA'06)*. 179–190. https://doi.org/10.1109/ISCA.2006.22

[59] Jun Yang, Youtao Zhang, and Lan Gao. 2003. Fast secure processor for inhibiting software piracy and tampering. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.* 351–360. https://doi.org/10.1109/MICRO.2003.1253209

[60] Danfeng Zhang, Aslan Askarov, and Andrew C. Myers. 2011. Predictive Mitigation of Timing Channels in Interactive Systems. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*. 563–574. https://doi.org/10.1145/2046707.2046772

[61] Xian Zhang, Guangyu Sun, Chao Zhang, Weiqi Zhang, Yun Liang, Tao Wang, Yiran Chen, and Jia Di. 2015. Fork Path: Improving Efficiency of ORAM by Removing Redundant Memory Accesses. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48)*. 102–114. https://doi.org/10.1145/2830772.2830787

[62] Xiaotong Zhuang, Tao Zhang, and Santosh Pande. 2004. 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)*. 72–84. https://doi.org/10.1145/1024393.1024403