Efficient Distributed Secure Memory with Migratable Merkle Tree

Erhu Feng[†], Dong Du[†], Yubin Xia[†§], Haibo Chen[†]
[†]*Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University*
[§]*Shanghai Artificial Intelligence Laboratory*

***Abstract*—Hardware-assisted enclaves with memory encryption have been widely adopted in the prevailing architectures, e.g., Intel SGX/TDX, AMD SEV, ARM CCA, etc. However, existing enclave designs fall short in supporting efficient cooperation among cross-node enclaves (i.e., multi-machines) because the range of hardware memory protection is within a single node. A naive approach is to leverage cryptography at the application level and transfer data between nodes through secure channels (e.g., SSL). However, it incurs orders of magnitude costs due to expensive encryption/decryption, especially for distributed applications with large data transfer, e.g., MapReduce and graph computing. A secure and efficient mechanism of distributed secure memory is necessary but still missing.**

This paper proposes Migratable Merkle Tree (MMT), a design enabling efficient distributed secure memory to support distributed confidential computing. MMT sets up an *integrity forest* for distributed memory on multiple nodes. It allows an enclave to securely delegate an *MMT closure*, which contains both data and metadata of a *subtree*, to a remote enclave. By reusing the memory encryption mechanisms of existing enclaves, our design achieves secure data transfer without software re-encryption. We have implemented a prototype of MMT and a trusted firmware for management, and further applied MMT to real-world distributed applications. The evaluation results show that compared with existing systems using the AES-NI instruction, MMT can achieve up to 13x speedup on data transferring, and gain 12%∼58% improvement on the end-to-end performance of MapReduce and PageRank.

I. Introduction

There has been a surge of interest in utilizing hardware-assisted enclaves [3], [6], [14], [17], [22], [25], [37] like Intel SGX, AMD SEV, ARM CCA and RISC-V Penglai [25], for protecting integrity and confidentiality for security-sensitive applications. Most of these enclaves adopt a hardware-based memory protection engine, such as Intel MEE [32], BMT [44], Vault [56] and Mountable MT [25], which provides strong security guarantees for physical memory, defending against memory spoofing [19], [70], aliasing [48], splicing and replay attacks [19], [44], [52], [56]. Meanwhile, some recent studies [25], [33], [46], [55] have pointed out that using a scalable integrity tree scheme can support up to 512GB of enclave memory.

However, existing enclaves are designed to protect memory on a single node and cannot well fit the requirements of distributed applications. Since the network among computing nodes is considered untrusted, existing systems have to resort to software secure channel, e.g., TLS/SSL [24], to protect the confidentiality, integrity and freshness of the data transferring, and use a local memory protection engine to reconstruct the integrity tree in the remote node. For example, VC3 [47] deploys a crypto-based protocol to protect the secure distributed MapReduce computation; Civet [59] checks the enclave interfaces and inputs with the cryptographic protection; Graviton [61] and HIX [35] leverage the AES-SHA3/OCB-AES algorithm to shield the sensitive data transferring between the CPU and GPU.

It is known that crypto-based approaches incur significant overheads during data transferring. Table I shows the max bandwidth of different interconnections. The state-of-the-art RDMA-based NIC [7] can provide 400 Gbps bandwidth and reduce the round-trip latency to microseconds. In addition, the latest PCI-E 5.0/6.0 standards [10], [11], CXL [4] and NVLink [8] can support more than 800 Gbps bandwidth [10] However, the cryptographic algorithm such as AES-SHA3 has much less throughput, even for the state-of-the-art FPGA-based AES/GCM accelerator, which can only achieve 2∼40 Gbps throughput [2], [15], [58]. Graviton also shows that the AES-SHA3 algorithm will bring about 10∼20 times overhead compared with the direct memory copy between CPU and GPU nodes. sRDMA [58] extends RDMA protocol to encrypt and authenticate the payload in the RDMA packet, but sacrifices the RDMA bandwidth (only 10 Gbps for sRDMA compared with max 400 Gbps RDMA bandwidth).

To achieve (nearly) linear speed of secure data transfer between distributed enclaves, our idea is to scale the memory protection engine, which can already protect the confidentiality, integrity and freshness of physical memory, from one node to multiple nodes. By reusing the hardware protection mechanisms, no extra cryptographic-based operation is needed, and an enclave can directly transfer data to another enclave through various connections like PCIe, RDMA, etc.

Nevertheless, it is challenging to scale existing single-node memory protection engines to multiple nodes because of the untrusted network (or interconnect). First, hardware memory protection engines depend on essential metadata which is sealed within the CPU, e.g., the root of the integrity tree and one-time-pad are sealed in the CPU for memory encryption and integrity checks. The metadata cannot be safely transferred in the untrusted network. Second, a man-in-the-middle can easily launch the replay (e.g., using stale packets) or re-order attacks (e.g., exchanging the order of packets) in the untrusted network. For example, although TDX [6] uses the MKTME to guarantee the confidentiality and integrity of physical memory

Method	Throughput	Connection
PCI-E 5.0 [10]	32GT/s	CPU-Device
UCI-E [13]	32GT/s	Chiplets
RDMA [7]	400Gb/s	Remote Memory
NVLINK [8]	900GB/s	GPU

TABLE I: **Throughput of the different interconnection**.

(encrypted data with MAC), it discards freshness protection. As the network is unsafe, it is much easier for attackers to launch replay attacks in the network rather than the physical memory.

This paper proposes *Migratable Merkle Tree (MMT)*, which transfers both data and metadata to remote nodes without software involvement (e.g., re-encryption). A key insight of MMT is that (single-node) hardware memory protection already provides confidentiality, integrity and freshness guarantees for untrusted DRAM that can be reused for protection in the untrusted network. To this end, we first extend the single integrity tree to an *integrity forest* spanning multiple nodes and break the restriction of CPU-bound encryption metadata. Besides, we design a new protocol: *MMT closure delegation*, to protect the data transferred in the untrusted network. It can safely transfer integrity subtree root, nodes and data to remote nodes, which defend against revealing secrets or replay attacks. Finally, we implement a tiny-and-trusted module to manage enclaves and trusted hardware modules. It hides details of the hardware implementation and is responsible for the connection between local and remote enclaves.

We have implemented a prototype of the MMT system in the Gem5 [20] as well as all the software components. We extend the memory controller in the Gem5 to support MMT controller and add a new device: PCI-connector, which can connect two nodes using an RDMA-liked mechanism and trigger the MMT closure delegation. We also implement a secure monitor based on the Penglai enclave [25], which configures the secure hardware modules and manages local/remote enclaves. The evaluation result shows that MMT can speed up the performance of data transfer up to 169x compared with a CPU-only secure channel, and achieve 13x speedup compared with the AES accelerator. As for real-world secure distributed applications, MMT can gain 12%∼58% improvement on end-to-end performance of MapReduce in different workloads, and 35% improvement for PageRank using the GAS model [29].

II. Background and Motivation

A. Hardware-based Memory Protection

As shown in Figure 1, the modern memory protection engine uses a counter-based integrity tree [28], [44], [45], [49], [51], [56] to provide the integrity/freshness protection, and leverages counter-mode encryption [38], [45], [49]–[51], [53], [54] to guarantee the confidentiality of secure memory. Meanwhile, the memory protection engine also reserves a crypto-key in the secure storage which cannot be accessed outside the CPU (e.g., efuse).

Some extra metadata is required for integrity, encryption and freshness checking, such as integrity tree nodes, addresses and MACs. The modern memory protection engine adopts counter-mode encryption to guarantee confidentiality. As shown in Figure 1(b), the memory engine first leverages the memory address and counter to generate a one-time-pad (OTP). Later, it will use this OTP to perform the XOR operation on the plaintext to generate the ciphertext. As for integrity protection, the memory engine constructs an integrity tree covering all secure memory. Each integrity tree node contains multiple counters as well as the corresponding hash value. The hash value is calculated with the counter in the parent node and all counters in the current node (xoring the OTP and a Galois Field (GF) dot product result). The counter is increased for each write request. Therefore, if a parent node is protected, attackers cannot tamper with its child nodes or fabricate a valid hash value, and recursively, the data memory is also protected. In this procedure, only the XOR operation is in the critical path, and other operations like AES calculation and dot product can be overlapped with DRAM fetching.

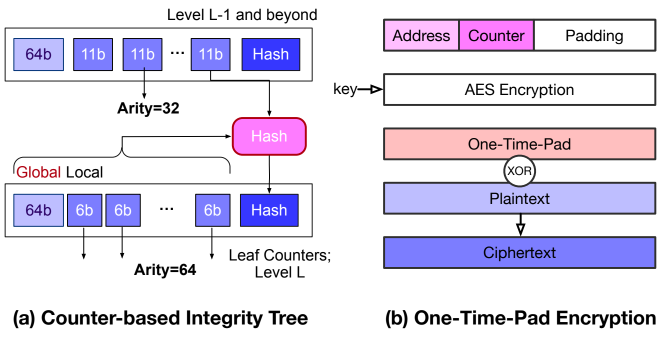


Fig. 1: **Hardware-based integrity/encryption protection for secure memory**.

B. Demand of Distributed Confidential Computing

Security guarantee for both code and data. To realize the distributed confidential computation, we need to protect the confidentiality and integrity for both code and data. Some state-of-the-arts [34], [36], [47], [59], [63], [64] choose the hardware enclave to protect the distributed computation in the cloud. Enclave can protect the confidentiality of code and data when executing in an isolated environment, but cannot guarantee the integrity of them. Therefore, users need to verify the integrity of the input data and attest the measurement of the executive code. Although using the hardware enclave has a better performance compared with the fully cryptographic-based methods, the security check for the input data brings a high overhead in the distributed computation.

Large-scale data transfer. Distributed computation is the basis of large data processing. During the computing, each node will communicate with others and transfer a large number of messages. Communication overhead is one of the key metrics for distributed computation. The traditional distributed computations may use the global-sharable files/memory pools to transfer the messages. MapReduce [23] stores the intermediate key-value result in the distributed file system, and one MapReduce task will transfer over 758TB [27] intermediate

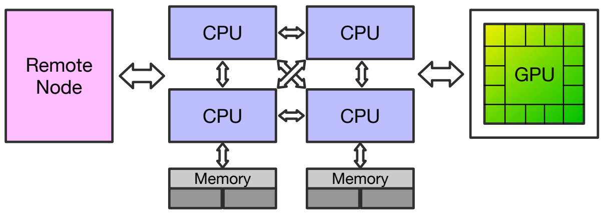


Fig. 2: **Interconnection between heterogeneous nodes**.

results. Graph computing [30], [40], [41] leverages the fast network connection to collect the messages from the neighbors (e.g., GNN [27] needs to transfer 72GB message in one epoch). Meanwhile, with the evolution of the fast interconnection mechanisms like RDMA, the distributed computation can further reduce the communication overhead and improve the end-to-end performance.

C. Inefficiency of Secure Channel over Untrusted Interconnection

Limited throughput over secure channel mechanism. Figure 2 shows the interconnection between heterogeneous nodes. Latest interconnections achieve over 400Gbps bandwidth and reduce the round trip latency to microseconds, as shown in Table I. However, these interconnections face a serious challenge: no protection for data transferring. Hence, users must leverage the cryptographic-based operation (i.e., secure channel) to protect the data in the untrusted network. Secure channel [12] can resist overhearing and tampering during the data transmission. It first executes the key exchange protocol (e.g., Diffie-Hellman [5]) to generate a key only known to the sender and receiver. Then, the sender will use the AEAD algorithm [1] (e.g., AES-GCM) to protect the associated data, and the receiver can check the integrity of both the encrypted and unencrypted messages. However, the throughput of AEAD algorithm is much smaller than interconnection mechanisms, which reduces the performance by orders of magnitude [42], [68], [69].

Establish the secure channel between enclaves. As the enclave memory is encrypted, the NIC driver cannot read/write the enclave memory directly. The state-of-the-art enclave design [3], [6], [14], [17], [22] uses a shared and untrusted memory buffer to copy I/O messages out of the enclave memory. Then, the NIC driver can read content in the shared buffer and send it to the remote enclave. As the network is untrusted, the enclave needs to encrypt and authenticate the message by itself. Hence, compared with the communication channel without any security guarantees, the secure channel used between enclaves needs two additional memory copies, one encryption and one decryption, which are considered to be time-consuming.

III. Design Overview

A. Design Goals

Performance. Our system should eliminate the overhead of re-encryption when transferring the secure memory to the remote nodes and saturate the maximum throughput of the interconnection.

Security. Our system should achieve the same security guarantee as the secure channel mechanisms (i.e., confidentiality, integrity and freshness of transferred data).

Compatibility. Our system should hide hardware details for user applications and inherit the programming paradigm of the distributed confidential computation, e.g., message passing.

B. Threat model

The TCB of our system only contains the CPU and most privileged software (e.g., EL3 monitor in Arm). Other hardware (e.g., PCI-e device, NIC, memory) and software (e.g., host OS in REE) are untrusted and can be comprised by attackers.

Physical attacks. We only trust on-chip hardware modules (e.g., CPU, memory controller), and any other off-chip hardware components and devices are untrusted, including the interconnection interface, memory and network. An attacker may issue off-chip physical attacks, such as spying, slicing and replay attacks by plugging a malicious micro-controller in the bus or network.

Privileged software attacks. We adopt a similar software threat model as TrustZone/CCA/Keystone, which is widely used in confidential computing. An attack may trigger a software attack from the REE, such as compromising the host kernel or obtaining full control of user applications. However, the firmware running in the most privileged mode (e.g., EL3 in Arm, M mode in RISC-V) cannot be comprised. In addition, TEEOS is trusted but optional in our design.

Our system does not consider the side-channel attacks [39], [60], [65], [66] and DoS attacks.

C. Challenges

Unlike the non-secure memory, the secure memory cannot be sent to others directly, as remote nodes cannot decrypt and authenticate the transferred encrypted memory. What's more, attackers can intercept packets in the network, exchange the order of packets or re-use a stale one to launch replay or re-order attacks. In summary, MMT needs to address the following challenges:

- **C1: How to decouple the memory protection from the CPU-bound secret in a single node?** Some security assumptions are held in a single node but fail in multiple nodes. For example, in a single node, a unique physical address can be used as the one-time-pad (OTP) in the memory encryption, and the encryption key is sealed in the hardware, which cannot be transferred among multiple nodes.
- **C2: How to securely transfer the secure memory to the remote node without re-encryption?** If we just send the secure memory to the remote, the remote memory engine cannot decrypt and authenticate the transferred memory. What's more, as the connection is unsafe, we need to protect the secret and defend against the replay attacks during data transferring.
- **C3: How to hide the hardware details and provide the suitable primitive for distributed computation?** The

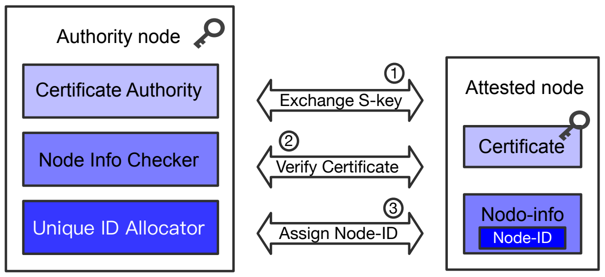


Fig. 3: **Global attestation for MMT nodes**. There are three steps for global attestation: (1) Discuss a session key; (2) Verify the hardware certificate; (3) Assign a node id that is used in the integrity check.

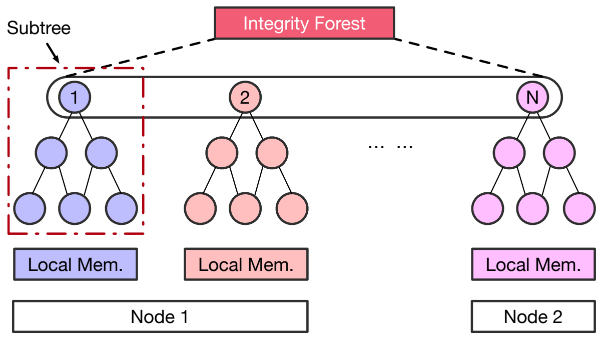


Fig. 4: **Integrity forest with multiple subtrees**. The subtree is the smallest unit in the integrity forest. Different subtrees can be located in the same or different nodes.

new hardware extension cannot complicate the implementation of user applications or break the paradigm in distributed confidential computation, such as using message passing for communication.

In this paper, we propose Migratable Merkle Tree (MMT), which can securely transfer the encrypted memory to remote nodes without software re-encryption. First, We propose a new abstraction: integrity forest which can span among multiple nodes. Meanwhile, a global attestation mechanism is adopted to verify each node that wants to join in the distributed confidential computation (in §IV-A). Second, we design a new transferred unit: *MMT closure* which integrates both data and metadata. A remote node can decrypt and authenticate the transfer memory according to the MMT closure. To be immune to the replay/re-order/revealing attacks in the untrusted network, we also design a protocol: *MMT closure delegation* to securely transfer the MMT closure to remote nodes (in §IV-B2). Third, we design a trusted-and-tiny module: MMT monitor in the most privileged mode to organize all secure hardware, and establish the connection between remote nodes. For scalability, we do not change the primitive of communication in distributed computation: message passing (in §IV-C).

IV. Detailed Design

We propose a new memory protection abstraction: Migratable Merkle Tree (MMT), to protect the distributed physical memory and messages in the untrusted network. To achieve it, we first set up an integrity forest among multiple nodes, and then extend each Merkle subtree to the MMT scheme.

A. Multiple-Nodes Integrity Tree

First, we establish mutual trust among multiple nodes. A global authority node will verify each participant node that wants to join in the distributed computing. Second, an integrity forest is constructed to protect the distributed memory among different nodes.

1) Global Attestation: As shown in Figure 3, when an MMT system boots, it will first attest itself to the global authority node (called global attestation). The global attestation includes three phases. First, an attested node generates a key agreement message to an authority node (as an attestation server) and discusses a session key (e.g., Diffie Hellman), which protects the sensitive data in the untrusted network. Second, the attested node sends a manufacturer certificate (sealed with a machine key) to the authority node. The authority node verifies the certificate with the manufacturer's public key, and responds a certificate authority (CA) report to the attested node. Last, the attested node sends node-related messages to the authority node. Node-related messages contain the measurement of the software module, node meta-info, etc. After receiving these messages, the authority node checks the software measurement and responds a global-unique node id to the attested node. This node id is essential to generate an integrity forest (see §IV-A2).

2) Integrity Forest: However, only global attestation is not enough. A modern memory encryption engine leverages an integrity tree to guarantee confidentiality, integrity and anti-replay attacks for secure memory. Hence, to realize distributed secure memory, we propose the integrity forest, which contains subtrees from multiple nodes, as shown in Figure 4.

Hardware-supported memory protection has two significant factors in cryptographic computing: memory address and counter. These two factors guarantee that any one-time-pad can only be used once. When considering the distributed memory across multiple nodes, however, the physical memory address will be duplicated, which violates the security assumption (an OTP can only be used once). To solve it, we use a global-unique address to replace the real physical address in memory protection. A global-unique address comprises two parts: the global-unique node id and a monotonic number. A global-unique node id is received from an authority node during the global attestation phase, and the monotonic number is generated by the hardware for each physical address. Hence, we do not need to synchronize the address info during the runtime. What's more, this global-unique address is only used by hardware during the integrity check and is transparent to software. Developers do not need to change the programming model of the application or sacrifice the degree of distributedness. All the integrity subtrees span among multi-nodes composing a large integrity forest, which protects the distributed secure memory.

Summary: To expand integrity trees to multiple nodes, we propose two techniques: global attestation and integrity forest.

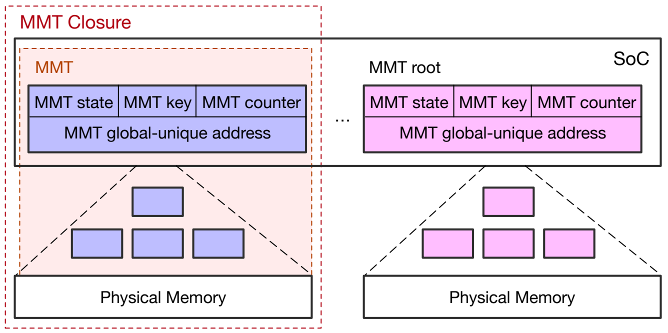


Fig. 5: **Migratable Merkle Tree and MMT Closure**. The MMT root is extended with the state, key, counter and global-unique address.

Global attestation guarantees that all nodes are verified and receive a global-unique identification. After global attestation, an attested node can send its attestation report to others and establishes mutual trust with remote nodes. The integrity forest contains subtrees among multiple nodes and guarantees the security assumption for the distributed secure memory.

B. MMT Scheme

MMT is a subtree in the integrity forest, which can be migrated to multi-nodes in the untrusted network.

1) MMT root: To migrate a Merkle tree, MMT extends the tree root with the extra metadata, as shown in Figure 5. There are four states for MMT: *valid*, *invalid*, *sending/read-only* and *waiting*; *Valid* state means the MMT is active and performs the security check for the memory access. *Invalid* state means the MMT is un-allocated or reclaimed, and the memory is regarded as non-secure memory. *Sending* and *waiting* states are used for transferring the secure memory. If the MMT state is *sending*, the content in this memory range cannot be modified (read-only); Meanwhile, the *waiting* state works together with the *sending* state, which waits for the remote transferred memory.

In addition to the MMT state, the MMT root also extends with key and global-unique address. Unlike the traditional crypto-key sealed in the hardware, MMT key is a user-defined secret used in memory encryption and authentication procedure. If two applications agree with the same MMT key, they can both decrypt and authenticate the same secure memory. Here, we decouple the crypto-key from the trusted hardware to the user space (similar to the TLS handshake), as users can flexibly trust the different nodes in a distributed system. The global-unique address is initialized after the global attestation, and will be re-assigned if the MMT state is changed to *Valid*. Notably, the value of the global-unique address is transparent to software, so the system still manages the memory using the physical address. Besides the key and address, the MMT root also contains a root counter to guarantee its freshness. Users can set/clear the initialized root counter when changing the MMT state to *Valid*/*Invalid*.

2) MMT closure delegation: MMT closure contains all data and metadata (i.e., tree nodes, root and data MACs) used in decryption and authentication procedure. Here, we propose a new data transfer primitive: MMT closure delegation.

Assume there are two nodes: receiver and sender. In the traditional method, the sender and receiver need to set up a secure channel, and later the sender can send the secret data to the receiver through this secure channel. However, using the secure channel will bring an expensive overhead including re-encryption and extra memory copy. In contrast, the MMT closure delegation transfers both secure memory and corresponding metadata to the remote directly. However, if we do not design this procedure carefully, there will be some security issues, such as tampering the MMT root and replay attacks.

As illustrated in Figure 6, we securely design the protocol of MMT closure delegation to guarantee the security assumption in the untrusted network.

1. Connect to the remote node: First, the sender registers a connection handler (similar to QP in RDMA connection) and sends an MMT key exchange request to the receiver. After agreeing on the same MMT key, the sender/receiver acquires a secure buffer (find a free memory region and change the MMT state to *valid* with the given root counter) and sets the same MMT key. Finally, the receiver responds to the sender with the buffer address and size.

2. Waiting phase: After the connection is established, the receiver will set the MMT state of the receiving buffer to *waiting*; and the sender will set the MMT state of the transferred buffer to *sending* (this memory region is read-only until the whole protocol is finished). After that, the receiver is waiting for the transferred MMT closure.

3. Securely Delegate the MMT Closure: To achieve it, we need to guarantee the security assumptions (i.e., confidentiality, integrity and freshness) during the MMT closure delegation. There are two facets in the MMT closure delegation: (1) how to transfer the MMT root and tree nodes; (2) how to transfer the data memory. Similar to the memory protection in a single node, we should first guarantee that the MMT root cannot be tampered by the malicious. Unfortunately, the network is untrusted during the MMT delegation, so an attacker can easily fabricate a malicious MMT root value which violates the security assumption. To protect the MMT root in the untrusted network, the sender uses the MMT key to seal the root value (encryption with a MAC value), and a receiver can decrypt and authenticate this MMT root with the same local MMT key. Hence, attackers cannot tamper with the MMT root during the delegation. Meanwhile, there is no need to encrypt intermediate tree nodes, as they are stored in memory as plaintext. As for the second facet, the confidentiality and integrity of the data memory are guaranteed by the corresponding MMT. Hence, the data memory can be transferred to the remote without any additional cryptographic-based protection.

4. Ack: When the receiver accepts all transferred secure memory and MMT nodes, it changes the MMT state to *valid/read-only* and sends an ack message to the sender's node. The sender will invalidate the original transferred MMT if it is needed (see more details in §V-B2). After that, the MMT closure delegation is done.

Defend Against Replay Attacks: In addition to confiden-

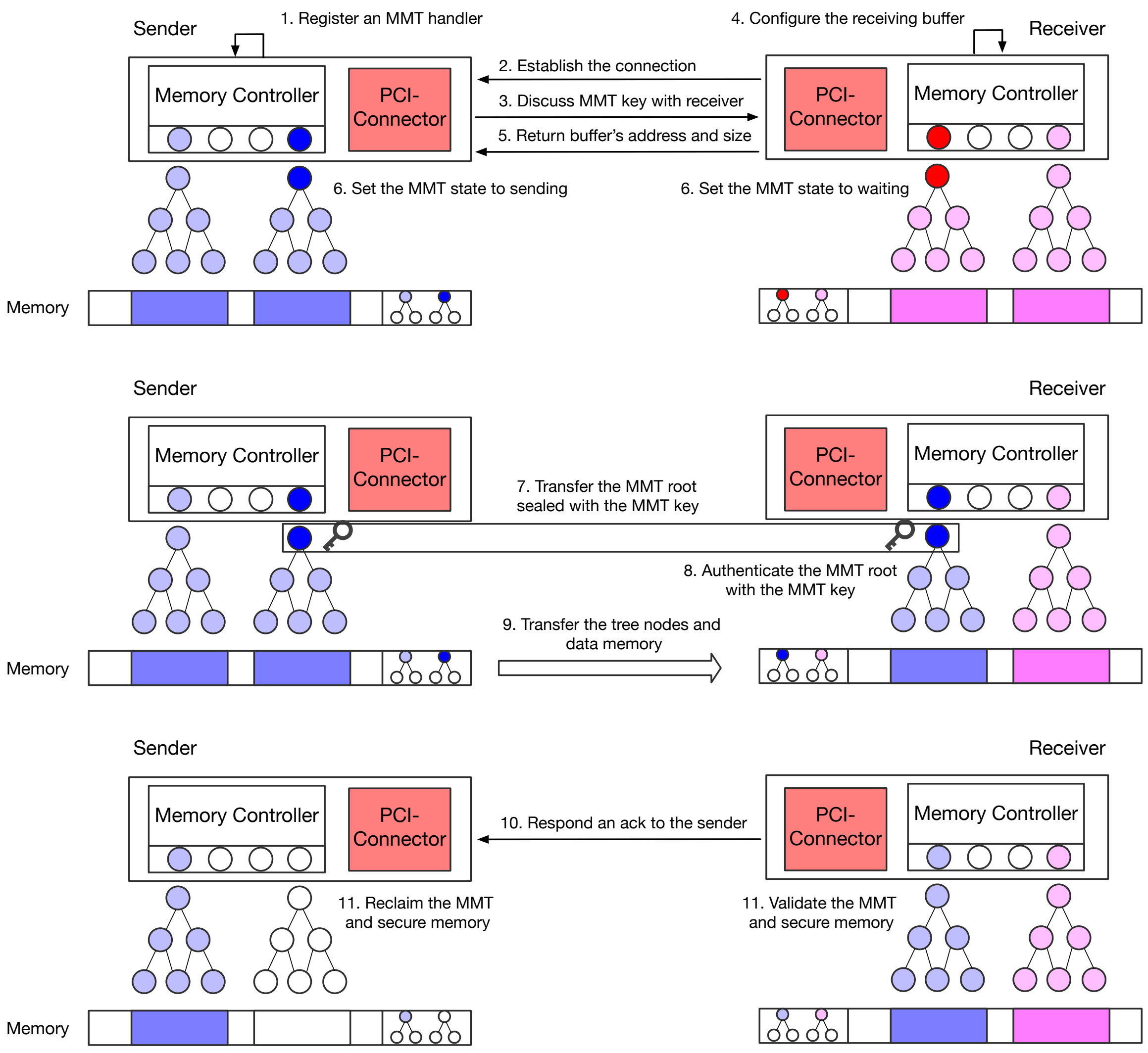

Sender
1. Register an MMT handler
4. Configure the receiving buffer
Receiver
Memory Controller
PCI-Connector
2. Establish the connection
3. Discuss MMT key with receiver
5. Return buffer's address and size
6. Set the MMT state to sending
6. Set the MMT state to waiting
Memory
7. Transfer the MMT root sealed with the MMT key
8. Authenticate the MMT root with the MMT key
9. Transfer the tree nodes and data memory
10. Respond an ack to the sender
11. Reclaim the MMT and secure memory
11. Validate the MMT and secure memory

Fig. 6: **MMT Closure Delegation**. The protocol of MMT closure delegation achieves the highest security assumption (confidentiality, integrity and freshness) for the transferred data.

tiality and integrity protection, the MMT closure delegation also defends against replay and re-order attacks. Although the Merkle tree can protect the secure memory from replay attacks in a single node, it does not consider the scenario of memory transferring in the network. In the untrusted network, attackers can easily (1) re-use a stale MMT closure with the same MMT key to the destination node; (2) exchange the order of MMT closure. MMT closure delegation leverages the counter-based freshness check and monotonic address ordering to defend against replay and re-order attacks. First, when a target node receives a transferred MMT closure, it will check the counter in the MMT closure with the local counter and reject any incoming MMT closure with less or the same counter value. The counter will be increased by hardware for each write request to guarantee any changes will cause a new version number. In addition, a user can initialize the root counter with a given value when the MMT state is changed to *valid*. According to these, we can ensure that the counter value in the sender is always larger than that in the receiver and is always increased during the delegation. Second, the target node will check the monotony of the global-unique address in two adjacent MMTs. It guarantees that the address in the MMT root of the latter is larger than the former. As the counter and address are sealed (using the MMT key to encrypt and authenticate the MMT root), attackers cannot tamper with these values in the network. Hence, the MMT closure delegation is immune to both replay and re-order attacks.

Summary: We design an MMT scheme to securely transfer the memory among multiple nodes without re-encryption. MMT closure is the smallest unit which can migrate in the untrusted network. When a remote node receives the MMT closure, it can decrypt and authenticate the secret data by itself. To guarantee the security assumptions for data transferring, we design a security protocol of MMT closure delegation to protect memory from spying, tampering and replay attacks. In summary, we reduce the overhead of transferring secure memory without re-encryption and extra memory copy, and achieve the same security assumption as the secure channel.

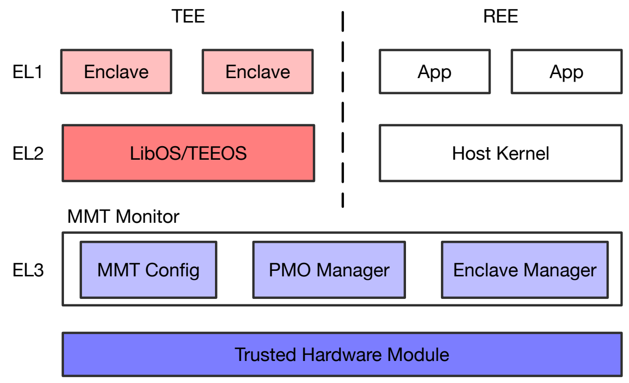


Fig. 7: **Software architecture for the MMT system**. We divide all hardware and software resources into two worlds: REE and TEE. MMT monitor runs in the most privileged mode.

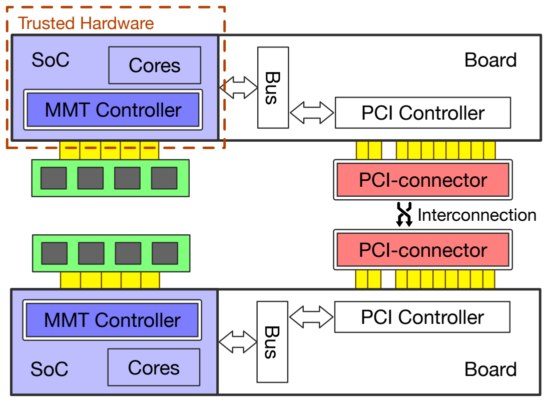


Fig. 8: **MMT hardware architecture.** Only in-SoC modules like the CPU and memory controller are in TCB, and other parts like pci-connector and DRAM remain untrusted.

C. MMT Monitor

Besides the hardware extension of the integrity forest and the MMT scheme, we also design a trusted-and-tiny trusted module to manage the MMT engine and perform the remote attestation for enclaves.

Figure 7 shows the software architecture of the MMT system. We introduce a trusted software module: MMT monitor, to manage enclaves and MMT-related operations in the most privileged mode (e.g., EL3 in Arm). MMT monitor uses the capability to organize the physical memory and hides the hardware details. It is also responsible for the remote attestation and setting up connection between local and remote enclaves.

MMT monitor has two key components: enclave manager and physical memory object (PMO) manager. The first component organizes the lifecycle of enclaves, attests to remote clients and establishes the connection. The enclave manager maintains a map between an enclave and its metadata (e.g., capability and attestation report, etc.). To establish the connection between the local and remote enclave, the local monitor can launch a connecting requests, and the remote monitor acquires for the target enclave, verifies the attestation report and finally establishes the connection. After that, the enclave manager can set up three kinds of channels between local and remote enclaves: non-secure, secure and MMT closure delegation. The second component is responsible for configuring the MMT state and checking the ownership of physical memory object (PMO). Physical memory object contains two parts: the secure memory and the corresponding MMT. As MMT configuration is closely related to the security of the whole system, we carefully design a capability mechanism to organize the MMT and secure memory. Only the owner of the PMO can configure its MMT root, and each PMO can have only one owner. Notably, the ownership can be revoked if the secure memory is assigned to another enclave or transferred to a remote one.

Summary: We design a trusted module: MMT monitor, to manage the lifecycle of enclaves, establish the connection and configure the MMT hardware. There are two major components in the MMT monitor: enclave manager and PMO manager. In addition, MMT monitor is the only privileged module that can configure the trusted hardware module (e.g., MMT controller).

V. Implementation

A. Hardware Changes for MMT

Figure 8 shows the hardware architecture of an MMT system.

1) PCI-connector: Modern smartNICs [15] expose RDMA interfaces to the host. Similarly, we implement a specific device for interconnection: pci-connector. To further control the data transferring latency in the network, we set delay cycles to emulate the different connection latency. If a local node wants to transfer its memory to a remote node, it will trigger a DMA request to copy the memory from the source to the pci-connector's buffer, and then, the remote pci-connector will initiate a DMA request to copy the memory from the device buffer to the destination in the remote node. In summary, to transfer memory from local to remote, we need to initiate two DMA requests and set the DMA delay cycles to emulate the connection latency.

There are two types of data transferring: data-only or MMT closure delegation. The data-only transferring is similar to RDMA operations. In our implementation, we reserve tx_buffer and rx_buffer, as the default device buffer. As for the MMT closure delegation, it needs to transfer both secure memory and MMT tree nodes to a remote node. First, pci-connector needs to set the source and destination of the remote memory address. Second, before transferring data memory to the remote node, it will initiate a request to transfer the MMT metadata (see §V-A2 for details). After receiving all MMT closure memory, the pci-connector will trigger a finished/ack command to local/remote memory controller to update MMT states (*valid* in receiver and *invalid* in sender). The MMT closure delegation is slightly slower than the data-only transferring, as it needs to transfer the extra metadata (MMT tree nodes). Meanwhile, the MMT root validation can be overlapped with the MMT delegation, which is not in the critical path.

2) MMT controller: As shown in Figure 9, we extend the memory controller to support MMT-related manipulations. The MMT controller contains two new components: integrity tree

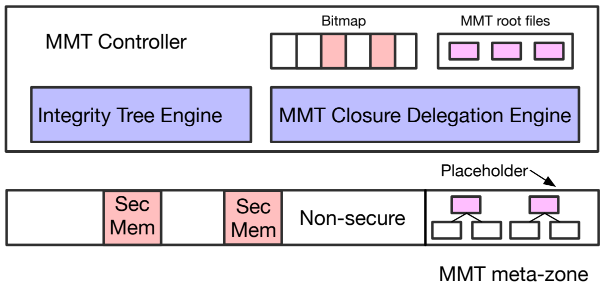


Fig. 9: **Hardware extensions for MMT controller.**

engine and MMT closure delegation engine. The integrity tree engine is designed to protect the physical memory from snooping, tampering and replay attacks. The MMT controller divides the whole physical memory into three types: the normal memory, the secure memory and the MMT meta-zone. When the MMT controller receives a memory request, it will first check a bitmap which records the type of physical memory. The granularity of non-secure and secure memory is equivalent to the protected size of the MMT (in our implementation is 2M). Meanwhile, the MMT meta-zone is a separate memory range which can only be accessed by MMT monitor. MMT meta-zone stores MMT metadata such as intermediate tree nodes and MMT root (only a placeholder, any read/write request will redirect to the real root in SoC), and each MMT metadata has a fixed mapping with its data memory. Each MMT root contains an extra 24 bytes metadata: 128-bit key, 2-bit state, 58-bit global-unique address and 4-bit is reserved.

Similar to the modern memory protection engine, MMT controller also adopts the counter-based integrity tree to protect the physical memory, and stores the integrity tree roots in the SoC. Figure 1 (a) shows the integrity tree structure for MMT. In each tree node, we adopt the global-local counter layout with hash value. Each counter comprises the global-shared part together with the local-individual part. When the global counter is exhausted, we need to update counters in all child nodes and perform the re-hash procedures. In our implementation, the leaf node has 64 local counters (64 arity), and other tree nodes have 32 or 16 local counters. Besides, each MMT is 3 level height.

The first component is integrity tree engine. Once the MMT controller receives a read request, it will traverse the integrity tree to fetch all intermediate nodes from the MMT meta-zone. After receiving all tree nodes and data packets, the tree engine checks hashes stored in tree nodes recursively up to the MMT root; As for a write request, the tree engine first initiates several read requests for intermediate tree nodes and checks data integrity before writing. After the validation, it increases node counters and updates the hash value in tree nodes. A write request will return immediately when it has been pushed into a write queue, as write-back operations are triggered periodically if the write queue is not empty.

The second component is MMT closure delegation engine. As mentioned above, the physical memory is divided into three types: the non-secure memory, secure memory and MMT meta-zone. For an MMT delegation request, the MMT closure delegation engine leverages the MMT key to seal the MMT root value (i.e., encrypted and authenticated). After that, the MMT root can be securely transferred to the destination through the pci-connector. As for the remote side, the MMT closure delegation engine will first unseal the transferred MMT root with the local MMT key. If the authentication fails, the delegation request will be rejected; Otherwise, the MMT engine will receive both MMT nodes and data memory. After receiving all data and MMT nodes, the MMT engine will trigger a *finished* command and set the MMT state from *waiting* to *valid*; Similarly, the sender node will receive an *ack* command, and invalidate the MMT in the sender side (change state from *sending* to *invalid*). After that, the ownership of MMT and data memory are transferred from the sender to receiver (see §V-B2 for more details).

B. Software Implementation

1) MMT Monitor and TEEOS: We implement the MMT monitor in Arm EL3 based on Penglai enclave [25] and adopt Chcore [31] — a microkernel-based kernel as the trusted OS in TEE. ChCore uses the capability to manage all physical resources (e.g., memory allocation, time slice) and software-defined objects (e.g., notification, file). MMT monitor is the most privileged module which manages enclaves, MMT state and trusted hardware modules. As mentioned in §IV-C, there are two key components in the MMT monitor: enclave manager and PMO manager. The MMT monitor uses enclave objects to manage the lifecycle of local enclaves and maintain the connection with remote enclaves. As for the PMO manager, MMT monitor extends the secure physical memory object (sPMO) with the extra MMT-related metadata (e.g., MMT state and key), and leverages the capability to organize its ownership. Meanwhile, to minimize the TCB of MMT monitor, we offload the allocation of sPMO into TEEOS. The sPMO can only be allocated from a pinned memory pool (like RDMA's buffer). If an enclave obtains the capability of a secure PMO, the TEEOS maps this physical memory range into its virtual memory space.

2) Enclave Application: There are two modes for user-level enclave applications: ownership-transfer mode or ownership-copy mode. In the ownership-transfer mode, the owner of the MMT can transfer the corresponding secure memory and MMT nodes to others. Once the remote node accepts the transferred MMT as well as the secure memory, the ownership of the MMT is transferred from the sender to the receiver. The sender needs to re-assign a new MMT if it wants to reuse the transferred memory region. This programming model is suitable for the DAG (Directed Acyclic Graph) scenario. As for ownership-copy mode, the owner of the MMT just sends a read-only copy of the secure memory and MMT nodes to the receiver. If the receiver wants to modify the content in the transferred memory, it needs to copy the content into another space. However, as the ownership of the transferred memory is still held by the sender, it can directly modify the content in the sender's side. This programming model is suitable for the send/receive protocol. In summary, we must ensure that there is only one writable copy of secure memory in the whole distributed system.

VI. Evaluation

A. Methodology

Processor Configuration	
CPU	8 Out-of-order cores @ 2.0GHz
I/D TLB	256 entries
L1 I/D Cache	32KB, 64B line, 2/4 Associativity
L2 Cache	1MB, 64B line, 15 Associativity
L2 Access Latency	data/tag (13 cycles), response (5 cycles)
MMT Controller Configuration	
Memory Type	LPDDR3_1600_1*32, 2GB
Write/Read queue	128/64 entries
MMT Roots in SoC	8KB
MMT Cache	32KB
MMT levels	3 levels
Encryption Latency	40 processor cycles

TABLE II: Gem5 configurations for processor and MMT controller.

We implement an MMT hardware based on Gem5 [20]: a full system, cycle-accurate simulator. Table II shows the Gem5 configuration for evaluation. We implement the full hardware extension (e.g., MMT controller, PCI-connector) in the GEM5. In addition, to compare the performance of MMT closure delegation with state-of-the-art secure channel mechanism using the AES accelerator (e.g., AES-NI instruction in the current Intel CPU), we implement a software MMT controller in a real machine and simulate the MMT closure delegation based on the RDMA connection. In short, we launch an additional one-sided RDMA operation to transfer the simulative MMT metadata to the remote node, and perform the software check of the MMT root. Table III shows the processor and MMT configurations for real Intel machines. As for the secure channel mechanism, we re-use the cryptographic library provided by OpenSSL [9], and establish the secure channel through the RDMA connection.

Processor Configuration	
CPU	Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
Cores	12 cores, 24 threads
Cache size	30720 KB
MMT Controller Configuration	
Memory Type	LPDDR4_2400 128GB
Secure Memory	16GB
Simulative MMT Roots	64KB
Simulative MMT levels	3 levels

TABLE III: Simulative MMT configurations on the Intel machine.

We first evaluate the performance of transferring secure memory to a remote node with two methods: MMT closure delegation and a software-based secure channel (e.g., using AES-GCM algorithm). To comprehensively evaluate the MMT performance and tradeoff in different settings, we test MMT on SPECCPU benchmark in different tree levels. We also set the various pci-connector delay cycles to simulate the different network connection latency. As for the real-world applications, we choose two well-known distributed tasks: MapReduce [23] and PageRank [21]. We implement a MapReduce framework and test the end-to-end latency with the different transferred memory sizes, workloads and settings. For the second application, we implement a Gather-Apply-Scatter (GAS) model [29] in our MMT system and adopt the PageRank algorithm in the apply phase.

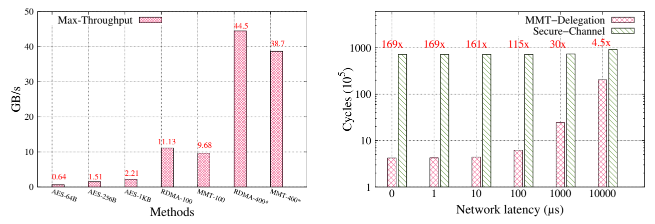
B. Microbenchmarks

MMT closure delegation. We evaluate the performance of MMT closure delegation with secure channel methods using CPU-only and AES-NI instructions. As for MMT closure delegation, an enclave needs to transfer the whole granularity of MMT closure to remote, even if the message size is much smaller (in our implementation, the default granularity of the MMT closure size is 2M). In contrast, the secure channel uses unsafe remote write interfaces without the memory size limitation. An enclave first needs to encrypt and copy the sensitive message to non-secure memory. Then, a remote write request is triggered to transfer this encrypted message to a remote receiving buffer. Finally, a remote enclave copies the encrypted message from the non-secure receiving buffer to its own secure memory and then decrypts it. In summary, there are two extra memory copies, one remote memory write and re-encryption in the secure channel method. Table IV:Gem5 shows the performance of MMT closure delegation and secure channel on MMT-enabled machines. In the Gem5 simulator, we only use CPU for cryptographic calculations. MMT closure delegation has a constant overhead when the transferred memory size is smaller than one MMT closure (2M). On the contrary, the secure channel does not have this limitation but needs four additional operations: memcpy, remote_w, encrypt and decrypt. Data encryption and decryption are the most costly operations in the secure channel mechanism, which both take up nearly 45% time for 2M message. Data copy between normal and secure memory takes up another 5% time, while the remote memory write only accounts for 0.5$\sim$4% of the transferring overhead. As for the end-to-end performance, MMT closure delegation gains the 169x speedup when transferring 2M secure memory. However, using the secure channel method has better performance in small message (e.g., memory size<8K). In the distributed computation scenarios, each node will transfer a large-scale memory to other nodes (see section §II-B), so the small size of transferred messages is rare in real-world workloads. Table IV:Intel compares the MMT performance with an enhanced baseline using AES accelerator. The current Intel CPU provides AES-NI instructions to accelerate the AES encryption. To simulate the MMT delegation performance, we issue two RDMA writes to transfer both data memory and MMT tree nodes. The evaluation result shows that even using AES-NI instructions, MMT delegation also gains 13x speedup compared with the secure channel mechanism.

Throughput. Figure 10 (a) shows the maximum throughput of AES-128-GCM, RDMA and MMT. We test the AES-GCM algorithm under different block sizes using AES-NI instruction. The evaluation result shows that the larger block size can gain a better performance (2.2GB/s in Intel E5-2650). To

Breakdown	Transferred Memory Size								
	Gem5 simlulator (10^3 cycles)						Intel E5-2650 AES-NI (ms)		
	2M	512K	128K	32K	8K	2K	32M	64M	128M
Memcpy*2	4288 (6%)	989 (5.6%)	211 (5%)	46.4 (3.9%)	6.26 (1.6%)	1.31 (0.7%)	8.84 (19.5%)	17.1 (19.6%)	34.0 (19.4%)
Remote_W	367 (0.5%)	102 (0.6%)	36 (0.8%)	15.9 (1.3%)	9.47 (2.4%)	7.69 (4%)	3.01 (6.7%)	6.02 (6.9%)	12.1 (6.9%)
Encrypt	34612 (48%)	8445 (47%)	2066 (47%)	530 (45%)	170.2 (43%)	77.4 (40%)	16.5 (36.5%)	31.8 (36.2%)	63.6 (36.2%)
Decrypt	32230 (45%)	8128 (46%)	2085 (47%)	580 (49%)	204.7 (52%)	104.6 (55%)	16.9 (37.3%)	32.7 (37.3%)	66.0 (37.5%)
Secure Channel	71498	17666	4409	1173	390.7	191.1	45.3	87.9	176
MMT Closure Delegation	422						3.47	6.92	13.9
Speedup	169.1x	41.77x	10.43x	2.77x	0.92x	0.45x	13.1x	12.7x	12.7x

TABLE IV: **Performance breakdown.** Compare the performance of MMT closure delegation and secure channel.



(a) Throughput. **(b) Network latency.**

Fig. 10: Evaluate the throughput and latency of MMT closure delegation.

evaluate the MMT throughput on a real machine, we trigger two RDMA transactions to send both data and metadata. In our test case, an r-NIC with 100Gbps bandwidth can achieve 11 GB/s throughputs, and the MMT delegation can gain 9.68GB/s bandwidth under the RDMA connection. If we escalate the r-NIC to 400Gbps [7], the maximum throughput of MMT delegation will increase accordingly. Therefore, even using the AES accelerator, the throughput of AES encryption is still an order of magnitude smaller than MMT delegation.

Network Latency. Figure 10 (b) demonstrates the relationship between the end-to-end latency of MMT closure delegation and network latency. We set the delay cycles in the pci-connector to emulate the different network latencies in Gem5. In the ideal situation (the network latency is zero), the MMT closure delegation can gain the 169x speedup for 2M transferred memory compared with the CPU-based secure channel. However, when network latency increases, it will narrow the improvement of end-to-end latency for the MMT closure delegation (only gain 4.5x speedup if the network latency is increased to ten milliseconds). Meanwhile, as for a real-world distributed computation, the major communication overhead comes from the data transfer, as each computing node needs to send large-scale messages, but the network latency is minor (microsecond using RDMA in the same rack).

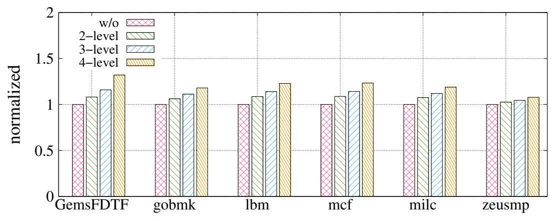


Fig. 11: **SPECCPU.** MMT overhead with different tree levels on SPECCPU benchmarks.

Tree level	Metrics		
	Root Size	MMT Size	Overhead
2-level	128K	128K	1.07
3-level	8K	2M	1.12
4-level	512B	32M	1.21

TABLE V: **Tree level.** Evaluate different MMT tree levels with multi-metrics.

Tree level. The MMT tree level is also a key parameter for MMT system, and we evaluate it with various metrics (e.g., performance, space overhead). Figure 11 shows the performance overhead of different MMT tree levels. We choose the SECCPU benchmark to evaluate the general performance of MMT. If we deepen the MMT level, the performance overhead will increase. What's worse, as the SoC space and tree node cache are limited, this overhead will rise rapidly when increasing the MMT level (one memory access will trigger extra tree node accesses, which occupy the read/write queue and tree node cache in the MMT controller). The evaluation result shows that if we adopt a 2-level MMT tree, the average overhead is only 1.07x, however, assuming the tree level is augmented to 4-level, the average overhead rises to 1.21x. Table V summarizes the effects of different MMT tree levels. A deeper MMT takes up less in-SoC storage. For example, a 4-level MMT only needs 512B extra space for MMT root in SoC. In contrast, a 2-level MMT occupies 128KB storage, which is orders of magnitude larger than a 4-level MMT. What's more, a higher MMT tree level also means a larger granularity of the transferred memory. A 4-level MMT closure contains 32MB contiguous secure memory, whereas this value is 128KB for a 2-level MMT. In summary, the higher MMT tree level saves the in-SoC storage, but increases the performance overhead and enlarges the granularity of the secure memory. In our evaluation, we choose a 3-level MMT as the default configuration.

C. Real-world Applications

1) Trusted MapReduce: MapReduce is a popular programming model in the distributed computation. Some state-of-the-arts [47], [59] propose trusted MapReduce schemes based on the hardware enclave like SGX. To protect the integrity of input/output for the mapper and reducer, these works add security guarantees such as read-write integrity check. VC3 [47] points out that the security guarantees will bring up to 63.4% overhead compared with the non-secure MapReduce tasks. We establish an in-memory MapReduce framework [18], [43], [57], [62], [67], which can transfer the intermediate result through the remote memory write interface (non-secure) or MMT

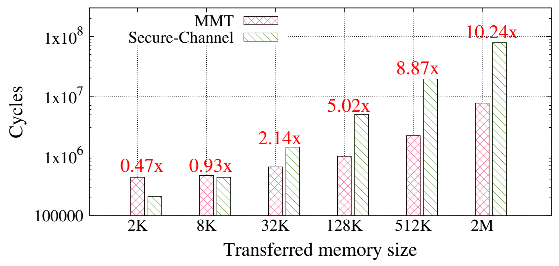
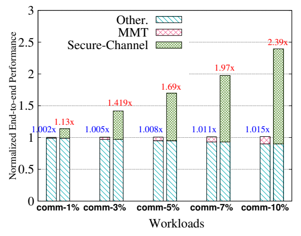
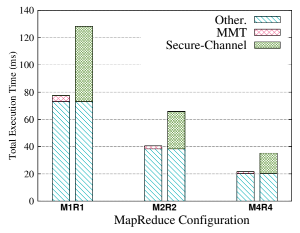


Fig. 12: Execute MapReduce with different sizes of the transferred memory.

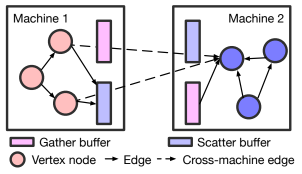


(a) End-to-end performance with different workloads.

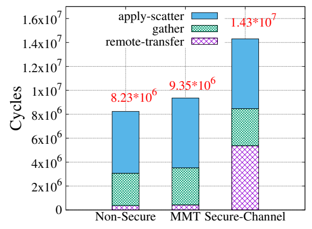


(b) Total execution time with different configurations.

Fig. 13: **End to end performance.** Comm-n% in figure (a) means that the communication cost takes up n% of total execution time in the baseline; MnRn in figure (b) means running n mappers and n reducers in 2n nodes, one worker per node.



(a) GAS model.



(b) PageRank result.

Fig. 14: Evaluate PageRank using the GAS model.

closure delegation. Figure 12 shows the end-to-end performance of WordCount tasks with different transferred memory sizes (MMT delegation v.s. secure channel in Gem5). The evaluation result shows that MMT delegation can gain up to 10x speedup compared with the secure channel (when transferred memory size is larger than 2M). To test the worst-case performance, we run the WordCount tasks with some extremely small workloads. As MMT delegation must transfer 2M memory region, using a secure channel has a better performance when the transferred memory size is less than 8K. Nevertheless, this case is almost negligible in the real-world distributed MapReduce tasks.

To further estimate the end-to-end performance of MMT, we run MapReduce jobs in the real Intel server and leverage the RDMA to transfer the intermediate results. The coordinator will connect a mapper with reducer using the RDMA queue pair (QP). There are three configurations: Baseline is running in the non-secure mode without any protection during the RDMA transaction; MMT uses MMT closure delegation to transfer the intermediate result; And secure-channel leverages AES-GCM algorithm to protect the confidentiality and integrity of the transferred data. As shown in Figure 13 (a), we evaluate the end-to-end performance of MapReduce in different workloads. Comm-n% represents that the communication cost takes up n% of total execution time in the baseline, and the y axis is the normalized performance. As for secure channel, AES-GCM encryption will augment the date transferring overhead. For example, in comm-5% workload, the communication only accounts for 5 percent of execution time. However, this percentage will rise to 74% in the secure channel mode and cause up to 69% overhead. MMT delegation has comparable performance to the baseline, as it does not need to re-encrypt transferred data. The end-to-end overhead is only 1.5% in the comm-10% workload, and gains 12%∼58% improvement compared with the secure channel mechanism.

We evaluate the scalability of MMT delegation by running MapReduce workers on the different nodes. As shown in Figure 13 (b), MnRn means running n mappers and n reducers in 2n nodes. Each mapper needs to set up connections with all reducers. The evaluation result shows that MMT delegation will not break the scalability of the MapReduce system, as MMT delegation leverages message passing-like interfaces for communication. Similar to the RDMA connection, MMT delegation will not introduce any coherence operations when transferring the intermediate results.

2) PageRank: PageRank [21] is a widely used graph computation algorithm. The recent study proposes a gather-apply-scatter (GAS) model [29] for the large-scale and distributed graph computation. One GAS iteration contains three phases: gather, apply and scatter. To migrate the distributed GAS model to the MMT system, we design a new phase called remote-transfer, which sends sensitive messages to a remote MMT machine. As shown in Figure 14 (a), we first set gather and scatter buffers in each MMT machine. As for a cross-machine edge, the source vertex node will copy messages to the scatter buffer, and trigger a remote memory write (non-secure) or MMT closure delegation to send messages to the gather buffer in the remote. In the gather phase, we can assure that all transferred messages are ready in the local machine.

Figure 14 (b) shows the evaluation result of the PageRank algorithm in the GAS model. There are three configurations: Non-secure, MMT and secure channel. Non-secure means we execute PageRank without any memory protection (disable the MMT engine); MMT/secure channel means using MMT closure delegation/secure channel to transfer remote messages in cross-machine edges. We initialize a graph with nearly 100,000 nodes and 60,000 cross-machine edges (similar to real-world graphs). The evaluation result shows that using MMT closure delegation, the remote-transfer phase only takes up 5% execution time for one iteration. As for the secure channel, nearly 37.5% of runtime cycles are spent in the remote-transfer phase. As for the end-to-end performance, the MMT closure delegation gains 35% improvement compared with the secure channel.

VII. Discussion

Fault tolerance. MMT adopts the same fault tolerance policy as the current RDMA-based distributed systems. First, the MMT delegation uses similar technology to the RDMA reliable connection (RC) mode, which guarantees that each packet will not be missing or reordered in the network. Second, the developer can adopt the primary-backup program mode in distributed computing.

Scalability. There are two programming models in distributed computing: distributed share memory (DSM) and message passing. DSM suffers from poor scalability due to memory coherence, write amplification, etc. MMT delegation chooses a more scalable primitive: message passing. Message passing does not have the scalability issue [30], [40], [41], [62] as it establishes a peer-to-peer connection and shifts the duty of synchronization to the developer. In addition, the MMT design is compatible with other scalable integrity tree schemes [25], [33], [46], [55]. The state-of-the-arts achieve to support up to 512GB of secure memory using a more compact tree structure, prefetching and mounting mechanisms.

Security. MMT provides the highest security protection for physical memory, which can defend against the malicious read or write, and even replay attacks. Some related works also claim to constitute a trusted environment for confidential computing, but compromise partial physical attacks. TDX [6] acknowledges that it does not defend against memory replay attacks. What's worse, the physical attacks become more severe in the untrusted network, as attackers can easily control the network infrastructure and inject malicious packages. Therefore, we cannot loosen the security assumption for MMT threat model.

VIII. Related Work

Hardware-based Memory Integrity Protection. Many prior works [22], [25], [28], [33], [44]–[46], [49], [50], [55], [56] also propose the memory integrity protection scheme and reduce the overhead of the integrity tree. Rogers et al. [45] propose an efficient data protection for the distributed shared memory. However, MMT has no assumption of the DSM system, and each node can have its own, non-sharable memory. What's more, MMT can transfer the secure memory directly without re-encryption. Penglai [25] designs a scalable memory protection scheme protecting 512GB secure memory in a single node. It dynamically mounts the subtree root into SoC to minimize the security check overhead. Synergy [50] stores data MACs in the place reserved for ECCs (e.g., ECC memory), which can reduce the memory overhead of the integrity tree. What's more, a dedicated cache can be applied in the memory protection controller to store the intermediate tree nodes and improve the performance of security checks. Morphable [46] dynamically adjusts the counter size in a tree node to reduce the overflow frequency. BMT [44] uses Merkle tree to protect the tree counter instead of the memory data, which reduces the overhead of the integrity checks. Vault [56] proposes a tree structure with the different sizes of the tree counters. ITESP [55] builds a separate integrity tree for each application to achieve a better cache locality. PCPT [33] leverages a Parallelized-Compressed-Prefetched-Tree to predict the memory fetch and reduce the integrity check overhead. BMF [26] aims to guarantee crash consistency on persistent memory, and to reduce the performance overhead of the BMT root updates. Although BMF and MMT both adopt the integrity forest schemes, BMF does not consider how to establish a distributed Merkle Forest among multiple nodes. In summary, these schemes only focus on the memory protection in the single node, and do not consider the secure data transferring between multiple nodes.

Hardware Enclave. Hardware-based enclaves have been widely adopted in the prevailing architecture, such as Intel SGX [22], TDX [6], Arm TrustZone [17], CCA [3], RISC-V Keystone [37], etc. Intel establishes a trusted environment for enclaves with the memory confidentiality, integrity and freshness protection. However, SGX [22] EPC memory is not available for device, and we cannot perform a DMA operation on this memory range. TDX [6] only considers the memory encryption and integrity protection, but does not defend against the replay attacks. CCA [3] only provides an encryption memory without integrity protection. TrustZone [17] and Keystone [37] do not consider the physical attacks for the memory system (no encryption, integrity protection). Graviton [61] proposes a GPU-enclave coordinating with the CPU enclave. To securely transfer the sensitive message between CPU and GPU, it leverages an AES-SHA3 algorithm to protect the transferred message. HIX [35] adopts the hardware memory protection engine to the GPU architecture and utilizes the memory access pattern to optimize the integrity tree structure. Elasticlave [16] only provides memory isolation using the PMP mechanism in the RISC-V. It does not consider any physical attacks on memory. However, MMT considers the memory encryption, integrity and freshness protection, and provides a mechanism to transfer the secure memory to others without violating the security assumptions.

IX. Conclusion

This paper presents *Migratable Merkle Tree* (MMT) to reduce the secure data transferring overhead in the confidential distributed computation. To achieve this, we design a secure protocol: *MMT closure delegation*, to securely transfer the MMT closure to others without re-encryption. The evaluation result shows that MMT can significantly reduce the transferring overhead compared with the traditional secure channel mechanism and promote the performance of the real-world distributed applications.

X. Acknowledgments

We sincerely thank anonymous reviewers for their insightful suggestions, and we also appreciate Dahu Feng and Yaodanjun Ren for their valuable advice to paper refinement. This work is supported in part by National Key Research and Development Program of China (No. 2020AAA0108500), China National NSF (No. 61972244, U19A2060, 61925206), and STCSM (No. 21511101502). Dong Du is the corresponding author.

References

[1] "Aead algorithm," https://en.wikipedia.org/wiki/Authenticated_encryption#Authenticated_encryption_with_associated_data_(AEAD), referenced April 2022.

[2] "Aes accelerators using fpga board," https://www.heliontech.com/aes_gcm.htm, referenced April 2022.

[3] "Arm confidential compute architecture," https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture, referenced April 2022.

[4] "Compute express link," https://en.wikipedia.org/wiki/Compute_Express_Link, referenced April 2022.

[5] "Diffie-hellman key exchange," https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange, referenced April 2022.

[6] "Intel trust domain extensions," https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html, referenced April 2022.

[7] "Nvidia gtc: New 400 gbps infiniband networking platform," https://www.embedded.com/nvidia-gtc-new-400-gbps-infiniband-networking-platform/, referenced April 2022.

[8] "Nvlink and nvswitch," https://www.nvidia.com/en-us/data-center/nvlink/, referenced April 2022.

[9] "Openssl," https://github.com/openssl/openssl, referenced April 2022.

[10] "Pci express 5.0," https://en.wikipedia.org/wiki/PCI_Express, referenced April 2022.

[11] "Pci express 6.0," https://pcisig.com/pci-express-6.0-specification, referenced April 2022.

[12] "Secure channel," https://en.wikipedia.org/wiki/Secure_channel, referenced April 2022.

[13] "Universal chiplet interconnect express (ucie): Building an open chiplet ecosystem," https://www.uciexpress.org/_files/ugd/0c1418_c5970a68ab214ffc97fab16d11581449.pdf, referenced April 2022.

[14] "Amd secure encrypted virtualization (sev) - amd," https://developer.amd.com/sev/, 2019.

[15] "Nvidia mellanox bluefield dpu," https://www.mellanox.com/products/bluefield-overview, 2021, referenced 2021.

[16] "Elasticlave: An efficient memory model for enclaves," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/yu-jason

[17] T. Alves, "Trustzone: Integrated hardware and software security," *White paper*, 2004.

[18] A. M. Aly, A. Sallam, B. M. Gnanasekaran, L.-V. Nguyen-Dinh, W. G. Aref, M. Ouzzani, and A. Ghafoor, "M3: Stream processing on main-memory mapreduce," in *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 2012, pp. 1253–1256.

[19] N. A. Anagnostopoulos, S. Katzenbeisser, J. Chandy, and F. Tehranipoor, "An overview of dram-based security primitives," *Cryptography*, vol. 2, no. 2, p. 7, 2018.

[20] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2024716.2024718

[21] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[22] V. Costan and S. Devadas, "Intel sgx explained. cryptology eprint archive," *Report 2016/086*, 2016.

[23] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," 2004.

[24] T. Dierks and E. Rescorla, "The transport layer security (tls) protocol version 1.2," 2008.

[25] E. Feng, X. Lu, D. Du, B. Yang, X. Jiang, Y. Xia, B. Zang, and H. Chen, "Scalable memory protection in the penglai enclave," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021, pp. 275–294.

[26] A. Freij, H. Zhou, and Y. Solihin, "Bonsai merkle forests: Efficiently achieving crash consistency in secure persistent memory," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1227–1240.

[27] S. Gandhi and A. P. Iyer, "P3: Distributed deep graph learning at scale," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021, pp. 551–568.

[28] B. Gassend, G. E. Suh, D. Clarke, M. Van Dijk, and S. Devadas, "Caches and hash trees for efficient memory integrity verification," in *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.* IEEE, 2003, pp. 295–306.

[29] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *10th USENIX symposium on operating systems design and implementation (OSDI 12)*, 2012, pp. 17–30.

[30] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework," in *11th USENIX symposium on operating systems design and implementation (OSDI 14)*, 2014, pp. 599–613.

[31] J. Gu, X. Wu, W. Li, N. Liu, Z. Mi, Y. Xia, and H. Chen, "Harmonizing performance and isolation in microkernels with efficient intra-kernel isolation and communication," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 401–417.

[32] S. Gueron, "A memory encryption engine suitable for general purpose processors," *Cryptology ePrint Archive*, 2016.

[33] Y. Guo, A. Zigerelli, Y. Cheng, Y. Zhang, and J. Yang, "Performance-enhanced integrity verification for large memories," in *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 2021, pp. 50–62.

[34] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, "Ryoan: A distributed sandbox for untrusted computation on secret data," *ACM Transactions on Computer Systems (TOCS)*, vol. 35, no. 4, pp. 1–32, 2018.

[35] I. Jang, A. Tang, T. Kim, S. Sethumadhavan, and J. Huh, "Heterogeneous isolated execution for commodity gpus," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 455–468.

[36] R. Ladjel, N. Anciaux, P. Pucheral, and G. Scerri, "Trustworthy distributed computations on personal data using trusted execution environments," in *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2019, pp. 381–388.

[37] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[38] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural support for copy and tamper resistant software," *Acm Sigplan Notices*, vol. 35, no. 11, pp. 168–177, 2000.

[39] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *2015 IEEE symposium on security and privacy*. IEEE, 2015, pp. 605–622.

[40] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning in the cloud," *arXiv preprint arXiv:1204.6078*, 2012.

[41] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 135–146.

[42] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "Efficient and high-performance parallel hardware architectures for the aes-gcm," *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1165–1178, 2011.

[43] S. H. Pugsley, J. Jestes, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, "Comparing implementations of near-data computing with in-memory mapreduce workloads," *IEEE Micro*, vol. 34, no. 4, pp. 44–52, 2014.

[44] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin, "Using address independent seed encryption and bonsai merkle trees to make secure processors os-and performance-friendly," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 2007, pp. 183–196.

[45] B. Rogers, M. Prvulovic, and Y. Solihin, "Efficient data protection for distributed shared memory multiprocessors," in *Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, 2006, pp. 84–94.

[46] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, J. A. Joao, and M. K. Qureshi, "Morphable counters: Enabling compact integrity trees for low-overhead secure memories," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 416–427.

[47] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "Vc3: Trustworthy data analytics in the cloud using sgx," in *2015 IEEE symposium on security and privacy*. IEEE, 2015, pp. 38–54.

[48] A. SEV-SNP, "Strengthening vm isolation with integrity protection and more," *White Paper, January*, 2020.

[49] W. Shi, H.-h. S. Lee, M. Ghosh, C. Lu, and A. Boldyreva, "High efficiency counter mode security architecture via prediction and precomputation," in *32nd International Symposium on Computer Architecture (ISCA'05)*. IEEE, 2005, pp. 14–24.

[50] W. Shi, H.-H. Lee, M. Ghosh, and C. Lu, "Architectural support for high speed protection of memory integrity and confidentiality in multiprocessor systems," in *Proceedings. 13th International Conference on Parallel Architecture and Compilation Techniques, 2004. PACT 2004.* IEEE, 2004, pp. 123–134.

[51] W. Shi, H.-H. S. Lee, C. Lu, and M. Ghosh, "Towards the issues in architectural support for protection of software execution," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 1, pp. 6–15, 2005.

[52] D. Skarlatos, M. Yan, B. Gopireddy, R. Sprabery, J. Torrellas, and C. W. Fletcher, "Microscope: Enabling microarchitectural replay attacks," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2019, pp. 318–331.

[53] G. E. Suh, D. Clarke, B. Gasend, M. Van Dijk, and S. Devadas, "Efficient memory integrity verification and encryption for secure processors," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.* IEEE, 2003, pp. 339–350.

[54] G. E. Suh, D. Clarke, B. Gassend, M. Van Dijk, and S. Devadas, "Aegis: Architecture for tamper-evident and tamper-resistant processing," in *ACM International Conference on Supercomputing 25th Anniversary Volume*, 2003, pp. 357–368.

[55] M. Taassori, R. Balasubramonian, S. Chhabra, A. R. Alameldeen, M. Peddireddy, R. Agarwal, and R. Stutsman, "Compact leakage-free support for integrity and reliability," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 735–748.

[56] M. Taassori, A. Shafiee, and R. Balasubramonian, "Vault: Reducing paging overheads in sgx with efficient integrity verification structures," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 665–678.

[57] J. Talbot, R. M. Yoo, and C. Kozyrakis, "Phoenix++ modular mapreduce for shared-memory systems," in *Proceedings of the second international workshop on MapReduce and its applications*, 2011, pp. 9–16.

[58] K. Taranov, B. Rothenberger, A. Perrig, and T. Hoefler, "srdma–efficient nic-based authentication and encryption for remote direct memory access," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 691–704.

[59] C.-C. Tsai, J. Son, B. Jain, J. McAvey, R. A. Popa, and D. E. Porter, "Civet: An efficient java partitioning framework for hardware enclaves," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 505–522.

[60] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 991–1008.

[61] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on gpus," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 681–696.

[62] M. Wasi-ur Rahman, N. S. Islam, X. Lu, J. Jose, H. Subramoni, H. Wang, and D. K. D. Panda, "High-performance rdma-based design of hadoop mapreduce over infiniband," in *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*. IEEE, 2013, pp. 1908–1917.

[63] P. Wu, J. Ning, W. Luo, X. Huang, and D. He, "Exploring dynamic task loading in sgx-based distributed computing," *IEEE Transactions on Services Computing*, 2021.

[64] P. Wu, Q. Shen, R. H. Deng, X. Liu, Y. Zhang, and Z. Wu, "Oblidc: an sgx-based oblivious distributed computing framework with formal proof," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, 2019, pp. 86–99.

[65] M. Yan, R. Sprabery, B. Gopireddy, C. Fletcher, R. Campbell, and J. Torrellas, "Attack directories, not caches: Side channel attacks in a non-inclusive world," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 888–904.

[66] Y. Yarom and N. Benger, "Recovering openssl ecdsa nonces using the flush+ reload cache side-channel attack," *Cryptology ePrint Archive*, 2014.

[67] R. M. Yoo, A. Romano, and C. Kozyrakis, "Phoenix rebirth: Scalable mapreduce on a large-scale shared-memory system," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2009, pp. 198–207.

[68] G. Zhou, H. Michalik, and L. Hinsenkamp, "Efficient and high-throughput implementations of aes-gcm on fpgas," in *2007 International Conference on Field-Programmable Technology*. IEEE, 2007, pp. 185–192.

[69] G. Zhou, H. Michalik, and L. Hinsenkamp, "Improving throughput of aes-gcm with pipelined karatsuba multipliers on fpgas," in *International Workshop on Applied Reconfigurable Computing*. Springer, 2009, pp. 193–203.

[70] X. Zhuang, T. Zhang, and S. Pande, "Hide: an infrastructure for efficiently protecting information leakage on the address bus," *ACM SIGOPS Operating Systems Review*, vol. 38, no. 5, pp. 72–84, 2004.