Lecture Note

Chapter 6: Memory Protections

CYENG 225: Microcontroller Essentials for Cyber Applications

Instructor: Dr. Shayan (Sean) Taheri

Gannon University (GU)

Chapter 6 Overview

- **Major Items**
  - It covers issues relating to protection of the main memory, typically **Dynamic Random-Access Memory** (DRAM), but also **Non-Volatile Random-Access Memory** (NVRAM) in the near future.
  - It overviews threats against main memory, dividing them into **active** and **passive** attacks.
  - It then discusses mechanisms which can help ensure confidentiality, integrity, and access pattern protections for the main memory.
  - It finally presents assumption relating to main memory protections.

Threats Against Main Memory

- The main memory, today typically DRAM, is <u>often a separate chip</u> from the main processor chip.
- The main memory is connected to the processor chip <u>via a communication bus</u> that is a set of wires on the mother board.
- **<u>Both the main memory and the communication bus</u>** are typically assumed untrusted.
- As they are located outside of the processor chip, it is assumed that they can be more easily probed than any wires inside the processor chip.
- Memory protection needs to consider both attacks from within the processor and from the outside of the system as the memory and the processor are <u>separate physical chips</u>.
- **<u>The penalty</u>** for these protections is the performance, e.g., encryption overhead, and area needed, e.g., to store root hash values in the processor.

Sources of Attacks on Memory

- Memory is vulnerable to the different **types of attacks** listed as follows.
  - Untrusted software running on the processor can attempt to access memory regions is not allowed to.
  - Malicious devices can attempt to use DMA to access memory regions they are not allowed to or snooping on the bus.
  - Physical attacks on the memory bus can be used to extract data that is being communicated on the bus.
  - Physical attacks on the memory itself, e.g., Rowhammer or Coldboot.
- In addition to attacks such as probing the memory bus, DRAM chips can be easily removed and analyzed off-line.
- With practical non-volatile memories (NVRAM) on the horizon, data stored in the main memory will no longer disappear when memory is powered off, further requiring the contents to be protected.

Passive Attacks

- Attacks on memory can be divided into **active and passive attacks**.
- In passive attacks, the attacker passively **eavesdrops on the communication**, but does not try to alter any data.
- **Passive attacker** can try to learn the contents of messages - this can be prevented with use of encryption, i.e., **encrypt all data before it leaves the chip boundary**.
- He or she can also try to learn some information from the patterns of the memory accesses - this can be prevented with **obfuscated memory accesses** (usually using oblivious RAM techniques).
- The passive attacks focus on **breaking confidentiality** protections and passive attacks usually succeed if the designers did not correctly protect the information, such as encryption is not used while untrusted components have access to the sensitive data.
- **Eavesdropping Attacks** focus on passively collecting information that is communicated to or from memory.
  - Getting access to data is not the only potential problem.
  - Passively observing the memory access patterns can also reveal sensitive information.
  - Encryption combined with hiding memory access patterns can protect against eavesdropping attacks.

Active Attacks

- In active attacks, the attacker can actively inject or manipulate the communication between the processor and the memory, or inject their own commands and request the memory to execute them.
- The active attacks focus on **breaking either confidentiality or integrity** - active attacks are usually successful if the designers do not properly authenticate communication, or if freshness is not ensured.
- **Spoofing Attacks**, aim to inject memory data (or memory operations) without being detected.
  - Attacker can inject memory reads or writes which will either change memory contents, or gain access to some memory. → It may be possible, for example, to spoof memory read operation while the main processor is in sleep or low-power mode, and read the memory contents undetected. → Such attacks can be mitigated by proper authentication of memory access commands, or through use of encryption (where only legitimate entity has the right cryptographic key).
  - Writing to memory can affect integrity of the system, for example the attacker can write new page table memory contents to attempt to modify the page table-based isolation mechanisms. → Such attacks can be prevented through use of hashing and message authentication codes.

Active Attacks

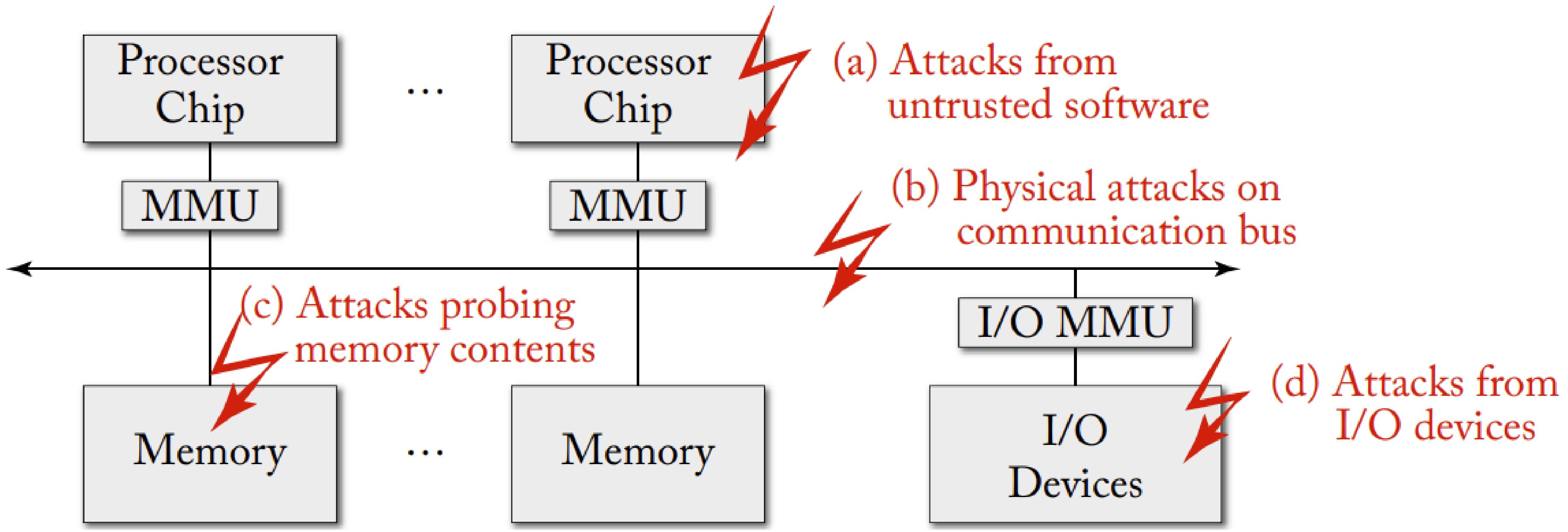
- **Splicing Attacks**, also called mix-and-match attacks, focus on combining data from two or more reads or writes to create a new, legitimate read or write.
  - The splicing attacks, as the name implies, splice portions of different message, e.g., data payload of one message, with authentication header of another.
  - If the whole read or write operation is not analyzed and authenticated correctly, attacker may be able to re-use portions of messages to create a valid memory operation.
  - To protect against this type of attack, hashing and MAC can be used, however, the authenticated data needs to make sure to cover whole packet so that attacker cannot change part of message, while leaving the hash or MAC the same.

- **Replay Attacks**, also called rewind attacks, leverage old messages, and send them again.
  - If a nonce is not used, then there is no guarantee of freshness.
  - Often, a monotonic counter can be used as a nonce to ensure that sender or receiver can discover that a message is old.
  - Replay protection involves both nonces (for freshness) and hashing or MACs to ensure that the correct nonce is always part of the message; otherwise **a reply attack** could combine part of one message, with a nonce from another message.

Main Memory Protection Mechanisms

- Memory protection can be achieved by leveraging the three following techniques:
  - **Confidentiality** protection with encryption.
  - **Integrity** protection with hashing (typically using a hash-tree).
  - **Access Pattern** protection (typically leveraging Oblivious RAM techniques).
- Confidentiality protection can be against attacks from different processor chips and I/O devices, or even against attacks from among different software running within a core (e.g., each trusted software module can have its memory encrypted and protected from other software, operating system or hypervisor).
- Integrity can be checked with respect to whole processor chip (one hash tree for each chip), or per trusted software module or per virtual machine.
- **Access pattern protection** requires special on-chip storage, to buffer and store some data locally, thus typically is done per processor chip. → Per software module or per virtual machine protections could be designed in the future.
- Some of the threats on the memories can be **mitigated thanks to memory management units (MMUs)** and I/O MMU which serve to logically isolate the different processors, memories, and devices.
- The interconnect and memories can still be physically probed, or the I/O devices can directly access the contents of memories if I/O MMU is not configured correctly, necessitating encryption and hashing to protect memory contents.

Main Memory Protection Mechanisms (Cont.)


Processor Chip
…
Processor Chip
(a) Attacks from untrusted software
MMU
MMU
(b) Physical attacks on communication bus
(c) Attacks probing memory contents
I/O MMU
(d) Attacks from I/O devices
Memory
…
Memory
I/O Devices

Simplified diagram of a multi-processor chip system with multiple memories and I/O devices all having access to the same interconnect.

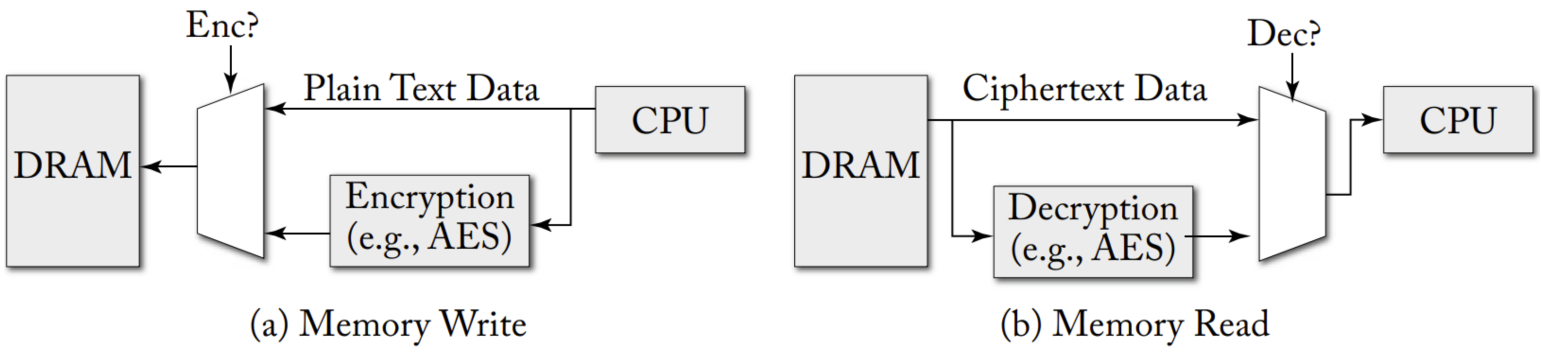
Confidentiality Protection with Encryption

- **Memory encryption typically uses AES block cipher** to transparently encrypt and decrypt data going to or from the DRAM, using an key re-generated on each system reboot.
- **The key has to be stored** on the processor chip, often in the MMU or in a special security management module.
- If same memory is **to be shared among different processors** they need access to the shared secret key.
- With multi-core processor chips, one MMU can be shared by the different processors, which actually removes need to have to explicitly share the key.
- In a multi-processor setup, the key has to be somehow shared by the processor chips.
- Sending of the key in plain text on the memory bus is not possible (attacker could easily snoop on the key).
- The solution can be **public-key cryptography**, however, this is very expensive in hardware and not often done.
- Most solutions today focus on having unique key(s) per chip without explicit sharing of the keys among different chips.
- Memory sharing with devices, via **Direct Memory Access (DMA)**, is often needed.
  - Direct memory access is a feature of computer systems that allows certain hardware subsystems to access main system memory independently of the central processing unit.
  - Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work.

Confidentiality Protection with Encryption (Cont.)

- The device would need to have access to the key, or the data is sent in plain text from the device to the **I/O MMU**, which then can encrypt data going to the main memory.
- Partly sending data in plain text is not a good solution, design should encrypt data end-to-end. In practice, the devices doing DMA are I/O devices (network card, or a disk) and they would not do any encryption.
- Instead, the data they are processing is already encrypted (at some higher level in the software stack) so even though data going on the memory or Peripheral Interface Controller Express (PICe) bus between device and memory is unencrypted per se.
- To facilitate enabling and disabling encryption for certain memory ranges (e.g., memory shared with I/O devices is unencrypted), an extra bit in page table entries can be used by the OS, hypervisor, or the hardware to mark data needing protection.
- Data marked as not needing protection can simply bypass the encryption or decryption engine.
- Data that does need protection will go through the engine—which add latency to the memory access.
- **Counter-mode encryption** can be used to speed up the process by pre-computing the encryption pads and then only doing one XOR operation between data and the pad.
  - This requires **proper caching of the encryption counters used in counter-mode encryption** to achieve good performance.
  - In addition to using one key for whole memory encryption, different keys can be used for different applications or virtual machines.

Confidentiality Protection with Encryption (Cont.)



Block diagram of key components in encryption and decryption, shown not to scale. (a) For memory writes, select data can be encrypted, typically with a symmetric key cipher such as AES. (b) For memory reads, if the data is not decrypted, e.g., read by software not authorized, the CPU will only see the ciphertext.

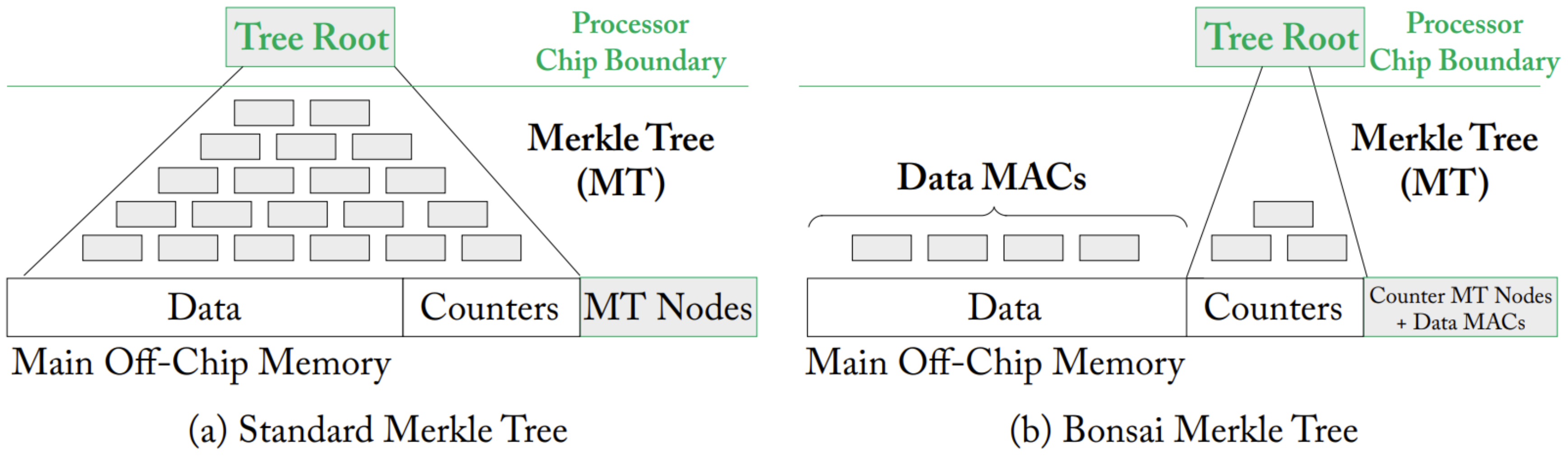
Confidentiality Protection with Encryption (Cont.)

- The keys are managed typically by **the MMU or a management engine** - this may be **the weakest link**, as the key is accessible to the MMU or management engine which can potentially decrypt all the memory traffic.
- As long as the keys are generated by the hardware and only accessible to the hardware, the management software, e.g., hypervisor, can help manage which keys are in use, without actually getting access to the plaintext of the keys.
- Outside the processor, memory is encrypted with the different keys, so each entity is protected from others.
- Inside the processor die, the hardware keeps track of the virtual machine IDs to prevent one entity from accessing another's code or data, while the code and data is in plain text.
- Secure architectures **should use hardware to generate memory encryption keys** on each boot cycle.
- Different keys can be used for different entities, to ensure data encrypted by one cannot be accessed by another.
- In addition to encryption, use of tagging or IDs is needed to control access to the data once it has been decrypted and brought into the processor.

Integrity Protection with Hashing

- Hashes are key cryptographic primitives used in **integrity verification**.
- Integrity protection leverages the fact that each input to a cryptographic hash will give a unique hash output (with negligible probability of two inputs giving same output, as determined by the size of the hash).
- The hash value uniquely represents the data, i.e., **it is a fingerprint for the data**.
- If the data is altered, the hash value will change. → Previously computed hash will be different from **freshly computed hash** and this difference indicates that someone changed the data.
- Hashing is then the key part in integrity protection of memory contents.
- Memory integrity protection most often focuses on the external attackers.
- The assumption is that the off-chip DRAM memory can be more easily manipulated or changed.
- After data is stored, and before it is used again, integrity checks are needed.
- Hashes require storage in a secure on-chip location so that there is a reference value which can be compared against.
- A simplistic solution would be to hash the whole memory and keep one hash value.
- A common alternative that is used are **hash trees**, also called **Merkle trees**.
  - In computer science, **a hash tree** is a persistent data structure that can be used to implement sets and maps, intended to replace hash tables in purely functional programming.
  - In cryptography and computer science, **a hash tree or Merkle tree** is a tree in which every "leaf" (node) is labelled with the cryptographic hash of a data block, and every node that is not a leaf (**called a branch, inner node, or inode**) is labelled with the cryptographic hash of the labels of its child nodes.

Integrity Protection with Hashing (Cont.)



Comparison of (a) standard Merkle Tree used for memory protection and (b) Bonsai Merkel Tree which uses MACs and build memory tree only over the counters used in the MACs. The root hash is stored within the trusted processor boundary, shown in green.

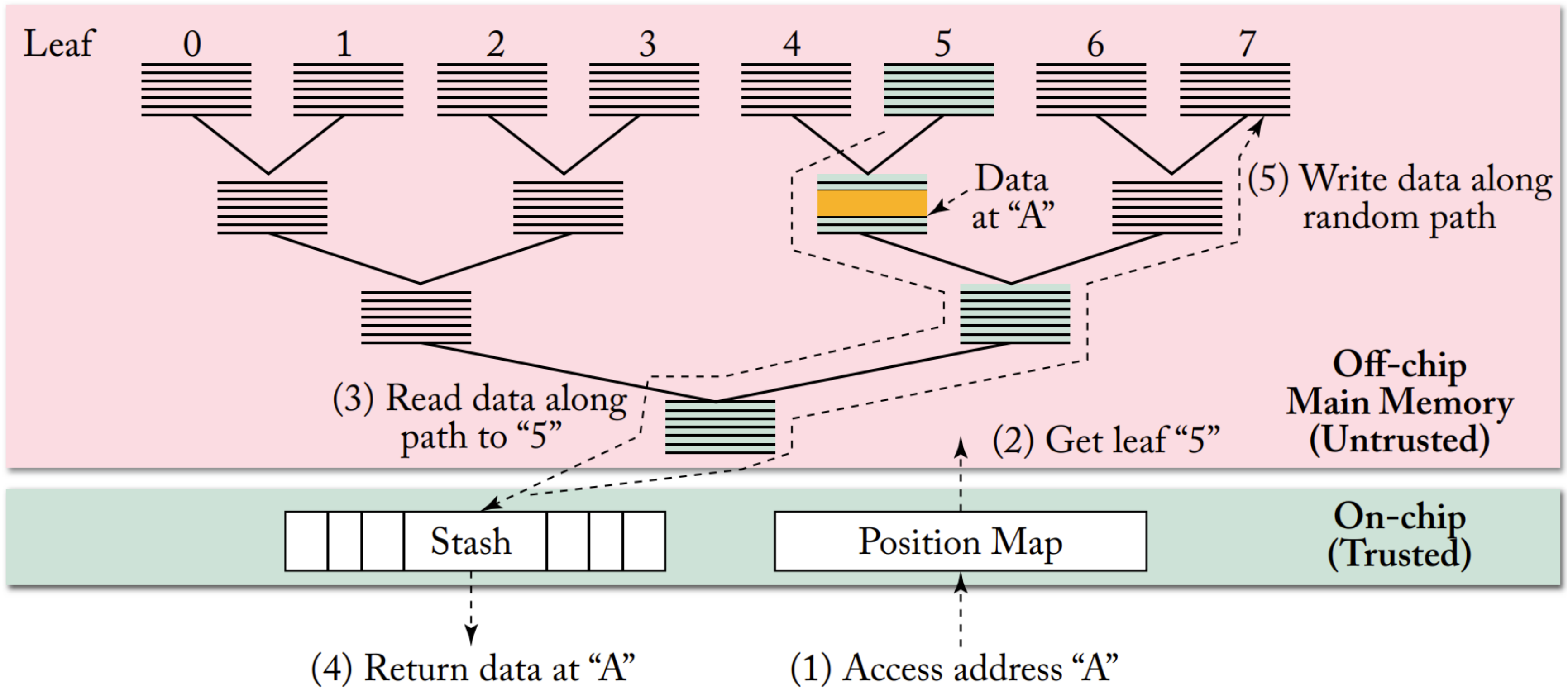
Integrity Protection with Hashing (Cont.)

- Secure architectures <u>should use hashing to check integrity of memory</u>, and store the root of the hash tree in a secure on-chip location.
- They should use <u>keyed hashes or MACs</u> to <u>ensure attackers are not able to generate their own hashes (to match their malicious data)</u>.
- For improved performance and to reduce the amount of data that needs to be integrity protected, approaches from Bonsai Merkle trees can be followed.
- <u>**Hash tress**</u> (using keyed hashes or MACs) <u>combined with nonces</u> can ensure <u>protection of spoofing, splicing, and replay attacks</u>.

Access Pattern Protection

- Besides getting access to raw data of the memory, observing memory access patterns can also reveal secrets.
- To address this issue, memory access pattern protection techniques have been proposed, most notably **Oblivious Random Access Memory (ORAM)**.
- ORAM keeps the semantics of a program, but hides the exact memory access pattern of the program.
- Conceptually, ORAM prevents information leakage by maintaining all memory locations randomly shuffled → On each memory access, data read or written, and then reshuffled.
- **The goal of ORAM** is that any memory access pattern is indistinguishable from any other access pattern.
- An attacker seeing seemingly random-access pattern does not know which data was actually accessed, and which one is a decoy access.
- It is composed of **two main components**: (**1**) A binary tree storage in off-chip main memory; and (**2**) an on-chip trusted ORAM controller.
- The binary tree stores the data content and is implemented using the main memory, DRAM.

Access Pattern Protection (Cont.)



Example operation of Path ORAM showing data access and how all data blocks along a path are read into the stash, data is returned to the processor, and finally data blocks along different, random, path are written back.

Access Pattern Protection (Cont.)

- **Each node in the tree** can hold up to **N** data blocks, and **any unused space in a node** is loaded with random dummy data blocks.
- The tree has a root node and number of leaf nodes, thus there is a path from the root to each leaf node, and on each path there are numerous data blocks (stored within the nodes of the tree).
- All nodes are encrypted and thus attacker cannot tell real data form random dummy data.
- **ORAM controller** is part of the trusted hardware that controls and manipulates the tree structure.
- The ORAM controller contains two main structures: a position map and a so-called stash.
- Data stored in Path ORAM either needs to be stored in the DRAM (in the tree structure) or be located in the on-chip stash.
- Protection of the raw data of memory is not sufficient. → **The access pattern behavior** needs to be protected as well, as it can leak information about what the protected software is doing.
- **Amount of information** that can be deduced from access patterns by the attacker can be reduced by shuffling the memory access patterns and hiding the true memory references among the random looking accesses.

Memory Protections Assumption

- **Off-Chip Memory is Untrusted** and **the contents is assumed to be protected** from the snooping, spoofing, splicing, replay, and disturbance attacks.
- **Encryption** should be used to prevent snooping and spoofing attacks.
- **Hashing** should be used to prevent spoofing, splicing, replay (nonces must be used), and disturbance attacks.
- **Oblivious Access** should be used to prevent snooping attacks.

Assignment

- **Reading Assignment:**
  - Zferer, J., 2018. **Principles of secure processor architecture design**, ser. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 9048, pp.1-175.
    - ✓ "Chapter 6: Memory Protections", Pages 65-74.

Questions?