Institut
Mines-Télécom

Performance Optimizations of Integrity Checking based on Merkle Trees

Salaheddine OUAARAB
<ouaarab@enst.fr>
June 14, 2015

Table of contents

Plan

Context

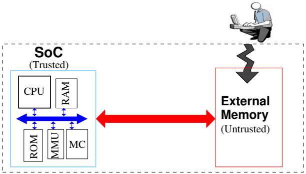
- The integrity protection of a large data structure stored on an untrusted medium is frequently one of the weakest points on the security point of view

Context

- The integrity protection of a large data structure stored on an untrusted medium is frequently one of the weakest points on the security point of view
- Many field are concerned like cloud computing, database and embedded systems

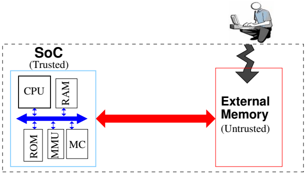
Context

- The integrity protection of a large data structure stored on an untrusted medium is frequently one of the weakest points on the security point of view
- Many field are concerned like cloud computing, database and embedded systems

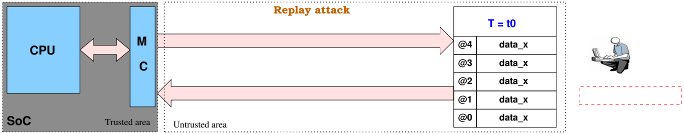


Context

- The integrity protection of a large data structure stored on an untrusted medium is frequently one of the weakest points on the security point of view
- Many field are concerned like cloud computing, database and embedded systems

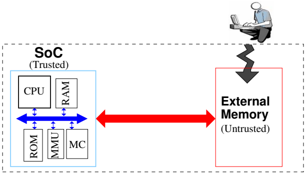


- The most difficult attack to counter is **replay attack**

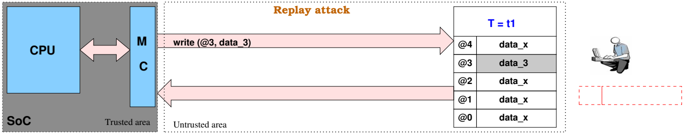


Context

- The integrity protection of a large data structure stored on an untrusted medium is frequently one of the weakest points on the security point of view
- Many field are concerned like cloud computing, database and embedded systems

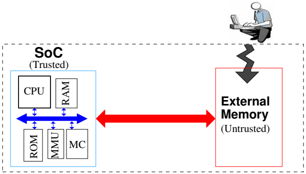


- The most difficult attack to counter is **replay attack**

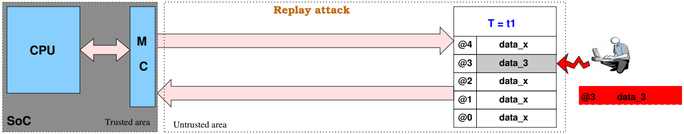


Context

- The integrity protection of a large data structure stored on an untrusted medium is frequently one of the weakest points on the security point of view
- Many field are concerned like cloud computing, database and embedded systems

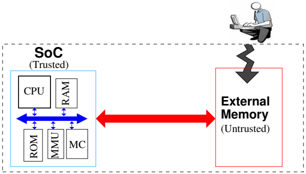


- The most difficult attack to counter is **replay attack**

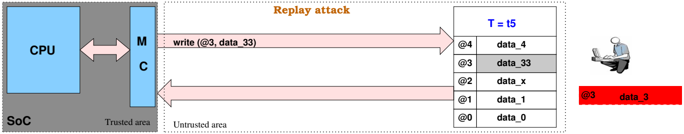


Context

- The integrity protection of a large data structure stored on an untrusted medium is frequently one of the weakest points on the security point of view
- Many field are concerned like cloud computing, database and embedded systems

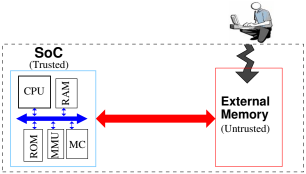


- The most difficult attack to counter is **replay attack**

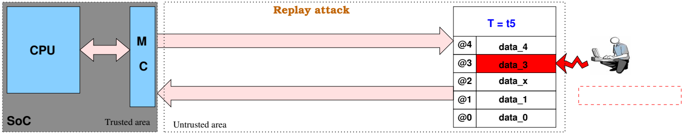


Context

- The integrity protection of a large data structure stored on an untrusted medium is frequently one of the weakest points on the security point of view
- Many field are concerned like cloud computing, database and embedded systems

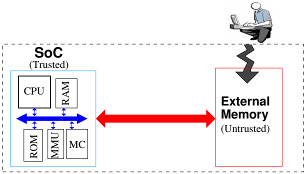


- The most difficult attack to counter is **replay attack**

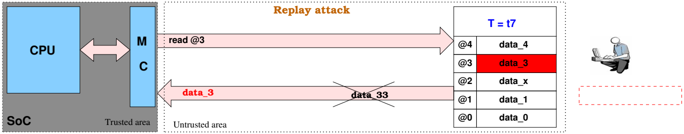


Context

- The integrity protection of a large data structure stored on an untrusted medium is frequently one of the weakest points on the security point of view
- Many field are concerned like cloud computing, database and embedded systems

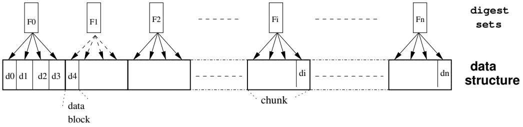


- The most difficult attack to counter is **replay attack**



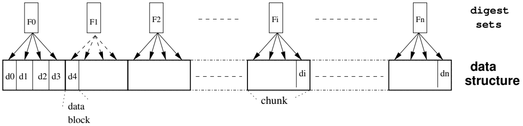
Context

- The integrity protection requests the use of one-way function (hash function or MAC)

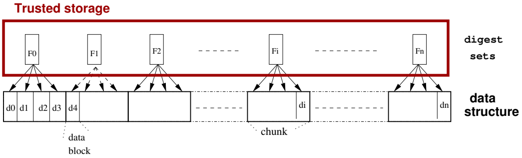


Context

- The integrity protection requests the use of one-way function (hash function or MAC)

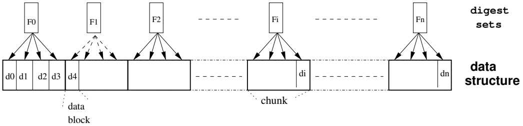


- **And** a secure storage (to counter the replay attack)

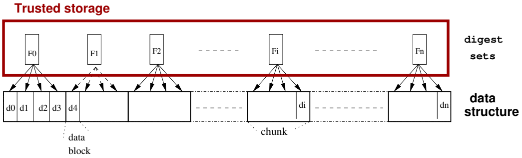


Context

- The integrity protection requests the use of one-way function (hash function or MAC)



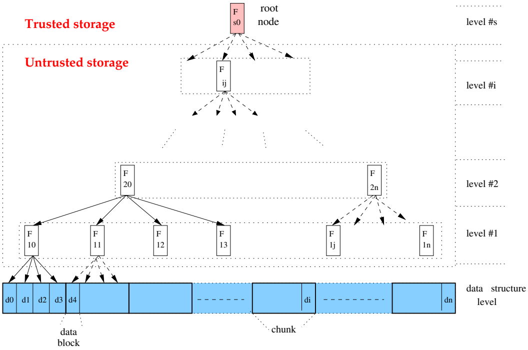
- **And** a secure storage (to counter the replay attack)



- **But** the secure storage is usually **small** and **expensive**

Merkle Tree
Definition

- Merkle Tree hierarchically organises the reference digests and stores the root in the secure storage



Plan

Merkle Tree

Problematic

Issue

Merkle tree leads to significant storage and performance overheads:

- `initialization`: based on iterative function
- `Integrity Checking / Update`: increase the number of untrusted storage access and digest computations

Merkle Tree

Problematic

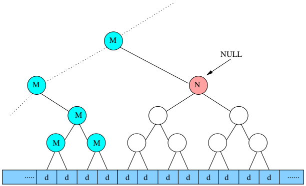
Issue

Merkle tree leads to significant storage and performance overheads:

- `initialization`: based on iterative function
- `Integrity Checking / Update`: increase the number of untrusted storage access and digest computations

Optimization

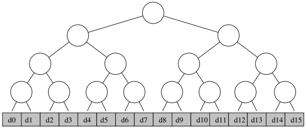
- `Initialization`: introduce **Hollow Merkle tree**
- `Integrity Checking / Update`: use of a customized **cache** located inside a secure area.



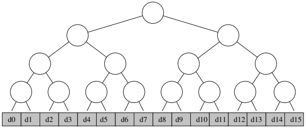
Merkle Trees

Initialization

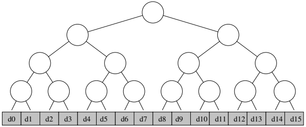
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)

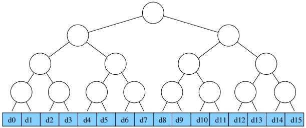


- Non-Initialized Hollow Merkle Trees (**NI-HMT**)

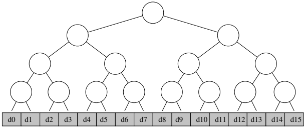


Merkle Trees
Initialization

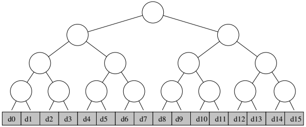
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)



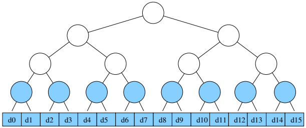
- Non-Initialized Hollow Merkle Trees (**NI-HMT**)



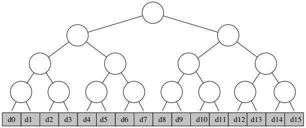
Merkle Trees

Initialization

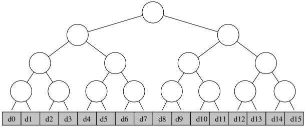
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)



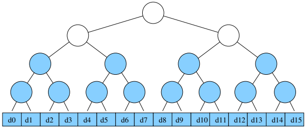
- Non-Initialized Hollow Merkle Trees (**NI-HMT**)



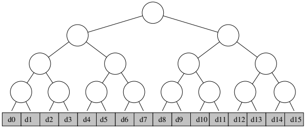
Merkle Trees

Initialization

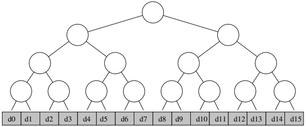
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)

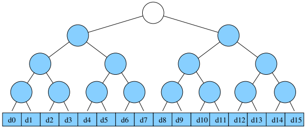


- Non-Initialized Hollow Merkle Trees (**NI-HMT**)

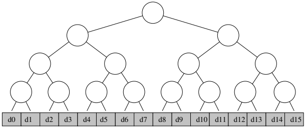


Merkle Trees
Initialization

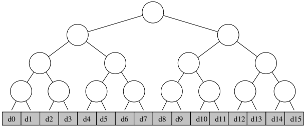
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)

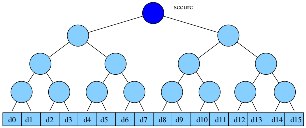


- Non-Initialized Hollow Merkle Trees (**NI-HMT**)

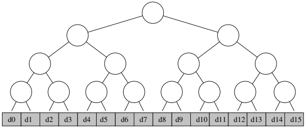


Merkle Trees
Initialization

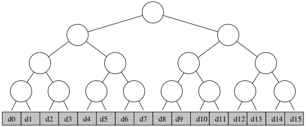
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)



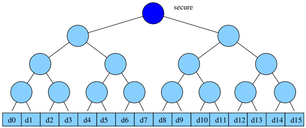
- Non-Initialized Hollow Merkle Trees (**NI-HMT**)



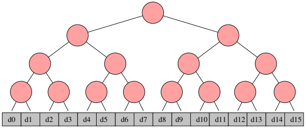
Merkle Trees

Initialization

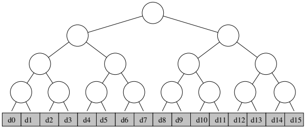
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)



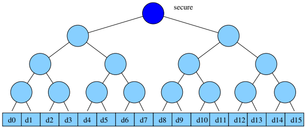
- Non-Initialized Hollow Merkle Trees (**NI-HMT**)



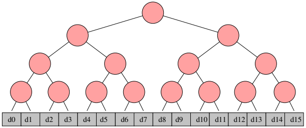
Merkle Trees

Initialization

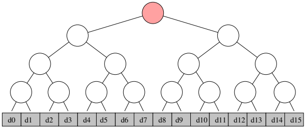
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)



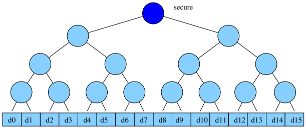
- Non-Initialized Hollow Merkle Trees (**NI-HMT**)



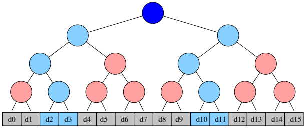
Merkle Trees

Update

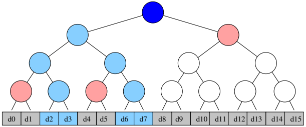
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)

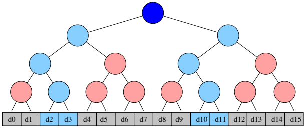


- Non-Initialized Hollow Merkle Trees (**NI-HMT**)

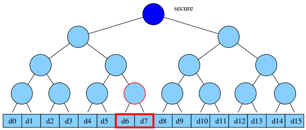


Merkle Trees
Update

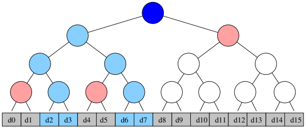
- Initialized Hollow Merkle Trees (**I-HMT**)



- Regular Merkle Trees (**RMT**)



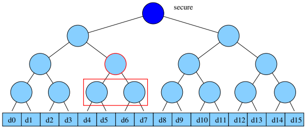
- Non-Initialized Hollow Merkle Trees (**NI-HMT**)



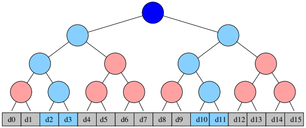
Merkle Trees

Update

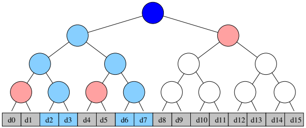
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)



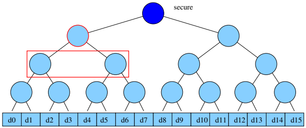
- Non-Initialized Hollow Merkle Trees (**NI-HMT**)



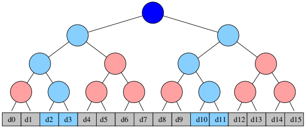
Merkle Trees

Update

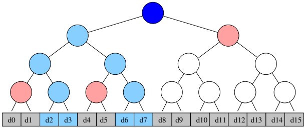
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)



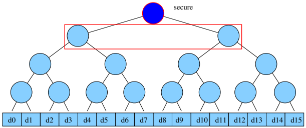
- Non-Initialized Hollow Merkle Trees (**NI-HMT**)



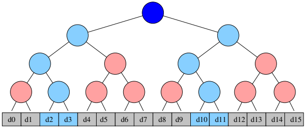
Merkle Trees

Update

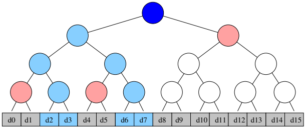
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)



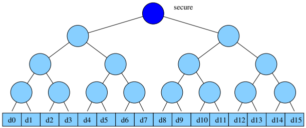
- Non-Initialized Hollow Merkle Trees (**NI-HMT**)



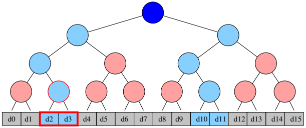
Merkle Trees

Update

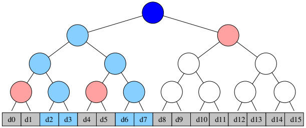
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)

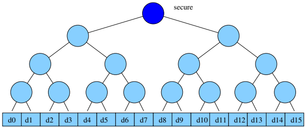


- Non-Initialized Hollow Merkle Trees (**NI-HMT**)

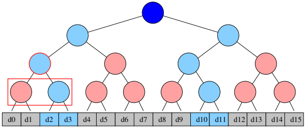


Merkle Trees
Update

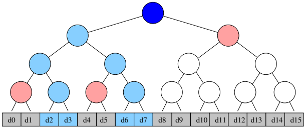
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)

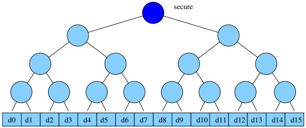


- Non-Initialized Hollow Merkle Trees (**NI-HMT**)

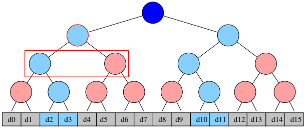


Merkle Trees
Update

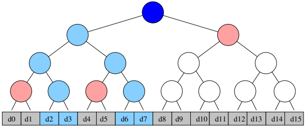
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)



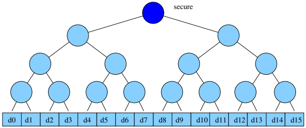
- Non-Initialized Hollow Merkle Trees (**NI-HMT**)



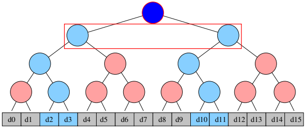
Merkle Trees

Update

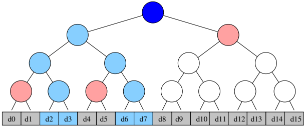
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)

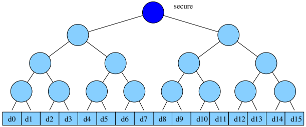


- Non-Initialized Hollow Merkle Trees (**NI-HMT**)

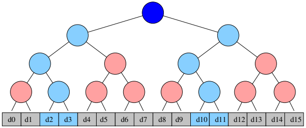


Merkle Trees
Update

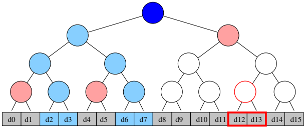
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)

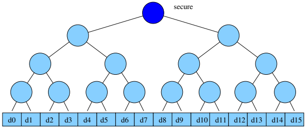


- Non-Initialized Hollow Merkle Trees (**NI-HMT**)

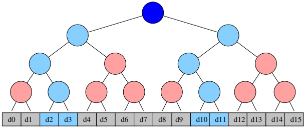


Merkle Trees
Update

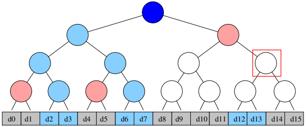
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)



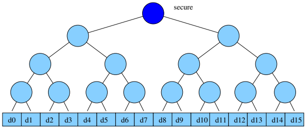
- Non-Initialized Hollow Merkle Trees (**NI-HMT**)



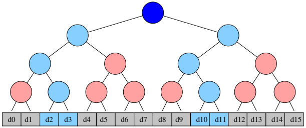
Merkle Trees

Update

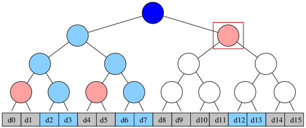
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)

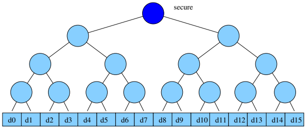


- Non-Initialized Hollow Merkle Trees (**NI-HMT**)

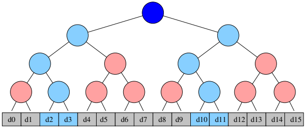


Merkle Trees
Update

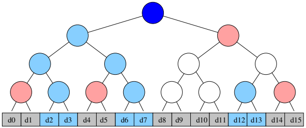
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)



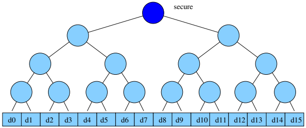
- Non-Initialized Hollow Merkle Trees (**NI-HMT**)



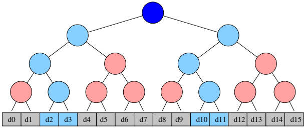
Merkle Trees

Update

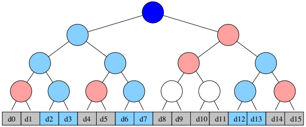
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)

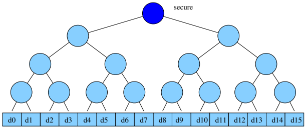


- Non-Initialized Hollow Merkle Trees (**NI-HMT**)

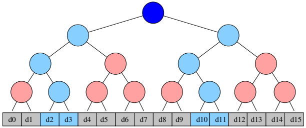


Merkle Trees
Update

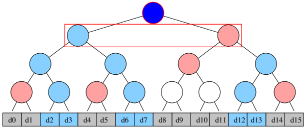
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)

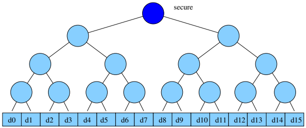


- Non-Initialized Hollow Merkle Trees (**NI-HMT**)

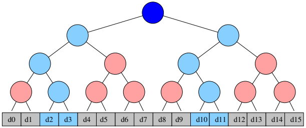


Merkle Trees
Update

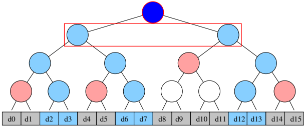
- Regular Merkle Trees (**RMT**)



- Initialized Hollow Merkle Trees (**I-HMT**)



- Non-Initialized Hollow Merkle Trees (**NI-HMT**)



Plan

Use of Cache

- The use of cache decreases the bandwidth with the mass storage and, also reduces the number of digest computations
- Two particular algorithms are introduced:
  - **ASAP**: the integrity checking ends as soon as we match an intermediate node into the cache
  - **ALAP**: the update of node into the untrusted storage is delayed as late as possible storage

Use of Cache

- The use of cache decreases the bandwidth with the mass storage and, also reduces the number of digest computations
- Two particular algorithms are introduced:
  - **ASAP**: the integrity checking ends as soon as we match an intermediate node into the cache
  - **ALAP**: the update of node into the untrusted storage is delayed as late as possible storage

Issue

ALAP algorithm causes Merkle tree incoherency between the nodes stored into the cache and that stored into the untrusted storage

Use of Cache

- The use of cache decreases the bandwidth with the mass storage and, also reduces the number of digest computations
- Two particular algorithms are introduced:
  - **ASAP**: the integrity checking ends as soon as we match an intermediate node into the cache
  - **ALAP**: the update of node into the untrusted storage is delayed as late as possible storage

Issue

ALAP algorithm causes Merkle tree incoherency between the nodes stored into the cache and that stored into the untrusted storage

Cache customisation

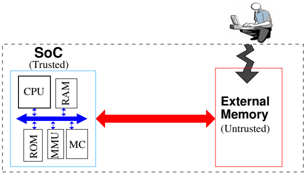
Modify the behavior of **Write-back** policy (i.e. modify `READ` and `WRITE` functions of cache controller and append new functions)

Plan

Case Study: SecBus Project

Purpose

Provide a strong confidentiality and integrity protection against on-board attacks (including replay attacks)



Initialization

- Memory interconnect latency: 100 CPU clock cycles
- DES algorithm latency: 4 CPU clock cycles
- Size of MT and memory page: 4 KB
- Number of random writes: 12.000
- Merkle Tree (MT) cache: Set associative, 64 sets, 8 blocks, 8-byte blocks, LRU, write-back

Use of HMT

- I-HMT: 22.5 times faster vs RMT
- NI-HMT: 186.5 times faster vs RMT

Schemes	cycle	access	MAC
RMT	4,219,439	29,142	10,885
NI-HMT	101,559	637	372
I-HMT	250,988	21,033	378

Table 1: Initialization step without cache

Schemes	cycle	access	MAC
RMT	4,097,234	28,480	10,331
NI-HMT	21,975	186	72
I-HMT	181,646	20,638	47

Table 2: Initialization step with cache

Random Writes

- Memory interconnect latency: 100 CPU clock cycles
- DES algorithm latency: 4 CPU clock cycles
- Size of MT and memory page: 4 KB
- Number of random writes: 12.000
- Merkle Tree (MT) cache: Set associative, 64 sets, 8 blocks, 8-byte blocks, LRU, write-back

Use of Cache

with cache is 6.5 times faster vs without cache

Schemes	cycle	access	MAC
RMT	102,515,933	684,103	432,000
NI-HMT	102,929,333	688,063	432,264
I-HMT	102,515,933	684,103	432,000

Table 3: Random writes without cache

Schemes	cycle	access	MAC
RMT	15,808,417	130,917	36,91
NI-HMT	16,116,007	135,472	38,183
I-HMT	15,921,940	130,988	37,099

Table 4: Random writes with cache

Plan

Conclusion

- Two optimizations of Merkle trees have been introduced to speed up initialization, integrity checking and tree updates
- The results shows an improvement of the trees initialization by using Hollow Merkle trees
- The cache improves the performance after the initialization
- The choice between the two types of Hollow Merkle trees depends on the use case

Thank you for your attention