Performance Optimizations of Integrity Checking based on Merkle Trees

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

Introduction

Merkle Trees Management

Merkle Tree Caches

Experiments and Results

Conclusion
Plan

Introduction

Merkle Trees Management

Merkle Tree Caches

Experiments and Results

Conclusion
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.
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 fields are concerned like cloud computing, database, and embedded systems.
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 fields are concerned like cloud computing, database and embedded systems.

The most difficult attack to counter is replay attack.
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.
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 fields are concerned like cloud computing, database and embedded systems.

The most difficult attack to counter is **replay attack**.
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 fields 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 fields are concerned like cloud computing, database and embedded systems.

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

---

**Figure:**

- **CPU**
- **RAM**
- **ROM**
- **MMU**
- **MC**
- **External Memory (Untrusted)**
- **SoC (Trusted)**

**Diagram Diagram:**

- **Replay attack**
- **T = t7**
  - @4 data_4
  - @3 data_3
  - @2 data_x
  - @1 data_1
  - @0 data_0
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)
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

Introduction

Merkle Trees Management

Merkle Tree Caches

Experiments and Results

Conclusion
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**)
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

- 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**)
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

Introduction

Merkle Trees Management

Merkle Tree Caches

Experiments and Results

Conclusion
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 behaviour of write-back policy (i.e. modify read and write functions of cache controller and append new functions).
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

Introduction

Merkle Trees Management

Merkle Tree Caches

Experiments and Results

Conclusion
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

<table>
<thead>
<tr>
<th>Schemes</th>
<th>cycle</th>
<th>access</th>
<th>MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMT</td>
<td>4,219,439</td>
<td>29,142</td>
<td>10,885</td>
</tr>
<tr>
<td>NI-HMT</td>
<td>101,559</td>
<td>637</td>
<td>372</td>
</tr>
<tr>
<td>I-HMT</td>
<td>250,988</td>
<td>21,033</td>
<td>378</td>
</tr>
</tbody>
</table>

**Table 1:** Initialization step without cache

<table>
<thead>
<tr>
<th>Schemes</th>
<th>cycle</th>
<th>access</th>
<th>MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMT</td>
<td>4,097,234</td>
<td>28,480</td>
<td>10,331</td>
</tr>
<tr>
<td>NI-HMT</td>
<td>21,975</td>
<td>186</td>
<td>72</td>
</tr>
<tr>
<td>I-HMT</td>
<td>181,646</td>
<td>20,638</td>
<td>47</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Schemes</th>
<th>cycle</th>
<th>access</th>
<th>MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMT</td>
<td>102,515,933</td>
<td>684,103</td>
<td>432,000</td>
</tr>
<tr>
<td>NI-HMT</td>
<td>102,929,333</td>
<td>688,063</td>
<td>432,264</td>
</tr>
<tr>
<td>I-HMT</td>
<td>102,515,933</td>
<td>684,103</td>
<td>432,000</td>
</tr>
</tbody>
</table>

Table 3: Random writes without cache

<table>
<thead>
<tr>
<th>Schemes</th>
<th>cycle</th>
<th>access</th>
<th>MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMT</td>
<td>15,808,417</td>
<td>130,917</td>
<td>36,91</td>
</tr>
<tr>
<td>NI-HMT</td>
<td>16,116,007</td>
<td>135,472</td>
<td>38,183</td>
</tr>
<tr>
<td>I-HMT</td>
<td>15,921,940</td>
<td>130,988</td>
<td>37,099</td>
</tr>
</tbody>
</table>

Table 4: Random writes with cache
Plan

Introduction

Merkle Trees Management

Merkle Tree Caches

Experiments and Results

Conclusion
Conclusion

- Two optimizations of Merkle trees have been introduced to speed up initialization, integrity checking and tree updates.
- The results show 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