Project Acronym:	TRESCCA
Project Title:	Trustworthy Embedded systems for Secure Cloud Computing
Project number:	European Commission – 318036
Call identifier	FP7-ICT-2011-8
Start date of project:	01 Oct. 2012 Duration: 36 months

Document reference number:	D4.3
Document title:	Report on evaluation and security assessment of trustworthy cloud platform
Version:	1.1
Due date of document:	30st of September 2015
Actual submission date:	23rd of January 2016
Lead beneficiary:	WT
Participants:	Mercedes Castaño (WT), I. Garcia (WT), M. Grammatikakis (TEI), M. Paolino (VOSYS), B. Katzmarski (OFFIS), R. Pacalet (IMT), A. Herrholz (CS), M. Coppola (ST)

Project co-founded by the European Commission within the 7th Framework Programme		
DISSEMINATION LEVEL		
PU	Public	X
PCA	Public with confidential annex	
CO	Confidential, only for members of the consortium (including Commission Services)	

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

1.1 Purpose of the Document

This deliverable describes the setup of the fully working evaluation platform of the cloud environment using the client prototype and a standard cloud server installation.

After the design and implementation of the evaluation environment, the application use cases will be implemented and deployed including a full assessment of the security objectives.

1.2 Document Versions Sheet

Version	Date	Description, modifications, authors
0.1	12/08/2015	First version of the deliverable. Mercedes Castaño Torres (WT), All
1.0	11/11/2015	Final version for 3rd Technical Review. Mercedes Castaño Torres (WT), All
1.1	23/01/2016	Revised version following review recommendations. Add section on setup, attack and protection of WeSave application (Section 5.1, Scenario 1). Rework HSM-mem performance evaluation. (Section 5.5). Add section on validation of integration effort (Section 5.6). Add remarks on limits of performed evaluation throughout document.

2 Evaluation method and plan

As introduced in D1.3, TRESCCA is being developed based on a Scenario-Based Design (SBD) approach that enables to experiment and validate the results in a real-world environment. Within the different phases of the SDB framework, the last stage involves the implementation of a prototype and the evaluation plan of the designed TRESCCA cloud environment.

In D1.3 the scenarios that should be used to evaluate TRESCCA features was described as well. It helped to define the overall user, functional and security requirements. Based on this previous work, within this task the consortium defined altogether the evaluation methodology in order to evaluate TRESCCA in experimental use, involving one or more secured devices connected to a cloud environment.

The primary goals of evaluation of the TRESCCA platform components are:

1. Show effective protection of the TRESCCA platform against those security threats which have been defined in D1.1 and are considered to be covered by TRESCCA.

2. Show general applicability and flexibility of the TRESCCA platform components to be deployed in several different application scenarios and cloud environment setups.

The evaluation is being executed using the following steps:

1. Investigation and definition of requirements and limitations of the evaluation environment.

2. Selection of target technologies and HW/SW components for implementation of TRESCCA platform demonstrators. Technologies and components for cloud and client side may be specific for certain scenarios depending on the particular requirements of the application.

3. Setup of cloud-side evaluation environment.

4. Development of TRESCCA client demonstrators integrating all or selected HW or SW TRESCCA security components. Specific demonstrators may be developed for certain scenarios.

5. Initial integration and test of TRESCCA client(s) with cloud-side evaluation environment.

6. Definition and implementation of a library of security attacks that may either be executed internally as part of an application (software attack, either on hypervisor, kernel or application level) or externally (client hardware or communication infrastructure attack). There needs to be one or more attacks for each Use Case whereas each Use Case may cover one or more security threats.

7. Design and implementation of evaluation scenarios by developing applications demonstrating the general applicability of the TRESCCA platform.

8. Execution of the security attacks within the context of the different scenarios and demonstration of the effective protection against the corresponding threat.

9. Report of evaluation results of the different scenarios and protection against security threats.

This Scenario evaluation and validation methodology is important because the initial analysis done may be overly influenced by the consortium's a priori understanding of TRESCCA platform and its

use context. An explicit validation in real scenarios will allow the consortium to check with end-users and experts to ensure that the results fulfil the security threats and objectives detected.

While and after this evaluation stage a claim analysis will be done to reflect the consequences of scenarios, the results of interaction between all the components, system features and likely consequences of such security objectives in each of the scenarios. This claim analysis involves aggregating and interpreting claims, with the respective improvements in the TRESCCA platform and the evaluation report.

3 Security use cases

In D1.3 a set of seven Security Use Cases has been defined. It was expected that each use case would be covered by at least one of the scenarios. This chapter summarizes the use cases and includes an overview on their final coverage. Details on the use cases can be found in D1.3. Details on how the scenarios specifically covered a use case are given in chapter 5.

3.1 Summary

This section includes a short summary of the use cases defined in WP3.

UC1. Shared resources as a threat for integrity, confidentiality and availability.	
Summary:	Malicious software or even the user himself, are often a threat for the integrity, confidentiality and availability of a system. A strong isolation between sensible components of a system helps to prevent this kind of attacks.

UC2. Data leakage while storing or transferring data between client and Cloud	
Summary:	The storage services must ensure that the information stored in persistent mediums remains confidential and with integrity for all except the user of the service and authorized personnel of the underline service. Data may be stored remotely or locally and should be protected (confidentiality & integrity) in both storage and transmission.

UC3. Malicious access or modification of critical information on external memory pages	
Summary:	The access or modification of the content of a page of the external memory that contains data identified as critical for the proper working of the platform or its sensitive applications should be impossible for an unauthorized entity. This protection shall take into account both software and hardware attempts to access these critical memory pages. Software attacks are prevented using strong isolation between processes and virtual machines, and hardware attacks are prevented using external memory encryption and integrity protection.

UC4. Uncontrolled access to memory regions	
Summary:	Typically, different regions of memory are used and shared by different HW and SW components. Isolation of SW services and applications as well as HW components can only be accomplished if the access to these memory regions is controlled and protected. Therefore different regions of the memory may only be accessed and/or modified if the accessing component can provide the proper access rights. Control and enforcement of these access rights should be ensured by a combination of HW and SW

	measures.
UC5. Integrity and confidentiality of memory shared between VMs	
Summary:	A VM needs to interact and communicate with the outside world. This means that it is necessary to share information with other VMs, such as network packets, secure service requests, data coming from devices or the user, etc.
UC6. Integrity of boot process and software stacks	
Summary:	In collaborative distributed systems, it is mandatory for a remote platform to provide its integrity status in a trustworthy way to others in order to detect and prevent applications from being deployed on untrusted even hostile platforms and the software components are authentic. It is essential to check if the integrity of its code and data is protected and that it can be only updated by its authority. Chained integrity checking during the boot sequence is an important feature that guarantees to boot a system into a known and trustworthy operating system state.
UC7. Protection of migrating applications	
Summary:	Today's services are facing a lack of trust which is twofold: providers cannot trust their user's devices and users do not have a way to control their personal data shared with their providers. Virtualization and migration technology could unlock much more potential if the platform could be trusted. While data simply can be protected by encryption no method exists for computations. Applications still need to decrypt data in order to process it. Offering a secure platform where applications can migrate seamlessly between clouds and devices can fill this gap. On the one hand location aware data access can protect sensitive information and on the other hand secure migration of applications introduces attractive possibilities for cloud and mobile computing.

3.2 Coverage of Use Cases by Scenarios

While none of the scenarios covered all use cases, each use cases is at least covered by one scenario.

Scenario	UC1	UC2	UC3	UC4	UC5	UC6	UC7
Digital Rights Management	x				x	x	
Smart grids and cloud systems	x		x	x		x	
Securing transactions and authentications		x					x

4 TRESCCA Requirements

At the beginning of the project, TRESCCA has defined a set of requirements, initially reported in D1.3 and partitioned into user requirements, functional requirements and security requirements. In the following sections, these requirements will be revisited and evaluated if they have been met and covered by a specific scenario and/or corresponding hardware or software component developed by TRESCCA.

For some of the requirements it proved to be very challenging to give precise answers whether they have been met or not. This mostly applies to the more general and cloud oriented requirements. The identified gap between the high level cloud scenario and the low level technology developed in TRESCCA could not always be bridged. Nonetheless the porting of the WeSave application to the integrated prototype as part of Scenario 2 has proven the general applicability of the TRESCCA technologies for cloud services.

Due to the limitations of the final prototyping platform and the implemented demonstration applications, the requirements have not be validated in the context of real applications. Except for the WeSave applications, all other applications are only proof-of-concepts and no actual real-world application. Though for Scenario

4.1 User requirements

The rich catalogue of risks identified in the deliverable D1.1 “Security Analysis of cloud application of security objectives” could be seen as the main difficulties for the applicability of cloud computing, and even more when there is a hardware device at the user side with computing capabilities.

After a first phase of identification of the target users, TRESCCA also identified from the beginning a large variety of user requirements. These user requirements have been written from the user’s point of view and describe the key security properties that must be provided to satisfy the user’s needs and to boost the trustiness in cloud computing technologies. These user requirements will be translated to functional and security requirements, to describe what the software and hardware shall do in terms of security services and tasks. So the user requirements are the basis for the technical partners to develop the TRESCCA platform.

As previously the target user has been identified, each of them show different but also overlapping user needs, thus giving birth to too many different user requirements. In this section, TRESCCA identifies the most high-level and relevant user requirements that suit the needs of multiple target users, and will satisfy critical mass of users.

USER.1	
Description	Ensure confidentiality of the information
User / Rationale	The Cloud Consumer will generally expect all communication between them and the Cloud Provider will remain private. If client information kept by the cloud provider is not confidential, some people might be reluctant to seek critical information on it. From this point of view, the light client must ensure

	that they comply with the applicable provision of confidentiality when storing and transmitting client data. From the other side, the Cloud Provider might also be interested in the confidentiality of the information he delivers to the customer or the light client hardware device, in order to ensure the no replication and proper use of information, as for example to ensure digital right of media.
Priority	High
Result	**Requirement met and covered in scenario 1, 2 and 3 through TEE and secure communication channel.**

USER.2	
Description	Integrity Assurance
User / Rationale	The overall cloud computing infrastructure must exhibit key features to enable the assurance of service and data integrity for the users. To ensure this integrity, the Cloud Provider needs a cloud based integrity management service coupled with a trustworthy client component as a mean to increase the quality of the security evaluation of the client.
Priority	High
Result	**Requirement met and covered in scenario 1, 2 and 3 through TEE and secure communication channel.**

USER.3	
Description	Authenticity
User / Rationale	End-to-end identity management and third party authentication are key elements of cloud security for both the Cloud Provider and the Cloud Consumer. They require a proper identity security mechanism to make sure the preservation of the integrity and confidentiality of data and application while making access readily available to the appropriate users.
Priority	High
Result	**Requirement met and covered in scenario 1, 2 and 3 through TEE, secure communication channel and VM migration.**

USER.4	
Description	Privacy
User / Rationale	Most customers are aware of the danger of letting data control out of their hands and storing data with an outside cloud computing provider. Data could

	be compromised by the Cloud Computing Provider itself, or other competitive enterprises which are customers with the same Cloud Computing Provider. There is still a lack of transparency for customers on how, when, why and where their data is processed. Moreover, data protection and privacy legislation is not even similar in many countries around the globe yet and cloud computing is a global service of the future.
Priority	High
Result	**Requirement met and covered in scenario 2 and 3 through TEE, secure communication channel and VM migration.**

USER.5	
Description	Traceability and non-repudiation
User / Rationale	Cloud Providers should track the deployment options from the data center to the business process to make sure the value chain is uncompromised. Moreover the Cloud Provider is very interested in having non-repudiation mechanisms as a methodology for ensuring the traceability to conform the agreed policies and the accounting.
Priority	High
Result	**Requirement indirectly covered in scenario 2 but not directly related to TRESCCA outcomes.**

4.2 Functional requirements

FUNCT.1	
Description	TRESCCA must ensure Secure Hypervisor and Virtualization.
Rationale	The TRESCCA hypervisor provides an isolation layer that is a key feature of the whole project. This component must enable the isolation between VMs and the possibility to access to the Trusted Compartment. A secure abstraction of the underlying hardware will be provided by the hypervisor to the virtual machines. A secure migration mechanism ensures that the computation is decupled from data. Moving the VMs to the devices where sensitive user's data are stored permit to not share security assets outside the secure platform.
Reference	UC1, UC5, UC7
Priority	High
Result	**Requirement covered in scenario 1 and 3 through TEE and secure hypervisor. Indirectly supported by HSM-Mem and HSM-NoC.**

FUNCT.2	
Description	Provide HSM-mem and SSM-mem to protect external memory.
Rationale	TRESCCA platform architecture must guarantee confidentiality and the integrity of critical software applications even in the presence of an adversary who has physical access to every system component, other than internal components of the main SoC. TRESCCA platform aims to protect confidentiality and integrity of the communications between on-chip CPUs and off-chip external memories by guaranteeing that sensitive information does not leave the SoC in plaintext form and cannot be tampered with without causing a trap due to an integrity check. For each physical page on the memory bus, the SSM-mem will assign a security policy in terms of confidentiality and integrity, and the hardware HSM-mem will apply this policy on all memory accesses to that page. This simple organization will ease the design of the SSM-mem module and HSM-mem and allow protecting sensitive memory pages with the most appropriate cryptographic primitive for the defined security policy.
Reference	UC1, UC2, UC3, (UC6, UC7)
Priority	High
Result	**Requirement covered in scenario 1 and 2 through HSM-Mem.**

FUNCT.3	
Description	Provide HSM-NoC and Software NoC Configuration Manager (SNoCCM).
Rationale	The NoC Firewall regulates accesses to privileged memory (page tables, virtualization contexts and interrupt information). This functionality is crucial to most TRESCCA use cases, since it supports malware detection of viruses, Trojan horses or rogue applications, as well as subverting Denial-of-Service attacks by appropriately policing the NoC Firewall access point configured at the network interfaces of the initiator or target IPs. Moreover, it supports hardware virtualization extensions for hypervisor and/or guest VM isolation via a NoC firewall implementing Mandatory Access Control policies
Reference	UC1, UC3, UC4, UC5
Priority	High
Result	**Requirement covered in scenario 1 and 2 through HSM-NoC and NoC firewall Linux drivers.**

FUNCT.4	
Description	Ensure secure chained booting from Boot ROM to Hypervisor.
Rationale	For TRESCCA the hypervisor will be a critical component. It has to be made sure that the hypervisor is in a well-defined state and it has not been tampered with. Attacks against the hypervisor are threatening the whole TRESCCA platform. Therefore the integrity of the hypervisor has to be measured at boot time and maintained during runtime. Common practise is to rely on a chained integrity measurement mechanism to boot the system with its hypervisor into a trustable state. This functionality is essential for virtual machines and applications putting their trust into its hypervisor.
Reference	UC6, UC7
Priority	High
Result	**Requirement partly covered in scenario 1 through TEE. Supported but not fully validated using HSM-Mem.**

FUNCT.5	
Description	Provide a Software Trusted Platform Module.
Rationale	The Trusted Platform Module (TPM) is a hardware chip specified by the Trusted Computing Group (TCG). It provides cryptographic primitives and a secure storage location for encryption keys. TRESCCA will borrow some concepts from the TPM specification and may implement a software based TPM solution to achieve security features like trusted boot. The HSM-mem and

	its SSM-mem software driver itself will not offer any TPM functionality. The HSM-mem provides the base for a 100% software based TPM implementation that can also be used for protection at runtime. HSM-mem is responsible for enforcing confidentiality and integrity of external memory pages. To build a Software TPM on top of the HSM-mem will facilitate the ability to implement chained integrity checking and applications can benefit from a well-defined interface for cryptographic functionalities and key management. While a software-based TPM cannot build trust on its own, in combination with HSM memory protection this can be extended to a trustable base on which multiple instances will be possible. Especially for cloud application this is desirable.
Reference	UC2, UC6
Priority	Medium
Result	**Requirement partly covered in scenario 1 through TEE. No full support for TPM functionality.**

4.3 Security requirements

SEC.1	
Description	Guarantee that a hardware and software trusted compartment exists inside the light client
Rationale	The most important security objective of the TRESCCA platform in order to prevent that information cannot be compromised neither by software exploits or hardware attacks
Reference	UC2, UC6, UC7
Priority	High
Result	**Partly implemented in integrated prototype (HSM-Mem, HSM-NoC). TEE implemented on Versatile Express, Chromebook and Juno board.**

SEC.2	
Description	TRESCCA will ensure robustness against software denial of service
Rationale	The hardware and software extensions of the light client shall guarantee that, whatever the currently running applications in the untrusted area and the load they put on shared resources, the trusted compartment is still reactive and is granted a sufficient access to the shared resources to render its nominal services.
Reference	UC2
Priority	High
Result	**Requirement covered by HSM-NoC.**

SEC.3	
Description	Remote attestation mechanism between the cloud provider and the light client
Rationale	The attestation mechanism will certificate that the trusted compartment has not been tampered with. For applications that are making use of the distributed approach of TRESCCA need to rely on proper attestation of remote platforms. This includes cloud and client side. Before sending sensitive information or computing tasks the platform has to make sure that the other party is trustable
Reference	UC2, UC7
Priority	High
Result	**Requirement partly met by HSM-Mem, TEE and VM migration. However**

	not fully validated as part of a scenario or test.

SEC.4	
Description	Remote authentication of delegated processing in the light client
Rationale	The trusted compartment shall provide means to authenticate computing tasks that the cloud infrastructure or the end user delegates to the light client.
Reference	UC2, UC7
Priority	High
Result	**Requirement covered and validated in scenario 1 and 3 through TEE and VM migration.**

SEC.5	
Description	Software update and boot-to-boot integrity
Rationale	This is required to fix discovered bugs or security flaws or to add new features. Updating the software stack shall not compromise the trusted compartment. Upgrades of the trusted compartment shall be authentic and there must be a way for the cloud infrastructure and/or the end user to control the upgrade.
Reference	UC2, UC6, UC7
Priority	High
Result	**Requirement covered by HSM-Mem and TEE though not fully validated as part of a scenario.**

SEC.6	
Description	Isolation between applications
Rationale	The trusted compartment shall guarantee the perfect memory isolation between services unless they require memory sharing. The level of isolation is an important requirement. The highest possible isolation would be achieved by using completely different machines. To achieve a high isolation within a platform, TRESCCA will integrate it from bottom up and use hardware supported isolation. This is especially required when it comes to run different applications with different level of trust on the same hardware platform.
Reference	UC1, UC3, UC4, UC5
Priority	High
Result	**Requirement met in scenario 1, 2 and 3 through HSM-NoC, TEE and**

	secure hypervisor.

SEC.7	
Description	Robustness against software exploits and other logical attacks
Rationale	The consequences of a software exploit shall thus be limited to the attacked application (containment of software exploits), will be one of the key features of TRESCCA project.
Reference	UC1, UC2, UC6
Priority	High
Result	**Requirement covered by HSM-NoC and TEE. For TEE validated in Scenario 1.**

SEC.8	
Description	Computing and Storage Security
Rationale	When it comes to computing and storing confidential or sensitive data reliable functionality is needed. This includes the strength of used cryptographic algorithms, proper length of encryption keys and securely encrypted storage to store them as well as other sensitive data needed by applications.
Reference	UC2
Priority	Medium
Result	**Requirement not directly addressed in TRESCCA. Analysis of crypto algorithms as part of HSM-Mem development and application of good practices when designing security components.**

SEC.9	
Description	Network Security
Rationale	Whenever an application requires multiple parties for the fulfilment of a task it needs the assurance of the platform that data is delivered over a secured channel. This can be achieved by using end to end encryption via TLS connections.
Reference	UC2, UC7
Priority	Low
Result	**Requirement covered in all scenarios through use of secure communication. Though no new contribution by TRESCCA.**

SEC.10	
Description	Alteration Detection
Rationale	Alteration detection includes the protection of code as well as data. At no time it should be possible for an adversary to tamper with data or code. The highest goal is to prevent any attempts coming from malicious users.
Reference	UC2
Priority	Medium
Result	**Requirement covered in scenario 2 through HSM-Mem protecting integrity of data including protection against replay attacks.**

5 Scenario-specific evaluations and tests

5.1 Scenario 1 – Smart metering/energy management

5.1.1 Background

Wellness Telecom provides a cloud based energy monitoring platform, called WeSave. It consists of the cloud infrastructure and a Universal Monitoring Device (UMD) located at the end-user premises. In this scenario the UMD is represented and implemented on the Zedboard based demonstrator. The UMD is a Python application running on top of an ARM-based Linux OS. The UMD is responsible for collecting and storing metering data from connected smart meters. Additionally the UMD submits collected data to the cloud and provides a Web UI. This UI is used by administrators for configuration purposes and is the attack target. In this scenario it is assumed, that the attacker has physical access to the hardware and the skills and abilities to read out and manipulate any RAM area used by the application.

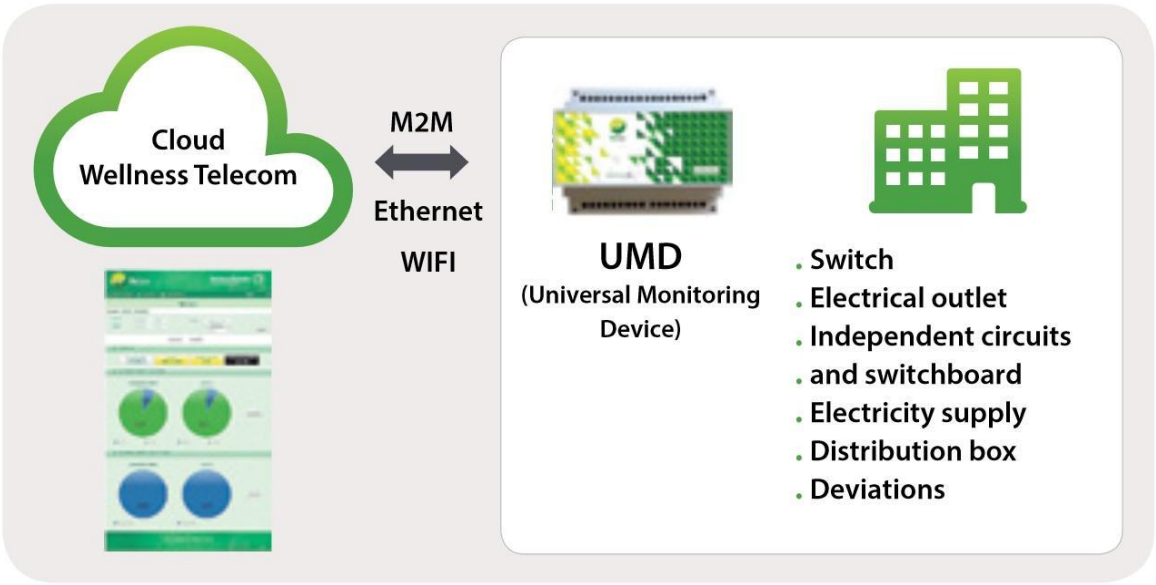


Figure 1 Architecture of WeSave application

5.1.2 Running the application

The application is started by typing “python main.py” at the command prompt which will then shows the following output to the console.

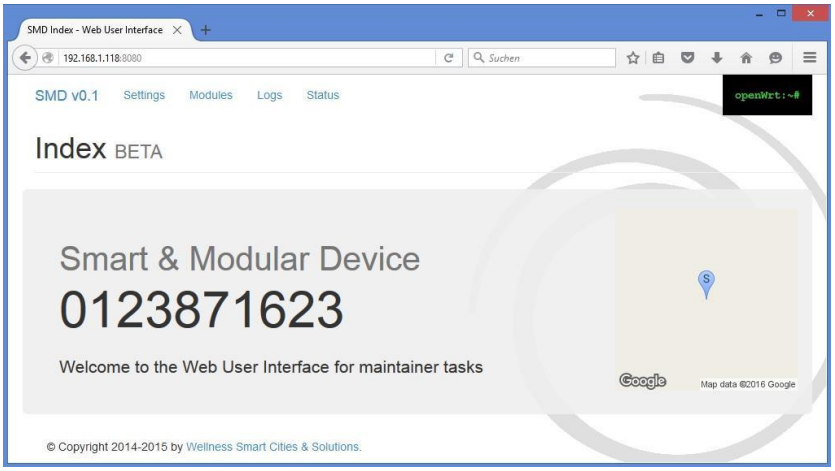
```
root@secbus> python main.py
['/mnt/usb/smd.git/src', '/usr/lib/python27.zip', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-linux
2', '/usr/lib/python2.7/lib-tk', '/usr/lib/python2.7/lib-old', '/usr/lib/python2.7/lib-dynload', '/usr
/lib/python2.7/site-packages', '/mnt/usb/smd.git/src', '/mnt/usb/smd.git/src/libs']
1970-01-01 04:26:54,085 [failsafe:INFO] Installed SIGTERM handler
1970-01-01 04:26:54,119 [failsafe:INFO] 256-bit Secure key installed
1970-01-01 04:26:54,122 [failsafe:INFO] Autodetected 6 apps: ['gatherer', 'base', 'ptp', 'sender', 'uc
av4', 'contactor']
1970-01-01 04:26:54,167 [failsafe:INFO] Main config successfully loaded
1970-01-01 04:26:54,168 [failsafe:INFO] Switching to /tmp/smd.log
1970-01-01 04:26:54,170 [main:INFO] Main config for SMD [0123871623] has been loaded
------------ ADDRESS ------------: 0x552e9740L
Bottle v0.12.7 server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:8080/
Hit Ctrl-C to quit.

1970-01-01 04:26:54,681 [main:INFO] Bottle.py server started at 0.0.0.0:8080
1970-01-01 04:26:54,699 [main:INFO] Adding log /tmp/coap.log for CoAP
1970-01-01 04:26:54,727 [main:INFO] CoAP server started at 0.0.0.0:5683
1970-01-01 04:26:54,738 [main:INFO] Adding log /tmp/soap.log for SOAP
1970-01-01 04:26:54,886 [main:INFO] SOAP server started at 0.0.0.0:8800
1970-01-01 04:26:54,906 [base:INFO] BASE app is ready for start
1970-01-01 04:26:54,908 [base:INFO] Loading specific API REST for base
1970-01-01 04:26:54,911 [main:INFO] Added REST resource GET /api/base
1970-01-01 04:26:54,913 [main:INFO] Added REST resource POST /api/base
1970-01-01 04:26:54,915 [main:INFO] Added REST resource PUT /api/base
1970-01-01 04:26:54,916 [main:INFO] Added REST resource DELETE /api/base
1970-01-01 04:26:54,919 [base:INFO] Loading specific API CoAP for base
1970-01-01 04:26:54,921 [main:INFO] Added CoAP resource "base"
/mnt/usb/smd.git/src/tools/launcher.py:35: ImportWarning: Not importing directory '/mnt/usb/smd.git/sr
c/apps/sender': missing __init__.py
  imported = getattr(__import__('%s.%s' % (package, module), fromlist=[module]), module)
1970-01-01 04:26:54,923 [failsafe:ERROR] Could not import module sender from package apps: No module n
amed sender
/mnt/usb/smd.git/src/tools/launcher.py:35: ImportWarning: Not importing directory '/mnt/usb/smd.git/sr
c/apps/contactor': missing __init__.py
  imported = getattr(__import__('%s.%s' % (package, module), fromlist=[module]), module)
1970-01-01 04:26:54,925 [failsafe:ERROR] Could not import module contactor from package apps: No modul
e named contactor
1970-01-01 04:26:54,927 [main:INFO] Thread MainThread is running
1970-01-01 04:26:54,928 [main:INFO] Thread bottleServer@0.0.0.0:8080 is running
1970-01-01 04:26:54,930 [main:INFO] Thread socketUdpRead@0.0.0.0:5683 is running
1970-01-01 04:26:54,931 [main:INFO] Thread SOAPServer@0.0.0.0:8800 is running
1970-01-01 04:26:54,932 [main:INFO] Going to sleep
```

The application has been modified to output the physical address of a sensitive data used by the application. In a real attack scenario this address would not be directly available to the attacker. However, there are several techniques to search and analyze any memory area for such sensitive data.

The marked line shows the physical address (0x552E_9740) of a “secret key” of the UMD.

When connecting to the Zedboards IP on Port 8080 with a browser, the WeSafe configuration UI is displayed including the value of the sensitive data.



5.1.3 Unauthorized Access

When an attacker has physical access to the DMU he may read out the content of some sensitive memory areas by sniffing directly on the wires between the CPU and the external RAM. In our scenario, we use the UART-Attacker to emulate this attack. The UART-Attacker is UART-controlled bus master located within the programmable logic (PL). Through commands received via the UART the attacker can read any RAM location in the Regular Address Space (RAS) from 0x0000_0000 to 0x1FFF_FFFF. The same memory area can also be accessed through HSM-Mem and HSM-NoC in the Alternative Address Space (AAS) from 0x4000_0000 to 0x5FFF_FFFF.

The UART module is connected via an RS232 interface to a PC. The PC runs two software tools, which can either read or write memory locations through commands send to the UART-attacker via the RS232 interface.

When HSM-Mem is deactivated, the attacker can read some sensitive data from the RAS by reading out the content of the mentioned address 0x552E_9740. (The actual address of that location in the RAS is accessed by subtracting 0x4000_0000 from the given physical address RAS_address = 0x552E_9740 – 0x4000_0000 = 0x152E_9740):

```
# ./u2m_m2f 0x152e9740 10 weSaveRAS
```

The given command writes the received data for that address to a file called “weSaveRAS”. The content of the file can be inspected by entering:

```
# cat weSaveRAS
```

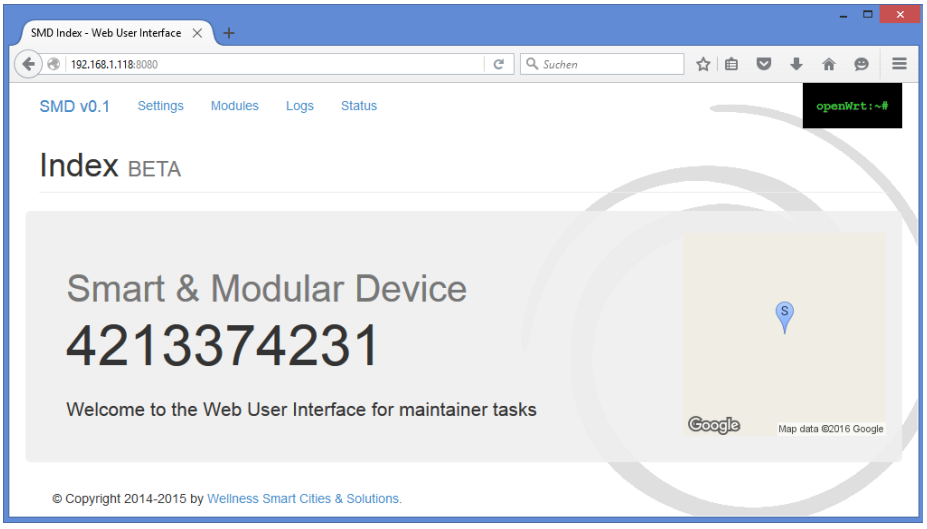
Which will output the following data:

```
# 0123871623
```

We now try to manipulate the data on that address by writing “4213374231” to this file and then writing it back to memory by entering:

```
# ./u2m_f2m 0x152e9740 weSaveRAS
```

This results in the WeSave Application displaying the manipulated data



This attack has demonstrated a simple scenario for attacking a real world application like the DMU of WeSave. In a similar way, any sensitive data of the application stored in the external RAM can be read out and manipulated without any notice of the application. While the application has been modified to output the physical address of a specific date, the attack would also be possible without it, given enough time to read out and analyze the memory of used by the application.

The next section will demonstrate how the HSM-Mem and the HSM-NoC security measures can be used to effectively prevent the demonstrated attack.

5.1.4 Defending the attack with HSM-NOC and HSM-Mem

The shown attack was realized by sniffing the signals directly from the wires to RAM which are available in address space from 0x0000_0000 to 0x1FFF_FFFF and can be described as an external attack. An attack can also occur as an internal attack, when the access to internal busses, e.g. through compromised hardware or manipulated USB devices. In our scenario, the UART-attacker additionally simulates an internal attacker in the address space from 0x6000_0000 to 0x7FFF_FFFF as shown in the demonstrator architecture.

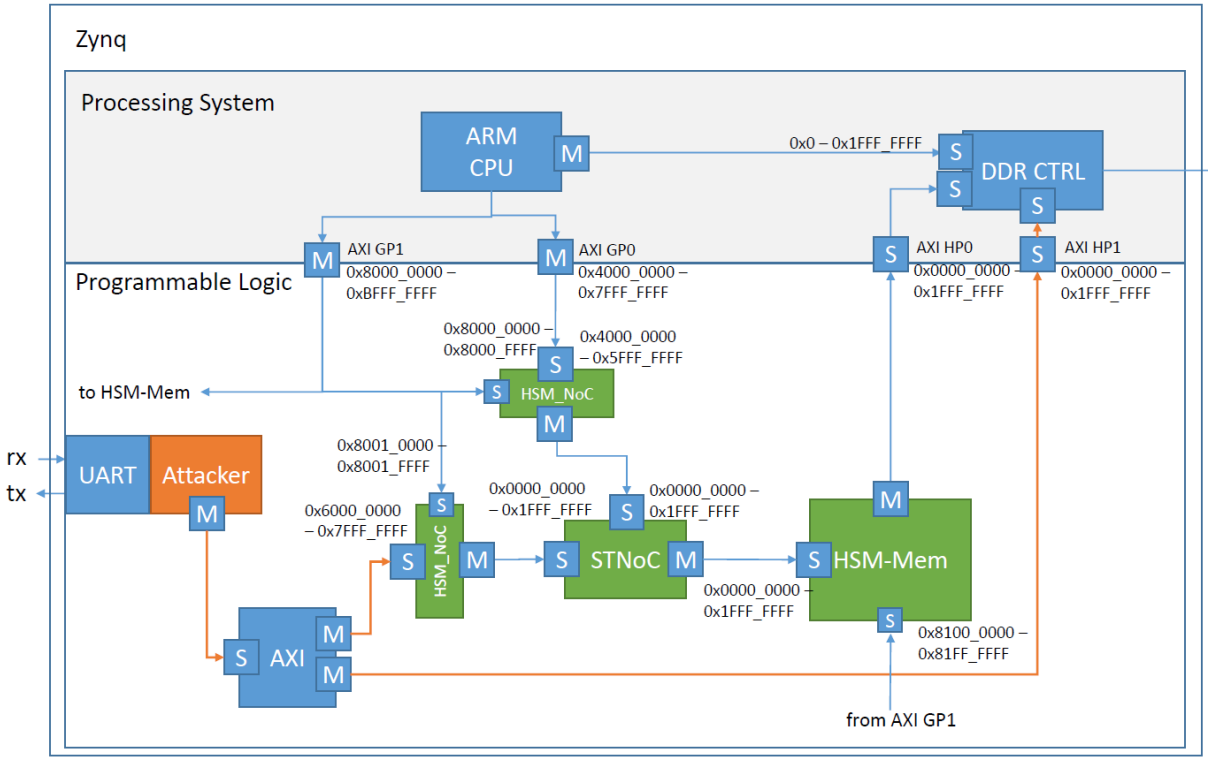


Figure 2 Architecture of TRESCCA prototype platform and address ranges.

5.1.5 Preventing an internal attack by HSM-NoC

The UART-Attacker is able to read out sensitive data (in this case only 4 bytes) by using the following command

```
# ./u2m_read 0x75819360
```

and shows

```
0x75819360: 0x33323130
```

Which are the first four bytes of the sensitive data string “0123871623”

After enabling the HSM_NoC driver

```
# insmod nocfw1.ko noc_address_base=0x80010000 paddress=0x55819360
```

and denying read access with

```
# echo 127 > /dev/noc_firewall_245
```

reading the same address

```
# ./u2m_read 0x75819360
```

results in

```
0x75819360: 0x00000000
```

After re-enabling the read access

```
# echo 255 > /dev/noc_firewall_245
```

and writing some new data to the regarding address with

```
# ./u2m_write 0x75819360 0x11223344
```

the new data can be displayed with

```
# ./u2m_read 0x75819360
```

and shows

```
0x75819360: 0x11223344
```

When the write access now is disabled by

```
# echo 191 > /dev/noc_firewall_245
```

Writing the old value with

```
# ./u2m_write 0x75819360 0x33323130
```

has no effect. A read command at the regarding address displays no change in the contained value

```
0x75819360: 0x11223344
```

5.1.6 Preventing an external attack by HSM-Mem encryption functionality

When HSM-Mems encryption functionality is active, every value written by the OS in the AAS is encrypted. Therefore the direct read out on memory wires (RAS) results in useless data. Accessing the same memory cells over the two different address spaces

```
# ./u2m_m2f 0x150a4360 10 weSaveRAS
# ./u2m_m2f 0x750a4360 10 weSaveAAS
```

results in encrypted

```
# cat weSaveRAS
```

```
|Z
```

and unencrypted data

```
# cat weSaveAAS
```

```
# 0123871623
```

An unauthorized write to 0x150a4360 with some new value e.g. “4213374231” contained in the file “writeNewValue”

```
# ./u2m_f2m 0x150a4360 writeNewValue
```

results in some useless content in the regarding address

```
# ./u2m_m2f 0x750a4360 10 weSaveAAS
```

```
# cat weSaveAAS
```

```
e▒c,▒T
```

5.1.7 Preventing an internal attack by HSM-Mem integrity check

An unauthorised write to 0x750a4360 with some new value simulates an internal attack via the HSM-NoC. The command

```
# ./u2m_f2m 0x750a4360 writeNewValue
```

results in

```
u2m_getc error - could not read char from FTDI device: 0 (all fine)
```

After this, the AXI-bus does not accept any new transactions, which means the systems freezes and no manipulation or reading of sensitive data in possible anymore.

5.1.8 Summary and conclusion

This section demonstrated and verified the functionality of the HSM-Mem and the HSM-NoC in the context of the WeSave application. The DMU components running on the remote device has been implemented on top of the Zedboard integrated prototype including the HSM-Mem and the HSM-NoC hardware prototypes. The program was modified to output the physical address of a sensitive data that is being attacked using the also integrated UART attack module.

As was shown, with no protection reading and manipulating the sensitive data through the UART attacker was easily possible. With HSM-Mem enabled, reading the data could be prevented through data encryption while manipulation could be prevented using the integrity protection feature of the HSM-Mem. The internal attack on the memory bus could be prevented by using a rule in the HSM-NoC firewall.

In this configuration the whole memory used by the Linux kernel and user processes was protected. This made protection of the DMU very easy as no further setup and manipulation was necessary.

Though there was a noticeable performance impact as all memory accesses have been processed by the HSM-Mem. In an improved setup it would be preferred to configure protection only for those parts of the memory that contain critical data.

5.2 Scenario 2 – Digital Right Management

Scenario 1 shows how TRESCCA features could improve the security of the DRM devices we use everyday. However, it is out of the scope for the project to provide a full featured and integrated DRM solution. In fact, the main objective of this scenario is to show how, integrating many of the TRESCCA concepts and features in a DRM system, it is possible to avoid attacks to the protected content and to the system's security assets.

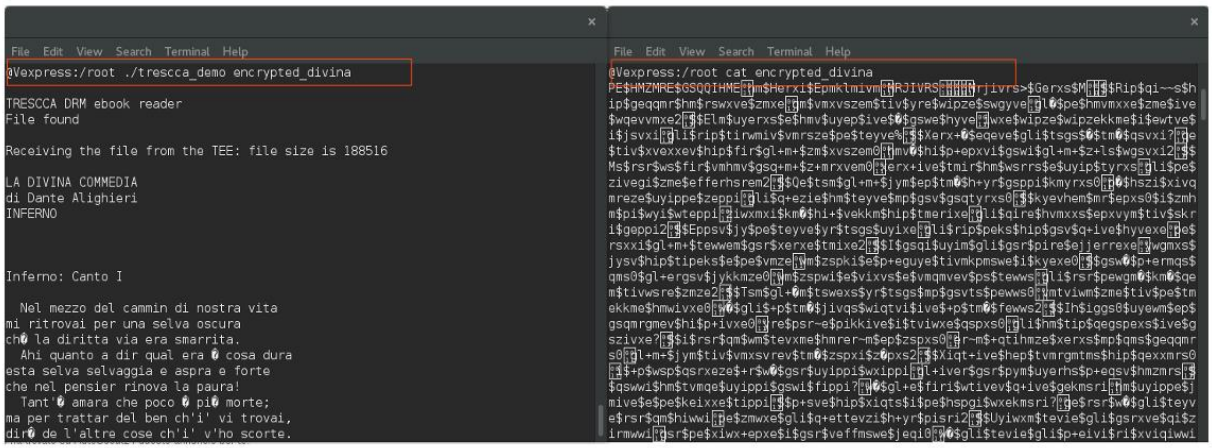


Figure 5-3*: DRM ebook reader evaluation*

For example, if an attacker in the guest tries to read a book file, the book content will be encrypted (right part of Illustration 1). An unencrypted copy of the file can't be found anywhere in the guest, because the decryption is done on the fly only when the TEE is involved in the DRM file reproduction (left part of Illustration 1).

Moreover, any unauthorized access from the guest and from the host to the security assets stored in the TEE are forbidden. This will be demonstrated by developing a TEE attacker program, which randomly tries to access to the Secure World applications.

Finally, in addition to ARM virtualization and security extensions, the DRM scenario could also benefit from other TRESCCA features such as HSM-mem (e.g., to protect external memory), HSM-NoC (e.g., to secure VM memory from attackers on the host) and hybrid migration (e.g., to move the DRM application at the content provider side to upgrade software).

5.3 Scenario 3 – Securing transactions or authentications

Scenario 3 is based in an E-commerce context, where features of TRESCCA shall be used in order to make the process more secure. However, it is out of the scope for the project to provide a full integrated E-commerce solution but an integration of the main components and concepts required, has been achieved. Thus far, the approach does not offer any direct security functionality and is by no means more secure. However, by integrating other components developed in TRESCCA the foreseen approach could demonstrate its benefits. Most promising is the integration of HSM-Mem to protect

external memory of moving VMs. Application developers can then decide to move to a target device in order to benefit from memory protection for sensitive operations.

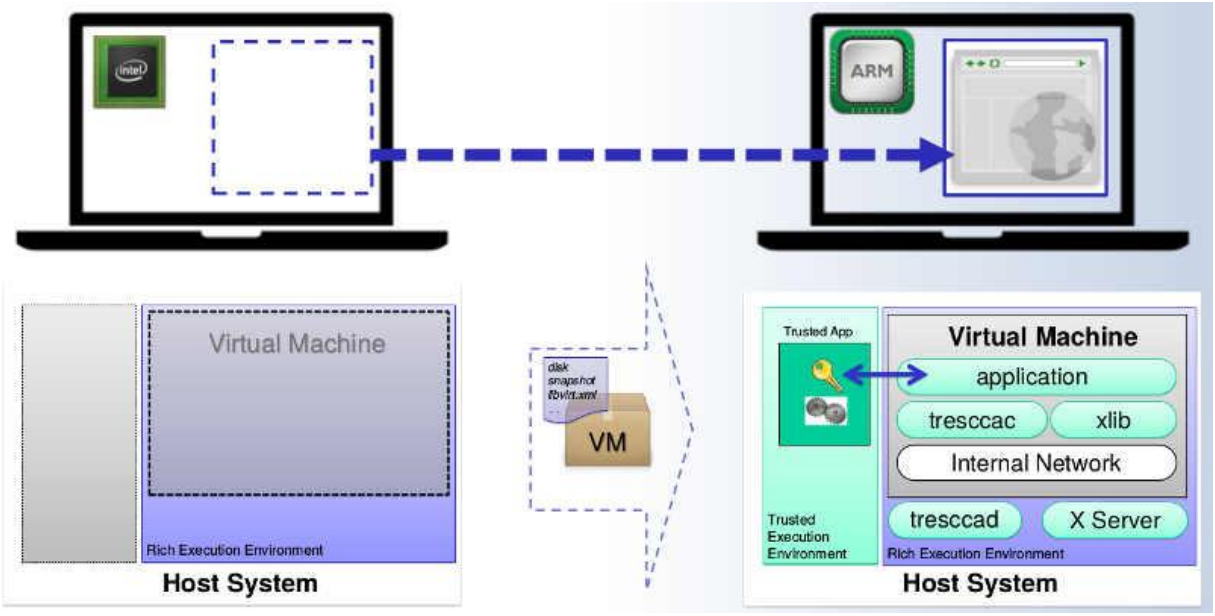


Figure 5-4 Illustration of implemented VM migration prototyp

The objective of this scenario is to demonstrate integrity and confidentiality of virtual machines while they are being migrated. VMs are transferred between cloud and client in a serialized package containing all necessary data to restore the current state of the VM. If an attacker gains access to the serialized snapshot of a given VM he can easily leak or tamper with the data which can be seen in **Fehler! Verweisquelle konnte nicht gefunden werden.**. It is inevitable to protect this data by sending it over a secured channel. Even if an attacker will manage to sniff communication, it will not be possible for him to resume execution of the VM in a foreign environment. This property can directly be demonstrated by placing a man in the middle attack.

```
root@zeus:~/inst# xxd snapshot | grep "my password is" -A 3 -B 3
0d30970: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0d30980: 0000 0000 0000 0000 0000 0000 0000 1100  ................
0d30990: 0000 0000 0000 0000 0000 0000 0000 890c  ................
0d309a0: 0000 6d79 2070 6173 7377 6f72 6420 6973  ..my password is
0d309b0: 2073 6563 7572 6520 2020 2020 2020 2020   secure
0d309c0: 2020 2020 2020 2020 2020 2020 2020 2020
0d309d0: 2020 2020 2020 2020 2020 2020 2020 2020
root@zeus:~/inst#
```

```
root@zeus:/home/bernd/work/trescca_code/Xagent# ssh root@10.0.0.2
root@zeus:/home/bernd/work/trescca_code/Xagent# ssh root@10.0.0.2
root@10.0.0.2's password:
/root # echo "my password is secure"
my password is secure
/root #
```

Figure 5-5:VM snapshots are exposed to integrity and confidentiality threats

Protecting VM packages during transfers from eavesdropping can be achieved in many ways. On the one hand TLS is used to protect the communication channel between two tresccad instances. Furthermore, a basic Public-Private Key Scheme has been implemented to protect VM packages. In this way, they can be encrypted using the public Key PK_Host#x of the respective destination host. These keys need to be exchanged before moving VM packages. Private keys need further runtime protection that may come from other TRESCCA components. However, for implementing a real use case, some PKI environment would be needed as well. Before executing foreign VMs, their origin would need to be attested to assure they were created by trustworthy parties. Required keys could either be stored in TPMs or the TEE. In this way, a key infrastructure would also allow signing VM packages on each host, so that their entire execution trace can be validated.

5.4 HSM-NoC performance and cost analysis

5.4.1 Hardware performance and area costs

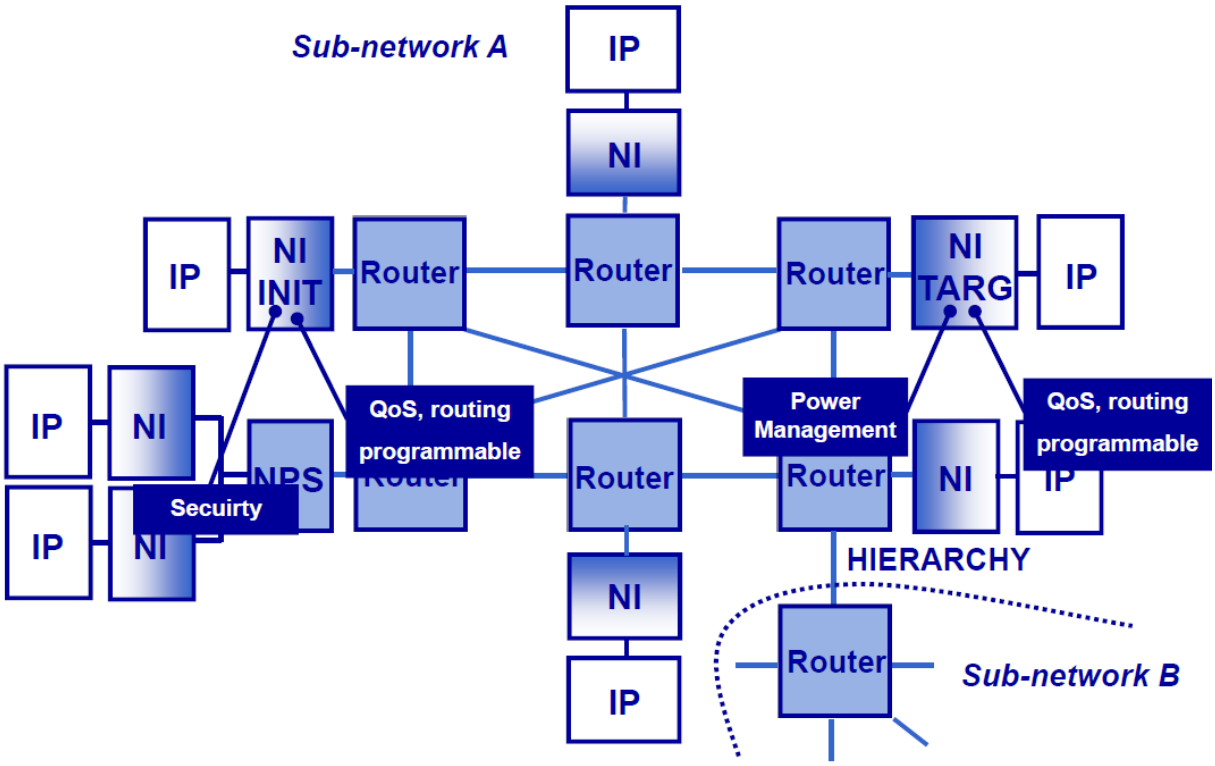


Figure 5-6 STNoC services at the network interface

The NoC Firewall service implemented at the initiator network interface (see *Figure 5-6*) is one of many lightweight services that can be deployed at the network interface. It is based on simple low-level primitives (coarse-grain segment-level protection) which can be virtualized to support isolation of virtual machines (and guest OSs) from each other by protecting both logical attacks (virus, Trojans) and security vulnerabilities, e.g. corrupt DMA engines.

Component	LUTs	Registers	RAMS	Frequency (MHz)
HSM-NoC (16 seg.)	655	1082	12	348
AXI4 Bridge	723	971		220.6
STNoC (3x7)	24939	17983		129.3
IOMMU	11150	9860	14 + 2.3K LUTRAM	204

Table 1 Synthesis results on the Zedboard Z7020 FPGA

As shown in Table 1, HSM-NoC synthesis results (area cost and frequency) for the Zedboard Z7020 FPGA compare favorably with other modules (STNoC and IOMMU) implemented on the same technology. The complexity is in fact even smaller than an AXI4 bridge. In terms of performance, HSM-NoC is non-intrusive, except when handling interrupts, which account for *90% of total access time*. For this reason, TEI is investigating into alternative kernel-assisted probing, as well as interrupt coalescing mechanisms.

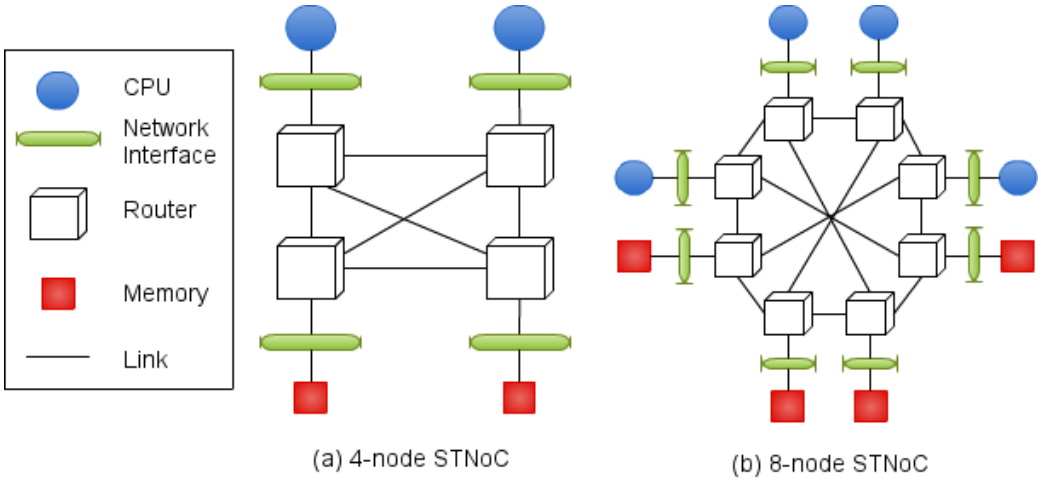


Figure 5-7 Spidergon STNoC system configurations to study the effect of security.We perform experiments for up to 32-nodes connected to STNoC.

Evaluation based only on the implementation of a NoC Firewall module on Zedboard is rather limited for extracting the value in terms of its implementation on top of a commercial NoC, such as STM's STNoC. In terms of scalability, we have examined behavior of the NoC Firewall (time-annotated based on RTL models) when it is implemented at the network interface for different network configurations (e.g. see *Figure 5-7*) based on a cycle-approximate STNoC instance modeled in Gem5;

a full version of this model will be released to Gem5 community in July 2016, while a preliminary beta version will also be available shortly as free software on http://gem5stnoc.sourceforge.net.

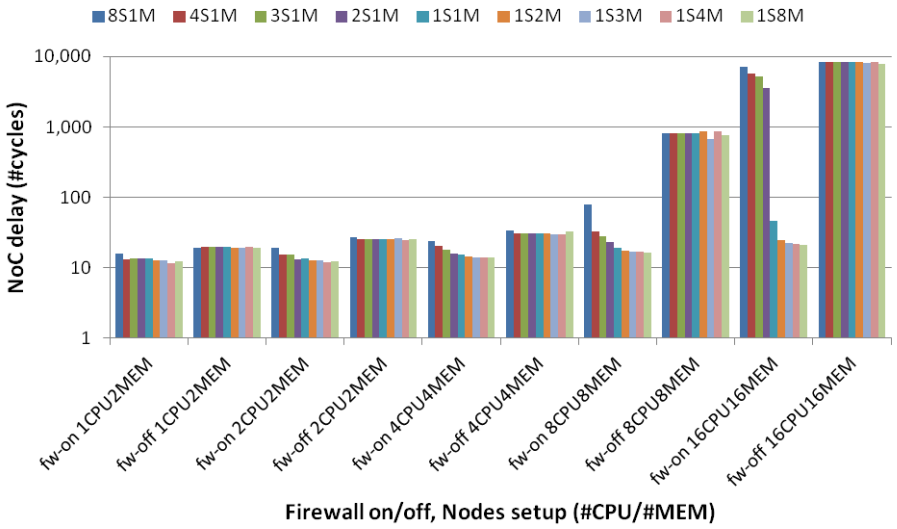


Figure 5-8 NoC delay for different STNoC setups, when firewall is on or off (logarithmic scale).

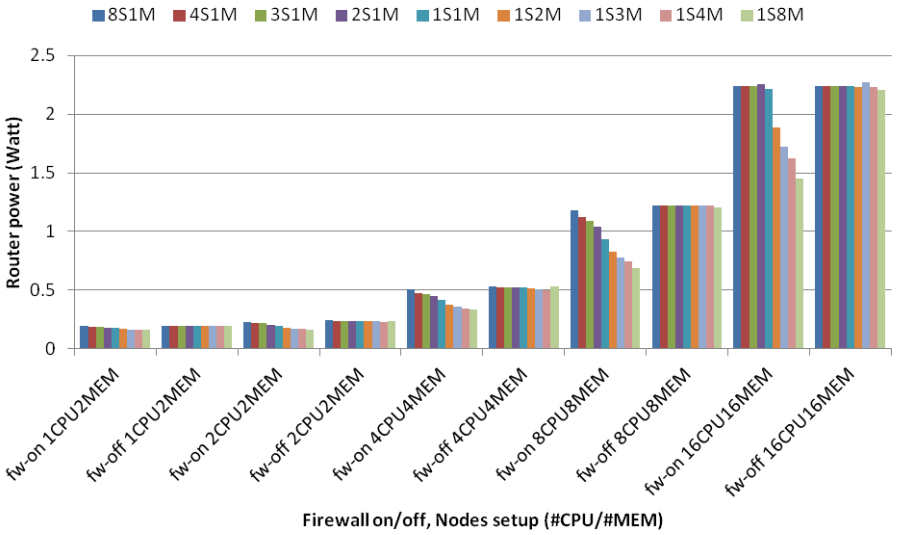


Figure 5-9 NoC power for different STNoC setups, when firewall is on or off.

Assuming a typical STNoC configuration, with delays of 1 cycle at the NI and 2 cycles at the router and link, a link width of 16 bytes, a 72 bytes 5-flit STNoC packet (with 8 bytes header and 64 bytes body) and with NI and router buffers sets to 5 flits, *Figure 5-8* and *Figure 5-9* show that an HSM-NoC implemented at each NI initiator can relieve the NoC from unnecessary load in the presence of malicious processes by stopping them prior to entering the NoC, thus reducing both NoC power and total NoC delay and often avoiding network saturation (e.g. in 8CPU/8MEM and 16CPU/16MEM cases). The greater benefits come when the number of relative malicious requests is large and also when the NoC topology is large.

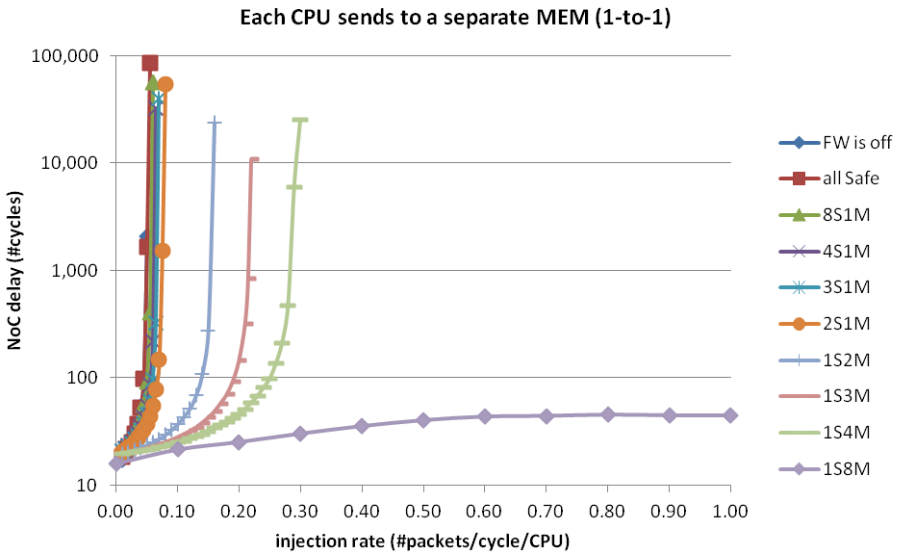


Figure 5-10 NoC delay for different packet injection rates per cycle per CPU (scale is logarithmic), in a setup with 2 CPUs and 2 memories interconnected with STNoC, link-width=8 Bytes. Injection rate is the same across all CPUs.

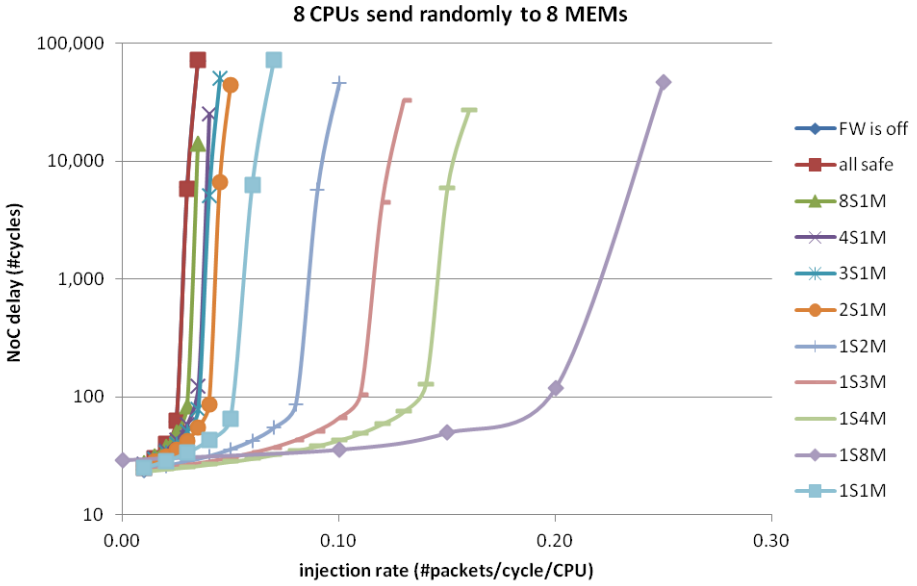


Figure 5-11 NoC delay for different packet injection rates per cycle per CPU (scale is logarithmic), in a setup with 8 CPUs and 8 memories interconnected with STNoC, link-width=8 Bytes. Injection rate is the same across all CPUs.

Network saturation is related to the packet injection rate, and possibly with the incapability of the network interface buffers to serve it, while the routing infrastructure and physical links eventually often remain largely underutilized. *Figure 5-10* and *Figure 5-11* consider simulations with a 4-node Spidergon (2 CPUs, 2 Memories) and 16-node Spidergon topology (8 CPUs, 8 Memories), respectively. When the firewall is disabled the NoC becomes saturated for smaller values of the

injection rate (#packets/cycle/CPU), i.e. less than 0:05 packets per cycle and CPU. On the other hand, when the firewall is active, as the number of malicious requests increases over the safe ones, the injection rate that the system can serve without becoming saturated increases. In fact, in some cases, e.g. 1S8M in 4-node network (see Figure x4), the firewall even prevents the system from becoming saturated at any injection rate. Network saturation is also affected by different STNoC parameters, e.g. link width or the number of packet flits. We have found that when the link-width is set to 8 Bytes the system is not able to break down the packets in flits within a reasonable time, and consequently the network experiences long delays.

5.4.2 Profiling of the NoC Firewall Driver

TEI has developed a single kernel module for HSM-NoC allowing simpler manipulation and providing the possibility to examine driver performance issues. In this scenario, the main module allocates two memory regions: one freely accessed where no rule exists (alternatively, an allow rule could be in place), and another one which is denied read and/or write access by the NoC Firewall. Then, two threads are generated using `kthread()` function: one *protected thread* accessing the rule-free memory region, and another *exploitation thread* accessing the protected memory region (hence, it falls in interrupt mode). Similar to the previous scenario, the exploitation thread represents a malicious (or corrupt) driver. All accesses to memory regions are routed via the NoC Firewall and are timed by calling the `ktime_get()` function to enable functional profiling and obtaining performance characteristics.

During module entry:

- *kmalloc*, *data initialization*, *virtual to physical* address translation, as well as writing a segment or rule register take on the order of 100 to 1K ns
- *request_mem_region*, *ioremap* (for range and rule registers) take on the order of 1K to 10K ns
- *ioremap* for IRQ register takes 10K to 100K ns
- registering the interrupt handler takes 70K to 700K ns
- creating each *kthread* is delayed 100K to 1.5M ns

During normal operation:

- *successful data reads* and writes take 100 to 1K ns, while reads that cause interrupt (denied read) take 10K to 100K ns
- *reads that cause interrupt* (denied read) take 10K to 100K ns;

During module exit: *freeing memory* takes 100K ns.

Based on the above profiling, in relation to performance, we note that interrupt handling under linux is very slow, taking approximately 90% of the total access time for both read and write accesses, thus ratifying our results from the bare-metal examples. This high interrupt costs can be relieved by redesigning the NoC Firewall, or implementing interrupt coalescing to reduce driver overhead.

5.4.3 Conclusions and outlook based on HSM-NoC evaluation tests

In order to shed some light towards future research directions in this exciting area, it is interesting to present a glimpse at the (hidden from this deliverable) complete HSM-NoC research world. Several results directly relate to HSM-NoC (NoC Firewall) research, including

- patenting virtualization-aware hardware mechanisms,
- *developing open source system-level models* (SystemC and Gem5) of the HSM-NoC and integrating them at the network interface of a cycle-approximate gem5 STNoC model for design space exploration, and
- designing the HSM-NoC architecture and an implementation in VHDL, performing synthesis and integration with STNoC, prototyping on Zedboard (and Versatile Express), design and profiling of bare-metal and GNU/Linux system driver and development of high-level security services; the latter is actually the **only HSM-NoC related result covered in this deliverable** (in Section 4.2).

HSM-NoC research described above has laid the foundations for further R&D and commercialization efforts concentrating on the following interesting topics.

- The **design of an enhanced process-aware NoC Firewall** which provides an improved interrupt service backward compatibility with previous version.
- The **design and integration of an extended NoC Firewall into each port of a multicast router**, with deny rules depending on physical address, as well as input and output ports.
- The design of distributed NoC Firewall synchronization mechanisms for consistent update of rules (add/delete/modify).
- **Integration of the NoC Firewall within an IOMMU**, either at device table and/or page walk interface, thus enhancing page-level protection.

5.5 HSM-mem hardware costs and performance overheads

This section is a summary of the performance evaluation of the HSM-mem. Its functional and security evaluation is presented in D4.2.

5.5.1 Hardware costs

The resources usage of the HSM-mem plus the attacker, plus the debugging facilities (not presented in D4.2) on the Xilinx Zynq-7020 FPGA, the Zynq core that equips the Zedboard prototyping board, is summarized in the following table:

Resource type	**Usage**
Glue logic (LUTs)	59.69%
Registers	3.67%
Memory (Block RAMs)	6.29%
DSPs	0%

The Zynq-7020 being the second smallest member of the family and the FPGA technology being far less dense than its equivalent integrated circuits technology, this can be considered as very reasonable. Using the (usually optimistic) Xilinx conversion ratio this can be estimated as about 750 k-gates or 3M transistors. Manufactured in a 14 nm process the HSM-mem would occupy a silicon area of about 0.3 mm².

5.5.2 Memory footprint overhead

The flexibility of the HSM-mem protection, the ability to fine-tune it and to control it by software comes at a price: the data structures used by the control are stored in the external memory. Moreover, the Message Authentication Codes (MAC) use to protect the integrity of memory pages are also stored in external memory.

The data structures occupy exactly 1/256 (0.39%) of the total memory space to protect. A 4GB memory space, for instance, requires 16MB of data structures.

An integrity-protected read-only memory page adds to this a set of MACs that occupy 25% of the size of the memory page. This overhead is 33% for integrity-protected read-write memory pages.

All in all, a system with 4GB external memory, 256MB of integrity-protected read-write memory and 1GB of integrity-protected read-only memory, would dedicate 16+256/3+1024/4=357MB (8.7%) of its external memory to the data structures and MACs. Of course, this highly depends on the type of applications and system memory layouts.

5.5.3 Performance overhead

In order to estimate the performance overhead introduced by the HSM-mem, several experiments have been conducted in 21 different configurations that differ by the monolithic security policy applied to the memory space of the software stack (the Linux kernel and the software applications running on top of the kernel). They differ also by the granularity of the protection: the HSM-mem operates on memory pages and supports various page sizes (4kB, 64kB, 1MB and 16MB). For various reasons that go beyond the purpose of this chapter, both the security policy and the page size on which it is applied impact the performance overhead.

In all these configurations a complete software stack (Linux, BusyBox, applications) has been run. On top of it, several classical benchmarks for integer processing (Dhrystone), floating point processing (Whetstone) and memory bandwidth (RAMSpeed, RAMSMP) have been used. Finally, the Linux kernel initialization time has also been measured using the timed kernel messages.

The detailed results are presented in a dedicated chapter of the D2.4 deliverable. Two important remarks must be considered when analysing these results:

- In 20 of the studied configurations the complete memory space is protected with a given, unique, security policy. They are a worst case. In a realistic situation only memory pages requiring protection would be protected and only with the lightest cryptographic primitive that fulfils their security requirements. The results presented must thus be considered as absolute upper-bounds. Implementing and evaluating more realistic situations, while perfectly feasible, would have required huge software developments, including significant modifications to the memory management sub-system of the Linux kernel, going far beyond the scope of TRESCCA.

- The performance overhead introduced by the HSM-mem is mainly due to the increased external memory latency: before returning read data to the processor, the HSM-mem must discover what security policy to apply and apply it. A first consequence is that it is only on CPU cache misses that the overhead is paid for. A second consequence is that, as the HSM-mem itself implements several cache-based optimization techniques, the overhead, when any, highly depends on the regularity of the memory accesses.

5.5.4 Conclusions on HSM-mem performance evaluations

The hardware cost of the presented version of the HSM-mem is comparable to that of a typical hardware accelerator and can be considered as perfectly acceptable for the typical TRESCCA edge device target.

As can be seen from the D2.4 results, the Dhrystone and Whetstone benchmarks show no or negligible differences between the configurations. This is due to the fact that most of the time they entirely fit in the CPU caches and / or HSM-mem caches. The Linux start and init times are more interesting because they are typical of an application that fetches a lot of non-cached data and code from the external memory. The overhead introduced is more significant because of the very low level of locality. It ranges from 1.3x to 50x slowdown. As logically expected, the worst overhead corresponds to configurations where the integrity is protected by Merkle MAC trees and the confidentiality by Cipher Block Chaining. Moreover, for any given security policy, it is worse for fine grain and better for coarse grain granularity.

The memory bandwidth benchmarks show similar results: for small data blocks (1kB) the overhead is very limited (less than 10% in the worst case) because most of the accesses fall in the CPU caches and those which do not highly benefit from the HSM-mem internal caches. For very large blocks (32MB) that render the CPU and HSM-mem internal caches completely useless, however, the overhead ranges from 1.6x to 90x bandwidth reduction. And there again, the configurations where the integrity is protected by Merkle MAC trees and the confidentiality by Cipher Block Chaining are the worst and, among them, the finest the granularity, the worse the overhead.

The memory footprint and performance overheads, in the non-realistic worst cases, can be significant but the protected device is still usable. In realistic situations where only the most sensitive assets are protected and only with the most appropriate cryptographic primitives, these overheads would probably be perfectly acceptable.

5.6 Validation of integration effort

During the development of the integrated platform as reported in D4.1, the components developed in WP2 and WP3 have been integrated into one demonstration platform. The following table lists the evaluation results of that integration process with respect to interoperability and integration effort for the different components.

It should be noted, that effort is only estimated and validated on a technical level, analysing the complexity and compatibility of the given technology with todays common System-on-Chip design. No actual effort in terms of cost and effort are given.

Component	Level	Results	
HSM-Mem	On-chip communication level	The resulting HSM-Mem hardware design provides a 32-bit AXI-bus interface. This interface is very common in ARM-based System on Chips. Integration into the Zedboard prototype was very easy as the whole design was based on AXI. Internally the HSM-Mem uses a VCI bus interface. The adaptor can be used with different	+ + +

		bus adaptors to use the same components with different bus interfaces.	
	System-on-Chip level	The HSM-Mem works on the level of memory addresses and data. No specific customization to different System-on-Chip architectures, not even instruction set architectures would necessary.	+ + +
	Software level	Software integration proved to be more challenging. No full integration into Linux or any other operating system is available yet. Configuration is generally possible from operating system, though full integration requires modification of memory manager of that operating system. Currently only full protection of operating system and application enabled by First Stage Bootloader is available.	- -
HSM-NoC	On-chip communication level	HSM-NoC is designed with an AXI-interface. Integration into the Zedboard prototype using AXI-interconnect was possible without extra effort. Integration with other on-chip bus architecture would require modification of HSM-NoC interface.	+ +
	System-on-Chip level	HSM-NoC rules are defined and checked on the level of physical address and segments. This works practically in all system and processor architectures. However if rule checking should be extended towards Virtual Machines, more changes and specific support are needed for different architectures.	+ +
	Software level	Linux drivers for HSM-NoC are available enabling setting and checking of access rules from applications. Though currently no direct support is yet available which would make sure that only the kernel or the hypervisor enables or disables rules during run time.	+
Secure Hypervisor	System-on-Chip level	The Secure Hypervisor could not be integrated on the Zedboard due to the limitations of the underlying prototyping platform. However it was prototyped and tested on the VersatileExpress and the Juno prototyping systems. The Secure Hypervisor requires hardware virtualization support available through the KVM	-

		hypervisor and the ARM TrustZone hardware feature. This limits the use of the Secure Hypervisor to ARM-based Systems with hardware virtualization support. Though all recent SoC products released during the last two years include that features. Secure Hypervisor is not dependent on TRESCCA HSMs though it would benefit from the additional security. It is up to the application and the developer whether this additional security is necessary. However, products integrating TRESCCA HSMs will not be available before 2017.	
	Software level	No full integration has been done on the Zedboard prototype. The prototypes based on VersatileExress and Juno provided all necessary features to setup and use the Trusted Execution Environment of the Secure Hypervisor from applications.	+

The development of the integrated platform has proven the general applicability and compatibility of the developed TRESCCA hardware components. Though the Secure Hypervisor could not be integrated into the same platform due to limitation of the prototyping system. This limited the validation of the Secure Hypervisor and prevented using the Secure Hypervisor in combination with the TRESCCA HSMs.

Also there are several requirements for integrating the HSMs into a System-on-Chip design. There is no full compatibility with any architecture. Though the HSMs have shown to be adaptable to different architectures or they at least support one very common bus architecture like AXI and the ARM processor architecture. The effort for integration into future products is roughly comparable to any other existing hardware IP components available on the market. Given the fact, that IP-based design (design of SoCs based on third party IP core) is very common nowadays, integration of TRESCCA HSMs does not generate more effort and cost than other typical IP components.

Software support for the HSMs is still limited. While there are drivers and example source code files are available to initialize and configure the HSMs, more software is needed to integrate the HSM features on the functional level of operating system. Currently it is either impossible or very difficult to use specific HSM features from an application. For example the HSM-Mem could be configured from the Linux kernel, however integration with applications would only work, if the memory manager of the kernel is modified to be HSM-Mem aware, so that allocated memory segments could be enabled with a configurable security policy.

6 Summary, conclusions and outlook

This report presented an overview on the activities and results generated by the TRESCCA consortium in evaluating and validating technologies and application scenarios developed by the project.

In general, evaluation of TRESCCA has shown to be very challenging. There is a significant gap between the low level hardware and software security components and the general cloud-based applications scenarios identified in WP1. In some cases, this gap could not be fully bridged. However the security technology developed by TRESCCA has been demonstrated and evaluated to full extend and also proved to be applicable to a complex use case such as a Smart Meter Gateway (WeSave).

There are several approaches for evaluating and validating security solutions, though they are not generally applicable. While TRESCCA has chosen the Scenario Based Design method, the approaches chosen for evaluation were different depending on the evaluated technology, ranging from hardware prototypes over proof-of-concept software demos to complete applications. This is mostly due to the fact that the evaluation had to cover hardware security modules, protecting either on-chip or off-chip communication as well as Virtual Machine migration and virtualization-based Trusted Execution Environments.

The Consortium revisited the requirements, use cases and security threats initially defined in D1.1 and D1.3. While some of them could be covered by TRESCCA directly, others were too general to be addressed by TRESCCA. In this cases, future directions or potential uses of TRESCCA technology to address such requirements were given.

The functional specifications and requirements of the different TRESCCA technologies have all been validated in prototypes and proof-of-concept implementations. An integrated prototype running the WeSave application and including all components (except virtualization-depending components) has proven that the components can be combined and orthogonally enhance the security of embedded cloud edge devices.

Though limitations of the underlying prototyping platforms prevented a full integration. The Consortium had to find a good trade-off between cost, complexity and limitations of the prototyping platform. The Zedboard proved to be the best trade-off but at the cost of not being able to integrate hardware-supported virtualization. Still alternative prototype platforms, such as the Versatile Express and the Chromebook enabled validation of those technologies that could not be fully integrated.

There are several different directions in which the outcomes of TRESCCA can be used and exploited. For instance, the hardware security could be scaled up towards integration into server platforms but also scaled down towards smaller and more light-weight systems. Also TRESCCA technologies could be used in future R&D projects in a more application centric scenario. In this case, the gaps between the security technology and the cloud application scenarios could be bridged by an additional middleware layer, making the security provided by the hardware easier to access and exploit by the software services.