PREparing SEcuRe VEhicle-to-X Communication Systems

Deliverable 3.1.1

FOT Trial 1 Results

Project:	**PRESERVE**
Project Number:	**IST-269994**
Deliverable:	**D3.1.1**
Title:	**FOT Trial 1 Results**
Version:	**1.0**
Confidentiality:	**Public**
Editor:	**Renault**
Cont. Authors:	**J. Petit, M. Feiri, D. Broekhuis, B. Lonc, R. Moalla, F. Kargl**
Date:	**2013-02-18**

Part of the Seventh Framework Program
Funded by the EC-DG INFSO

Document History

Version	Date	Main author	Summary of changes
v0.1	2013-01-07	B. Lonc, R. Moalla (Renault)	Initial structure of document created
v0.2	2013-01-18	J. Petit (UT)	Inputs for FOT 1
v0.3	2013-02-10	D. Broekhuis (UT)	Figures, Setup
v0.4	2013-02-14	M. Sall (Trialog)	Input for Assessment
v1.0	2013-02-15	B. Lonc (Renault), R. Moalla (Renault), M.Feiri (UT), F. Kargl (UT)	Finalization of the document

Table 1: PIM methods

Approval		
	Name	Date
Prepared	B. Lonc	2013-02-15
Reviewed	All Project Partners	2013-02-18
Authorized	Frank Kargl	2013-02-18

Circulation	
Recipient	Date of submission
Project Partners	2013-02-18
European Commission	2013-02-18

Contents

List of Figures

List of Tables

Glossary

Abbrev	Synonyms	Description	Details
API		Application Programming Interface	An API is a particular set of specifications that software programs can follow to communicate with each other.
ASIC		Application-Specific Integrated Circuit	As its name suggests an ASIC is an integradted circuit specifically customized for a particular function
CL		Convergence Layer	Module that connects the external on-board entities (e.g. communication stack or applications) to the PRESERVE Vehicle Security Subsystem (VSS)
CRS		Cryptographic Services	Module acting as proxy for accessing different cryptographic algorithm implementations. Originates from the EVITA project
EAM		Entity Authentication Module	Module responsible for ensuring entity authentication of in-vehicle components. Originates from the EVITA project
FPGA		Field-Programmable Gate Array	A FPGA is an integrated circuit that can be reprogrammed by the customer after manufacturing
HSM		Hardware Security Module	
IDE		Integrated development environment	An IDE is a software application that provides comprehensive facilities to computer programmers for software development
IDM		ID and Trust Management Module	Module responsible for ID management originating from SeVeCom project.
OEM		Original Equipment Manufacturer	Refers to an generic car manufacturer

Abbrev	Synonyms	Description	Details
PAP		Policy Administration Point	Module related to the PDM originating from EVITA project
PC	Short Term Certificate	Pseudonym Certificate	A short term certificate authenticates stations in G5A communication and contains data reduced to a minimum.
PCA		Pseudonym Certificate Authority	Certificate authority entity in the PKI that issues pseudonym certificates
PDM		Policy Decision Module	Module responsible for enforcing the use of policies originating from EVITA project
PDP		Policy Decision Point	Module related to the Policy Decision Module originating from EVITA project
PeRA		Privacy-enforcing Runtime Architecture	Module responsible for enforcing privacy protection policies originating from PRECIOSA project
PEP		Policy Enforcement Point	Module related to the Policy Decision Module originating from EVITA project
PIM		Platform Integrity Module	Module responsible for ensuring in-vehicle component integrity originating from EVITA project
PKI		Public Key Infrastructure	A PKI is a set of hardware, software, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates.
PMM		Pseudonym Management Module	Module responsible for management of the station's pseudonym certificates originating from SEVECOM project
SCM		Secure Communication Module	A generic name for the complete secure communication stack
SEP		Security Event Processor	Module responsible for security event management (e.g. checking message plausibility, station reputation calculation)
VSS		V2X Security Subsystem	Close-to-market implementation of the PRESERVE VSA that is the outcome of PRESERVE work package 2
WSU		Wireless Safety Unit	

1 Introduction

This deliverable D3.1.1 intends to summarize the evaluation of measurements performed on PRESERVE FPGA-based V2X Security Subsystem (VSS) through two distinct field-testing activities of the V2X security system:

1. An internal FPGA-based VSS test done at University of Twente, called Internal FOT Trial 1 (IFT1).
2. A joint trial with the Score@F French FoT project (JFT), using the VSS integrated into the OBUs and RSUs platforms developed by Score@F.

Note: The joint tests with Score@F are currently not covered in this initial draft deliverable (D3.1.1) but will be covered in a second version (D3.1.2). This is due to Score@F project delays and priorities given by Score@F partners to user experimentations on natural/open roads environment in Versailles/Yvelines area. Consequently, the time schedule for testing security and safety applications on SATORY tracks is constantly changing and pushed towards the end of Score@F project (planned for 2013-Q2 or Q3).

Section 2 presents the assessment plan for PRESERVE VSS implementation based on the FESTA test methodology. This section details the steps of the test methodology, the use cases and research questions (regarding the challenges of PRESERVE solutions for security and privacy-protected V2X communications), and the performance indicators and measurement procedures used to evaluate PRESERVE VSS implementation. Section 2 integrates the specification of a list of test cases that can be used in various trials during the project duration. It was initially prepared as an Technical Report 4 “Testing Handbook” for dissemination to other projects (e.g. FOTNET, Drive C2X).

Section 3 presents the two initial PRESERVE trials: the internal and the joint test. For both, the test environment and set-up, the test purpose and main functions and operational requirements which are being tested during the concerned field-testing activities is presented. D3.1.1 focuses on the internal trial activities only (based on VSS Kit 1).

Section 4 presents the evaluation results of the internal field-test at UT. This deliverable D3.1.1 includes preliminary conclusions, based on first measurements evaluation from the Internal FOT Trial 1 (IFT1).

The tests presented in this first version (D3.1.1) primarily serve to validate the correct functionality of PRESERVE VSS Kit 1 and to provide initial performance measurements. The latter are primarily aimed at demonstrating the insufficiency of pure software-based solutions. While we also benchmark the FPGA-based solution, it is important to note that the FPGA has the primary purpose to validate the later ASIC design and to allow early integration of hardware. It also proved useful in discovering problems with USB connectivity

that is currently still enhanced and tuned. The FPGA is therefore not optimized for speed and cannot serve as an indication of later ASIC speed.

2 Assessment plan of VSS Kit 1

This section presents the assessment plan of VSS Kit 1 (HSM SW version and FPGA-based HSM). It describes the test bench and test environment used for the assessment of VSS Kit 1 and especially presents a logging facilities developed for such testing. Secondly the specification of test cases used to validate VSS Kit 1 is given and the test results of assessment is presented in section 2.3.

2.1 Test bench for validation

The validation tests of VSS functionalities were performed using VSS Kit 1, version 1.3.1. The figure 2.1 presents the configuration of the test bench.

In order to be able to evaluate the performance of our VSS implementation, the class **LogMemoryAndTimeStats** has been implemented. With the help of other internal classes, it offers to the caller the possibility to know the time consumed by a function. For convenience' sake, some macros have been defined. Thus, for knowing the time consumed by a method, the implementer has just to add CHECKPOINT(label) at the beginning of his method where label is any text that identifies the checkpoint. When this method is entered, the current time is registered (with the label). When the control leaves the method, the corresponding time is automatically registered (again with the label). All the checkpoints are stored in one or several records that can be analyzed afterwards. The implementer can also used the mechanism described above for storing special events like the result of a function (e.g. the result of the verification of a signature). The memory consumption can also be traced by the class **LogMemoryAndTimeStats**.

This mechanism can be activated or deactivated in the configuration file of the VSS (attribute **logging_with_statistics**). All the logging data are periodically stored in one file. But in case of platform with a limited disk space, a special mechanism has been implemented. This mechanism is triggered by the parameter **logging.report.circular** When it is present, a new file is created each time logging data must be stored. This file is suffixed by an counter. The value of the parameter indicates the maximum number of files that can be created. When this number of files is reached, the counter is set to 1. For example, if the value of the parameter is 20, the files logging.1 up to logging.20 are created. And the file logging.1 is created again (replacing the file with the same name) and so on.

It has been used for measuring the time consumed for signing a message and for verifying the signature of a message. Some results on different platforms can be found in the next sections.

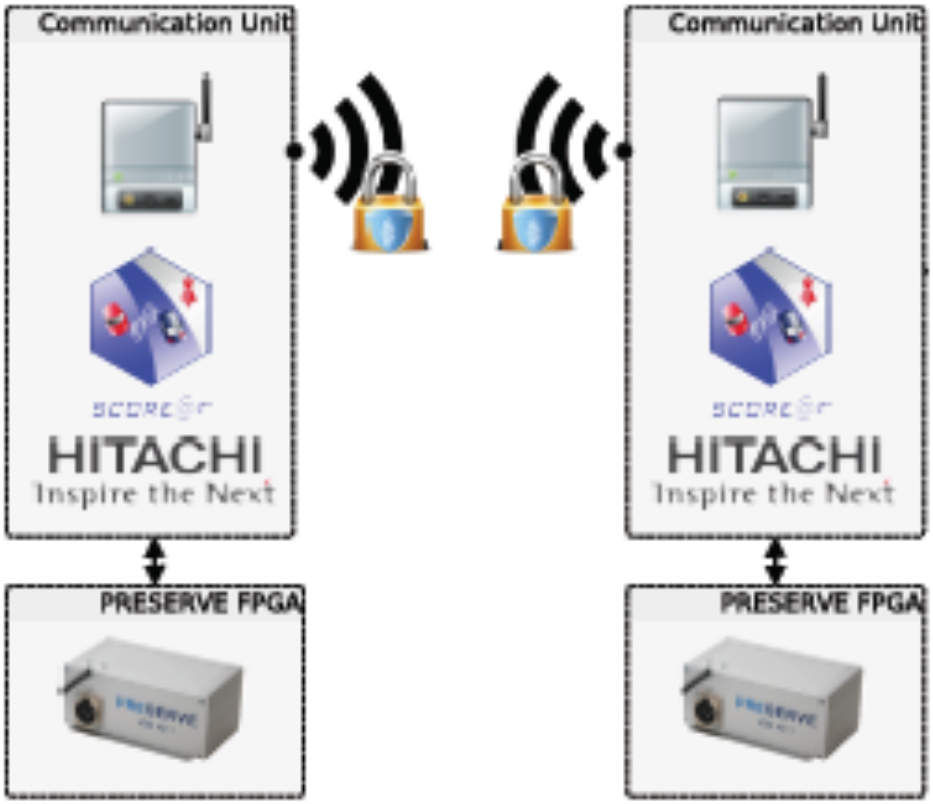

Figure 2.1: The Denso and Cohda modems

2.2 Test of the VSS implementation

In this section we describe the different tests that verify that all the functionalities implemented in the VSS are working as expected. This description is divided in two parts. The first part concerns the test of the underlying functions that support the functionalities visible to the FOTs. The second part tests the functionalities required by the FOTs.

A big part of the behaviour of the VSS can be configured in the configuration file which name is **pcom.cfg**. In particular, the directory where the certificates are stored, the directory where the cryptographic keys are stored for the software version of the HSM.

2.2.1 Test of the basic functionalities of the VSS

Prior to be able to offer services to the FOTs, the VSS must have a bunch of certificates (at least one long term certificate and one or more pseudonyms). The first step is then to test that the VSS is able to obtained certificates from a PKI.

- The communication with the PKI in order to obtain new certificates. In Preserve we have two PKIs, so, we passed the tests with the two PKIs.
- The request of a long term certificate from both PKIs.
- The request of n pseudonyms from both PKIs.
- The request of n pseudonyms for the one PKI with the use of the long term certificate obtained from the other PKI. The objective here is to test the interoperability of the two PKIs.

These high level functionalities use a great number of underlying functions which have been tested individually. Among these functions are the following ones:

- The creation and the management of 1609.2 messages
- The serialization and the deserialization of 1609.2 messages
- The creation of symmetric cryptographic key
- The creation of private/public cryptographic key pair
- The import of a remote cryptographic key
- The export of a public cryptographic key
- The signing of a bloc of data,
- The verification of a signature associated to a bloc of data
- The cipher of a bloc of data
- The deciphering of data

2.2.1.1 Test_LTCREQ_VB_01

Purpose: Verify that a long term certificate request can be issued and the response was treated correctly

Precondition: A LTCA must be reachable

Context: The software version of the HSM is used

The PKI of Fraunhofer is used

Procedure	Remarks
Preamble: Remove all the keys from the HSM	This is not mandatory. It is just for being sure that there is enough space in the HSM for creating new keys

Procedure	Remarks
Body: Creation of the long term certificate request Ciphering of the request The request is sent to the LTCA The response from the LTCA is received The response is deciphering The LTC is extracted It's signature is verified The LTC is stored	 The request is ciphered with the IDK and the public key of the LTCA Verify that the LTC has been effectively stored in the repository together with the corresponding .meta that contains the HSM key handles created for the LTC
Postamble:	Verify that the HSM keys associated to the LTC are still there

2.2.1.2 Test_LTCREQ_VB_02

Purpose: Same purpose as above but the PKI of Escrypt

2.2.1.3 Test_LTCREQ_VB_03

Purpose: Same purpose as above but with the FPGA

2.2.1.4 Test_PSNYMREQ_VB_01

Purpose: Verify that a pseudonym certificate request can be issued and the response was treated correctly

Preconditions: A PCA must be reachable

A long term certificate must exist and must be valid. It must be stored in the directory specified in the configuration file

The HSM keys associated to the long term certificate must exist. They are in the directory specified in the configuration file

Context: The software version of the HSM is used

The PKI of Fraunhofer is used

Procedure	Remarks
Preamble:	
Body: Creation of the pseudonym certificate request Ciphering of the request The request is sent to the PCA The response from the PCA is received The response is deciphering The pseudonyms are extracted Their signature is verified The pseudonyms are stored	The request is signed with the LTC and ciphered with the IDK and the public key of the PCA
Postamble:	Verify that the pseudonyms have been effectively stored in the repository together with their corresponding .meta that contains the HSM key handles created for the different pseudonyms Verify that the HSM keys associated to the pseudonyms are still there

2.2.1.5 Test_PSNYMREQ_VB_02

Purpose: Same purpose as above but with the PKI of Escrypt

2.2.1.6 Test_PSNYMREQ_VB_03

Purpose: Same purpose as above but with the PCA of Escrypt and the LTC obtained from the LTCA of Fraunhofer

2.2.1.7 Test_PSNYMREQ_VB_04

Purpose: Same purpose as above but with the PCA of Fraunhofer and the LTC obtained from the LTCA of Escrypt

2.2.1.8 Test_PSNYMREQ_VB_05

Purpose: Verify that a pseudonym certificate request can be issued and the response was treated correctly with the FPGA

Precondition: A PCA must be reachable

A long term certificate must exist and must be valid. It must be stored in the directory specified in the configuration file

The HSM keys associated to the long term certificate must exist. Either pre-defined keys are used or the FPGA has not been reloaded between the long term certificate request and the pseudonym request.

Context: The FPGA is used

The PKI of Fraunhofer is used

Procedure	Remarks
Preamble:	
Body: Creation of the pseudonym certificate request Ciphering of the request The request is sent to the PCA The response from the PCA is received The response is deciphering The pseudonyms are extracted Their signature is verified The pseudonyms are stored	 The predefined keys of the FPGA must be used. The range of these keys is specified in the configuration file The request is signed with the LTC and ciphered with the IDK and the public key of the PCA
Postamble:	Verify that the pseudonyms have been effectively stored in the repository together with their corresponding .meta that contains the HSM key handles created for the different pseudonyms

2.2.2 test of the functionalities offered to the FOTs

In order to test our VSS implementation we first list all the functionalities needed by the FOTs. For each functionality we specified some test scenario (as done in the previous section).

Having test the basic functionalities, we test the following ones:

- The creation for the signature of a message that is to be sent by the vehicle
- The verification of the signature of a message received by the vehicle
- The change of pseudonym when the VSS implementation finds that the current pseudonym has been used to many times or for a too long time. This depends on the configuration parameters.
- The last functionality that has been tested is **Verification on demand** which has been recently added to the VSS in response to a demand of a FOT. This functionality has been requested to overcome the poor performance of the current implementation of the FPGA.

2.2.2.1 Test_Sign_Verify_VB_01

Purpose: Verify that the VSS is able to sign a message and to verify the signature of an incoming message

Preconditions: Pseudonyms must exist and must be valid. They must be stored in the directory specified in the configuration file

The HSM keys associated to these pseudonyms must exist. They must be stored in the directory specified in the configuration file or in the default directory if not specified in the configuration file

Context: The software version of the HSM is used. No customer function has been registered so that pseudonym change is not possible.

The test calls the functions sign/verify 200 times. At each invocation, the size of the buffer to sign is incremented by one (= a simple solution for having different buffers).

Procedure	Remarks
Preamble: No customer function has been registered.	
Body: Creation of the signature of a bloc of data	Verify that the result of the operation is success
Verification of the signature created at the previous step	Verify that the result of the operation is success

Procedure	Remarks
Postamble:	As the two functions are called 200 times, we just verify that we have 200 success in the log file with the tools wc. We also verify that no pseudonym change has occurred.

2.2.2.2 Test_Sign_Verify_VB_02

Purpose: Verify that the VSS is able to sign a message and to verify the signature of an incoming message

Preconditions: Pseudonyms must exist and must be valid. They must be stored in the directory specified in the configuration file or in the default directory if not specified in the configuration file

The HSM keys associated to these pseudonyms must exist. They must be stored in the directory specified in the configuration file

Context: The software version of the HSM is used. Pseudonym change is possible.

The request for more pseudonyms has been deactivated in the configuration file.

The test calls the functions sign/verify 200 times. At each invocation, the size of the buffer to sign is incremented by one (= a simple solution for having different buffers).

Procedure	Remarks
Preamble: A customer function has been registered The attribute **pm.dont_ask_new_pseudonyms** must be present in the configuration file and set to 1	
Body: Creation of the signature of a bloc of data Verification of the signature created at the previous step	 Verify that the result of the operation is success Verify that the result of the operation is success
Postamble:	

Procedure	Remarks
	As the two functions are called 200 times, we just verify that we have 200 success in the log file with the tools wc. At least one pseudonyms changed has been done. In this case, verify that at least one pseudonym has been removed from the repository. In fact, during the 200 invocations, 3 different pseudonyms were used.

2.2.2.3 Test_Sign_Verify_VB_03

Purpose: Verify that when the number of available pseudonyms is under a specified number, a pseudonym request is sent to a PCA

Preconditions: Pseudonyms must exist and must be valid. They must be stored in the directory specified in the configuration file or in the default directory if not specified in the configuration file

The HSM keys associated to these pseudonyms must exist. They must be stored in the directory specified in the configuration file

A PCA must be reachable

Context: The software version of the HSM is used. Pseudonym change is possible.

The test calls the functions sign/verify 200 times. At each invocation, the size of the buffer to sign is incremented by one (= a simple solution for having different buffers).

Procedure	Remarks
Preamble: A customer function has been registered. The attribute **pm.dont_ask_new_pseudonyms** of the configuration file must be set to 0	
Body: Creation of the signature of a bloc of data Verification of the signature created at the previous step	 Verify that the result of the operation is success Verify that the result of the operation is success

Procedure	Remarks
Postamble:	As the two functions are called 200 times, we just verify that we have 200 success in the log file with the tools wc. At least one pseudonyms changed has been done. In this case, verify that at least one pseudonym has been removed from the repository. We must see that a pseudonym request has been sent to a PCA The new pseudonyms must have been stored in the repository

2.2.2.4 Test_Sign_Verify_VB_04

Purpose: Verify that the functionality blockPseudonymChange is operational.

Preconditions: Pseudonyms must exist and must be valid. They must be stored in the directory specified in the configuration file or in the default directory if not specified in the configuration file

The HSM keys associated to these pseudonyms must exist. They must be stored in the directory specified in the configuration file

A PCA must be reachable

Context: The software version of the HSM is used. Pseudonym change is possible.

The test calls the functions sign/verify 200 times. At each invocation, the size of the buffer to sign is incremented by one (= a simple solution for having different buffers).

Procedure	Remarks
Preamble: A customer function has been registered. The attribute **pm.dont_ask_new_pseudonyms** of the configuration file must be set to 0	
Body: the function blockPseudonymChange is called with a duration greater than the time of the test Creation of the signature of a bloc of data	Verify that the result of the operation is success

Procedure	Remarks
Verification of the signature created at the previous step	Verify that the result of the operation is success
Postamble:	As the two functions are called 200 times, we just verify that we have 200 success in the log file with the tools wc Verify that no pseudonym changed has occurred

2.2.2.5 Test_Sign_Verify_VB_05

Purpose: Verify that the functionality blockPseudonymChange is operational.

Preconditions: Pseudonyms must exist and must be valid. They must be stored in the directory specified in the configuration file or in the default directory if not specified in the configuration file

The HSM keys associated to these pseudonyms must exist. They must be stored in the directory specified in the configuration file

A PCA must be reachable

Context: The software version of the HSM is used. Pseudonym change is possible.

The test calls the functions sign/verify 200 times. At each invocation, the size of the buffer to sign is incremented by one (= a simple solution for having different buffers).

Procedure	Remarks
Preamble: A customer function has been registered. The attribute **pm.dont_ask_new_pseudonyms** of the configuration file must be set to 0	
Body: the function blockPseudonymChange is called with a duration smaller than the time of the test Creation of the signature of a bloc of data Verification of the signature created at the previous step	 Verify that the result of the operation is success Verify that the result of the operation is success
Postamble:	

Procedure	Remarks
	As the two functions are called 200 times, we just verify that we have 200 success in the log file with the tools wc Verify that at least one pseudonym changed has occurred

2.2.2.6 Test_Sign_Verify_VB_06

Purpose: Verify that the functionality freePseudonymChange is operational.

Preconditions: Pseudonyms must exist and must be valid. They must be stored in the directory specified in the configuration file or in the default directory if not specified in the configuration file

The HSM keys associated to these pseudonyms must exist. They must be stored in the directory specified in the configuration file

A PCA must be reachable

Context: The software version of the HSM is used. Pseudonym change is possible.

The test calls the functions sign/verify 200 times. At each invocation, the size of the buffer to sign is incremented by one (= a simple solution for having different buffers).

Procedure	Remarks
Preamble: A customer function has been registered. The attribute **pm.dont_ask_new_pseudonyms** of the configuration file must be set to 0	
Body: the function blockPseudonymChange is called with a duration greater than the time of the test Creation of the signature of a bloc of data Verification of the signature created at the previous step the function freePseudonymChange before the end of the test	 Verify that the result of the operation is success Verify that the result of the operation is success
Postamble:	

Procedure	Remarks
	As the two functions are called 200 times, we just verify that we have 200 success in the log file with the tools wc Verify that at least one pseudonym changed has occurred

2.2.2.7 Test_Verification_On_Demand_VB_01

Purpose: Verify that the verification does not occur at each call

Preconditions: Pseudonyms must exist and must be valid. They must be stored in the directory specified in the configuration file

The HSM keys associated to these pseudonyms must exist. They must be stored in the directory specified in the configuration file or in the default directory if not specified in the configuration file

Context: The software version of the HSM is used.

The test calls the functions sign/verify 200 times. At each invocation, the size of the buffer to sign is incremented by one (= a simple solution for having different buffers).

Procedure	Remarks
Preamble: The attribute **pm.policy.verificationpolicy** has the value **sampling** and the attribute **pm.policy.verificationvalue** is set to a threshold	
Body: Creation of the signature of a bloc of data Verification of the signature created at the previous step	 Verify that the result of the operation is success Verify that the result of the operation is success
Postamble:	At the end of the test, the number of verifications is coherent with the value specified in the configuration file

All the above tests are relevant for use cases F-SIG-01, F-SIG-02, F-SIG-03, F-CER-01, F-PSN-02 of PRESERVE testing handbook.

2.3 Test results

All the tests described above have been successfully passed on an x86 platform for the software version of the HSM and for the FPGA.

Below we indicate the status of the higher level functionalities offered to customers by VSS (see deliverable D2.1). Test results are summarized in the following table:

Functionality	Status
blockPseudonymChange	not yet requested by FOTs but successfully tested
freePseudonymChange	not yet requested by FOTs but successfully tested
changePseudonym	not tested. In fact the action of provoking the changing of pseudonym by a customer has not been tested. But the test **Test_Sign_Verify_VB_02** shows that the changing of pseudonyms works
createSignature	successfully tested
verifySignature	successfully tested
encryptSendingFrame	not yet requested by FOTs so not tested. But the low-level method that ciphers data is working as it has been successfully tested by the tests on certificate requests
decryptReceivedFrame	not yet requested by FOTs so not tested. Same remark as above
registerRequestNetworkIdChange	successfully tested
indicateNetworkIdChange	successfully tested

3 PRESERVE Initial Validation Tests Setup

This section presents the initial tests used for PRESERVE trials: the test environment and set-up, the test purpose and main functions and operational requirements which are being tested during the concerned field-testing activities. D3.1.1 focuses on the internal trial activities only (based on VSS Kit 1). D3.1.2 will then also detail results of joint tests with Score@F.

3.1 Internal FOT 1 Setup

The first field operational test (FOT) is an internal test of the first (FPGA-based) VSS Kit. Its aim is to perform testing to verify overall functionality and to benchmark timings in less-loaded environments.
A typical VSS Kit setup consists of modems, and connected to each of these is an FPGA. The modems have antennas to communicate with each other. The FPGA acts as a hardware accelerator for cryptographic operations, such as signing and verifying messages. Thus a setup such as this allows a modem to for example send a message to the connected FPGA, which will generate a signature on this message. The modem then uses the radio connection to send this signature to another modem, which then uses its FPGA to verify the received signature. Before this entire setup can be tested even in a low-load environment, the individual components need to be tested for functionality and benchmarked to give some indication of performance.

To achieve this goal, a number of tests were performed. For these tests the following hardware was available:

- Denso WSU-01 (B)
- Cohda Wireless MK3
- Asus Eee PC netbook
- ZTEX USB FPGA Module 1.15 with Power Supply Module 1.1

The Denso WSU and the Cohda Wireless MK3 are both modems, and can be seen in figure 3.1. The FPGA can be seen in figure 3.2.

The Denso WSU uses a PowerPC based processor, and the Cohda MK3 uses an Arm based processor. Because of this, a cross-compilation toolchain is necessary to build applications for these systems. Furthermore, the FPGA connects to the modems through

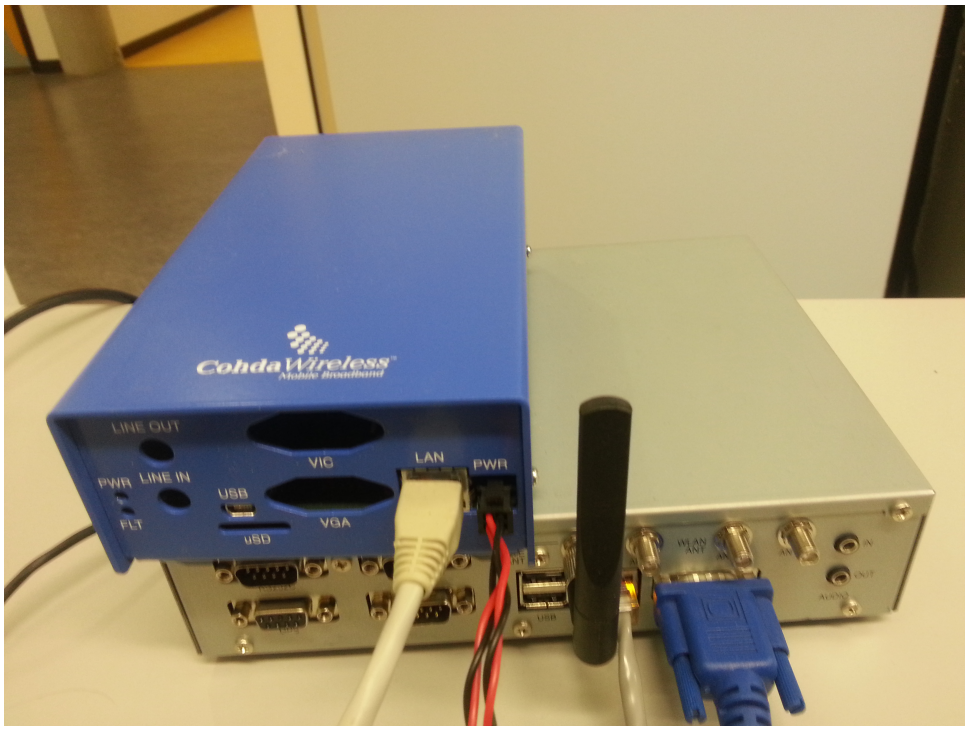

Figure 3.1: The Denso and Cohda modems

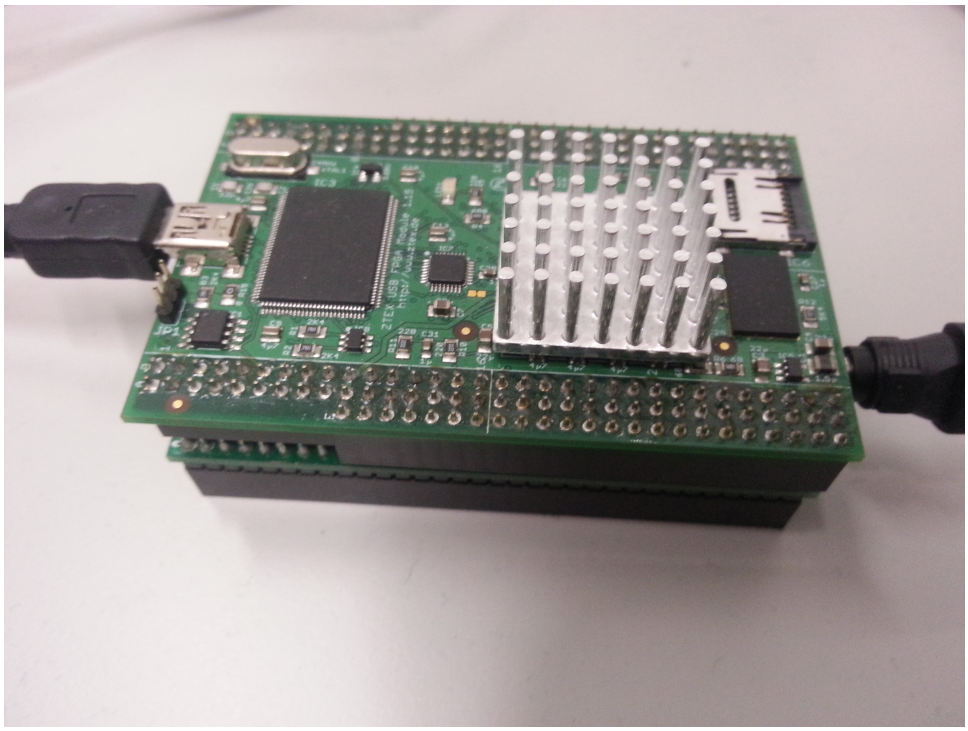

Figure 3.2: The FPGA

USB, which means that we need some library on the modems that gives them the functionality to connect and use USB devices. As a cross-platform library was needed, libusb was used as the USB library.
Lastly, the modems also have an ethernet connection, which means that they can be put online and accessed remotely. For security, standard services such as telnet and ftp are disabled, allowing SSH access only from authorized users.

3.2 Test cases

To test the performance of the modems and the FPGA, a benchmarking application was created. The benchmarks are based on the performance of signing messages and verifying these signatures. The benchmarking process works as follows: A message is signed and then verified, and the time taken for each of these two steps is calculated. This is repeated a number of times to give the average time taken for generating signatures and verifying signatures respectively. From this the average number of signature generations/verifications per second is calculated. Additionally, for each message it is checked whether or not the signing and verifying were successful. This way both the functionality and the timings can be evaluated.

To compare the performance, the same benchmark application is run on the different components. To give an indication of the effects of including an FPGA in the setup, software only benchmarks will also be performed alongside a hardware (FPGA) test. The software only benchmarks will be done on the two modems as well as on the Asus netbook. Finally benchmarking will be done using one of the modems or the laptop as a host, and the FPGA as a hardware accelerator.

All of the above leads to the following four test cases:

3.2.1 Test1: x86

For the x86 test an Asus Eee PC with a 1.6GHz Intel Atom N270 processor was used. The benchmark application was built using GCC 4.4.3.

3.2.2 Test2: Denso WSU

The next tests and benchmarks were performed on a Denso Box, which contains a 400MHz MPC5200B PowerPC processor. The benchmark was compiled using GCC 3.3.2.

3.2.3 Test3: Cohda Wireless MK3

The final VSS Kit software only tests were performed on a Cohda Wireless MK3 which has a 533MHz ARM11 processor. The benchmark application was built using GCC 4.1.2.

3.2.4 Test4: FPGA

The processing performance of the FPGA should not be dependent on the host that it is connected to, as all processing is done on the FPGA itself and the host is only used for sending and receiving packets from the FPGA. Thus we only need to run the benchmarks from a single host to get performance benchmarks for the FPGA. In this case the Asus Eee PC is used as a host and connected to the FPGA with a USB cable. The signing and verification of messages is performed on the FPGA itself.

3.2.5 All tests

For all tests, the benchmarking application was built against VSS kit 1.4.0 (svn release r2883) and the FPGA ran firmware version 1.5. It was also of interest how different compiler flags affect the benchmarking performance. For this reason, all test are performed twice, first with the application compiled with the -Os flag which optimizes for size, and second with the -O3 flag, which optimizes for speed.

All the above tests are relevant for use cases F-SIG-01, F-SIG-02 and F-SIG-03 of the Preserve testing handbook.

4 Performance evaluation results of the Internal FOT 1

In this section the results of the four test cases are presented and described. For each test a table with 10 different test runs is given. For each test run the time to sign a message, the time to verify a signature, as well as whether or not the sign/verify process was successful is given. These data lead to the average time of signature generations/verifications, and from this how many signature generations/verifications can be performed on average per second.

4.1 Test1: x86

The results of the size optimized test can be seen in table 4.1 and the results of speed optimized test can be seen in table 4.2.

We can see that for the size optimized test it takes on average 16.5 and 93.5 milliseconds respectively to sign and verify messages. This leads to on average 60.6 signature generations per seconds and 10.7 signature verifications per second.

For the speed optimized test, the netbook can perform on average 69.4 signatures per second and 12 verifications per second. This is an improvement of roughly 14.5% and

Test #	Sign Duration (ms)	Verify Duration (ms)	Success
1	20	94	yes
2	15	94	yes
3	16	93	yes
4	16	93	yes
5	18	94	yes
6	16	93	yes
7	16	93	yes
8	16	93	yes
9	15	95	yes
10	17	93	yes
average	16.5	93.5	-
average verification per second	60.6	10.7	-

Table 4.1: x86 results (size optimized)

Test #	Sign Duration (ms)	Verify Duration (ms)	Success
1	19	83	yes
2	14	83	yes
3	13	85	yes
4	14	83	yes
5	14	83	yes
6	14	82	yes
7	14	83	yes
8	14	83	yes
9	14	84	yes
10	14	83	yes
average	14.4	83.2	-
average verification per second	69.4	12.0	-

Table 4.2: x86 results (speed optimized)

Test #	Sign Duration (ms)	Verify Duration (ms)	Success
1	107	458	yes
2	76	452	yes
3	74	456	yes
4	78	457	yes
5	77	457	yes
6	77	455	yes
7	74	460	yes
8	76	457	yes
9	77	451	yes
10	75	454	yes
average	79.1	455.7	-
average verification per second	12.6	2.19	-

Table 4.3: Denso WSU results (size optimized)

12.1 for signature generations and verifications respectively. For both tests we can see that all the tests complete successfully, so the signing and verifying works as it should.

4.2 Test2: Denso Box

The results of the size optimized test can be seen in table 4.3 and the results of speed optimized test can be seen in table 4.4.

The size optimized benchmarks give verification and generations rates of 12.6 and 2.19 messages per second respectively. For the speed optimized benchmarks there is an improvement to 15.2 generations and 2.66 verifications per second. In this case, optimizing

Test #	Sign Duration (ms)	Verify Duration (ms)	Success
1	90	378	yes
2	66	375	yes
3	63	375	yes
4	64	377	yes
5	62	372	yes
6	62	372	yes
7	64	377	yes
8	63	373	yes
9	61	377	yes
10	63	377	yes
average	65.8	375.3	-
average verification per second	15.2	2.66	-

Table 4.4: Denso WSU results (speed optimized)

Test #	Sign Duration (ms)	Verify Duration (ms)	Success
1	144	685	yes
2	115	682	yes
3	116	678	yes
4	150	677	yes
5	118	680	yes
6	117	678	yes
7	122	675	yes
8	115	678	yes
9	112	677	yes
10	114	711	yes
average	122.3	682.1	-
average verification per second	8.18	1.47	-

Table 4.5: Cohda Wireless MK3 results (size optimized)

for speed gives a 20.6% and 21.5% improvement in signature generation time and verification time respectively.

4.3 Test3: Cohda Wireless MK3

The results of the size optimized test can be seen in table 4.5 and the results of speed optimized test can be seen in table 4.6.

The Cohda wireless seems to give the slowest software performance, with on average 8.18 generations and 1.47 verifications per second. Again optimizing for speed shows some gain in performance, giving 9.28 generations and 1.62 verifications per second.

Test #	Sign Duration (ms)	Verify Duration (ms)	Success
1	130	620	yes
2	101	613	yes
3	103	614	yes
4	107	605	yes
5	104	647	yes
6	106	616	yes
7	105	619	yes
8	108	608	yes
9	107	623	yes
10	107	603	yes
average	107.8	616.8	-
average verification per second	9.28	1.62	-

Table 4.6: Cohda Wireless MK3 results (speed optimized)

Test #	Sign Duration (ms)	Verify Duration (ms)	Success
1	86	75	yes
2	86	76	yes
3	87	75	yes
4	89	75	yes
5	89	75	yes
6	90	75	yes
7	89	77	yes
8	90	76	yes
9	89	79	yes
10	90	75	yes
average	88.5	75.8	-
average verification per second	11.3	13.2	-

Table 4.7: FPGA results

This is an improvement of 13.4% for signature generations and 10.2% for signature verifications.

4.4 Test4: FPGA

The result of the hardware tests can be seen in table 4.7.

Here we see that the FPGA can perform circa 11.3 signature generations per second and 13.2 signature verifications per second. It is important to note that these benchmarks include all USB latencies. This means that the total time to sign or verify a message consists of sending packets over the USB connection to the FPGA, the FPGA processing

the message, and then the FPGA sending back the result over the USB connection. In terms of functionality the FPGA works as expected, successfully completing all tests.

4.5 Discussion

From the benchmark results, it is clear that the modems are very slow at verifying signatures in software, and including the FPGA as a cryptographic accelerator certainly provides improvements in this area. In terms of generating signatures, the software and FPGA performance does not seem to be very different. Compared to the x86 benchmarks, generating signatures seems to be much faster on the netbook, whereas verifying signatures is slightly slower. Overall it is seen that in software tests, around 10-20% improvement can be found in compiling with a flag so that the application is optimized for speed, though this may go at the cost of requiring more storage/memory.

5 Conclusion

As a result of our initial tests we have validated the correct functionality of the PRESERVE VSS Kit 1 including the FPGA-based HSM. During these tests, initial problems, e.g., with USB connectivity of the HSM were identified and removed. Tests were conducted in two environments, one in a lab environment at University of Twente and the second during integration of the PRESERVE VSS Kit 1 into Score@F cars. In the first, we also conducted preliminary performance measurements.

As a result of these tests, we have an increased confidence in the correctness of the PRESERVE VSS software and especially the FPGA Hardware Security Module that is now the basis for the design of the ASIC. Next steps now include practical FOT tests with Score@F in Q2 or Q3 2013 that will further allow to enhance the reliability of the PRESERVE VSS Kit 1 and then internal and joint tests with PRESERVE VSS Kit 2 including the ASIC. Due to significant changes in ASIC design (55 nm production process instead of originally envisioned 90 respectively 180 nm), ASICs will likely not become available before the end of the year, so these tests will likely be conducted in 2014.