PREparing SEcuRe VEhicle-to-X Communication Systems

Deliverable 3.2

FOT Trial 2 Results

Project:	PRESERVE
Project Number:	IST-269994
Deliverable:	D3.2
Title:	FOT Trial 2 Results
Version:	1.0
Confidentiality:	Public
Editor:	Carsten Rolfes
Cont. Authors:	Michael Feiri, Frank Kargl, Athanasios Giannetsos, Stelios Gisdakis, Hongyu Jin, Mohammad Khodaei, Michel Sall
Date:	2015-07-31

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

Document History

Version	Date	Main author	Summary of changes
v0.1	2015-04-21	Carsten Rolfes (FhG AISEC)	Initial structure of document created
v0.2	2015-06-10	Carsten Rolfes (FhG AISEC)	Update of use cases and results.
v0.3	2015-06-10	Norbert Bißmeyer (FhG SIT)	Draft for review
v0.4	2015-07-16	Thanassis Giannetsos (KTH)	Testbed updates
v0.5	2015-07-17	Michael Feiri (UT)	Performance updates
v0.9	2015-07-20	Carsten Rolfes (FhG AISEC)	Draft for final review
v1.0	2015-07-25	Frank Kargl (UT)	Final version

Table 1: Version history

Approval		
	Name	Date
Prepared	Carsten Rolfes, Frank Kargl	2015-07-20
Reviewed	All Project Partners	2015-07-31
Authorized	Frank Kargl	2015-07-31

Circulation	
Recipient	Date of submission
Project Partners	2015-07-20
European Commission	2015-07-31

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 integrated 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 D3.2 summarizes the evaluation of measurements performed on PRESERVE's ASIC-based V2X Security Subsystem (VSS Kit 2) through tests in its internal testbed. Previous tests using VSS Kit 1 conducted internally and jointly with the Score@F project are described in D3.1.2 [5].

The FOT 2 field operational test is an internal test of the second (ASIC-based) VSS Kit. Its aim is to perform testing to verify overall functionality and to benchmark timings in low-, medium- and high-loaded environments.

A typical VSS Kit setup consists of 20 modems, and connected to 6 of these is an ASIC-based HSM. The other boxes are equipped with the software-only VSS kit based on OpenSSL as crypto backend. The ASIC 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 ASIC, 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 ASIC 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 possible performance.

The tests include two distinct field-testing activities of the V2X security system:

1. An internal performance test of the VSS software-based version done at University of Twente.
2. A large scale FOT, using the VSS (software and ASIC-based) integrated into the several OBUs platforms at the Technical University of Stockholm, called Internal FOT Trial 2 (IFT2).

Section 2 presents the assessment plan for PRESERVE VSS implementation based on the FOTnet test methodology. This section details the steps of the test methodology, the performance indicators and measurement procedures used to evaluate PRESERVE VSS implementation. This Section also integrates the specification of a list of test cases that evolved during various trials during the project duration. These test cases are part of the Technical Report 4 “Testing Handbook”, which was disseminated to other research projects (e.g., FOTNET, Drive C2X). D3.2 focuses on the internal trial activities only (based on VSS Kit 2) while D3.3 [1] focuses on tests conducted with external partners, esp. the ETSI plugtests carried out during the ITS Cooperative Mobility Services Event 4.

Section 2.2 presents the internal PRESERVE testbed setup at KTH in Stockholm, Sweden. 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 are presented here.

Section 3 presents benchmarks of VSS software-based version tested on a single box.

While Section 4 presents the evaluation results of the internal field-test at KTH running on the testbed with 20 boxes. This deliverable D3.2 includes preliminary conclusions, based on first measurements evaluation from the Internal FOT Trial 2 (IFT2).

The tests presented in this deliverable aim at providing both a functional and performance analysis of VSS Kit 2. We both aim at demonstrating the performance issues of pure software-based solutions, identifying impact of V2X security payload and security processing to V2X network operation, and provide benchmarks for our VSS Kit 2 to show that it meets operational requirements described in D1.1 [8].

2 Assessment Plan of VSS Kit 2

This section presents the assessment plan of VSS Kit 2 (SW-only version and ASIC-based HSM). It describes the test bench and test environment used for the assessment of VSS Kit 2 with a particular focus on: (*i*) the test cases generated for validating the effectiveness and reliability of the system under various loads (varying msgs/sec and number of senders), and (*ii*) the testing methodology followed for performing these evaluations (i.e., logging facilities used and orchestration scripts developed for extracting the various test bench performance benchmarks). Furthermore, Section 2.3 specifies the use cases considered for validating the functionality of VSS Kit 2 along with the accompanying test results (Section 2.4).

2.1 Test Bench for Validation

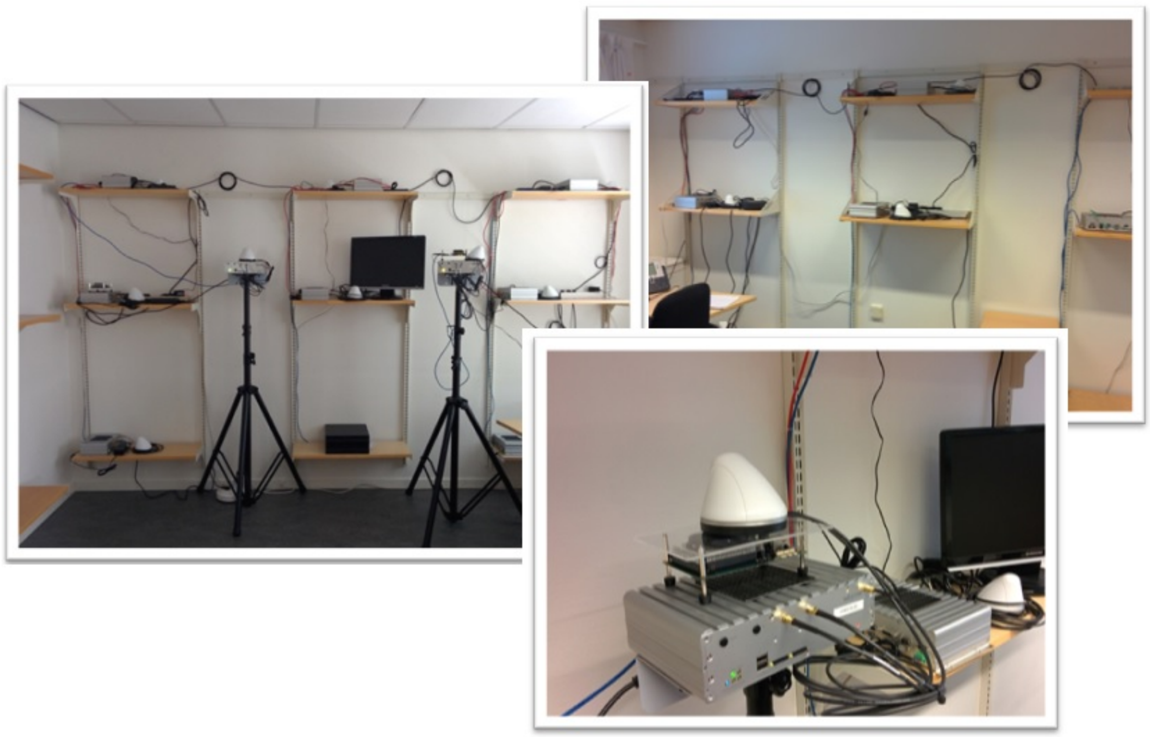


Figure 2.1: The Nexcom testbed in Stockholm

Our testbed comprises 20 NEXCOM VTC 6201 platforms and an analysis server. Each NEXCOM box is equipped with a 64bit dual core, dual HT Intel Atom CPU D510 @ 1.66GHz processor and 1GB of RAM. The operating system is Ubuntu 12.04.5. Each box is equipped with an 802.11p ITRI networking card and DRIVE C2X antenna and leverages the HITACHI communication stack. The analysis server is used for orchestrating the test scenarios of Sec. 4 and for analyzing the test results.

The validation tests of VSS functionality were performed using VSS Kit 2, version 2.1.0. Figure 2.1 presents the configuration of the test bench. Two of the NEXCOM boxes are currently mounted on tripods; they serve as the receivers for the validation tests described in Sec. 4 and they are also equipped with PRESERVE ASICs.

2.2 Test Bench Environment

In this section, we give an overview of the tests performed on the test bench for verifying the overall correctness and performance of the VSS Kit. Two main properties are of interest: *end-to-end latency* and *packet loss*. It needs to be stressed that all extracted figures are the results of various experiments run on the large-scale testbed described in the previous section (more details can be found in Section 4). The goal is to provide strong evidence on the suitability of the VSS for such wide-ranging deployments in pilot tests.

Moreover, in Section 2.2.2, we describe our testing approach and the facilities created for performing the defined test cases. In order to rigorously evaluate the performance of the VSS Kit, a number of orchestration and extraction scripts were created for automating the test bench to be able to collect the necessary measurements and perform (some kind of) data processing to exfiltrate the final (collective) results in the analysis server.

2.2.1 Tests Overview

PRESERVE had decided on an extensive test plan for better evaluating the overall performance of the VSS Kit both in the case of SW-only version and ASIC-based HSM. In this section, we give an overview on the individual test cases that were defined towards this direction. Our main focus is not on the internal mechanisms of specific system components (i.e., ASIC), as such results are presented in Deliverables 2.3/2.4, but on the outputs generated in response to selected inputs and execution conditions in the testbed.

The three tests planned are based on different load scenarios to better emulate diverse conditions. We varied the number of “sending” ITS stations in order to achieve different levels of *density* and *message rate* (i.e., traffic load). This also results in different probabilities for packet collisions and will allow us to investigate the effect that increased packet size due to security payload has on packet loss. In all cases, we had one “receiving” station that served as the data collection point of all measurements to be then sent to the analysis server.

1. **Low Load:** 2 ITS stations (one of which is the receiver) in communication range with 1 ITS sender broadcasting at increased message rates; i.e., 1 Hz, 10 Hz, 20 Hz and 100 Hz. This test case was primarily used to test correct functionality and performance in absence of collisions.
2. **Medium Load:** Up to 6 ITS stations (one of which is the receiver) in communication range. More specifically, we had 5 ITS senders broadcasting at increased message rates; i.e., 1 Hz, 10 Hz, 20 Hz and 100 Hz each.
3. **High Load:** Up to 16 ITS stations (one of which is the receiver) in communication range. By having such a large number of ITS stations and letting them communicate with increased message rates (1 Hz, 10 Hz, 20 Hz and 100 Hz each), we can emulate high load scenarios where channel capacity can be totally saturated. Such increasing packets rates is (to some extent) a valid approach to emulate a higher number of nodes in a wireless communication environment sending at lower rates. E.g., 15 stations sending at 100 Hz are compatible to 150 stations sending at 10 Hz or 300 stations sending at 5 Hz. However, overall number of collisions at medium access will be lower in our scenario. Still, the comparatively high number of ITS stations gives us good insights on broadcast collision behavior.

These tests serve as a performance verification of the VSS Kit 2 in various-load environments with different numbers of ITS stations but without mobility. As mentioned before, our goal is to provide strong evidence on the suitability of the VSS for such large-scale deployments. Details on the extracted performance results and their interpretation can be found in Section 4.

2.2.2 Test Methodology

In order to be able to evaluate the performance of the VSS implementation, we developed a number of orchestration scripts for automating the tasks of firing up the ITS sending and receiving stations and collecting the measurements back to the analysis server for further processing. For the latter, we employed the Spark [9] data engine which enables low-latency processing of large amounts of data. In what follows, we give the details behind our testing approach and the steps followed for extracting the final (collective) results.

Increased message rates (1 Hz, 10 Hz, 20 Hz, 100 Hz)

Prior to be able to run all the previously described tests, we had to investigate and adjust the configuration of the HITACHI networking stack which technically limits the maximum possible transmission rate to 10 Hz. In our case, we wish to emulate high load scenarios with increased message rates that go up to 100 Hz (per ITS sender). As source-code access was not possible to make modifications to the stack itself, we had to find a work-around.

Therefore, we run at each ITS sender multiple instances of the VSS Kit 2 program image (isolated from each other) for cases where we want to achieve transmission rates greater than 10 Hz. For instance, in the case of a per-sender message rate equal to 20 Hz, we start (in each NEXCOM box) 2 VSS program threads, each transmitting at the maximum possible rate of 10 Hz. Similarly, in the case of a per-sender message rate of 100 Hz, we have 10 VSS program threads running in each one of the ITS senders. This allows us to achieve the targeted message rates and evaluate the performance of the system under various traffic loads. Investigations showed that NEXCOM devices were capable of running the 10 instances in parallel without prohibitive multiplexing delay or packet loss.

Test bench Set-up & Orchestration Scripts

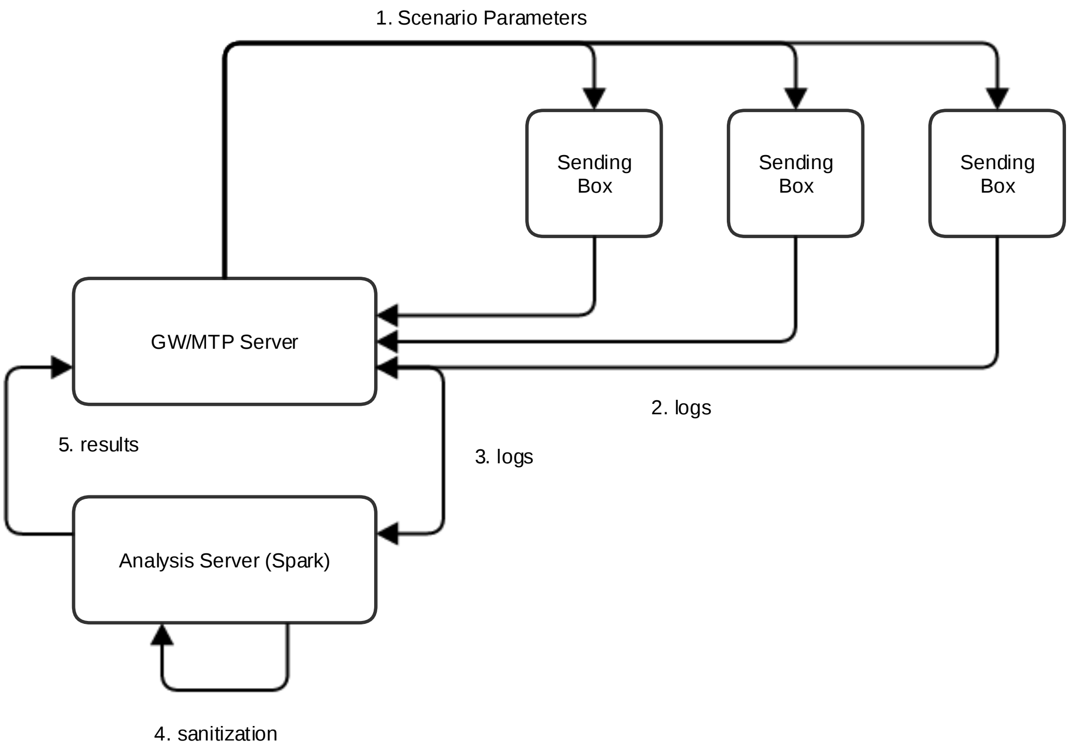


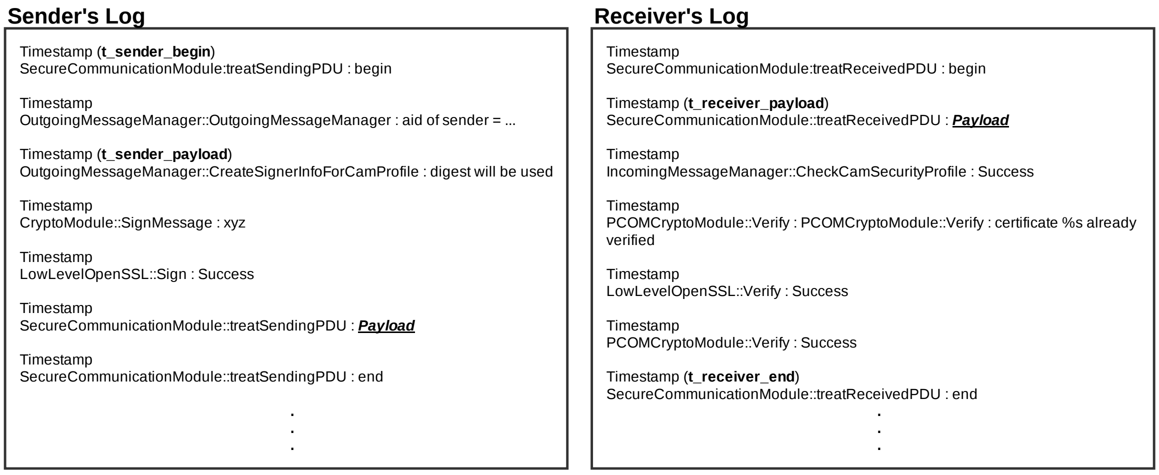
Figure 2.2: Set-up of the test bench for running the performance tests

Figure 2.2 shows the conceptual set-up of the test bench for running the above presented performance tests. A number of orchestration scripts were created for automating the completion of each test. Prior to running each test script, one has to specify two parameters to the ***run_numberOfSenders_messageRate.sh*** overall program.These input arguments are:

- The number of VSS Kit program instances that should be launched at each ITS sender.
- The overall execution time.

Once these parameters have been specified correctly, the analysis server will execute the test by launching the desired number of threads to the ITS sender stations specified in a separate file, named **hosts.start**. We have to highlight that before the HITACHI networking stack is fired up (in the ITS stations), each NEXCOM box will synchronize its local time by querying the NTP server running on the analysis server. This enables the correct synchronization of all ITS stations, prior to the execution of each test, and guarantees the correctness of the extracted measurements.

As aforementioned, once an experiment is completed, the analysis server collects all the necessary log files from all the hosts (i.e., sender & receiver ITS stations) and performs some type of processing to extract the final (collective) results. Figure 2.3 shows an example of such log files that were retrieved from the ITS sender and receiver stations (once the necessary sanitization was performed).



Sender's Log

```
Timestamp (t_sender_begin)
SecureCommunicationModule:treatSendingPDU : begin

Timestamp
OutgoingMessageManager::OutgoingMessageManager : aid of sender = ...

Timestamp (t_sender_payload)
OutgoingMessageManager::CreateSignerInfoForCamProfile : digest will be used

Timestamp
CryptoModule::SignMessage : xyz

Timestamp
LowLevelOpenSSL::Sign : Success

Timestamp
SecureCommunicationModule::treatSendingPDU : Payload

Timestamp
SecureCommunicationModule::treatSendingPDU : end
                              .
                              .
                              .
```

Receiver's Log

```
Timestamp
SecureCommunicationModule:treatReceivedPDU : begin

Timestamp (t_receiver_payload)
SecureCommunicationModule::treatReceivedPDU : Payload

Timestamp
IncomingMessageManager::CheckCamSecurityProfile : Success

Timestamp
PCOMCryptoModule::Verify : PCOMCryptoModule::Verify : certificate %s already
verified

Timestamp
LowLevelOpenSSL::Verify : Success

Timestamp
PCOMCryptoModule::Verify : Success

Timestamp (t_receiver_end)
SecureCommunicationModule::treatReceivedPDU : end
                              .
                              .
                              .
```

Figure 2.3: Example log files extracted from an ITS sender & receiver, respectively

Two main properties are of interest: *network latency* and *packet loss*. To compute the **network latency**, we subtract the timestamp $t\_receiver\_payload$ from $t\_sender\_payload$. To map the packets between senders and receivers, we include a unique counter in the CAM $payload$. Furthermore, to compute the **packet loss** we construct two sets: one with all the packets "seen" at the senders' log files (***p_sender***) and one with all the packets captured at the receivers' log files (***p_receiver***). By comparing their *cardinality*, we can correctly identify the amount of correctly transmitted messages.

All detailed results, along with a rigorous analysis of the system's performance, can be found in Section 4.

2.3 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.

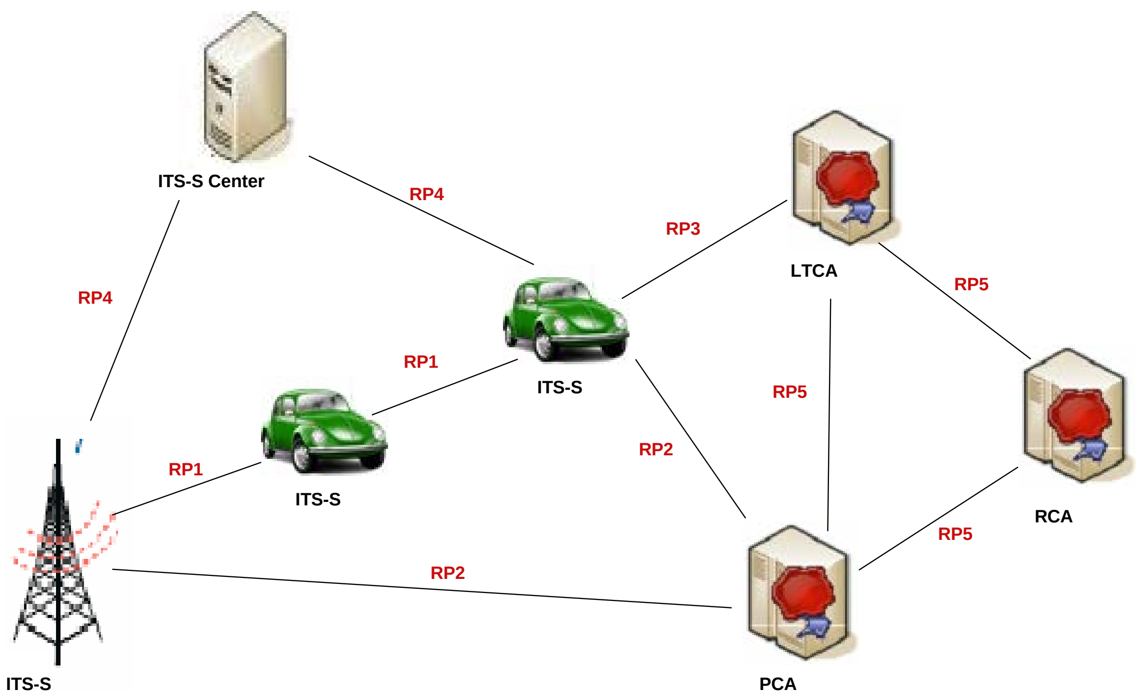


Figure 2.4: Reference points on ITS for PRESERVE tests

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.

2.3.1 Functional Use Cases Overview

As shown in Figure 2.4, we define five Reference Points (RP) between system entities on ITS in which we will test PRESERVE functionalities.

- (RP1) Reference Point between an ITS-S and another ITS-S
- (RP2) Reference Point between an ITS-S and PCA
- (RP3) Reference Point between an ITS-S and LTCA
- (RP4) Reference Point between an ITS-S and ITS-S Center
- (RP5) Reference Point between a CA and another CA

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 ETSI-compliant CAM or DENM messages with security payload

- The serialization and the deserialization of ETSI-compliant CAM or DENM messages with security payload
- The creation of a symmetric cryptographic key
- The creation of a private/public cryptographic key pair
- The import of a remote cryptographic key
- The export of a public cryptographic key
- The signing of a block of data,
- The verification of a signature associated to a block of data
- The cipher of a block of data
- The deciphering of data

We present uses cases that will be tested on each RP. Use cases are split into functional use-cases that tests the correct functionality described above and attack use cases that are evaluating behaviour of the VSS under certain attacks. The use-cases listed here are the ones that are tested in our testbed and are only a subset of all use-cases defined in the testing handbook.

2.3.2 Attack Use Cases

Table 2.8 shows the collection of various *attack scenarios* that were implemented to further verify the behavior and correctness of the VSS Kit. The overall purpose of these tests is to showcase the performance of the VSS, when different types of attackers (i.e., different penetration ratio, increased malicious packet rate) are present, and measure their impact on the performance of the benign ITS stations.

Tables 2.1 - 2.5 give a more detailed description of each attack case; the goal of the attacker and the actions she performs, the prerequisites for launching the attack and the values that are of interest. The impact on the system's performance (calculated from these extracted values) is presented in Section 4.2.

Table 2.1: A-SIG-01

Use Case Name	Send CAMs with invalid signatures
Use Case Code	A-SIG-01
Prerequisites	VSS has valid certificates, and is able to attach invalid signatures to CAMs
Actions	Introduce attackers that attach invalid signature on CAMs
Measured values	End-to-End delay, packet loss and goodput for CAMs sent from benign VSSs (see Sec. 4 for more details)

Table 2.2: A-SIG-03

Use Case Name	Send CAMs without signature
Use Case Code	A-SIG-03
Prerequisites	VSS has valid certificates, and is able to generate CAMs without signatures attached
Actions	Introduce attackers that send CAMs without signature
Measured values	End-to-End delay, packet loss and goodput for CAMs sent from benign VSSs (see Sec. 4 for more details)

Table 2.3: A-SIG-06

Use Case Name	Send CAMs with old timestamps
Use Case Code	A-SIG-06
Prerequisites	VSS has valid certificates, and is able to generate CAMs with old timestamps
Actions	Introduce attackers that send CAMs with old timestamps
Measured values	End-to-End delay, packet loss and goodput for CAMs sent from benign VSSs (see Sec. 4 for more details)

Table 2.4: A-CER-03

Use Case Name	Send CAMs signed under expired certificate
Use Case Code	A-CER-03
Prerequisites	VSS has expired certificates, and is able to sign CAMs under expired certificates
Actions	Introduce attackers that send CAMs signed under expired certificate
Measured values	End-to-End delay, packet loss and goodput for CAMs sent from benign VSSs (see Sec. 4 for more details)

Table 2.5: A-SIG-04

Use Case Name	DoS overload Attack
Use Case Code	A-SIG-04
Prerequisites	VSS is able to send CAMs with higher frequencies than allowed
Actions	Introduce attackers that send invalid CAMs with high frequencies to clog the receiver
Measured values	End-to-End delay, packet loss and goodput for CAMs sent from benign VSSs (see Sec. 4 for more details)

2.3.3 Use cases on RP1

Table 2.6: Mandatory Functional use cases on RP1

1	F-SIG-01	Signature of CAM in cooperative safety applications, e.g. RHS
2	F-SIG-03	CAM/DENM Processing at very high rate
3	F-SIG-04	Signing a CAM message with its-aid-ssp compatible with the current pseudonym
4	F-SIG-05	Signing a CAM message with its-aid-ssp not compatible with the current pseudonym
5	F-SIG-06	Signing a CAM message with its-aid-ssp not compatible with any of the pseudonyms
6	F-SIG-07	Signing a CAM message with its-aid-ssp not compatible with the long term certificate
7	F-CER-02	PCA certificate missing – Pseudonym certificate cannot be verified
8	F-PSN-01	Pseudonym Change on request or at system startup (OBU)
9	F-PSN-05	Pseudonym Certificate Refill via a RSU

Table 2.7: Optional functional use cases on RP1

1	F-SIG-02	Signature of DENM messages
2	F-SNS-01	Usage of other signed safety messages in application
3	F-SNS-02	Usage of signed message for service advertisement from a RSU, e.g. SAM
4	F-SNS-03	Usage of pseudonym certificates with compressed public keys
5	F-PSN-03	Lock pseudonym change
6	F-PSN-04	Pseudonym Change at system startup
7	F-ENC-01	Encrypted sending of Traffic Information to TCC

Table 2.8: Mandatory attack use cases on RP1

1	A-SIG-01	Send CAMs with invalid signatures
2	A-SIG-03	Send CAMs without signature
3	A-SIG-06	Send CAMs with old timestamps
4	A-CER-03	Send CAMs signed with expired certificates
5	A-SIG-04	DoS Overload Attack

Table 2.9: Optional attack use cases on RP1

1	A-SIG-02	Using invalid signatures in DENMs
2	A-CON-01	Sending correctly signed messages with invalid content
3	A-PSN-01	Attacker trying to identify pseudonym change
4	A-SIG-05	Time adjustment / replay attacks

2.3.4 Use cases on RP2

Table 2.10: Mandatory functional use cases on RP2

1	F-PSN-02	Pseudonym Certificate Refill

No optional functional use cases
No mandatory attack use cases

Table 2.11: Optional attack use cases on RP2

1	A-CER-06	Attacks on PCA

2.3.5 Use cases on RP3

Table 2.12: Mandatory functional use cases on RP3

1	F-CER-01	Issuing a LT certificate

No optional functional use cases
No mandatory attack use cases

Table 2.13: Optional attack use cases on RP3

1	A-CER-05	Attacks on LTCA

2.3.6 Use cases on RP4

No mandatory functional use cases

Table 2.14: Optional functional use cases on RP4

1	F-ENC-02	Encrypted sending of Traffic Information to TCC

No mandatory attack use cases
No optional attack use cases

2.3.7 Use cases on RP5

Table 2.15: Mandatory functional use cases on RP5

1	F-CER-04	PCA requests an authorization from LTCA for providing new pseudonyms reloading to a requesting ITS-S station

No optional functional use cases
No mandatory attack use cases

Table 2.16: Optional attack use cases on RP5

1	A-CER-04	Attacks on RCA

2.4 Test Results

Test results of the functional tests are summarized in Table 2.17. All test are independent of the underlying cryptographic service used (OpenSSL or ASIC). They are grouped into mandatory and optional tests. A detailed description of each test can be found in 2.3.1. The second column indicates if the test was successful or could not be tested due to software implementation issues.

Table 2.17: VSS Kit 2 functional test results

Tests	**Status**
Mandatory functional tests without ASIC	
F-PSN-01	passed
F-PSN-02	passed
F-PSN-05	not tested
F-SIG-01	passed
F-SIG-03	passed
F-SIG-04	passed
F-SIG-05	passed
F-SIG-06	passed
F-SIG-07	passed
F-CER-01	passed
F-CER-02	passed
F-CER-04	passed
A-SIG-01	passed
A-SIG-03	passed
A-SIG-04	passed
A-CER-01	passed

Table 2.17: VSS Kit 2 functional test results

Tests	Status
A-CER-03	passed
Optional functional tests without ASIC	
F-SIG-02	passed
F-SNS-01	not tested
F-SNS-02	not tested
F-SNS-03	passed
F-PSN-03	passed
F-PSN-04	not tested
F-ENC-01	not tested
F-ENC-02	not tested
F-IVS-01	not tested
F-CON-01	not tested
A-SIG-02	passed
A-SIG-05	not tested
A-CON-01	not tested
A-PSN-01	not tested
A-CER-04	not tested
A-CER-05	not tested
A-CER-06	not tested
A-IVS-01	not tested

3 PRESERVE FOT 2 Test

To evaluate the performance characteristics of the PRESERVE VSSKit we collect benchmarks of the basic sign and verify operations, which provide the foundation for the security of vehicular communication. These operations are sometimes referred to as encap and decap, to highlight that the encapsulation and decapsulation of messages can include more work than the application of raw cryptographic primitives.

3.1 FOT 2 Platform Setup

To derive the overhead introduced by trust management tasks, such as parsing of certificates and validity checks of certificate attributes, we also present benchmarks of raw cryptographic primitives of the cryptographic backends. The PRESERVE VSSKit supports multiple cryptographic backend. Among these are the following:

1. OpenSSL libcrypto (software)
2. Escrypt CycurLib (software)
3. VSSKit v1 FPGA (hardware)
4. VSSKit v2 ASIC (hardware)

For the purpose of collecting FOT 2 reference benchmarks we exclusively use the "OpenSSL libcrypto" backend for pure software test and the "VSSKit v2 ASIC" backend for hardware assisted benchmarks. The details of the "VSSKit v2 ASIC" backend are discussed in the PRESERVE Deliverable D2.3 "ASIC-based VSS Prototype" [6]. The "OpenSSL libcrypto" backend uses version 1.0.2a of this software. The libcrypto library was compiled with the optimization options as specified by the OpenSSL default configuration settings. The performance of the software backends highly depends on the processor architecture of the host platform and also on the availability of suitable assembly or compiler optimizations for the library. For the measurements performed within our test we used the compiler toolchain provided by the manufacturer of the host on-board unit. For the NEXCOM VTC 6201 platform the attributes of our hardware and software environment are summarized in Table 3.1. Figure 3.1 shows the hardware setup.

In previous tests all measurements were collected on 32bit systems running one single CPU core. This is also the case in our end-to-end performance tests in Chapter 4, It can be argued that these properties will continue to be typical for embedded systems in the foreseeable future, including the first generation deployment of on-board-units for deployments of vehicular communication technology. For the evaluation of software performance

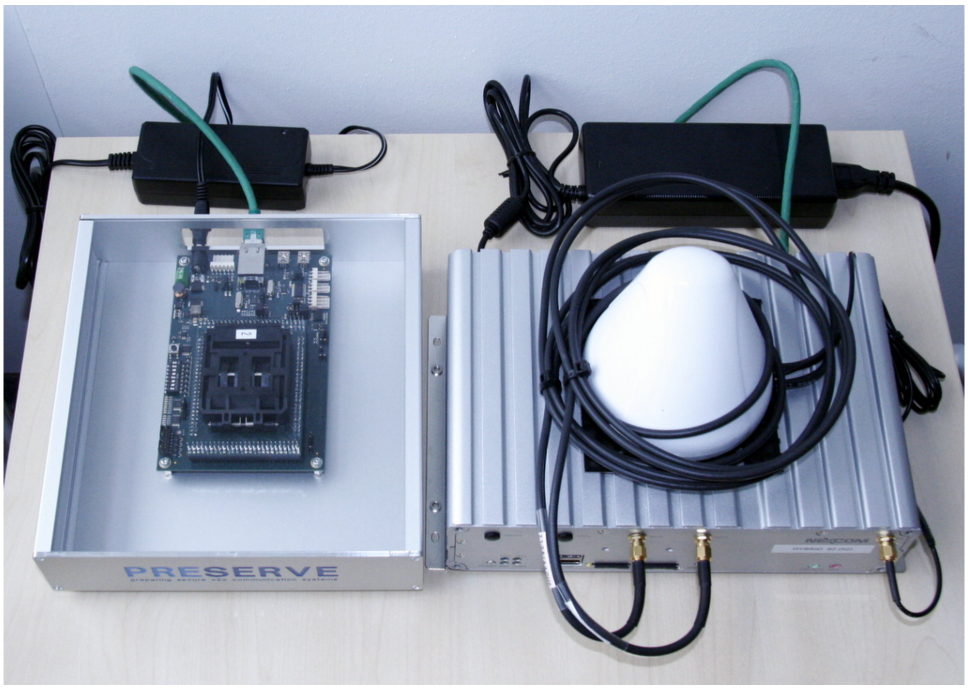


Figure 3.1: Nexcom modem and VSS Kit 2 HSM module

Platform	NEXCOM VTC 6201
Operating system	Ubuntu 12.05 LTS
Compiler	GCC 4.6.3-1ubuntu5
Target	x86_64-linux-gnu
Processor type	Intel Atom D510
Processor speed	1660MHz
ISA extensions	SSE2, SSE3, SSSE3
Physical cores	2
Hyperthreads	4
Memory type	DDR2 667/800

Table 3.1: NEXCOM VTC 6201 test environment

it is however useful to also consider more powerful processor architectures, which might become available only in the next generation of vehicular on-board-units. The NEXCOM VTC 6201 platform can actually be considered to be a sufficiently powerful platform as to represent the next iteration in the hardware evolution cycle for on-board units. This system uses a modern dual core processor with dual hyperthreading in each core, offering a total of 4 logical CPUs. The system also supports operation in 64bit mode and a set of 128 bit vector instructions, which can accelerate big integer calculations such as those necessary in the relevant variants of elliptic curve arithmetic.

Meanwhile new research was published about substantial enhancements in the software optimizations for ECDSA over 256-bit prime fields [4]. Accompanying patches have been accepted upstream into OpenSSL and are included in the x86_64 builds of OpenSSL starting at version 1.0.2a. Table 3.2 shows a set of benchmark results, which takes all the aforementioned enhancements info considerations. The values show latency and throughput values of raw OpenSSL speed tests as well as values for the PRESERVE VSSKit2 with the same version 1.0.2a of OpenSSL used as a backend. The recorded measurements represent the fastest run of at least 10 executions of a benchmark tool which executes the basic preserve_sign() and preserve_verify() operations. This methodology is applicable because the goal is to derive the speed of the pure operational runtime of these operations without consideration of unrelated interruptions by the operating system or other processes on the on-board unit. The operating systems did not offer any real time guarantees, but the system was kept idle during measurements.

3.2 FOT 2 Benchmark Results

Setup #	sign (ms)	sign/s	verify (ms)	verify/s
VSSKit2 OpenSSL (32bit)	7.1	146	7.9	132
VSSKit2 OpenSSL (64bit)	3.9	256	2.8	387
Raw OpenSSL (32bit)	1.5	1901.7	5.7	490.9
Raw OpenSSL (64bit)	0.4	6742.2	1.1	2780.6

Table 3.2: 64bit multicore performance combined with new ECDSA software enhancements in OpenSSL 1.0.2a

The signature generation performance generally fulfills the basic requirements to provide sufficient performance for secure vehicular broadcast communication. We therefore focus on analyzing the performance of signature verification results in Table 3.2. The VSSKit2 OpenSSL setups were configured to not perform full validation of the entire certificate trust chain. This means that only one cryptographic verification is performed for each invocation of the preserve_verify() function. The overall throughput of 132 verifications per second for the 32bit variant of the VSSKit2 test indicates that the 4 logical cores of the CPU have not been used concurrently. Indeed the architecture of the VSSKit2 is adapted to the designs of contemporary event-driven geo-networking stacks, which favor low latency single threaded performance and eschew support for asynchronous scheduling of parallel tasks. It turns out that this is a major limitation of common geo-networking stacks, which needs to be addressed in future iterations of such software packages.

The VSSKit2 OpenSSL (64bit) configuration uses the same setup as the 32bit configuration, with the major difference of utilizing 64 bit instructions as well as taking advantage of new software optimizations presented in [4]. We observe an increase a performance increase by almost a factor of 3, due to these changes.

The Raw OpenSSL (32bit) benchmark shows the potential of utilizing all computational resources by fully exploiting the multithreading capabilities of the CPU. The latency for individual signature verification operations is only slightly enhanced over the single threaded

VSSKit2 performance. This is solely due to the reduced overhead of raw OpenSSL compared to fully message processing in VSSKit2. The overall throughput measurement however demonstrates the impact of multithreading on overall performance. In this setup a total of 8 threads were configured to work in parallel to keep all 4 logical cores busy. The result is an overall throughput of 490 signature verification per second, instead of an expected throughput of 175 verifications per seconds for single threaded execution. This represents an enhancement of a factor of 2.8.

Finally the Raw OpenSSL (64bit) setup shows the overall peak performance achievable on the NEXCOM VTC 6201 on-board unit. With close to 1 millisecond of latency and overall throughput of 2780 verifications per second, it appears realistic that such an on-board unit can provide enough computational resources to sustain acceptable cryptographic performance even under heavy load. At least under the assumption that multithreading could be exploited and that the load and calibration of the operating system is carefully adjusted to minimize preemptions and interruptions through context switches.

4 Performance Evaluation Results of the Internal FOT Trial 2

In this section we present the results of the test cases described in Sec. 2.2. We focus on the *signature verification* time, the *End-to-End latency*, the *experienced packet loss* and the overall *resource consumption* of the communication stack and the VSS kit.

For all of the above, we consider different cases. More specifically, we begin with a presentation and an analysis of the system's performance under various loads (Sec. 2.2.1) and we proceed with a presentation of the results for the mandatory use cases of Sec 2.3. Results were collected according to the methodology presented in Sec. 2.2.1.

4.1 Performance tests

4.1.1 Performance analysis of Signature Verification

Fig. 4.1 presents the average time needed for signature verification at the receiving NEXCOM box. For this scenario we consider different numbers of senders ($1, 5$ and 15) and a single receiver. The receiving box uses the software-only version of the VSS Kit 2.

The average time needed for signature verification is plotted as a function of the sending frequency of the receivers. These frequencies range from $1Hz$ (i.e., 15 CAM msg/s) to $100Hz$ (1500 CAM msg/s). These values correspond to the different load levels defined in Sec. 2.2.1.

As the figure shows, signature verification takes, on average, $0.009\ sec$. In the most demanding configuration, i.e., $15\ senders$ sending at a frequency of $20Hz$, signature verification time foes to approximately $0.011\ sec$. Generally, raw generation of ECDSA signatures is approximately 3x faster than verification of ECDSA signatures. However, the values discussed here are measured at the PCOM level, where signature generation includes significantly more processing, e.g. key lookup, serialization and memory copies among others. In fact, the presented latency times are agreeing with the ones of Table 3.2 for the case of *VSSKit2 OpenSSL (32bit)* performance on the Nexcom box.

Next, we show the performance achieved in a configuration where signature verification is performed by the ASIC. For this case we consider a configurations of 5 senders and a single receiver. As Fig. 4.2 shows, the ASIC configuration outperforms the software verification. More specifically, the signature verification time for the by the ASIC is 0.0086 and $0.0091\ s$ for a receiver load of 100 and $500\ msg/s$ respectively.

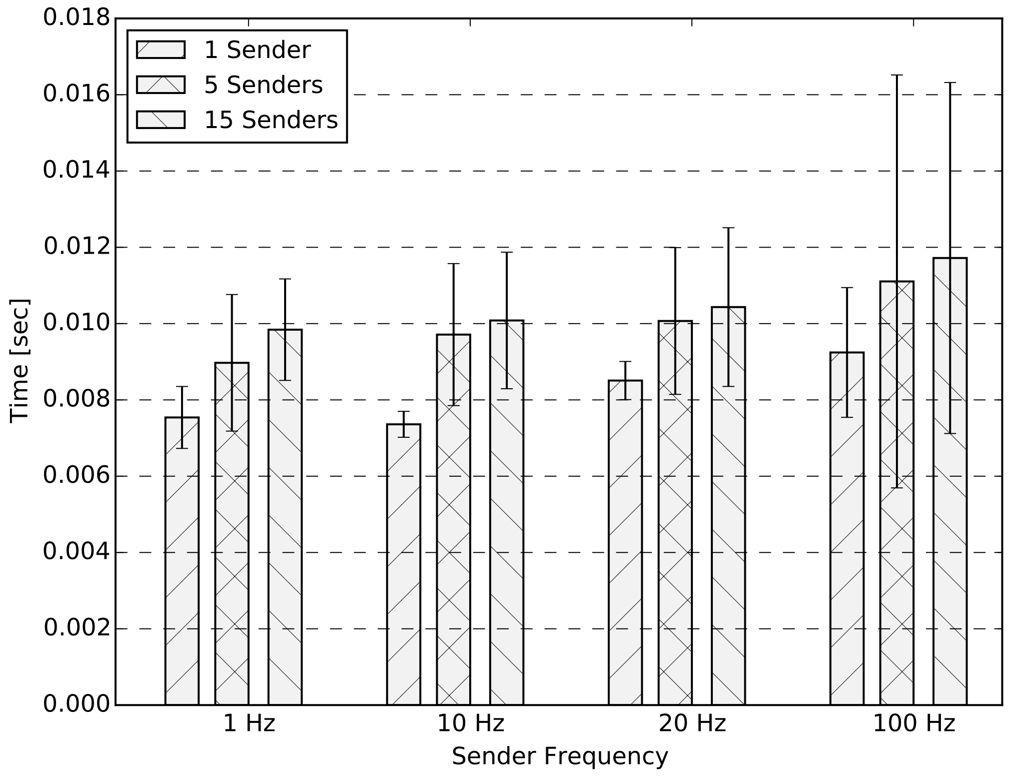


Figure 4.1: Signature verification time

Moreover, please note that ASIC solution is connected to the NEXCOM box via Ethernet. As a result, every request and verification requires a round trip through the kernel's network stack. The OS scheduler treats this fact very differently than the pure software computations of the OpenSSL back-end, which are consequently far less affected than the ASIC solution. These values are in line with single core ASIC performance values at 104MHz clock speed presented in Deliverable 2.3.

4.1.2 End-to-End Latency

We now proceed with an analysis of the End-to-End delay for CAM messages transmitted from different ITS stations. In this case, we plot the total time needed from the point that a CAM message is generated by the facility layer (at the sending NEXCOM box) until it is processed and verified at the receiver. Of course, the presented results also include the time needed for the message to be transmitted and processed by the HITACHI communication stack.

We plot the End-to-End latency as a function of the receiver's load. More specifically, we consider five senders broadcasting messages at $1, 10, 20$ and 100 Hz thus, achieving a

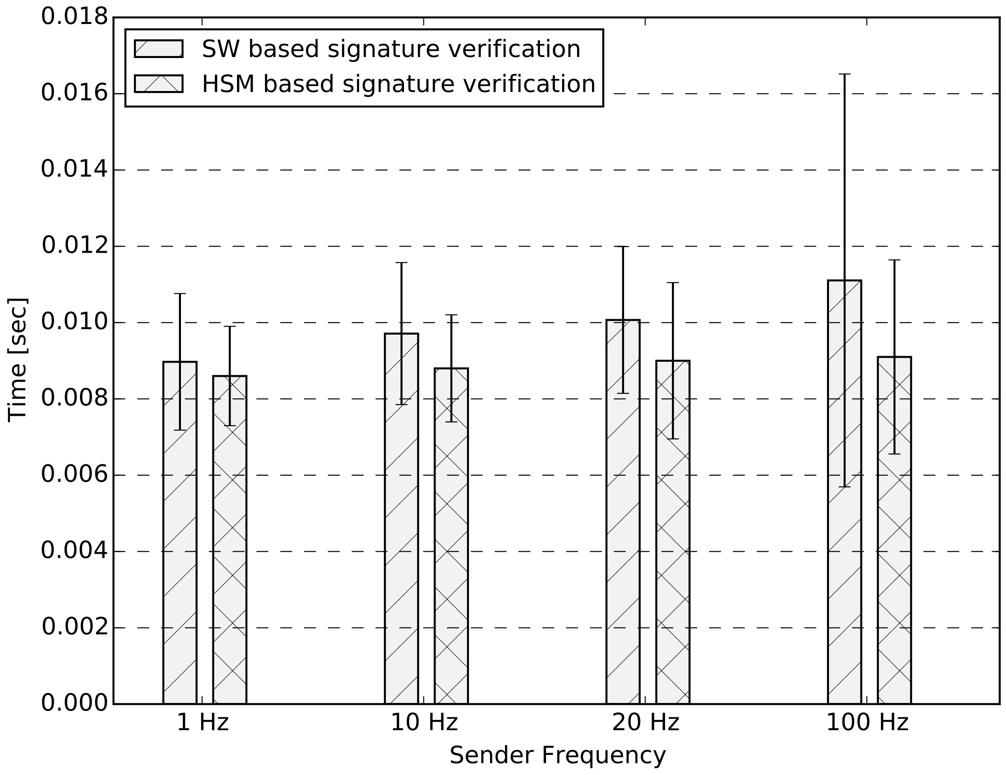


Figure 4.2: Performance Analysis of HSM based verification

receiver load of $5, 50, 100$ and $500\ msg/s$ respectively. For this scenario we try two configurations (described by the two different lines of the figure). The first line of Fig. 4.3 , labeled as *Full Security*, shows the time needed for generating a signature for a CAM message (at the sender), broadcasting the message and verifying the signature (at the receiver). This is compared to a configuration in which no signature generation and verification is performed at the sender / receiver. But even in this scenario, the CAM message includes the security header to have comparable load on the network.

As the figure shows, the End-to-End latency for both configurations are comparable. This serves as an indication that the time needed for signature generation and verification is not significant (both take approximately $0.0171s$). Nonetheless, we can observe an increase of the End-to-End latency for the cases that the 5 senders broadcast messages at a frequencies of 100 and $500\ msg/s$. The insignificant difference of the two configurations in this cases shows that this increase is not a result of the signature generation and verification processes but of the channel saturation and the handling of messages by the communication stack. This, in turn, results in a latency of $0.6s$ per message for the case of $500\ msg/s$ (compared to $0.0204s$ for $50\ msg/s$).

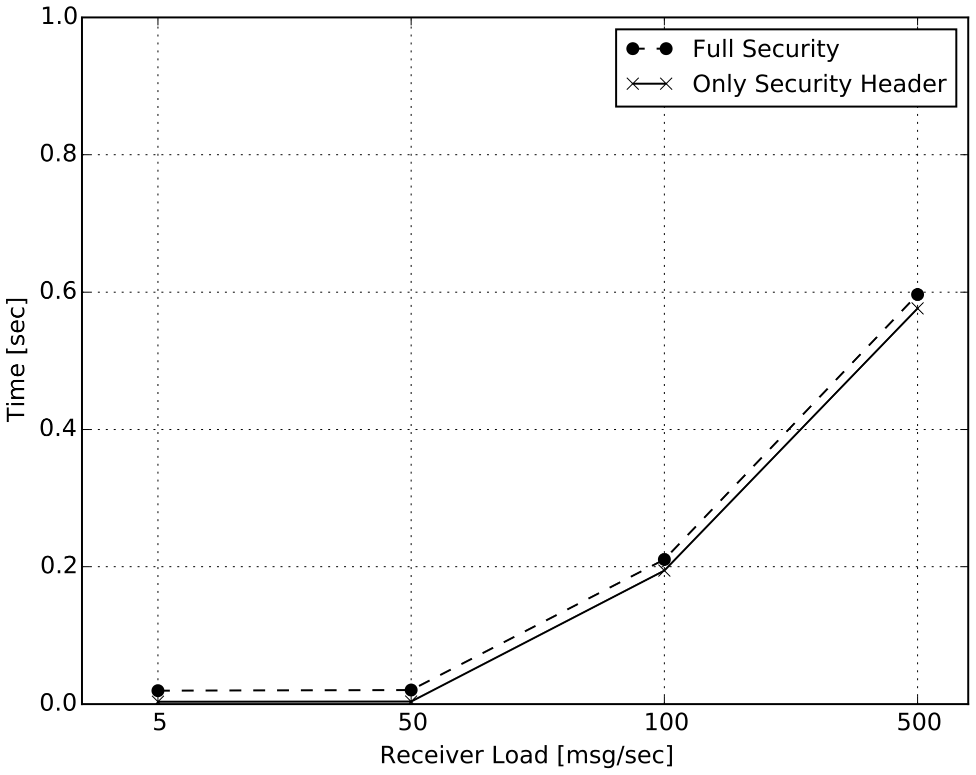


Figure 4.3: Experienced latency CAM messages between different ITS stations

4.1.3 Packet Loss

We now investigate what is the packet loss experienced for different configurations. To do this, we considered the most stressful scenario our testbed could support; 15 senders broadcasting and a single receiver verifying. Again, we plot the percentage of lost packets as a function of the sending frequency (i.e., $1, 10, 20$ and 100 Hz). We consider three cases: one case where packets are signed (by the senders) and verified (by the receiver), one where packets are not verified but are transmitted with the security header attached and one with no security (i.e., no security header is attached to the CAM messages).

As Fig. 4.4 shows, the packet loss in the case of no security is insignificant (1.41%) even for the high-load configuration (i.e., 1500 msg/s). Nonetheless, including the security header in the CAM messages significantly increases the packet loss (85% packet loss for a load of 1500 msg/s). It is worth mentioning that for the case that the receiver verifies the CAM messages no packet was lost by the VSS kit; all packets delivered by the communication stack were successfully verified by the receiver.

We believe that the main reason for the packet loss is mainly due to buffer effects. More specifically, when the channel is full packets will be lost since buffers can't queue enough

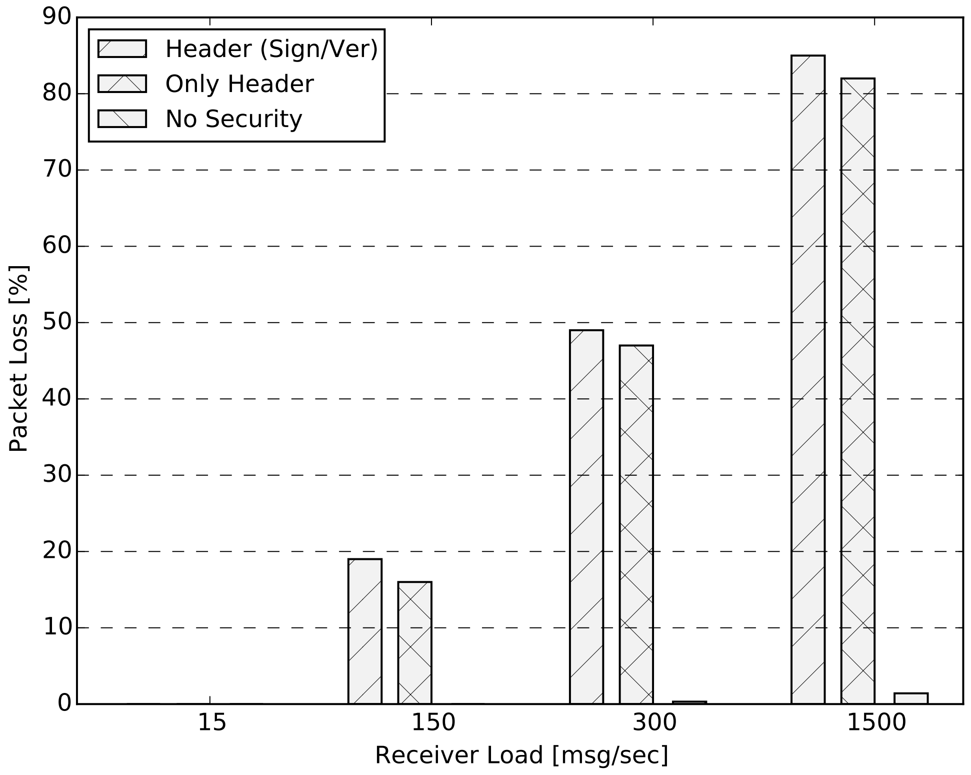


Figure 4.4: Packet Loss

packets or because the packets simply expire while waiting in the queues. Since the CPU also performs other operations it is more likely that packets get delayed and become stale. Furthermore, packets are delivered in bursts that overflow buffers. These effects depend mostly on the 802.11p driver.

Additionally, the observed packet loss is also deteriorated by collisions occurring during transmission. This is, in fact, confirmed by the insignificant packet loss when no security header is included. More specifically, the dominating packet-loss factor is the packet size: the size of a CAM message along with its security header is approximately 450 bytes if the complete signing pseudonym is included or 180 bytes if only the pseudonym's digest is attached. When no header is attached the size of the CAM message is 89 bytes.

Although these aspects merits further investigation, we believe that it also underlines the importance of a significant body of research performed in the scope of our project which focuses on optimal strategies for certificate omission [2, 3, 7].

Finally, we would like to highlight that the purpose of this scenario is to stress the channel and the VSS kit; thus, it is not representative of the channel load expected in real-life ITS deployments. Additionally, the multi-path effect, an artifact of our testbed set-up (i.e., 20

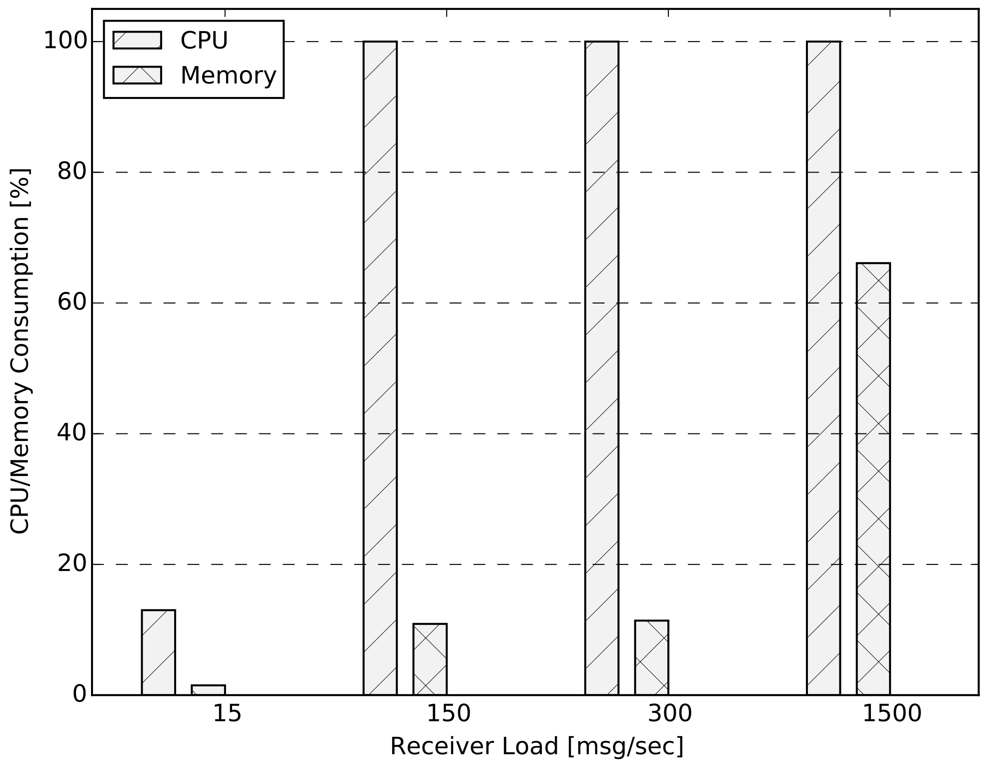


Figure 4.5: CPU Memory Consumption

ITS stations deployed in a confined room) becomes as an additional deteriorating factor for the performance of the wireless channel.

4.1.4 Resource Consumption

Finally, we examine the amount of resources consumed by the HITACHI communication stack and the PRESERVE VSS kit on the receiver. We plot the fraction of CPU and memory used during a the heavy-load scenario described in the previous section (i.e., maximum receiver load $1500\ msg/s$). As Fig. 4.5 in the most stressful configuration, the ITS components (network stack and VSS) consume 100% of *one* CPU core and 60% of the available memory.

4.2 Performance Tests under Attack

In this section, we investigate the performance of our VSS in the presence of adversaries launching different attacks. Table 4.1 summarizes the scenario parameters employed during the emulation of the attacking scenarios. *Senders* denote legitimate ITS stations whereas *Attackers* correspond to adversaries. For each case, we configure the legitimate ITS stations to a *Full Security* setting: legitimate senders correctly sign their messages and the receiver verifies the signatures. For the analysis we consider that a single NEXCOM box serves as the receiver.

Table 4.1: System parameters

No. of Senders	5
No. of Attackers	$2, 4$
Sender Frequency (each)	$10\ Hz$
Attacker Frequency (each)	$10, 20, 50, 100\ Hz$

Table 4.2 shows the results for different attacking scenarios. Besides the End-to-End latency and packet loss (considered in the previous sections), here, we additionally examine the *Packet Delivery Ratio* on the receiver side; the ratio of packets received from benign VSSs to all received packets, i.e., from both benign VSSs and attackers. This metric serves as an indication of the impact that adversaries impose to the system. More specifically, messages sent by the attackers will compete with the ones sent by the legitimate users. Overall, we see that the frequency at which the attackers broadcast their messages (i.e., the intensity of the clogging attack) significantly affects the ratio of packet delivery. This is an expected result and confirms the evaluation presented in Sec. 4.1.3.

In addition, from the table, we see with moderate frequencies ($10\ Hz$ and $20\ Hz$) from the attackers, the performance of VSS (especially for the cases of End-to-End delay and packet loss) does not significantly deteriorate, compared to the results from benign settings in Sec. 4.1. However, increasing the number of attackers and of their corresponding message frequencies (thus, a DoS attack case) results in a drastic slump in performance; higher End-to-End delay and packet loss, and lower goodput. For the DoS attacks, the attacker behavior corresponds to use cases A-SIG-01 and A-SIG-06; since they affect the receiver's performance comparatively more than the ones of A-SIG-03 and A-CER-03. For the former cases, the receiver verifies the attached certificates before the corresponding signatures or their timestamps on CAMs. For the later cases, the receiver can easily detect invalid packets by checking the packet format or certificate validity periods. At the same time, we see that the End-to-End latency for received packets from benign VSSs never exceeds $60\ msec$. This result is in accordance with the scenario of $15\ Senders$ broadcasting CAM messages at $100\ Hz$ each (Sec. 4.1).

Use Case Code	**No. of Attackers**	**Attacker Frequency (Hz)**	**Packet Delivery Ratio (%)**	**End-to-End Delay ($msec$)**	**Packet Loss (%)**
A-SIG-01	2	10	71.42	16.44	0.10
		20	55.55	18.92	0.07
		50	33.33	57.05	17.37
		100	20	58.76	49.09
	4	100	11.11	59.04	72.66

Use Case Code	No. of Attackers	Attacker Frequency (Hz)	Packet Delivery Ratio (%)	End-to-End Delay ($msec$)	Packet Loss (%)
A-SIG-06	2	10	71.42	15.21	0.05
		20	55.55	17.65	0.19
		50	33.33	57.13	17.05
		100	20	53.75	46.20
	4	100	11.11	55.46	69.57
A-SIG-03	2	10	71.42	16.52	0.20
		20	55.55	16.60	0.17
A-CER-03	2	10	71.42	17.87	0.13
		20	55.55	16.10	0.13

Table 4.2: Performance tests under different attacks

5 Conclusion

The tests presented in this deliverable aim at providing both a functional and performance analysis of VSS Kit 2. The VSS Kit testbed setup consists of 20 modems, and connected to 6 of these is an ASIC-based HSM. The other boxes are equipped with the software-only VSS kit based on OpenSSL as crypto backend. This setup qualifies us to perform testing to verify overall functionality and to benchmark timings in low-, medium- and high-loaded environments.

We deeply investigated the signature verification time, the End-to-End latency, the experienced packet loss and the overall resource consumption of the communication stack and the VSS kit.

The results for the signature verification time on the reciever box for a high load scenario (1500 CAM msg/s) show an average time for the SW of 0.011 sec and the ASIC based VSS Kit of 0.0091 s. Thus the HW accelerated VSS Kit outperforms the pure SW based implementation as expected, even with the additional delay introduced by the message processing of the Ethernet stack and the ASIC firmware.

During the end-to-end latency test for CAM messages the effect of channel saturation and the handling of messages by the communication stack became visible for higher load scenarios. Resulting in a latency of $0.6s$ per message for the case of $500\ msg/s$ (compared to $0.0204s$ for $50\ msg/s$).

An additional effect influencing the performance of the VSS Kit signature verification is the the packet loss. In the case of no security is insignificant (1.41%) even for the high-load configuration (i.e., 1500 msg/s). Nonetheless, including the security header in the CAM messages significantly increases the packet loss (85% packet loss for a load of 1500 msg/s). Our investigation show that this due to buffer effects depending mostly on the 802.11p driver. Additionally, the observed packet loss is also deteriorated by collisions occurring during transmission when the packet size increases. This is confirmed by the insignificant packet loss when no security header is included. While this effect is still under investigation, it shows the importance of optimal strategies for certificate omission to prohibit packet loss.

The OpenSource VSS Kit 2 and the testbed at KTH empower researchers and field operational test projects to understand the previous mentioned effects introduced by the security processing and adapt their implementation accordingly.

Bibliography

[1] N. Bißmeyer and F. Kargl, “PRESERVE D3.3 Joined FOT Trial Results,” PRESERVE consortium, Deliverable, June 2015.

[2] M. Feiri, J. Petit, and F. Kargl, “Evaluation of congestion-based certificate omission in vanets,” in *2012 IEEE Vehicular Networking Conference, VNC 2012, Seoul, Korea (South), November 14-16, 2012*, 2012, pp. 101–108.

[3] ——, “Formal model of certificate omission schemes in VANET,” in *2014 IEEE Vehicular Networking Conference, VNC 2014, Paderborn, Germany, December 3-5, 2014*, 2014, pp. 41–44.

[4] S. Gueron and V. Krasnov, “Fast prime field elliptic-curve cryptography with 256-bit primes,” *Journal of Cryptographic Engineering*, pp. 1–11, 2013.

[5] B. Lonc, D. Broekhuis, R. Moalla, M. Lagana, and J. Petit, “PRESERVE D3.1.2 Joint FOT test results V2X,” PRESERVE consortium, Deliverable, January 2014.

[6] M. Moser, N. Bißmeyer, D. Estor, M. Feiri, S. Joost, F. Kargl, M. Lange, S. Mauthofer, R. Moalla, J. Petit, M. Sall, and F. Smailbegovic, “PRESERVE D2.3 ASIC-based VSS Prototype,” PRESERVE consortium, Deliverable, July 2015.

[7] N. Ristanovic, P. Papadimitratos, G. Theodorakopoulos, J.-P. Hubaux, and J.-Y. L. Boudec, “Adaptive Message Authentication for Multi-Hop Networks,” in *International Conference on Wireless On-Demand Network Systems and Services (IEEE/IFIP WONS)*, January 2011, pp. 96–103.

[8] J. P. Stotz, N. Bißmeyer, F. Kargl, S. Dietzel, P. Papadimitratos, and C. Schleiffer, “PRESERVE D1.1 Security Requirements of Vehicle Security Architecture,” PRESERVE consortium, Deliverable, July 2011.

[9] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets,” in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud’10, 2010.