

Vanet Simulator

Report for the Computer Security exam at the Politecnico di Torino

Walter Dal Mut (161600)

Armand Sofack (157938)

tutor: Giorgio Calandriello

June 2009

Contents

1	Introduction	3
2	Framework Introduction	3
2.1	Security Implementations	3
2.2	From RSA to EC	4
2.3	Role of Certification Authority and CRL	4
2.4	Vehicle movement's	4
2.5	Security Implementations	6
2.5.1	Baseline Pseudonyms	6
2.5.2	Hybrid Scheme	7
2.5.3	Optimizations	8
3	UML Diagrams	9
3.1	Framework implementation	9
3.2	Class Diagram	9
3.3	Sequence Diagrams	9
4	Implementation, Simulations and Tests	13
4.1	Real Vanet implementation to Simulator	13
4.2	Simulation of Baseline Pseudonyms	13
4.3	Performance of Simulator	14

5	Comparisons and conclusions	15
6	Documentation	17
6.1	User Manual	17
6.1.1	Base Configurations	17
6.1.2	Vehicle Configurations	18
6.1.3	Why many log configurations	18
6.1.4	Log configuration	19
6.1.5	MySQL database configuration	19
6.1.6	Output Reading	19
6.1.7	Read vanet simulator operation from log	20
6.1.8	Install Vanet Simulator on Windows	21
6.1.9	Install Vanet Simulator on generic OS	22
6.1.10	Add certificates and private keys for Baseline Pseudonyms	22

1 Introduction

A Vehicular Ad-Hoc Network, or VANET, is a form of Mobile ad-hoc network, to provide communications among nearby vehicles and between vehicles and nearby fixed equipment, usually described as roadside equipment. It will contribute to safety and more efficient roads by providing information to drivers and concerned authorities around street condition, traffic problems and others. As all wireless networks, VANET can be easily subject to any kinds of security attacks then, for ensure a good security, VANET needs an appropriate security architecture that should be based on different security aspects like: Threat model, Authentication and key management, Privacy, Secure positioning. We focused our work in on the Authentication and Key management.

Our project was to implement a simple framework for the simulation of secure messages exchange in VANET. The simulator is essentially based on signing messages before send them in network area, to be able to verify the integrity of any messages received from the network area. All the messages are sent in broadcast way in a limited area

Techniques used to provide these security features are essentially based on two mains approaches provided by[1] which are :

- BaseLine Pseudonyms,

In this aspect a considerable amount of certified public key called pseudonym will stored in a vehicle, in order the ensure the privacy of the sender it will continuously change the public key certificate or pseudonym to sign a given amount of messages or to sign for a limited time this to avoid the sender to be retrieved by some receivers from a subsequent signed messages. These pseudonym should issued by a well known Certificated Authority in order to provide authentication.

- Hybric Scheme

This technique combines two approaches, coupling the pseudonym generation and the Group signature scheme. Instead of stored the pseudonyms like in the previous case, the pseudonym is generated on the fly when the vehicle wants to send a message, it has to be sign by a group public key provided to any vehicle by a Certificate Authority that operate as a group manager in order to guaranty the authentication between users assuming that all legitimate vehicles belongs to the same group.

<http://www.sigmobile.org/workshops/vanet2007/slides/3.pdf>

These two techniques during the implementations has been subject to some optimizations.

2 Framework Introduction

2.1 Security Implementations

Before starting with a real implementation of VANET Simulator we want to focus on a security features that has been chosen. This technology is focused around four main task: privacy, authentication, integrity and no-repudiation; and we can note that these properties are related to asymmetric cryptography.

We have two principals techniques for asymmetric cryptography that can provide these requires properties : RSA and Elliptic curves. At the beginning of this project, we have implemented the first one, RSA with 1024 bit of security level because almost useful functionality are already available for Java Platform. After few tests performed we meet a lot of problem related to the

implementation of the solution. First of all we have a real good time responses, see table 1 at page 4, for sign and verify and authority verification but the big problems where at the trasmission level caused by packet dimensions.

Table 1: RSA Speed Analysis

Provider	Sign	Verify	CA verification	sign/s	verify/s	CA verify/s
Java2 1.6u14	0.009284s	0.000521s	0.000525s	107.7	1919	1904

2.2 From RSA to EC

If we consider to send on network beacons of 200 bytes with RSA technique we have an enormous cryptography overhead, see table the following table 2 at page 4. The real problem with RSA

Table 2: RSA Cryptography Overhead

Payload	Signature	Certificate
200	128	1029

is the signature length and more precisely the certificate length. If we want 1024 bit of security level we cannot modify signature length but we can modify the certificate length. To do that, we have constructed a X509 striped certificate for reduce dimension of packet removing all useless information from the certificate. But the length of the certificate reduced dimension to 750 bytes, that still too big. The real problem is length of public key inside certificate and we can not modify that unless we reduce security level.

At this point we have decided to use elliptic curves according to data provided in [1]. Consider the same data, for payload, around 200 bytes with 163 bit of security level using elliptic curves we have very little certificates and public key. The longest message in the network is around 500 bytes, during this monography we analyze optimizations for reduce packet dimension to 250 bytes. See table 3 at page 4 for more details on cryptography overhead.

Table 3: Elliptic curves cryptography overhead

Payload	Signature	Certificate
200	48	234

2.3 Role of Certification Authority and CRL

Certification authority and CRL for VANET Simulator is actually a point of discussion [1], in particular the positioning of CA and CRL is really difficult for vehicular networks. VANET Simulator implement only one CA and not use certificate revocation list.

2.4 Vehicle movement's

The simulator is also able to simulate vehicle movements, it's implementation is really simple because simulator do not consider the collisions of two or more vehicles or particular position for a given vehicle. VANET simulator move all the vehicles on wireless area according to the

velocity information, computing how many cells of WIFI it have to jump. Vehicle movements is only in horizontal direction, from left to right, no others movements are considered. If a vehicle go out of the wireless area it will remain there for one cycle, the vehicle stop any transmission and reception. After a cycle time it will re-enter into WIFI with 180 degree of rotation, y coordinates do not change.

2.5 Security Implementations

2.5.1 Baseline Pseudonyms

For the i -th pseudonym the CA provide a valid certificate with the public key signed by the CA and a private key for signing messages.

The easy way for realize this security implementation is to attach after every message a digital signature and the certificate used for signing message. In this way a vehicle which receive the message can verify the validity of certificate and so verify the digital signature of message. It's really important that pseudonyms is used only for a few time τ for provide security against the track message reception, because if I use the same certificate for a long period of time an attacker can follow the certificate and identify the car which send messages. This implementation it's really difficult to implement because the message is too long because if we sum the message length plus the signature plus certificate with public key we have a big packet to send on network. For resolve the overhead problem we can use three optimizations [1].

Certificate and identify the car which send messages. This implementation it's really difficult to implement because the message is too long because if we sum the message length plus the signature plus certificate with public key we have a big packet to send on network. For resolve the overhead problem we can use three optimizations [1].

The main problem of Baseline Pseudonyms is based on certificate because this implementation use a preset of certificates, around a thousand, for sign beacons and send on network. For privacy reasons we can not re-use certificates and after a period of time, for that the life-time of Baseline Pseudonyms is dependently from certificate preloaded and this is a real problem because we have to insert new certificates after a period of time and for users that could really invasive.

2.5.2 Hybrid Scheme

Brings the two approaches together the pseudonym generation and the group signature.

A Group signature scheme allows each member of the group to anonymously sign a message on behalf of the group. The group members are supposed to be all the vehicles registered with the same Certificate authority. Each vehicle is equipped with a secret group key or also called private group used to sign a group message and a group public that allow a validation of any group signature generated by any group member. The group keys are installed when the car is manufactures and refreshed at the periodic checkup.

Each node generates its own pseudonym on the fly then uses its private group key to sign the pseudonym in other to generate the corresponding self certificate. This self certificate generated will be verify at the reception by any group member using the group public key.

We can also note that in other to ensure the privacy, the group signature allows at each user to not be traced by another group member, but in case of necessity liability is guarantee so, the certificate authority can disclose the identity given a signature transcript.

We can also added that, there is no need to reuse the pseudonym because they are generated on the fly and messages can also be linked if needed, but according to some optimization scheme pseudonym could be used for a given amount of time.

The main functions implemented are describe are the following:

- Securize

Basically used to generate a secure payload to sent according the scheme shown in [2] which consist on generate on the fly a pseudonym and try to provide the followings features :

- Integrity : we generate for each new message to sent a new pseudonym that will used to sign the message. The signature on that message is added to the payload and also the correspondent self certificate generated by the group signature on the pseudonym generated.
- Authentication : The pseudonym is certified , signed by the private group key in other to authenticate the sender. But due to the complexity of the group signature algorithm the function to generate the self certify will be dummy taking into account the signature time and the signature bytes size provided by [2]. Both pseudonym and the signature on it will also added at the message and will be consider as the related certificate.

Finally the packet generated by the function securize will have the following structure:

KeyID is a random number generated to identify the current pseudonym used to sign the payload : 4 bytes.

The payload needed data : 200 bytes

The signature on the payload : 69 bytes

The length of the signature in order to recover it form the final packet : 4 bytes

The self certify of the pseudonym generated on the fly : 298 bytes

Total packet size represented as a vector of bytes is 570 bytes.

- Privacy : the group signature works in other to avoid that some user can be identity through a subsequent messages by it signed.Vehicles are traceable only when using the same pseudonym (as in BaseLine Pseudonym scheme.)

- Verify

The function verify specially check the integrity of the received message and the authen-

ticity of the sender. Is a Boolean function that receive from the transceiver a message already reconstructed in a class and can extract all additional information added in the securize function in order to run the verification .

Form these information like the signature performed on the message , the pseudonym used to signed the message and the self certificate , the verify check the received messages and return true or false in order in the checking is right or wrong.

2.5.3 Optimizations

We can note in the securize part that we sent principally two kinds of messages the first one generated in long mode or without optimization and the second one in the short mode according the optimization scheme. The size of the message sent in long mode is 570 respect to the size of the payload, the overhead here is 370.

One of the improvement we did here were to try to reduce that overhead keeping always the security level . The technique implemented here is base on the optimization 1&2 provided by [1].

- Optimization 1: At the sender side, the self certificate is computed only once per pseudonym, because it will remains unchanged throughout the pseudonym lifetime that can be set in a configuration file of the simulator. For the same reason, at the verifier's side the certificate is validated upon the first reception and stored, even though the sender appends it to multiple (all) messages. For all subsequent receptions, if the certificate has already been seen, the verifier skips its validation. .
- Optimization 2: The sender appends the signature on the message, the pseudonym and the self certificate once every α messages (beacons); it appends only the signature on the message on the remaining $\alpha - 1$ ones. We call α the Certificate period. In this case the value of α can change through the configuration file. The short mode message remain unchanged, and all messages will carry a 4-byte ID field called Key ID, that is, a random number indicating which pseudonym must be used to validate the signature on the message. That ID field does not affect privacy as there is a 1:1 correspondence with the pseudonym. When a pseudonym change occurs, the new triplet signature on the message, pseudonym and self certificate must be computed and transmitted
These two optimization combined allow us to reduce to 72 bytes the overhead for the $\alpha - 1$ messages sent after any new computation on the pseudonym, which is relatively significant.

3 UML Diagrams

3.1 Framework implementation

The Vanet Simulator is based on stack representation, in particular is composed by a vehicles with a transceiver for sending and receiving messages on the network and under a transceiver there is a security box which use the security implementation that could be set before any simulation simulation.

3.2 Class Diagram

3.3 Sequence Diagrams

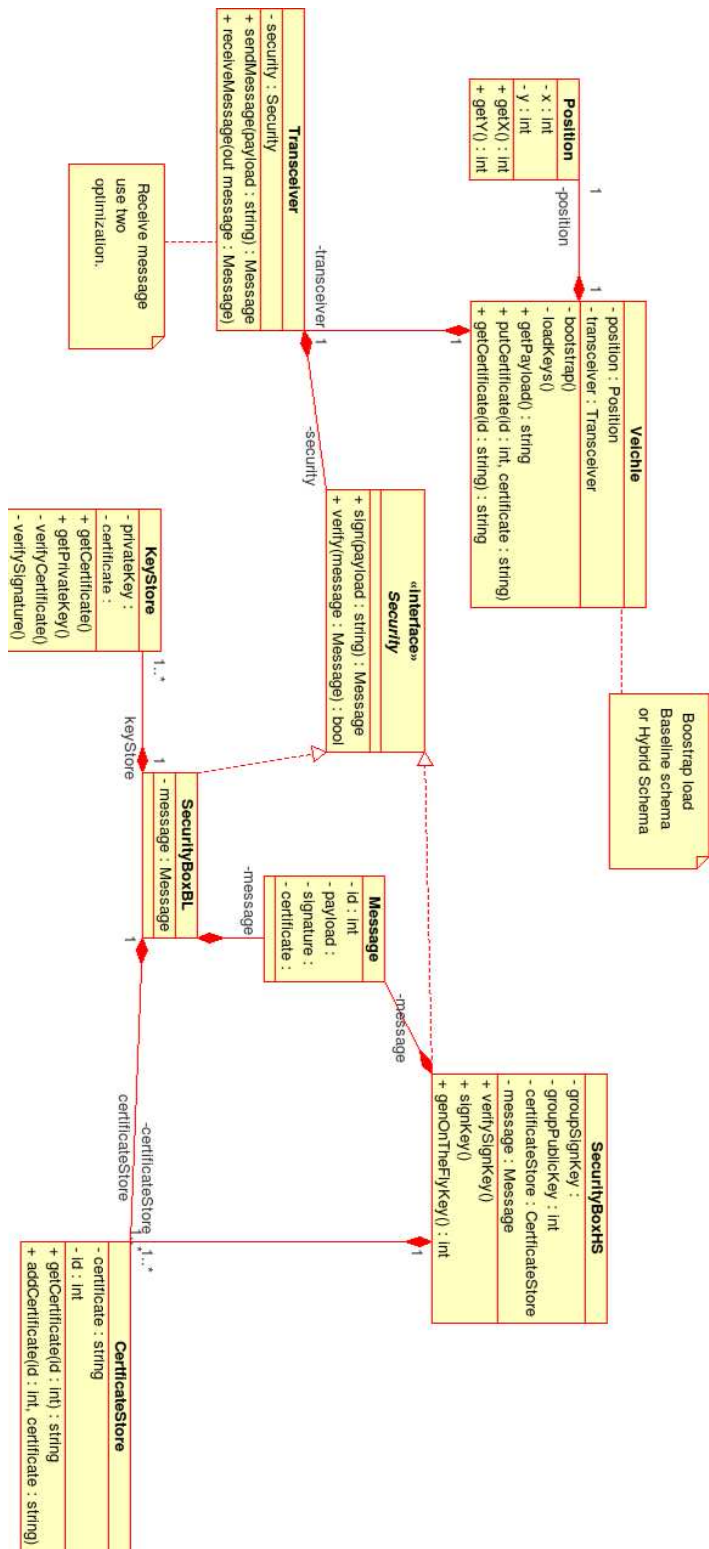


Figure 1: Class Diagram

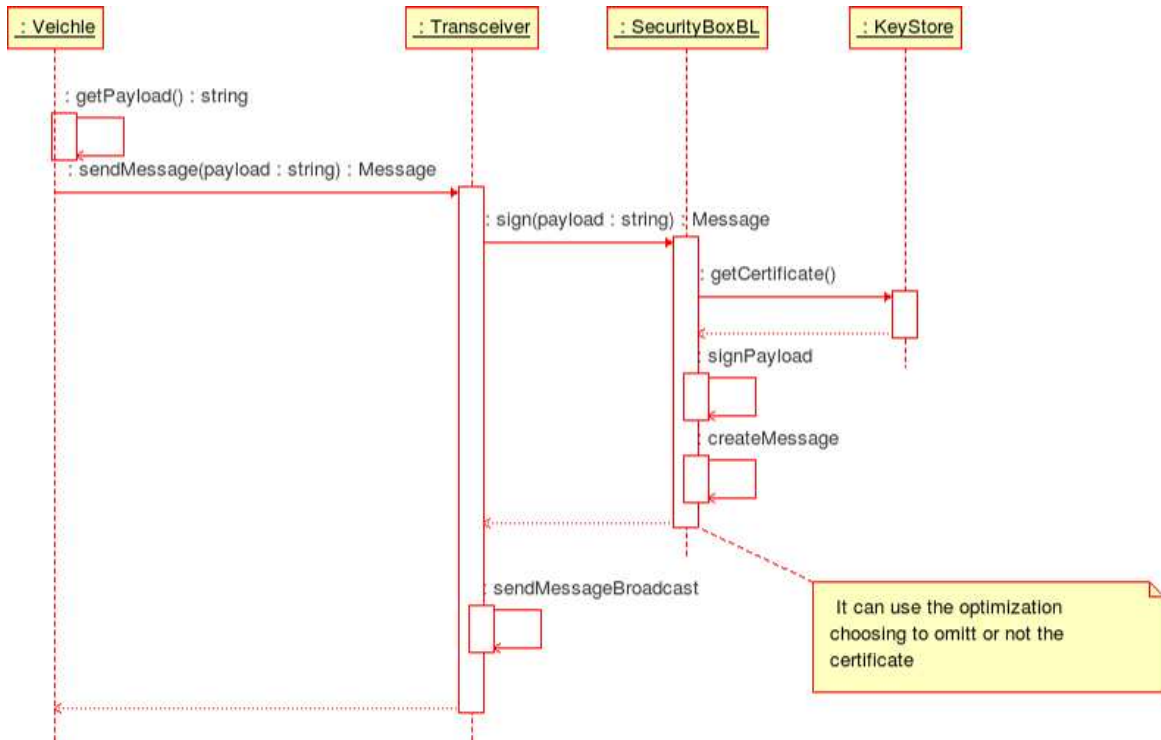


Figure 2: Sequence Diagram Baseline Pseudonyms securize and send message

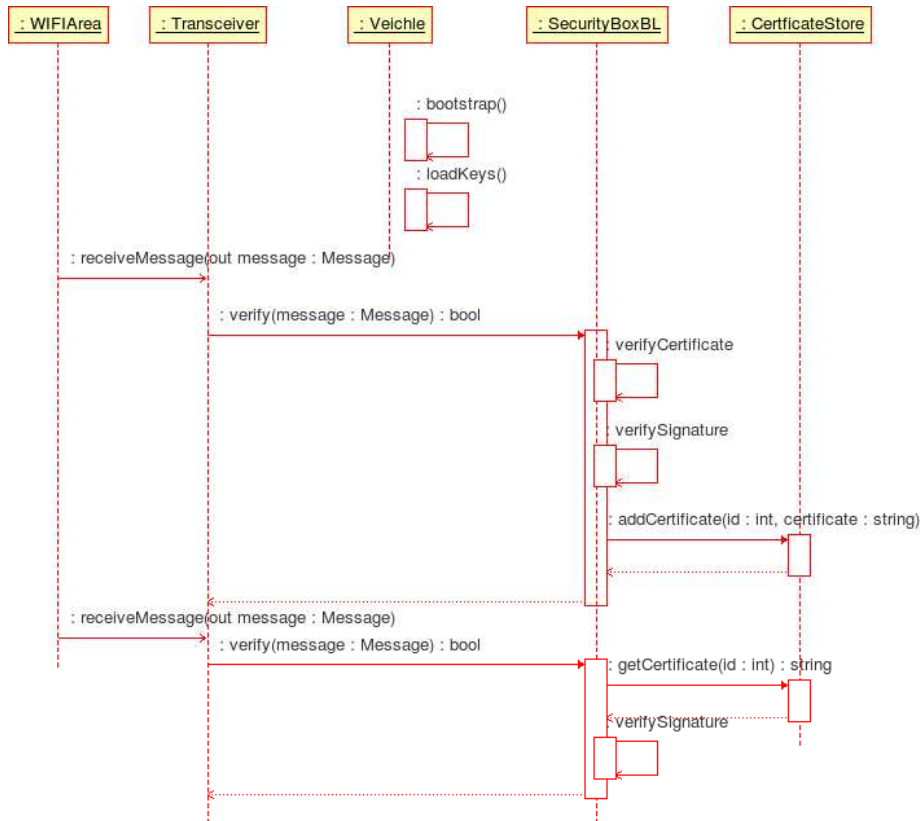


Figure 3: Sequence Diagram Baseline Pseudonyms receive and check message

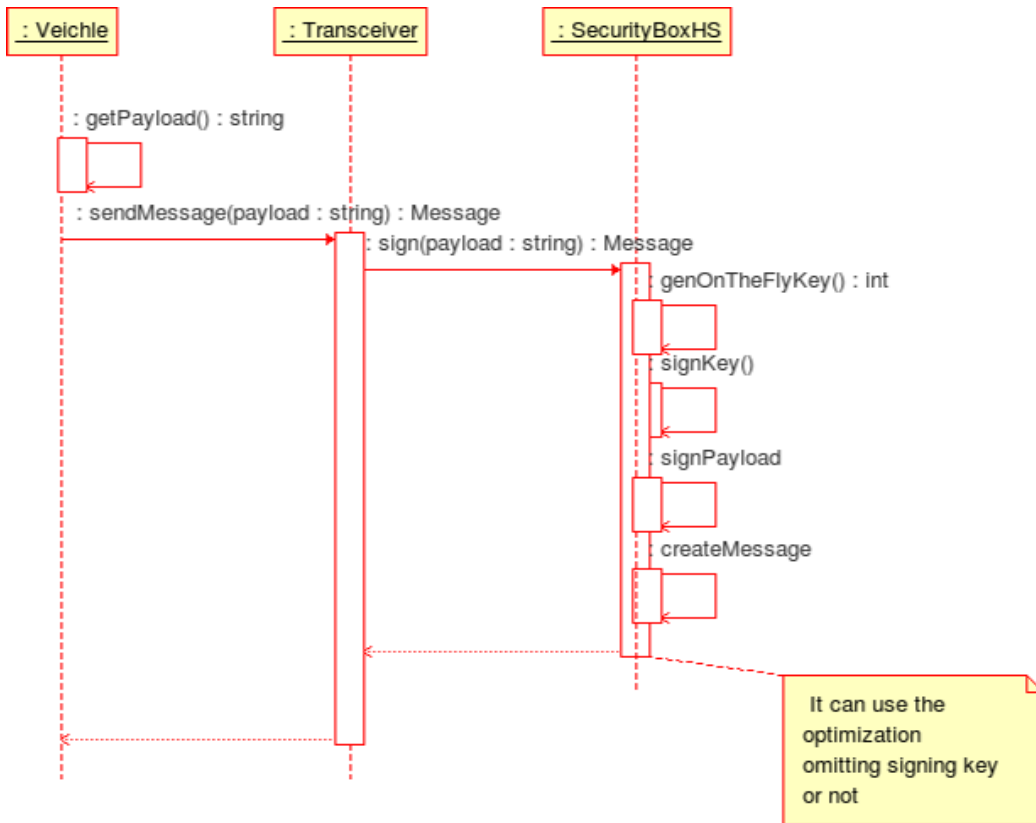


Figure 4: Sequence Diagram Hybrid Scheme securize and send

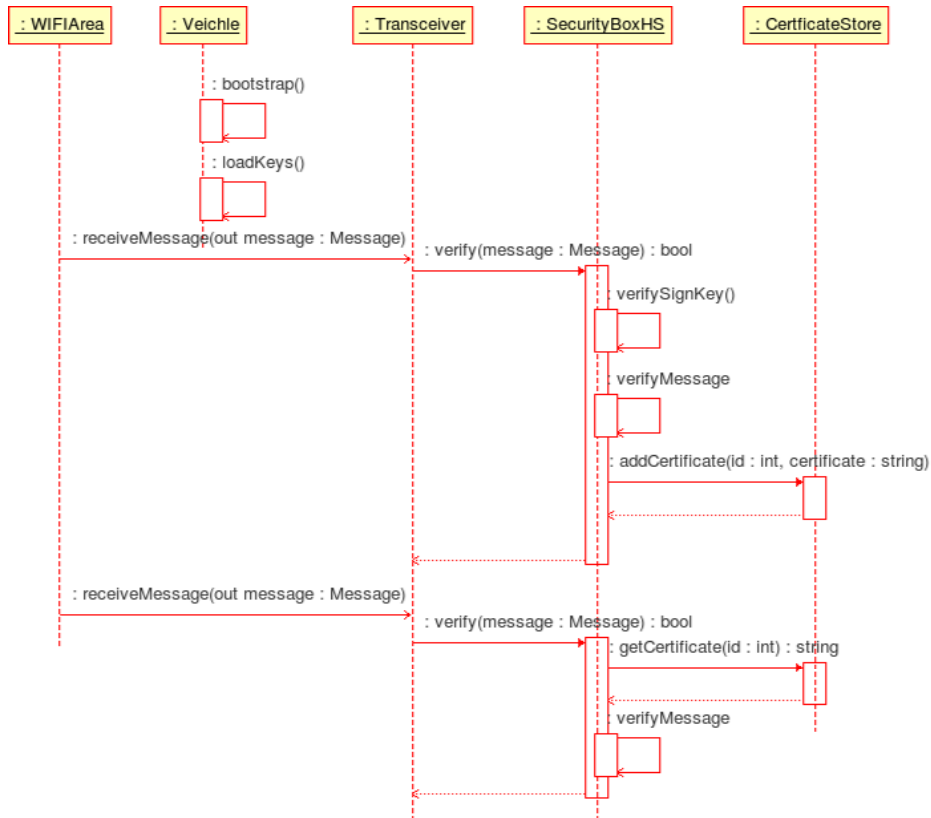


Figure 5: Sequence Diagram Hybrid Scheme receive and check

4 Implementation, Simulations and Tests

VANET Simulator is written in Java because this programming language is really powerful and flexible to use but more over we see the Java problems with cryptography. First of all the Java security implementation, at this time (version 1.6.x), do not include elliptic curves for asymmetric cryptography and we have been able to find on Internet a provider which release elliptic curves : found *Bouncy Castle*¹, then we decided to use 163 bits of security level of digital signature *nistb*, after that we have also computed with *openssl*² the typical speed³ for sign and verify a digital message, see table 4 at page 13. The Bouncy Castle implementation

Table 4: OpenSSL Speed Analysis

Provider	Sign	Verify	sign/s	verify/s
OpenSSL	0.0017s	0.0050s	583.5	199.3
Bouncy Castle	0.028031s	0.017342s	35.6	57.6

is slowly than OpenSLL and we have found a difference time order between C (OpenSSL) implementation and Java implementation (Bouncy Castle), see table 4 at page After additional analysis we verified the certificates with the certification authority and for doing that in Java it last typically 0.038708s roughly $25.83 \frac{verify}{s}$

4.1 Real Vanet implementation to Simulator

The main task of Vehicular Networks is privacy, integrity and not repudiation of messages and for do that we use a lot of methods, like pseudonyms change or group keys. In the simulator we are obliged to use all network stack from application level to physical level and this feature remove privacy because we are not able to change MAC address dynamically and if we analyze the network traffic we can found MAC address and track vehicle moves. Real system do not use the complete network stack and work only on level two (MAC layer) with changing MAC address for every beacon sent on the network.

4.2 Simulation of Baseline Pseudonyms

Before starting simulation in baseline pseudonyms you have to modify the configuration file *base.properties*⁴ under folder *properties* in root directory of VANET Simulator.

For Baseline Pseudonyms we can realize three different optimizations picked up from [1] but in this simulator we have implemented the second optimization, in particular the sender append *beacon identification*, *signature*, *public key* and *certificate* only for α messages and transmit only *beacon identification* and *signature* for remaining $\alpha - 1$ beacons.

The beacon identification is random number compute on four byte without consider other vehicles, is reasonable to use this method without remember other IDs is because probability which two vehicle use the same identification number is really small. Baseline Pseudonyms use optimization two from [1] for limit cryptography overhead, see section 2.1 at page 3 for importance

¹See bibliography at 24 for more details on this provider

²See bibliography at page 24 for more details

³For computation we have used: Pentium 4 Core 2 Duo, processor: Intel T5450 (1.66GHz, 667 Mhz FSB, 2 MB L2 Cache, 2 GB DDR2 RAM

⁴See the user manual for more information around configurations. Section 6.1.1 at page 17

of cryptography overhead, and you can change parameters for realize you personal optimization, for example certificate reattach after tot beacons and certificate or change certificate after a period of time.

4.3 Performance of Simulator

For analyze performance of Simulator (figure 6 at page 14) we have computed statistics for understand the number of signature which we are able to do with the simulator. We have set up the Baseline Pseudonyms simulator and set logger into console, after that we have ran system and observed behaviors after changing the number of beacons/sec and the number of vehicles into wireless area.

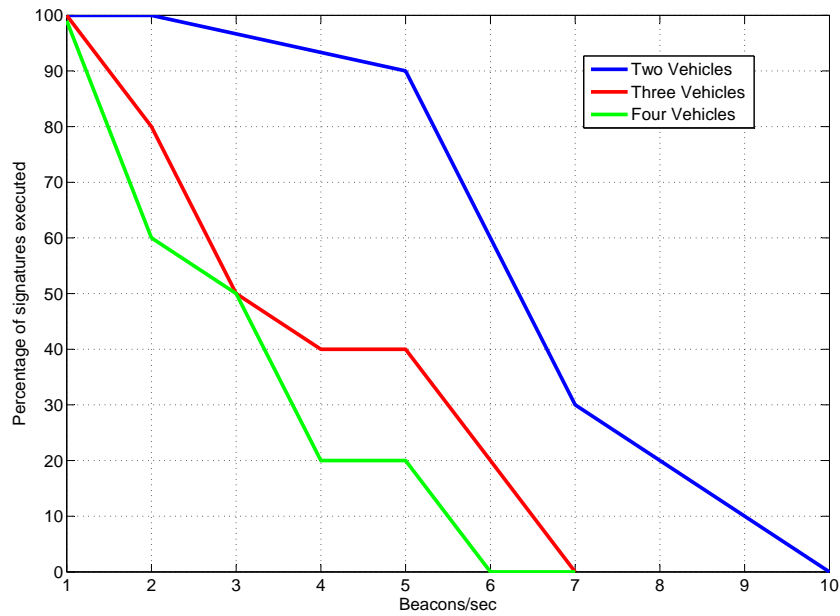


Figure 6: Performance of simulator

5 Comparisons and conclusions

We have able at the end of this work to design a very simple as possible simulator that were able to provide the principals properties required to exchange secure messages in broadcast between vehicles in movement in a given WiFi area. During the simulation and the development the behavior of the two scheme were little different in some points the principal difference has been noted at the test where for the Baseline Pseudonyms increasing the sending rate (number of messages sent per second by each vehicle) some wrong messages start been appeared due to the impossibility of some vehicle to sign some messages or the overlap time of the life time of two consecutive s pseudonyms. The main reason of this can also be due to the fact that simulator run in a same computer.After have testing the exchange of the message between two vehicles by installing the framework on two separated computers connected we have noted the difference and some improvements This situation is not more present in the hybrid scheme due to the fact that some of the mains functions are not really implemented like the group signature and the group verification that are only dummy function.

The log report below display the result after running Baseline Pseudonyms and Hybrid Scheme scheme with the same parameters.

Baseline Pseudonyms typical response

```
19:43:10,752 [1664] [Vehicle - 1]
    DEBUG vanet.Vehicle:105 - Vehicle 1 position, x=37 y=20
19:43:10,752 [1664] [Vehicle - 1]
    DEBUG vanet.Vehicle:108 - Vehicle 1 send new message
19:43:10,754 [1666] [Vehicle - 1]
    DEBUG vanet.security.BaseLinePseudonyms:128 -
    Message sent in LONG MODE. Certificate expired use new pseudonymous.
    Certificate ID: 39883
19:43:10,789 [1701] [Vehicle - 2]
    DEBUG vanet.Vehicle:105 - Vehicle 2 position, x=53 y=50
19:43:10,789 [1701] [Vehicle - 2]
    DEBUG vanet.Vehicle:108 - Vehicle 2 send new message
19:43:10,789 [1701] [Vehicle - 2]
    DEBUG vanet.security.BaseLinePseudonyms:128 -
    Message sent in LONG MODE. Certificate expired use new pseudonymous.
    Certificate ID: 35990
19:43:10,851 [1763] [Transceiver For Vehicle]
    INFO vanet.Transceiver:123 - Skip auto-verification for message: 39883
19:43:10,973 [1885] [Transceiver For Vehicle]
    INFO vanet.Transceiver:113 - Message secure: 39883
```

Hybrid Scheme typical response

```
20:17:00,886 [3152] [Vehicle - 1]
    DEBUG vanet.Vehicle:105 - Vehicle 1 position, x=91 y=20
20:17:00,886 [3152] [Vehicle - 1]
    DEBUG vanet.Vehicle:108 - Vehicle 1 send new message
20:17:00,887 [3153] [Vehicle - 2]
    DEBUG vanet.Vehicle:105 - Vehicle 2 position, x=119 y=50
```

```
20:17:00,887 [3153] [Vehicle - 2]
    DEBUG vanet.Vehicle:108 - Vehicle 2 send new message
20:17:00,914 [3180] [Transceiver For Vehicle]
    INFO vanet.Transceiver:123 -
        Skip auto-verification for message: 56512
20:17:00,954 [3220] [Transceiver For Vehicle]
    INFO vanet.Transceiver:113 -
        Message secure: 56512
20:17:00,954 [3220] [Transceiver For Vehicle]
    INFO vanet.Transceiver:123 -
        Skip auto-verification for message: 3442
20:17:00,962 [3228] [Transceiver For Vehicle]
    INFO vanet.Transceiver:113 -
        Message secure: 3442
```

We can said that up to these basics functionalities of the framework implemented, some of the improvements can be added. In the optimization scheme used in this case is not so efficient because we did n't tacking in account the case where some receiver lost two consecutive messages that can be the last signed by the old pseudonym and the first signed by the new pseudonym . This could be improved by adding in the combination scheme the optimization 3 provided by [2] and try to implemented in some robust scheme both in secure and in verify.

6 Documentation

6.1 User Manual

In this part of monography we explain the possibilities offered by configurations for using the Vanet Simulator in all of its parts. In particular the main features offered by simulator are changing the security implementation passing by Baseline Pseudonyms and Hybrid Scheme implementations and configure vehicles on the road, the number of beacons sent during the simulations and modifying logging system for understanding results of simulation.

The Vanet Simulator is completely configurable modifying its configurations files under the folder *properties* in the root directory of simulator.

6.1.1 Base Configurations

The base configurations provide modifications in the core of Vanet Simulator, in particular you can change the *base.properties* file for change core properties like beacons sent, security implementations and others base properties. If you open the configuration file you see:

```
#Max speed in km/h
max_speed = 140
#Max 802.11 cover in meters
wifi_cover = 200
#Access Point broadcast point
server_broadcast_point = 127.255.255.255
#Server Port
server_port = 55055
#Beacons/sec
beacons_sec = 0.1
#If you want no moves of vehicles
no_moves = true
#choose simulator BP or GS
simulator = bp
#max certificate validity into area in seconds
maxCertificateValidityTime = 33
#Reattach certificate every tot beacons
reattachCertificate = 5
#MYSQL properties
mysql_host=127.0.0.1
mysql_username=root
mysql_password=
mysql_database=vanet
#Define the log system
# 0 MYSQL log
# 1 File log
# 2 StdOut log
logSystem=0
```

For detailed information you can see table 5 at page 18.

Table 5: Base Configuration specifications

Property Name	Property Translation	Property Type
max_speed	The maximum speed of vehicles	int
wifi_cover	The maximum wireless area coverage	int
server_broadcast_point	The broadcast node for send messages	string
server_port	The port for receive messages	int
beacons_sec	The number of beacons sent in one second	float
no_moves	Lock vehicles into the map	boolean
simulator	The simulator which you want use. BP for baseline implementations. GS for group signature implementation	string
maxCertificateValidityTime	The maximum time for certificate validity	int
reattachCertificate	Reattach the certificate every tot beacons	int
mysql_host	The host for mysql	string
mysql_username	Username for authenticate into mysql	string
mysql_password	Password for authenticate into mysql	string
mysql_database	Database to use	string
logSystem	The log system which you want use. 0 for mysql log system. 1 for log data into a files 2 for log data on console	int

6.1.2 Vehicle Configurations

The vehicles configurations set the status of roads into the simulator. In particular you can modify the number of vehicles into the road, velocity and initial position.

The configuration file for vehicle is XML (eXtensible Markup Language) based and is named *vehicles.xml* and positioned into folder *vehicles*; if you open this file you see:

```
<?xml version="1.0" encoding="UTF-8"?>
<Vehicles>
  <Vehicle id="1" speed="100" x="10" y="20" />
  <Vehicle id="2" speed="120" x="20" y="50" />
</Vehicles>
```

Every tag, excluding root tag, identify a new vehicle with attributes like options, in particular you can modify the vehicle identification number changing the *id* attribute or you can change the vehicle speed modifying the *speed* attribute or the position of the vehicle using the *x* or *y* attributes. For the Baseline Pseudonyms operating mode you have to link certificates and private keys to each vehicles, for doing that you have to follow instruction in section 6.1.10 at page 22

6.1.3 Why many log configurations

The log system for this application is really difficult, in fact the normal std out log system is too slow and produce conflicts if you send a lot of beacons during sign and verify operation but it's really useful because you can understand immediately what the system are doing in real time, the other methods are a middle solution for see result and velocity during sign and verify

and the best solution for velocity but difficult to understand in real time the system but it's useful for post-processing. For this reason we have written three type of log system for use the best method when that are compatible with the simulation.

6.1.4 Log configuration

The log system use the *log4j* module for write sensible information of simulator. The system provide three log configurations, on standard output stream, file stream or on MySQL database. The configuration of log system it's really powerful and you can set the level of logging or change the log representation for standard out stream or file stream. The configuration of log system is divided into three file, ones for each method and it's collected into folder *properties* which names *stdout.properties* for standard out, *file.properties* for file stream or *mysql.properties* for MySQL database log system.

6.1.5 MySQL database configuration

For using MySQL database log system you have to configure the database before launching the Vanet Simulator. You have to create or import a database with tables definition into MySQL using the *vanet.sql* file under the *properties* directory.

For create the database and tables definition you have to enter in you MySQL command line and create a new database using command:

```
mysql> CREATE DATABASE vanet;
```

After this step you have to create a table in the new database using commands:

```
mysql> use vanet;
...
mysql>CREATE TABLE IF NOT EXISTS 'logs' (
  'log_id' int(11) unsigned zerofill NOT NULL auto_increment,
  'level' varchar(255) NOT NULL,
  'class_name' varchar(255) NOT NULL,
  'method_name' varchar(255) NOT NULL,
  'message' text NOT NULL,
  PRIMARY KEY ('log_id'),
  KEY 'level' ('level')
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
...
```

After this step you have configured the MySQL database for record logs from Vanet Simulator.

6.1.6 Output Reading

VANET simulator have a log system for showing information, this lines are leveled on five states:

trace Normal trace operation for follow program execution line

debug Debug general information

warn Warning, little problem, not critical

error Severe error

fatal Impossible to run over, the program terminate immediatly

The simulator can provided log information's in three differents ways, directly on your screen using *std out* or writing into a new file or storing into a database table. All log system show the same information but structured in different ways, a typical std out representation is:

```
07:22:22,870 [1632] [Vehicle - 1] (new line for readability reasons)
DEBUG vanet.Vehicle:105 - (new line for readability reasons)
Vehicle 1 position, x=37 y=20(new line for readability reasons)
```

```
07:22:22,870 [1632] [Vehicle - 1] (new line for readability reasons)
DEBUG vanet.Vehicle:108 - (new line for readability reasons)
Vehicle 1 send new message(new line for readability reasons)
```

```
07:22:22,905 [1667] [Vehicle - 2] (new line for readability reasons)
DEBUG vanet.Vehicle:105 - (new line for readability reasons)
Vehicle 2 position, x=53 y=50 (new line for readability reasons)
```

All lines are written with the same logic, in particular start with the time using format: hour, minutes and seconds followed by microseconds. After that into square brackets you have the number of milliseconds elapsed since the start of program, followed by the name of the Java class which execute log, always into square brackets. Then in upper case you have the level of log, followed by the complete name of class followed by double dots and the line of program execution. After that a free text message which inform you on operations, like state changes or errors.

Logging into files follows the same representation of stdout. Database implementation instead use one table organized in five fields, see table 6 at page 20. **Log identification** is an unique

Table 6: Database table for logging system

Log identification	Level of log	Class name	Method name	Free message
--------------------	--------------	------------	-------------	--------------

number for log entry, other fields represent the same information already expressed for stdout log system, in particular the **level of log** is always on six step (trace, debug, info, warn, error, fatal), the **class name** is the name of java class which use the log system, the **method name** is name of method which call the log system and **free message** express how the system have logged into database.

6.1.7 Read vanet simulator operation from log

Before closing the output reading we want to explain with an example the normal flow of Vanet Simulator with vehicles on the roads. The simulator is multi-thread and don't use synchronization, for that reason the output could be confused. In this section we log pieces of log output but for readability reasons we remove all information which do not have sense for this section, like timestamp and other not useful parts. In addition long line are splitted into two lines, but we consider for one line only those who start with square brackets.

```
[Vehicle - 1] DEBUG vanet.Vehicle:105 - Vehicle 1 position, x=37 y=20
[Vehicle - 1] DEBUG vanet.Vehicle:108 - Vehicle 1 send new message
[Vehicle - 2] DEBUG vanet.Vehicle:105 - Vehicle 2 position, x=53 y=50
[Vehicle - 2] DEBUG vanet.Vehicle:108 - Vehicle 2 send new message
[Vehicle - 1] DEBUG vanet.security.BaseLinePseudonyms:128 - Message sent in
LONG MODE. Certificate expired use new pseudonymous. Certificate ID: 12884
[Vehicle - 2] DEBUG vanet.security.BaseLinePseudonyms:128 - Message sent in
LONG MODE. Certificate expired use new pseudonymous. Certificate ID: 54389
...
...
[Vehicle - 2] DEBUG vanet.security.BaseLinePseudonyms:178 - Message sent in
SHORT MODE for certificate. Certificate ID: 54389
```

In the first 6 line, two vehicles have sent new message and these messages are securized using a new certificate (pseudonym), when we use the word *long* we want identify the attach of certificate at the end of message instead with word *short* when we send a message without certificate, in line with optimization 2 in [1].

The system send message in broadcast and for that reason we receive also our messages and in line:

```
[Transceiver For Vehicle] INFO vanet.Transceiver:123 - Skip auto-verification
for message: 54389
```

the transceiver skip verification of self-messages for reduce computational resource. The same operation is executed by the other vehicle. When a vehicle receive a message securized by another vehicle, check it validity and write result of verification on log

```
[Transceiver For Vehicle] INFO vanet.Transceiver:113 - Message secure: 54389
```

If a signature is not valid or a vehicle do not have certificate for check a message, write a message insecure log:

```
[Transceiver For Vehicle] WARN vanet.Transceiver:118 - Message insecure: 2804
```

Another information expressed by logs is around positioning of vehicles, in particular when a vehicle go out of wifi range the system express this condition with a message like this:

```
[Vehicle - 1] DEBUG vanet.Vehicle:105 - Vehicle 1 position, x=276 y=20
[Vehicle - 1] INFO vanet.Vehicle:113 - Vehicle 1 is not in the WIFI area,
set x position equal to 0
```

6.1.8 Install Vanet Simulator on Windows

For install Vanet simulator on Windows operating system you can use the installer *setupVanet-Simulator.exe* and follow the screen information for complete the setup of application. After install procedure you have to open a new console and go into install directory and send the command

```
C:\Program File\Vanet Simulator\>java -jar vanetSimulator.jar
```

After this command you see the bootstrap procedure and after the system run. The default log operation is the standard out and you can see directly all the informations.

The common output on screen are this:

```
.:: Bootstrap :.:
Loading base properties
Base properties loaded
Loading vehicles configuration
security/certificates/1/c
security/certificates/2/c
Vehicles configuration loaded
.:: Bootstrap end :.:
```

6.1.9 Install Vanet Simulator on generic OS

For install Vanet Simulator you have to setup all folders and executable jar manually. Create new folder in a point of file system and enter in it, after that copy the content of *build* directory and now you can send the command for start the simulator.

```
name@domain$ java -jar vanetSimulator.jar
```

After this command you see the bootstrap procedure and after the system run. The default log operation is the standard out and you can see directly all the informations.

The common output on screen are this:

```
.:: Bootstrap :.:
Loading base properties
Base properties loaded
Loading vehicles configuration
security/certificates/1/c
security/certificates/2/c
Vehicles configuration loaded
.:: Bootstrap end :.:
```

6.1.10 Add certificates and private keys for Baseline Pseudonyms

When the system doing the bootstrap in Baseline Pseudonyms mode research certificate and private keys for doing digital signatures.

Security properties are in *security* folder, if you add one vehicle you must attach certificates and private keys for work with Baseline Pseudonyms implementation. For do that you have to create a set of folders and positioning certificates and keys in a right place. Under *security* you have one folder named *certificates* and under that folder you have another one folder for each vehicle named with the *vehicle id* (section 6.1.2 at page 18), create new folder named with unique integer identification. Under this folder you have to create another two folders named *c* for certificates and *p* for private keys; after this steps you have to copy your private keys in folder *p* and certificates in folder *c*. Certificates and private keys must have the same name for link, for example: *sec1.crt* → *sec1.key.tificates* and keys in a right place. Under *security* you have one folder named *certificates* and under that folder you have another one folder for each vehicle named with the *vehicle id* (section 6.1.2 at page 18), create new folder named with

unique integer identification. Under this folder you have to create another two folders named *c* for certificates and *p* for private keys; after this steps you have to copy your private keys in folder *p* and certificates in folder *c*. Certificates and private keys must have the same name for link, for example: *sec1.crt* \rightarrow *sec1.key*.

References

- [1] P. Papadimitratos, G. Calandriello, J.-P. Hubaux, A. Lioy, “Efficient and Robust Pseudonymous Authentication in VANET”, MOVE 2008: IEEE INFOCOM-2008 workshop on Mobile Networking for Vehicular Environments, Phoenix (AZ, USA), April 13-18, 2008, pp. 19-27
- [2] P. Papadimitratos, M. Raya, J.-P. Hubaux, “Securing Vehicular Communications”, MOVE 2006: IEEE Wireless Communications Inter vehicular Communications, October, 2006, pp. 8-15
- [3] F. Kargl, E. Schoch, B. Wiedersheim, T. Leinmuller, “Secure and Efficient Beaconing for Vehicular Networks”, VANET 2008: IEEE INFOCOM-2008 workshop on Mobile Networking for Vehicular Environments, San Francisco (CA, USA), September 15, 2008, pp. 1-2
- [4] Bouncy Castle Security Provider, “<http://www.bouncycastle.org>”, Legion of the Bouncy Castle, a provider for the Java Cryptography Extension and the Java Cryptography Architecture
- [5] OpenSSL, “<http://www.openssl.org>”, OpenSSL, C implementation for security.