A secure road traffic congestion detection and notification concept based on V2I communications

Vinh-Thong Ta[a,*], Amit Dvir[b]

[a] *School of Physical Sciences and Computing, University of Central Lancashire, Preston, PR1 2HE, UK*
[b] *Ariel Cyber Innovation Center, Department of Computer Science, Ariel University, Ariel, Israel*

Abstract

Applying vehicular (V2X) communications in detecting traffic congestion is a promising approach, as smart and self-driving vehicles are equipped with sensors that can be used to detect an incident anywhere in real-time. Unfortunately, without appropriate security measures and careful design the communication can be vulnerable to malicious attacks, causing even more damage on the roads. Addressing these problems, we propose a high-level system architecture and a security protocol specifically designed for congestion detection based on vehicle-to-infrastructure (V2I) type communication. The security properties of our proposed approach are formally verified using the ProVerif tool, and its efficiency compared to the traditional traffic light systems is demonstrated through simulations with the Veins framework. Results show that our system is secure against a large set of attacks, and can have lower total travelling time compared to three traditional traffic light approaches based on induction loop, lane area detector/camera (installed near the junctions), and static lights.

Keywords: Security, V2I communications, traffic congestion detection.
2010 MSC: 68M12

1. Introduction

Many studies on road traffic control problems address the efficiency aspect, for instance, proposing algorithms and methods for traffic lights to reduce congestion and CO2 emission (see, for example, the studies [1–12]). Besides the theoretical research studies, adaptive traffic control systems are already widely deployed on the roads, using technologies of induction loop or cameras to mitigate congestion. For instance, SCATS [8] and SCOOT [9] measure traffic intensity by installing vehicles detector loops at the junctions, while InSync [10] relies on camcorders. Although adaptive traffic control systems mitigate traffic

*Corresponding author

congestion, they are not capable of detecting any traffic congestion or incident taking place outside the range of the cameras or detector loops.

There are initiatives and studies, for example [13, 14], on using surveillance camera systems in smart cities to detect road congestion. Unfortunately, monitoring the entire roads, especially motorways, would require a huge number of detector loops or cameras throughout the road, which can be expensive. Further, surveillance camera systems for this purpose can be less effective in bad weather with poor visibility. Nowadays, self-driving vehicles are equipped with LIDAR and multi camera systems (e.g. [15, 16]) that could be used to detect a road incident. However, V2X communication would be necessary to report this.

The security problems of traditional traffic light systems were examined in the past, for instance, in the studies [17–20], where the authors found that the communications in some early versions of traffic light systems happened without any encryption, as well as the default passwords remaining unchanged. More recently, researchers also found the possibility for generating fake traffic congestion in social navigation systems such as Waze [21].

Adapting vehicular (V2X) communications in traffic management is a promising approach to overcoming the limitations of traditional methods. However, this would increase the possibility for attacks as the attackers can intercept and modify the messages as well as compromise the vehicles and road-side units with malware. Several projects investigated the security and privacy problems of vehicular (and intra-vehicular) communications such as SeVeCom[1] and EVITA[2]. The main goal of these projects is to secure the wireless vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I) as well as the intra-vehicular communications, preventing attackers from causing accidents by eavesdropping and manipulating the communications or stealing the secret keys stored inside the on-board modules. The anonymity of vehicles on the road has been also investigated, and solutions proposed such as using pseudonym or group signature [22]. These projects provided a generic concept of secure vehicular communications, which we adapted and modified for the road traffic control context and problems.

Addressing the above-mentioned problems, to fill the gap, we propose a secure road congestion detection and notification concept (namely, a system architecture and a secure communication protocol) based on the vehicle to infrastructure (V2I) type communication. The security of the proposed communication protocol against both the external and insider attackers is formally proved with the ProVerif verification tool [23], and the efficiency of the proposed concept is demonstrated through simulations with the Veins framework [24]. In our approach a vehicle would be able to identify the lane IDs based on the road information sent by the road side units.

We note that this paper is an improved version of our previous conference

[1] Secure vehicle communication project, 2006-2008, https://trimis.ec.europa.eu/project/secure-vehicle-communication

[2] E-safety vehicle intrusion protected applications project, 2008-2011, http://www.evita-project.org/

paper [25] with improved algorithms, security analysis and simulation. Section 2 highlights the related works. Section 3 discusses the adversary model, followed by our proposed smart traffic control concept based on V2I communications in Section 4. We provide a formal security analysis of our concept in Section 5, and a simulation to demonstrate its efficiency against different traditional approaches in Section 6. Section 8 summarizes our observations and findings, and we conclude the paper in Section 9.

2. Related Works

Zhou et. al. [2], proposed a decentralised traffic light control approach using wireless sensor network. Their system has a three-layer architecture that consists of a traffic flow policy model and a high-level coordination between the so-called intersection control agents. The authors in [3] proposed a real time vehicle detection and tracking system using surveillance videos. They proposed a video processing algorithm for vehicle detection and analysing traffic density on a particular lane. Unlike our V2I approach, video-based methods would require a large amount of cameras to detect congestion on long roads/motorways.

Leontiadis et. al. [6] evaluated the effectiveness of a decentralized traffic-based navigation system in which instead of distributing traffic information centrally, vehicles report their current information such as location, speed, and travel time to their neighborhood. Their solution assumes vehicle-to-vehicle (V2V) communications and that each vehicle is able to act as a traffic sensor, to measure the surrounding traffic density (vehicle quantity), and then exchange the information in an ad-hoc manner. Their results show that a decentralized approach can greatly reduce congestion in a realistic scenario.

The above studies focused mainly on the effectiveness of traffic control systems, but not the security problems. From the security aspect, Jeske [17] presented a man-in-the-middle attack on GPS coordinate transmissions in order to trick navigation services into inferring non-existent congestion while in an article, Zetter [18] outlined how the wireless vehicle detection systems (an earlier version of Sensys Networks VDS240) deployed in 40 U.S. cities were vulnerable and could be exploited to generate false traffic congestion. The author, C. Cerrudo, found that those Sensys networks devices communicated with each other without any encryption or security mechanism, making it possible for an attacker to modify the messages to mislead the control system. Similarly, in his article, Grad [19] outlined that by adjusting the signal times at the key intersections in Los Angeles, USA, insider attacks on command centers could also create massive delays.

A study from the University of Michigan in 2014 [20] points out that a large portion of traffic lights in the United States communicated with each other wireless over the 900Mhz and 5.8Ghz ISM band *without any encryption*. In order to connect to the 5.8Ghz traffic signals, attackers only needed the SSID of the corresponding device. The second security hole the researchers found was that the passwords were left default, which could be found on the traffic light manufacturer's website. In addition to these, to gain access to the 900Mhz

networks attackers also needed a 16-bit slave ID. The authors also found a vulnerability in the open debug port in the VxWorks OS which allowed the attacker to read-modify-write any memory register. In another attack (called remote keypad), the attacker could freeze the current intersection state, modify the signal timing, or change the state of any light. Fortunately, the hardware based malfunction management unit (MMU) would still be able to detect any illegal states (conflicting green or yellow lights), and took over with the 4-way red flashing. In this case, a technician needed to manually reset the traffic signal to recover from the illegal state, the attacker could turn every intersection on the network into a 4-way stop, causing traffic congestion.

Laszka et al. [26] introduced a novel approach for evaluating vulnerabilities of transportation networks by identifying traffic signals that have the greatest impact on congestion. Therefore, those signals may be the targets of attacks. The authors showed that this is a NP-hard problem, so a polynomial-time heuristic algorithm for computing approximately optimal attacks was presented. Yin et al. [27] presented a threat of data spoofing over the U.S. Department of Transportation traffic control system. The data spoofing can be done from one single attack vehicle only, with the attack goal of creating traffic congestion. Some further vulnerabilities are discussed in [28].

Machine learning has been used recently to estimate or predict traffic congestion. In [29] the authors proposed an efficient traffic congestion estimation method based on model-based estimation through virtual sensors using the Kalman filter. The procedure is based on the data collected from sensors such as induction loops and then applying the KNN classification to estimate congestion. The authors tested their method by deploying induction loops in the SR60 Highway of California to collect traffic data, and showed that their algorithm can estimate congestion with good accuracy. The authors in [30] discussed several methods between 2014-2019 on the application of machine learning to predict crowd flow (human mobility prediction), traffic flow and public transit flow (e.g. bike sharing and metro flows) on spatio-temporal datasets. They examined five groups of prediction approaches, including statistics-based, traditional machine learning-based and deep learning-based approaches.

Social networking concept has already been broadly applied to smart traffic control. Waze[3] is a community-based navigation application, where users can share traffic status on certain roads with each other, such as indicating traffic congestion, police presence, construction, etc. Researchers managed to exploit a vulnerability in an earlier version of Waze [21], and misled the application causing fake traffic congestion. As a result, the application suggested an alternative longer route for the nearby drivers. Finally, camera-based products for smart city are also widely deployed and used in self-driving vehicles that could potentially be applied for traffic management (e.g. MobilEye[4]).

Unfortunately, only very few studies proposed specific protocols and concepts

[3]Waze, https://www.waze.com/en-GB/about
[4]MobilEye, https://www.mobileye.com

for smart traffic control systems based on vehicular communications. The main difference between our work and the previous studies is that we propose an approach that combines those different areas, such as V2X communications, traffic management and security. The main focus of our work is the security design against different attacker models; however, we also show the effectiveness of our method compared to the traditional traffic management solutions based on induction loops, area lane detector or cameras.

3. Our Adversary Model

In our case, the main goal of the attackers is to mislead the traffic control system with incorrect or incomplete traffic data leading to inefficient traffic schedule. They can prevent normal light functionality, for instance, by setting all lights to red or green at the same time (e.g. [20]). Although recent traffic light systems are protected by a hardware-based malfunction management unit (MMU) that prevents inconsistent/unsafe light states avoiding accidents, this will not prevent the attackers from setting inefficient light programs (e.g. longer red time than required).

As for the attacker's ability, in our approach we consider the insider and external attackers, as well as a set of possible attack methods, as follows:

- ***Insider attackers*** are parts of the roads, such as compromised vehicles or road side units. They can be instructed to send incorrect, fake traffic messages to the other participants on the roads. The incorrect messages are encrypted or signed with the valid keys of the compromised nodes.

- ***External attackers*** are not parts of the road (i.e. they are neither compromised vehicles nor road side units), and hence, they do not possess any valid keys by default. In our case, the main goal of the external attackers is exploiting the design flaws in the communication and the security protocols. We assume that an external attacker is able to eavesdrop the entire wireless communication and modify, forge and create fake messages with the keys it possesses. Further, it can replay and relay any captured packets to any participant. This adversary model is strong as in vehicular ad-hoc networks attackers can normally only overhear certain parts of the roads, and hence their ability of intercepting and sending messages is more limited [31]. Since we will prove that our proposed system is secure against the stronger attacker model, it is also secure against a weaker model.

Furthermore, we also consider the attacks based on their nature, as follows:

- ***Local attacks*** are attacks that require the attacker to have either physical or local access to the RSUs, controllers or vehicles. Local attack is a means to recruit compromised vehicles or RSUs.

- ***Remote attacks*** are attacks where the victims (e.g. vehicles, RSUs) are exploited and taken control over from anywhere.

- ***Non-collusion attack***: In this case, we are talking about a single attacker who does not cooperate with the other attackers to reach a certain goal.

- ***Collusion attack***: In this case, the attackers cooperate with each other to reach the same goal. They can share resources and synchronize their activities. This includes the so-called Sybil attack [32] when a node uses several fake identities to mislead the traffic control systems.

The motivation of the attackers in our case includes causing false traffic congestion and unnecessary traffic diversion, like in [18]. Namely, the attackers' intent is to modify or fake the traffic data and then get it accepted by the controllers who are responsible for setting the traffic light programs. We consider the following cases where congestion can be caused if the attacker

1. achieves that the duration of the red light is prolonged on a busy road/lane segment, leading to a potential financial impact (e.g. [33]);
2. achieves that certain roads (e.g. the road on which the attacker is travelling) have longer green light duration than normal, so that the attacker can benefit from this (e.g. [18]);
3. achieves that the traffic flow on certain roads is diverted to other roads because of fake congestion reports (e.g. [21]).

4. Proposed Secure Traffic Congestion Detection and Management Methodology

We present our proposed concept for detecting and managing traffic congestion based on the so-called vehicular public key infrastructure (VPKI). As outlined in [31], asymmetric key cryptography seems to be the most suitable approach in securing vehicular communications as it has the potential to provide confidentiality, authenticity, privacy and accountability (e.g. for police investigation) at the same time. This concept has been applied in several previous studies and projects such as the SeVeCom project [22]. However, we modify it for the traffic congestion detection context that, to the best of our knowledge, has not been done before. Our system architecture, protocol and algorithms are different from the one in the project.

4.1. System architecture

The basic concept of our method is depicted in Fig. 1. In our approach, we define *road segments* as a part of the road from one junction to another junction with traffic lights, and denote them by R_i, $i \in \{1, \ldots, n\}$.

Each lane segment Ln_i is defined by a triple, $Ln_i = (Lid_i^j,\ D_i^j,\ RSU_i^n)$, where Lid_i^j is the unique ID of a lane segment, D_i^j is the direction of the lane that can be, for instance, north (N), south (S), east (E), west (W), or from the perspective of a vehicle it can be forward (FW) and reverse (REV), as well as a road side unit (RSU) that is installed at the beginning of a lane segment.

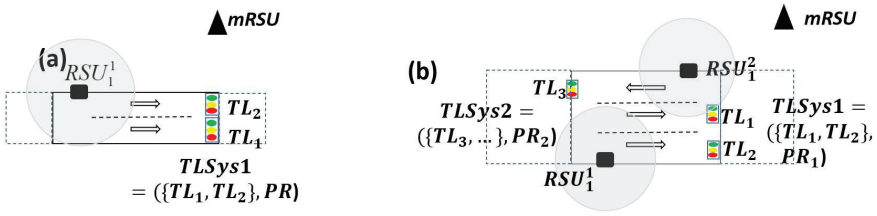

Figure 1: *An architecture overview of the road and lane segments with the RSUs, mRSUs. Point (a) shows a road defined by* R_1 = ({(Lid_1^j, D_1^j, RSU_1^1), j=1,2}, mRSU), *and in (b),* R_1 = ({(Lid_1^j, D_1^j, RSU_1^n), j=1,2,3, n=1,2}, mRSU).

Each road segment R_k is defined by a set of lane segments, and a so-called main road-side unit (mRSU), namely, $R_k = (Rid_k, \{Ln_j\}, \text{mRSU})$. One mRSU is installed in a region of several road segments and junction(s).

At a junction, we define the so-called traffic light system, denoted by TLSys, which includes a set of traffic lights ($\{\text{TL}_1, \ldots, \text{TL}_M\}$) for some natural M, and the program (PR) that controls those lights. Formally, a region (denoted by Reg_n) can be defined as $Reg_n = (\{R_k\}, \{TLSys_j\}, mRSU)$, where $\{R_k\}$ and $\{TLSys_j\}$ are a set of road segments and a set of traffic light systems, respectively, in this region, while mRSU is the main RSU installed in this region and is responsible for changing the traffic light programs in $\{TLSys_j\}$.

Fig. 1/(a) highlights a simple road segment containing two lane segments, and a traffic light system, TLSys1, with two lights TL_1, TL_2, and a default light program, PR. A RSU (RSU_1^1) is installed at the start of the road segment that covers both lanes. Finally, mRSU is the main road side unit assigned to this region. Fig. 1/(b) depicts a road segment and three lane segments with two RSUs. The same approach is applied for the bigger road segments.

The RSUs should be installed close to the traffic lights/junctions, at the start of the lane segments in order to inform the incoming vehicles about the lane segments they are about to enter. We assume that each RSU has a short-range radio antenna that enables transmission of data within the radius equal to the width of some lanes. With the short-range antennas we would like to minimise the number of messages that a vehicle has to deal with.

4.1.1. Architectural requirements

Our approach includes a few architectural requirements regarding the vehicles, the RSUs and mRSUs, such as:

- **For vehicles**: Each vehicle has a built-in satellite navigation system (SatNav/GNSS), e.g. GPS, Galileo, GLONASS, or BeiDou.

  Vehicles are equipped with a V2X capable internal module (IM) that stores the addresses of the mRSUs in a record (denoted by *MRSUrecord*). The format of *MRSUrecord* is defined in Table 1.

Table 1: The record MRSUrecord for some $N \geq 1$

$MRSUrecord = \{1.\ mRSU_1 : Re_1, 2.\ mRSU_2 : Re_2, \ldots, \text{N.}\ mRSU_N : Re_N\ \}$

Each row of *MRSUrecord* contains the ID of a mRSU and the region (Re_j) in which this mRSU is installed, which is specified by SatNav coordinates.

To reduce the size of *MRSUrecord*, only the information about the mRSUs inside a country/state is stored, where a vehicle was registered. This can be updated with new mRSUs when a vehicle arrived in a new country/state. Alternatively, this record can be built into the SatNav system used inside the vehicles.

In addition, an IM is responsible for handling non-sensitive cryptographic operations with the public keys (e.g., signature verification or asymmetric encryption). An IM is connected to a sensor (or to the odo/mileo-meter) and records the distance that a vehicle traveled within a road segment.

A vehicle is equipped with a good quality camera (or a set of cameras), with which it can take pictures about the road segment it arrived at, and the IM module is capable of extracting and differentiating the lanes from the pictures taken, as well as identifying the lane in which the vehicle is currently traveling (e.g. see [34]).

In addition to the IM module, each vehicle is equipped with a hardware security module (HSM) or a trusted platform module (TPM) that is responsible for performing security critical operations such as digital signature generation as well as the storage of the private keys. We will discuss further the choice between TPM and HSM in Section 8.

- **For RSUs**: Each RSU is equipped with a trusted platform module (TPM) for digital signature generation and storing the signature keys. In addition, they are installed with an internal module that is responsible for putting together and broadcasting messages for the vehicles.

- **For mRSUs**: Each mRSU is equipped with an internal module (IM) that can receive and verify the traffic status messages from the vehicles. An IM stores a record, *Roadrecord*, about the road segments in the region where the mRSU is installed, and the lane information in each road segment, such as lane IDs, number of lanes, lane and pavement width.

In order to broadcast road information for the vehicles to help them with identifying the lane they are traveling on, at the installation time each RSU stores a record called *RoadInfo* about the road segment where it has been installed. Specifically, the format of *RoadInfo* is defined in Table 2.

Table 2: The format of *RoadInfo* for some $NumLanes \geq 1$

$RoadInfo = (SatNvArea,\ NumLanes,\ LanesInfo,\ DistInterval)$
$LanesInfo = (Lanes,\ Pavements)$
$Lanes = \{(LaneID_i,\ Direction_i,\ Width_i) \mid i \in \{1,\ldots,\ NumLanes\}$
$Pavements = (Left\_\ Width,\ Right\_\ Width)$

RoadInfo contains the SatNav area of the road segment (*SatNvArea*), the number of lanes in the road (*NumLanes*), a set of lanes information (*LanesInfo*), and a distance interval *DistInterval*. *LanesInfo* includes information about the lanes and pavements in the road segment. Namely, *Lanes* contains the lane ID, the lane direction, and the width of the lanes. From the perspective of the arriving vehicles, $i = 1$ is the leftmost lane while $i = NumLanes$ is the rightmost, or vice versa. *Pavements* contains the width of the left and right pavements (zero in case of no pavement). Finally, *DistInterval* is the interval defined by two distances from the start of the road segment, during which vehicles are required to notify if they have not got stuck until that point. This interval depends on the length of the road segment. The road information can be edited at the installation time of the RSU, as normally, information about the road does not change often. In case of any change due to construction, the corresponding *RoadInfo* is updated accordingly.

4.2. Cryptographic Keys

As mentioned, we assume a public key infrastructure for vehicular systems, where communications between the vehicles and RSUs, mRSUs are digitally signed, and the certificate authority/authorities (CA) is responsible for issuing and revoking certificates for the public keys. Following the concept of SeVeCom [22, 31, 35], the vehicles have a pair of long-term public and private keys. These are generated by and stored inside the HSM or TMP in the vehicle at the time of registration or manufacture. This way nobody can read or obtain the private key. The long-term public key part is certified by the CA (there can be several CAs for different regions and countries). Each vehicle is also given a *unique long-term identity* and the public key of the CA at the registration time.

To prevent the vehicles from being tracked in the long term, *short-term key pairs* are used in the communications instead of the long-term key pairs. This kind of solution is referred to as “pseudonymous authentication" [22, 35]. The CA is given the authority to link long-term identities with several corresponding short-term credentials to provide accountability. The short-term public keys do not reveal the vehicles long-term identity, and each vehicle will switch to another (not previously used) short-term key pair at a junction. Junctions prevent the observer/attacker from tracking a certain vehicle due to the crowd, and to the fact that messages signed under different short-term private keys cannot be

Table 3: The variables and their values/calculations in the algorithms

Variable	**The calculation/value of the variable**
1. RSUBroadcast =	(RSUid, MRSUid, *RoadInfo*, curr_time)
2. signed_RSUBroadcast =	**Sign**(RSUBroadcast, SK_{RSU})
3. curr_loc =	(currLaneID, dist, loc)
4. notstuck_msg =	("Not Stuck", lanes_info, curr_time, curr_loc, curr_state, mRSUid)
5. signed_notstuck_msg =	**Sign**(notstuck_msg, SK^S_{VE})
6. lanes_info =	signed_RSUBroadcast[*RoadInfo*][*LanesInfo*]
7. taken_picset =	**enhancePics**(**takePics**(), lanes_info)
8. currLaneID =	**idenCurrLaneID**(taken_picset, lanes_info)
9. dist =	**updateDistance**()
10. state =	**updateState**(loc)
11. loc_of_stuck =	(currLaneID, dist, loc)
12. stuck_msg =	("Got Stuck", lanes_info, loc_of_stuck, curr_time, state, Vtype, mRSUid)
13. signed_stuck_msg =	**Sign**(stuck_msg, SK^S_{VE})
14. goagain_msg =	("Go Again", loc_of_stuck, curr_time)
15. signed_goagain_msg =	**Sign**(goagain_msg, SK^S_{VE})
16. $newSK^S_{VE}$ =	**SK**()
17. $newPK^S_{VE}$ =	**PK**($newSK^S_{VE}$)
18. loc =	**updateSatNavLoc**()
19. curr_time =	**updateTime**()
20. cert_request_msg =	(ID_{VE}, *CAid*, $newPK^S_{VE}$, loc, curr_time)
21. signed_request =	**Sign**(cert_request_msg, SK^L_{VE})
22. encrypted_signed_request =	**AEnc**(signed_request, PK^{Enc}_{CA})
23. mRSUid =	**updateMRSU**(loc, MRSUrecord)
24. no_br_alarm =	("No Br", loc, curr_time, mRSUid)
25. signed_no_br =	**Sign**(no_br_alarm, SK^S_{VE})
26. cert_new_key =	(*CAid*, ID_{VE}, $newCert^S_{VE}$, curr_time)
27. signed_cert =	**Sign**(cert_new_key, SK^L_{CA}))
28. encrypted_signed_cert =	**AEnc**(signed_cert, PK^{Enc}_{VE})
29. notstuck_time =	signed_notstuck_msg[curr_time]
30. notstuck_curr_loc =	signed_notstuck_msg[curr_loc]
31. notstuck_loc =	curr_loc[loc]
32. notstuck_state =	signed_notstuck_msg[curr_state]
33. XPK^S_{VE} =	Cert(PK^S_{VE})[PK^S_{VE}]
34. got_roadID =	**getRoadID**(loc, Roadrecord)
35. got_lanes_info =	**getLanes**(got_roadID)
36. stuck_time =	signed_stuck_msg[curr_time]
37. stuck_curr_loc =	signed_stuck_msg[curr_loc]
38. stuck_loc =	curr_loc[loc]
39. stuck_state =	signed_stuck_msg[curr_state]
40. goagain_loc =	signed_goagain_msg[loc_of_stuck]
41. SetTLSys =	{$\text{TLSys}_1, \ldots, \text{TLSys}_N$}
42. SetCV =	{$\text{CV}(L_1), \ldots, \text{CV}(L_M)$}

linked. New short-term key pairs are generated by the HSM/TPM inside the vehicles. The private keys are kept inside the HSM/TPM or stored encrypted inside the vehicle, while the public key parts are certified by the CA.

Table 4: Notations Used in The Algorithms (Constant values/records Part 1)

Notations	Explanation
RSUid/MRSUid	Long-term ID of an RSU/main RSU
CA/CAid	The certificate authority/its long-term ID
Vtype	A vehicle type (e.g., "emergency", "bus")
RoadInfo	(*SatNavLoc*, *NumLanes*, *LaneInfo*, *DistInterval*)
LaneInfo	(*Lanes*, *Pavements*)
SatNvLoc	The SatNav area/coord. of a road segment
NumLanes	The number of lanes in a road segment
Lanes	A set of properties about the lanes in a road segment
Pavements	Contains the width of the pavements in a road segment
DistInterval	An interval of values of distance from the road start
Roadrecord	A record of all the road segments in a region
Initime	An initialisation time value
Resendlimit	A limit value for resending a message
Current_time	The current time
Current_loc	The current SatNav location of a vehicle
$\triangle T_{RSU}$	A time period after which a RSU broadcasts
$\triangle T_{stuck}^{thres}$	A threshold after which a "Got Stuck" alarm is sent
$\triangle T_{goagain}^{thres}$	A threshold after which a "Go Again" msg is sent
$\triangle T_{resend}$	A Threshold after which a msg is re-sent
$\triangle T_{fresh}$	A time threshold to check message freshness
$\triangle T_{notstuck}$	A threshold after which a "Not Stuck" msg is sent
$\triangle T_{signal}$	A threshold until there is no satellite signal
$\triangle T_{light}^{update}$	A threshold after which the congestion states are checked

4.3. The Secure Traffic Congestion Detection Procedure

As discussed in the previous section, the RSUs are installed at the start of the lane segments in order to broadcast the record *RoadInfo* to the approaching vehicles so that they know which lane of a road segment they arrived at. The SatNav/GNSS system built in a vehicle is used to identify roads, junctions and a larger region with several road segments, but cannot be used to identify a single lane accurately [36].

For better readability, we keep the description of the algorithms as simple and short as possible, and refer the reader to Table 3 for the values or calculation of the variables used in the algorithms. In the algorithms, we use generic, implementation specific parameters and variables rather than proposing specific values. We examined some specific values in simulation (Section 6).

Algorithm 1 defines the operation of a RSU. After each $\triangle T_{RSU}$ time period, a RSU broadcasts a digitally signed message, *signed_RSUBroadcast*, along with the certificate of the public key of RSU (Cert(PK_{RSU}^{L})). As shown in row 1 of Table 3, *signed_RSUBroadcast* contains the ID of the RSU (RSUid), the ID of the main RSU for that region (mRSUid), the record *RoadInfo* and the timestamp (curr_time) for message freshness.

Algorithm 2, VE, outlines the behaviour of a vehicle. When VE receives a broadcast message from a RSU (line 3), it carries out several verification steps

Table 5: Notations Used in The Algorithms (Constant values/record Part 2)

Notations	Explanation
VE/ID_{VE}	A vehicle VE/its long-term ID
PK^L/PK^S	Long/short-term public (signature) key
SK^L/SK^S	Long/short-term private (signature) key
PK^{Enc}/SK^{Enc}	Long-term encryption public/private key
Cert(PK)	The digital certificate of the public key PK
CurrLaneID	The current lane ID of a vehicle
MRSUrecord	The record of the mRSUs and their regions
Msg[e1]	The element e1 in the message Msg
V_{ano}^{thres}	A threshold on the amount of the "not stuck" vehicles, above which "anomaly" is reported.
D_{ano}	Anomaly check will be done within D_{ano} distance before and after the location of the stuck
T_{ano}	Anomaly check will be done within T_{ano} time before and after the time of the stuck.
TL	An ID of a traffic light
TLSys	An ID of the traffic light system at a junction (TLSys = {TL_1, ..., TL_n} for some n)
SetTLSys	A set of traffic light systems (SetSTLSys = {$TLSys_1$, ..., $TLSys_n$} for some n)
$PR_{Default}$	The default program of a traffic light system.
PR_i	A program of a traffic light system for some i.
$\triangle T_{TL}$	A threshold after which a PR_i is updated
SetPR	The set of all pre-defined programs of all the traffic light systems in a region.
ST_j	A state of a traffic light system for some j. e.g. for TLSys = {TL_1, TL_2, TL_3, TL_4} the state Green-Red-Green-Red refers to the color of the lights TL_1,...,TL_4, respectively.
M	A pre-defined value to specify that the green light time in one light state is M times longer than in another light state.
SetST	The set of all pre-defined states of all the traffic light systems in a region.

Algorithm 1 RSU

```
1: procedure RSU
2:     Init values for RSU
3:     prev_time = Inittime
4:     while state == "functioning" do
5:         curr_time = updateTime()
6:         if (curr_time - prev_time = △T_RSU) then
7:             broadcast (signed_RSUBroadcast, Cert(PK^L_RSU))
8:         end if
9:         state = updateRSUState()
10:        prev_time = curr_time
11:    end while
12: end procedure
```

Table 6: Notations Used in The Algorithms (Variables)

Notations	Explanation
dist	The distance a vehicle travels since entering a new road segment
speed	Captures the current speed of VE (see getSpeed())
lanes_info	Captures the received lane properties
taken_picset	Captures a set of pictures taken by a vehicle
enh_picset	Captures a set of enhanced pictures of a vehicle
pic_set	Captures a set of pictures from a RSU
currLaneID	Captures a current lane ID of a vehicle
roadID	Captures an ID of a road segment
got_roadID	An ID of a road segment fetched by a mRSU from its road record
got_lanes_info	The lanes info fetched by a mRSU from its record
prev_time	A variable that captures the time
curr_time	A variable that captures the time
stuck_msg_sent	A Boolean variable, whether stuck alarm is sent
loc	A SatNav location/coordinate variable
loc_of_stuck	A variable for the location where a vehicle stuck
state	The state of a vehicle (onroad, parked, leave)
record(PK)	The traffic record stored at a mRSU about the vehicle whose public key is PK.
recordset	A set of Record(PK) maintained by a mRSU about the vehicles in its region with short-term public key PK of a vehicle within a road segment

(line 4). First, it checks the validity of the public key of the RSU in the attached certificate (Cert(PK^L_{RSU})), and then, the validity of the signature. Afterwards, VE checks if the time stamp in the message is fresh (i.e., the difference between the current time and the received timestamp is within a threshold $\triangle T_{fresh}$). In case of success, and the satellite signal is unavailable or weak (line 5), VE continues its operation as defined in procedure VEONROAD.

Otherwise, if the satellite signal is good in the area (line 7, Alg. 2), then VE checks if the received MRSUid (*signed_RSUBroadcast[MRSUid]*) is the same as the mRSU ID the vehicle stores in its MRSUrecord (line 8)[5]. In line 9, VE compares its current SatNav location (loc, row 18 of Table 3) with the SatNav area in the received broadcast, *signed_RSUBroadcast[RoadInfo][SatNvArea]*. In case they are consistent (VE is inside the correct road segment), VE continues its operation as defined in procedure VEONROAD (line 10, Alg. 2).

In procedure VEONROAD, first, VE resets the distance value *dist* and sets its state to “onroad" (lines 18-20). Then, until the vehicle VE leaves the road segment (i.e. state == ”leave"), in case its speed is less than a given threshold, S, for a certain period of time, $\triangle T^{thres}_{stuck}$, it sends a “Got Stuck" message to the

[5]Although a vehicle can fetch the mRSU ID from its MRSUrecord, it requires the availability of a satellite signal. To prepare for the case when there is no/weak signal in an area, the mRSU ID is included in a RSU broadcast. Besides, with this redundancy, a vehicle can also detect compromised RSUs that intentionally send wrong/fake mRSU IDs.

Table 7: Notations Used Inside The Algorithms (Functions/Activities)

Notations	Explanation
Sign(Msg, SK)	The signature of Msg with the key SK
AEnc(Msg, PK)	The encryption of Msg with the key PK
broadcast Msg	Broadcasts a message Msg
SK( )	Returns a new secret key, newSK
PK(newSK)	Returns a new public key for newSK
receive Msg	Receive a message Msg
delete Msg	Delete a message Msg
verif(Msg)	The verification of the message Msg returns **true** if successful
updateTime()	Returns the current time
updateSatNavLoc()	Returns the current satnav location
updateState(loc)	Returns the current state of a vehicle
getSpeed()	Returns the current speed of a vehicle
numNotStuckVeh (R, L, T, D, T')	Returns the number of NOT stuck vehicles within a distance d from the location L, and within the time T' from the time T, based on the set of records R.
enhancePics (pic_set, lanes_info)	Returns a set of enhanced pictures of the input set of pictures based on a set of lanes properties
drawLanes (pic_set, lanes_info)	Draw virtual lanes lines in the pictures in pic_set based on a set lanes_info
idenCurrLaneID (taken_picset, lanes_info)	Returns the current lane ID based on a set of taken pictures and lane info
takePics()	Returns a set of taken pictures
getRoadID(loc, Roadrecord)	Returns the ID of a road segment based on the SatNav location, and a record of road segments
getLanes(roadID)	Returns the lanes of a road segment
CV(L)	Returns the congestion value on the lane L (e.g., number of stuck cars)
Func1 | **Func2**	Func1 and Func2 are running in parallel
setLightProg(TLSys, PR)	Set the program of the traffic light system TLSys to PR
CurrProg(TLSys)	Returns the current program (PR) of the traffic light system TLSys

mRSU installed in this region (lines 22-24). When VE manages to go again (i.e. speed $> S$) for at least a pre-defined time period, $\triangle T^{thres}_{goagain}$, a signed "Go Again" message is sent to the same mRSU that was previously sent the corresponding "Got Stuck" message (lines 25-26). The signed "Go Again" message will also be sent (even if $\triangle T^{thres}_{goagain}$ has not elapsed) when VE leaves the road segment where it got stuck (lines 32-33).

If a vehicle VE does not get stuck from the point it received a RSU broadcast when entering a road segment up to a random distance in *DistInterval*, it sends the so-called "Not Stuck" message once to the corresponding mRSU (lines 27-29). This is to make the mRSU capable of detecting anomalies in case the compromised vehicles intentionally send fake "Got Stuck" messages.

When VE leaves a road segment and reaches a junction, it starts updating

Algorithm 2 VE

```
1: procedure VE
2:     Init values for VE
3:     if receive (signed_RSUBroadcast, Cert_RSU) then
4:         if verif(signed_RSUBroadcast, Cert_RSU) == true then
5:             if no or weak satellite signal for △T_signal then
6:                 VEOnRoad
7:             else
8:                 if signed_RSUBroadcast[MRSUid] == mRSUid then
9:                     if loc is in signed_RSUBroadcast[RoadInfo][SatNvArea] then
10:                        VEOnRoad
11:                    end if
12:                end if
13:            end if
14:        end if
15:    end if
16: end procedure
```

```
17: procedure VEOnRoad
18:     dist = 0                                  ▷ reset the travelled distance in this road
19:     stuck_counter = 0                         ▷ reset the number of stuck in this road
20:     state = "onroad"
21:     while updateState(updateSatNavLoc()) != "leave" do
22:         if (speed ≤ S) for △T_stuck^thres AND !stuck_msg_sent then
23:             SendStuckAlarmMsg
24:             stuck_counter = stuck_counter + 1   ▷ number of times VE got stuck
25:         else if (speed > S) for △T_goagain^thres AND stuck_msg_sent then
26:             SendGoAgainNotificationMsg
27:         else if stuck_counter==0 then
28:             at a random dist. in signed_RSUBroadcast[RoadInfo][DistInterval] do
29:             SendNotStuckMsg
30:         end if
31:     end while                    ▷ If state == "leave" send "Go Again" message
32:     if stuck_msg_sent then
33:         SendGoAgainNotificationMsg
34:         UpdateShortTermKeys
35:     else                                      ▷ ... and update the keys
36:         UpdateShortTermKeys
37:     end if
38: end procedure
```

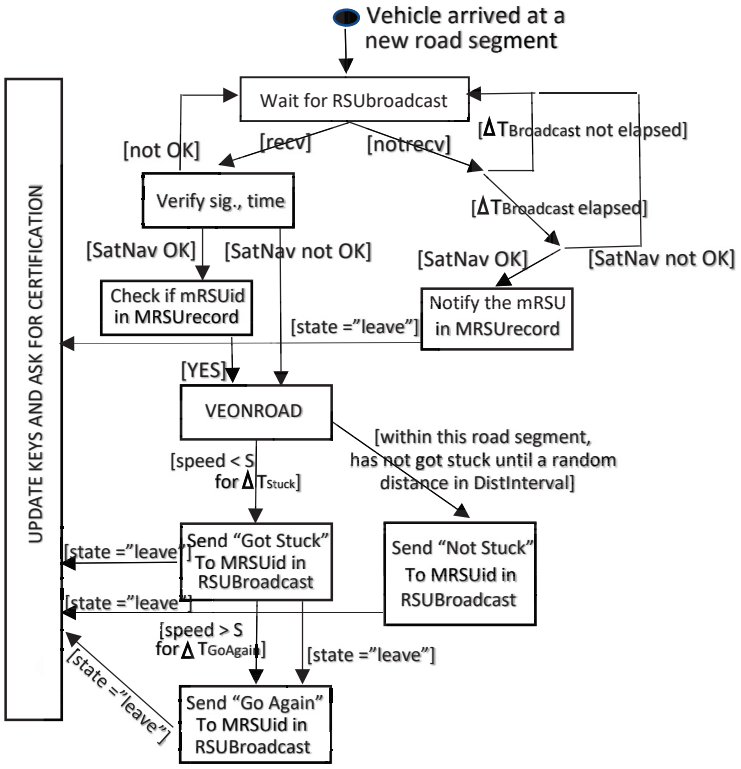

Figure 2: *The operation of a vehicle (VE) when arrived at a new road segment.*

the new short-term signature key pairs (lines 34 and 36, procedure VEOnroad). The rationale behind this is to achieve a higher degree of anonymity for VE, as this area is where we expect a large number of vehicles change their short-term keys at the same time, making it more difficult to (re)-identify a vehicle after that[6]. Although this concept would allow vehicles to be traceable within a single road segment, the long-term tracking is more difficult [37].

Algorithm 3 defines how VE sends a message. In line 2, the "Got Stuck" message, is signed using the actual short-term private key of VE (*signed_ stuck_ msg*), and as shown in rows 12-13 of Table 3, contains

1. the "Got Stuck" tag;
2. the information about the lanes, *lanes_ info*, which VE received from the RSU (see row 6 of Table 3);
3. the location where VE got stuck (*loc_ got_ stuck*, row 11 of Table 3), con-

[6]This solution is less effective in case there are only few vehicles on the roads. (Asymmetric) Encryption could be used to achieve confidentiality all time, but it would increase the computation and message overhead.

Algorithm 3 VE Send Messages

```
1: procedure SendStuckAlarmMsg
2:     sendmRSU(signed_stuck_msg, Cert(PK^S_VE))
3:     stuck_msg_sent = true
4: end procedure

5: procedure SendGoAgainNotificationMsg
6:     sendmRSU(signed_goagain_msg, Cert(PK^S_VE))
7:     stuck_msg_sent = false
8: end procedure

9: procedure SendNotStuckMsg
10:    sendmRSU(signed_notstuck_msg, Cert(PK^S_VE))
11: end procedure
```

sists of the current lane ID, (*currLaneID*, row 8 of Table 3), the distance that VE travelled so far within this road segment (*dist*, row 9 of Table 3), and the current SatNav location of VE (*loc*, row 18 of Table 3);

4. the timestamp (*curr_time*, row 19 of Table 3);
5. the state of VE (*state*, row 10 of Table 3), and the type of VE (which can be e.g., public, emergency, private);
6. finally, the ID of the actual mRSU as addressee (mRSUid, row 23 of Table 3), which VE extracts from its internal *MRSUrecord* based on its current SatNav location (*loc*, row 18 of Table 3).

The 'Go Again" message (line 6, Alg. 3), as shown in rows 14-15 of Table 3, is composed of the 'Go Again" tag, the location where it got stuck (loc_of_stuck, row 11 of Table 3) and the current time (row 19 of Table 3).

The "Not Stuck" message (line 10, Alg. 3) contains the tag "Not Stuck", the current time, current location (comprised of the current lane ID, the travelled distance, and the current SatNav location), the current state, and the ID of the addressee mRSU (rows 4-5 of Table 3).

Algorithm 4 captures when VE updates its short-term signature key pairs at a junction. In line 3, a new short-term secret key is generated based on the function SK, then the corresponding public key is derived from the new private key using the function PK (line 4). In line 5, an encrypted and signed request, *encrypted_signed_request*, along with the certificate of the long-term public (signature) key of VE is sent to the CA. As shown in row 20 of Table 3, the message *cert_request_msg* contains:

1. the long-term ID of VE and the ID of the CA,
2. the newly generated short-term public key ($newPK^S_{VE}$), for which VE requests the certificate, and
3. the current SatNav location of VE (loc), and the current time (curr_time).

The new short-term private key ($newSK^S_{VE}$) is kept in secret inside the HSM/TPM module of VE. As row 21 of Table 3 shows, this message is signed

Algorithm 4 VE Updates Short Term Keys

1: **procedure** UpdateShortTermKeys
2: already_recvd_newRSUbroadcast = **false**
3: $newSK^S_{VE}$= **SK**()
4: $newPK^S_{VE}$= **PK**($newSK^S_{VE}$)
5: **sendCA**(encrypted_signed_request, Cert(PK^L_{VE}))
6: **while !receive** (encrypted_signed_cert) **do**
7: **if** loc is inside the area of a new road segment **then**
8: **if receive** (signed_RSUBroadcast, Cert_{RSU}) for the first time **then**
9: **if verif**(signed_RSUBroadcast, Cert_{RSU}) == **true then**
10: already_recvd_newRSUbroadcast = **true**
11: **end if**
12: **end if**
13: **end if**
14: **end while** ▷ *once received (encrypted_signed_cert) from CA*
15: **if verif**(encrypted_signed_cert) == **true then**
16: **delete** SK^S_{VE}, PK^S_{VE}, Cert(PK^S_{VE}) ▷ *delete old key pair*
17: ▷ *still not received RSU broadcast for the new road*
18: **if** already_recvd_newRSUbroadcast == **false then**
19: ▷ *invoke VEAFTERKEY with the new key pair*
20: **VEafterKeys**($newSK^S_{VE}, newPK^S_{VE}, newCert^S_{VE}$)
21: **else**
22: **if** (**updateSatNavLoc**() == "nosignal") for $\triangle T_{signal}$ **then**
23: **VEOnRoad**
24: **else**
25: **if** signed_RSUBroadcast[MRSUid] == mRSUid **then**
26: **if** loc is in signed_RSUBroadcast[*RoadInfo*][*SatNvArea*] **then**
27: **VEOnRoad**
28: **end if**
29: **end if**
30: **end if**
31: **end if**
32: **end if**
33: **end procedure**

34: **procedure** VEafterKeys(SK^S_{VE}, PK^S_{VE}, Cert(PK^S_{VE}))
35: **Init values for VEAFTERKEYS**
36: **while !receive** (signed_RSUBroadcast, Cert_{RSU}) **do**
37: **if** loc is inside the area of a new road segment **then**
38: **if !receive** (signed_RSUBroadcast, Cert_{RSU}) for $\triangle T^{wait}_{broadcast}$ **then**
39: **sendmRSU**(signed_no_br, Cert(PK^S_{VE}))
40: **end if**
41: **end if**
42: **end while**
43: [Here we reuse the lines 3-15. of Algorithm 2 for brevity.]
44: **end procedure**

using the long-term signature private key of VE (SK^L_{VE}), which is then encrypted using the (long-term) encryption public key of the CA (PK^{Enc}_{CA}) in row 22. The encryption is necessary to keep the long-term ID (ID_{VE}) confidential to avoid the vehicle being tracked for a long time. For security purpose, different algorithms are used for signature and encryption.

Lines 6-14 of Algorithm 4 capture the case when VE, while waiting for the certificate of its new public key, enters a new road segment. If VE receives a RSU broadcast in this road segment (line 8), after a successful verification (line 9), it sets the flag "already received a new broadcast", and saves this broadcast (line 10). Later, when VE receives the key certification from the CA, after a successful verification steps (line 15), VE deletes the old short-term keys to free space and enhance its anonymity (line 16). In case VE has not received any RSU broadcast for the new road segment yet (line 18), it continues its operation with the new short-term keys and certificate as defined in lines 34-44. Otherwise, depending on the availability of the satellite signal in that area, it verifies the received mRSU ID against its MRSUrecord (lines 25-26), and continues as procedure VEONROAD (line 23 and 27).

Procedure VEafterKeys defines an operation of VE after the key certification from the CA. If VE has not received any RSU broadcast from the new road for a $\triangle T^{wait}_{broadcast}$ time period, then VE sends to the corresponding mRSU a so-called "no broadcast" message, *signed_ no_ br*, indicating a potential problem with the RSU. Upon receiving this, the mRSU notifies the authority so the broken/compromised RSU can be fixed. The message *signed_ no_ br*, as shown in rows 24-25 of Table 3, contains the tag "No Br", the current SatNav location and time, and the mRSUid fetched from the MRSUrecord (row 23 of Table 3).

Algorithm 5 CA certifies the short-term public keys

```
1:  procedure CA
2:      Init values for CA
3:      if receive (encrypted_signed_request, Cert^L_VE) then
4:          if verif(encrypted_signed_request, Cert^L_VE) == true then
5:              SendVE(encrypted_signed_cert)
6:                                          ▷ storing 3 elements from the received message
7:              Store(encrypted_signed_request[ID_VE],
8:                    encrypted_signed_request[newPK^S_VE],
9:                    encrypted_signed_request[loc])
10:         end if
11:     end if
12: end procedure
```

In Algorithm 5, once the CA receives a certificate request message (line 3), in line 4 it decrypts the message with its own decryption private key, then verifies the signature and checks whether the time stamp is fresh. Once successful, the CA checks if *CAid* is its own ID, then, in line 5 it sends back to VE a message that contains the certification for the new short term public key. Again, this message is signed with the private key of the CA, and then encrypted using

the encryption public key of VE. The CA stores the long-term ID of VE, the new short-term public key and the SatNav location of VE for accountability purposes, for example, during an investigation of a criminal incident (lines 7-9).

Algorithm 6 mRSU

```
1: procedure MRSU
2:     Init values and variables for mRSU
3:     mRSUHandleMsg | mRSUUpdateLights
4: end procedure
```

A mRSU, in Algorithm 6, performs more computation tasks than a RSU, including signature verification, as well as modifying the phases or states of the traffic lights under its control (region). The operation of a mRSU consists of two functions running in parallel, *mRSUHandleMsg* specifies how a mRSU handles a received message, while *mRSUUpdateLights* specifies how a mRSU regularly checks and updates the traffic light programs based on the congestion values.

In Algorithm 7 (*mRSUHandleMsg*), we distinguish among the cases when a mRSU receives a "Not Stuck" message from a vehicle (line 3), a "Got Stuck" message (line 5) or a "Go Again" message (line 7).

In the first case (lines 11-17), mRSU performs a verification including the validity of the signature and certificate, and the freshness of the timestamp (line 12). Next, the mRSU checks if the lanes information it stores about the road (*got_lanes_info*) is consistent with the received lanes information (line 13). In order to do this, as shown in rows 34-35 of Table 3, the mRSU extracts the road ID from its stored Roadrecord based on the received SatNav location (row 34), then reads the lanes information it stores about this road (row 35) to compare with the received one. In case of consistency, a so called "Not Stuck" entry with the timestamp, the location, the state and the public key inside the received message is (temporarily) cached/stored in the storage of the mRSU called *recordset* (line 14, Alg. 7). This verification is for security purposes, to detect if a compromised RSU sent incorrect lane information to the vehicle.

Once a signed "Got Stuck" message (lines 18-27, Alg. 7) has been received from VE, and in case this message has not been stored before (line 19), the verification of the message will take place. In line 20, mRSU checks whether it is the addressee (i.e., mRSUid is correct), and the attached public key certificate and the signature on the message are valid, as well as if the timestamp is fresh. Finally, in line 21, the mRSU double checks if the lane information the vehicle sent is correct. In case the verification is successful, in line 22, an entry with the flag "stuck" is added into *recordset*. Finally, in line 23 the congestion values of the lane segments are updated with the flag "stuck" indicating that the congestion value is updated after receiving a "Got Stuck" message (see Algorithm 8).

When a mRSU receives a "Go Again" message (lines 28-35, Alg. 7), this message should be related to a previous "Got Stuck" message from the same vehicle, indicating that a vehicle manages to go again after it got stuck. This relation is verified in line 29, checking if there is already a corresponding "stuck" entry in the record record(XPK^{S}_{VE}). Thereafter, further verification will be done

Algorithm 7 mRSU Handle Messages

procedure mRSUHandleMsg
  Init variables for mRSUHandleMsg
  if receive (signed_notstuck_msg, Cert(PK^S_{VE})) **then**
    mRSUHandleNotStuckMsg
  else if receive (signed_stuck_msg, Cert(PK^S_{VE})) **then**
    mRSUHandleStuckMsg
  else if receive (signed_goagain_msg, Cert(PK^S_{VE})) **then**
    mRSUHandleGoAgainMsg
  end if
end procedure

---

procedure mRSUHandleNotStuckMsg
  if verif(signed_notstuck_msg, Cert(PK^S_{VE})) == **true then**
    if got_lanes_info == signed_notstuck_msg[lanes_info] **then**
      Cache((“notstuck", notstuck_time, notstuck_loc, notstuck_state, XPK^S_{VE}), recordset)
    end if
  end if
end procedure

---

procedure mRSUHandleStuckMsg
  if (“stuck", stuck_time, stuck_loc, stuck_state, XPK^S_{VE}) NOT in record(XPK^S_{VE}), for any stuck_time, stuck_loc, stuck_state **then**
    if verif(signed_stuck_msg, Cert(PK^S_{VE})) == **true then**
      if got_lanes_info == signed_stuck_msg[lanes_info] **then**
        Cache((“stuck", stuck_time, stuck_loc, stuck_state, XPK^S_{VE}), recordset)
        updateCongestionState("stuck", recordset, stuck_time, stuck_loc, V^{thes}_{ano}, D_{ano}, T_{ano}, SetCV)
      end if
    end if
  end if
end procedure

---

procedure mRSUHandleGoAgainMsg
  if (“stuck", stuck_time, goagain_loc, stuck_state, XPK^S_{VE}) is in record(XPK^S_{VE}), for some stuck_time, goagain_loc, stuck_state **then**
    if verif(signed_goagain_msg, Cert(PK^S_{VE}) == **true then**
      updateCongestionState("goagain", recordset, goagain_loc, stuck_time, V^{thes}_{ano}, D_{ano}, T_{ano}, SetCV)
      delete (“stuck", stuck_time, goagain_loc, stuck_state, XPK^S_{VE}) from record(XPK^S_{VE})
    end if
  end if
end procedure

in line 30 and the congestion state will be updated, in line 31. Eventually, the "stuck" entry of XPK_{VE}^{S} will be deleted as the vehicle managed to go again.

The procedure *updateCongestionState* (Algorithm 8) expects as input the parameter *flag*, the set recordset, the location and the time of the stuck (loc and time) included in the received message, the threshold V_{ano}^{thres}, the pre-defined values D_{ano} and T_{ano}, and the set SetCV (see Tables 4-5 for the meaning of the notations). If a "Got Stuck" message is received (line 2) and no anomaly is detected (line 4), then the congestion value (CV) of the lane where the vehicle got stuck (lane currLaneID) is increased. Otherwise, if a "Go Again" message is received, then the congestion value of the lane is decreased. Lines 3 and 11 double check whether the lane segment with the ID loc[currLaneID] (i.e., currLaneID inside loc) is in the area under the control of the mRSU.

Lines 15-18 (Algorithm 8) define how a mRSU updates the programs of the traffic light systems under its control area. After every pre-defined delay $\triangle T_{lights}^{update}$, a mRSU checks the congestion values of the lane segments in its area and updates the programs of the light systems accordingly (in lines 19-39, procedure UPDATELIGHTS).

Our main goal is to *decrease the total travelling time of all the vehicles*, hence, we compare the total congestion values of the crossing roads at the junctions. In line 20, for each traffic light system *TLSys* under control of the mRSU, given two sets of crossing lane segments (L_1,..., L_n and L_{n+1}, ..., L_k) that follow the opposite light colours, the total congestion value of the two sets of lane segments will be compared with each other (lines 24 and 30), and an appropriate program will be set, which favours one or another sets of lane segments (lines 26 and 32). *PR1* is the program that gives green for the lane segments L_1,..., L_n, while *PR2* for L_{n+1},..., L_k. In lines 27 and 33, after prolonging the green time for (L_{n+1}, ..., L_k) and (L_1,..., L_n), respectively, the mRSU checks if a diversion of the traffic is needed (*CheckIfDiversionNeeded*). In case there is a set of lanes, denoted by *LaneSameDir*, with the same directions (*LaneSameDir* $\subseteq$ {L_{n+1}, ..., L_k} or {L_1,..., L_n}) in which all lanes are blocked (full road blockage), or the total congestion values of those lanes do not decrease for $\triangle T_{Div}$ time period then the traffic arriving in these lanes will be diverted (if possible), and the authority will be automatically notified to resolve this. Otherwise, the default program $PR_{Default}$ will be set at the junction (line 36).

Note that the specific approaches used in the procedures *increaseValue*, *decreaseValue*, and the specific values in Tables 4-5 are not defined in this paper. Instead we leave them in a generic form and carry out the simulation (in Seciton 6) on some example specific values and approaches for the functions.

Algorithm 9 deals with the anomaly detection done by a mRSU. Basically, in line 2, the amount of vehicles sent "Not Stuck" (NS) within a pre-defined distance D_{ano} before and after the location of the stuck (loc) and a pre-defined time period T_{ano} before and after the time of the "got stuck" message is derived from *recordset*. If NS is larger than a pre-defined threshold value V_{ano}^{thres} (line 3), then an anomaly is detected (i.e., true is returned). Intuitively, this means that the more vehicles notify the mRSU that they do not get stuck the more likely that there is an attack carried out by compromised vehicles.

Algorithm 8 mRSU Updates Congestion State and Lights

procedure updateCongestionState(flag, recordset, loc, time, V_{ano}^{thes}, D_{ano}, T_{ano}, SetCV)
  if flag == "stuck" **then**
    if CV(loc[currLaneID]) ∈ SetCV **then**
      if !isAnomaly(recordset, loc, time, V_{ano}^{thres}, D_{ano}, T_{ano}) **then**
        increaseValue(CV(loc[currLaneID]))
      else return -1 ▷ otherwise, exit
      end if
    end if
  else if flag == "goagain" **then**
    if CV(loc[currLaneID]) ∈ SetCV **then**
      decreaseValue(CV(loc[currLaneID]))
    end if
  end if
end procedure

procedure mRSUUpdateLights(SetCV, SetTLSys, SetST, SetPR, $\triangle T_{TL}$, $\triangle T_{light}^{update}$, M)
  after each $\triangle T_{light}^{update}$ time period **do**
    UpdateLights(SetCV, SetTLSys, SetST, SetPR, $\triangle T_{TL}$, $\triangle T_{Div}$, M)
end procedure

procedure UpdateLights(SetCV, SetTLSys, SetST, SetPR, $\triangle T_{TL}$, $\triangle T_{Div}$, M)
  for each TLSys ∈ SetTLSys **do**
    Let L_1,..., L_n and L_{n+1} ..., L_k be all the lane segments directly linked to a junction controlled by TLSys (e.g., L1, L2, L3, currLaneID in Fig. 3).
    Let ST_1 be the light state where L_1,..., L_n are on green, while L_{n+1} ..., L_k are not, and let ST_2 be the light state where L_{n+1} ..., L_k are on green, while L_1,..., L_n are not.
    Let PR_1 be the program where ST_1 lasts M times longer than ST_2 (i.e., favouring L_1,..., L_n), and let PR_2 be the program where ST_2 lasts M times longer than ST_1 (i.e., favouring L_{n+1} ..., L_k).
    if $\sum_k^m \mathrm{CV}(L_j) > \sum_1^n \mathrm{CV}(L_j)$ for a $\triangle T_{TL}$ period **then**
      if CurrProg(TL) != PR2 **then**
        setLightProg(TL, PR_2)
        CheckIfDiversionNeeded({L_{n+1}, ..., L_k}, $\triangle T_{Div}$)
      else goto point 2.
      end if
    else if $\sum_k^m \mathrm{CV}(L_j) < \sum_1^n \mathrm{CV}(L_j)$ for a $\triangle T_{TL}$ period **then**
      if CurrProg(TL) != PR1 **then**
        setLightProg(TL, PR_1)
        CheckIfDiversionNeeded({L_1,..., L_n}, $\triangle T_{Div}$)
      else goto point 2.
      end if
    else setLightProg(TL, $PR_{Default}$)
    end if
  end for
end procedure

Algorithm 9 Anomaly Detection

procedure isAnomaly(recordset, loc, time, V_{ano}^{thres}, D_{ano}, T_{ano})
  NS = **numNotStuckVeh**(recordset, loc, time, D_{ano}, T_{ano})
  if NS $\geq V_{ano}^{thres}$ **then**
    return true
  else
    return false
  end if
end procedure

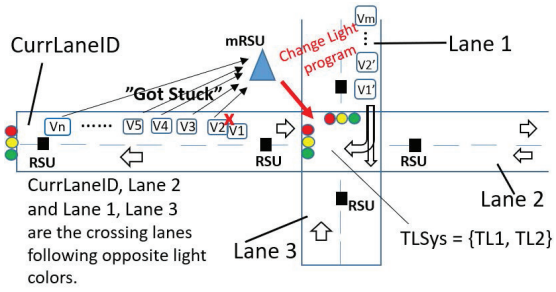

Figure 3: *There is an accident caused by V1 and V2 in the lane NewLaneID, and all the vehicles behind them got stuck. {CurrLaneID, Lane 2} and {Lane 1, Lane 3} are the two sets of crossing lanes following the opposite light colors. The vehicles from Lane 1 can turn to right preventing V3, . . . , Vn in overtaking V1/V2 in the opposite lane.*

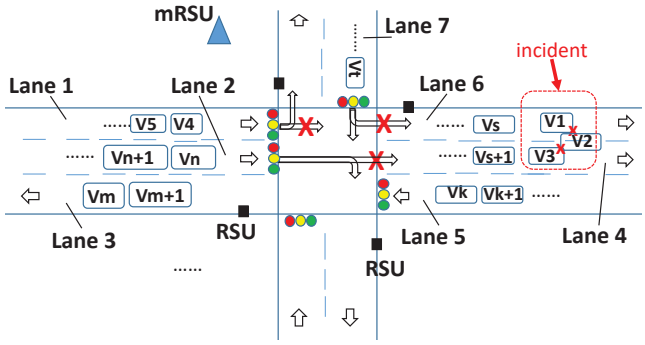

Figure 4: *An accident scenario involving V1, V2, V3 where both the two lanes, Lane 4, Lane 6 of the same direction are blocked for more than $\triangle T_{Div}$ time. The mRSU diverts the traffic from Lane 1, Lane 2, and Lane 7.*

5. Security Analysis

We discuss a security analysis of our proposed protocol against the attacker models and attack methods defined in Section 3. Our analysis is based on the ProVerif tool [23], which has been extensively applied in verifying the security

of different protocols (e.g. [38–40]). It assumes the attacker model in which the attacker can intercept, replay and relay as well as creating fake messages using the data it has and take part in different protocol sessions running in parallel.

5.1. The security against external attackers

For the external attackers who are not part of the system, we aim to analyze the key secrecy, authenticity and integrity properties of the protocol as three of the most relevant security properties, namely:

- **Key secrecy**: Here we formally prove the secrecy of the long-term and short-term private keys of the participants and long-term ID of the vehicles. Specifically,
  1. the attacker cannot obtain the long-term private key (SK_{CA}^{enc}, SK_{VE}^{enc}) for the asymmetric encryption used in the communication between a vehicle and the CA for certifying the new short term keys;
  2. the attacker cannot obtain the short-term private key SK_{VE}^{S} used for signing the traffic messages ("Not Stuck", "Got Stuck" and "Go Again");
  3. the attacker cannot obtain the long-term private keys SK_{VE}^{L}, SK_{RSU}^{L}, SK_{mRSU}^{L} and SK_{CA}^{L};
  4. the attacker cannot obtain the long-term ID (ID_{VE}) of the vehicles.

- **Authenticity**: We prove three different properties for authenticity.
  1. Property Auth1: In the communication between a vehicle VE and the CA to certify the new short-term keys (lines 34, 36 in Alg. 2 and Alg. 5), whenever VE finished the session and believes that it ran the session with the CA, then this is really the case, and vice versa. Hence, an external attacker cannot impersonate either VE or the CA, and cannot generate valid signatures in the name of VE or CA.
  2. Property Auth2: Whenever a vehicle receives and accepts a broadcast message about the road segment (lines 3-4 of Alg. 2), the message was really generated freshly by a RSU.
  3. Property Auth3: Whenever a mRSU accepts a ("Not Stuck", "Got Stuck" and "Go Again") message of a vehicle VE and accepts this as valid (i.e., the verification was successful), this message was really generated by VE, who sent it *freshly*.

- **Integrity**: We prove three different properties for integrity, as follows. Any modification of the messages broadcast by a RSU can be detected (Property Int1), any modification of the messages sent by a vehicle to a mRSU can be detected (Int2), and any modification of the messages exchanged between a vehicle and the CA can be detected (Int3).

To analyse the security properties of our system, we define a ProVerif process for each participant, namely, *procRSU*, *procMRSU*, *procVE*, and *procCA*. The protocol definition in ProVerif is as follows:

process
1. **new** skVE : skey; *(* a private key of VE, for decryption *)*
2. **new** skCA : skey; *(* a private key of the CA, for decryption *)*
3. **new** lskVE : lskey; *(* a long-term signature key of VE *)*
4. **new** lskCA : lskey; *(* a long-term signature key of CA *)*
5. **new** lskRSU : lskey; *(* a long-term signature key of RSU *)*
6. **let** pkVE = pk(skVE) **in out**(c , pkVE);
7. **let** pkCA = pk(skCA) **in out**(c , pkCA);
8. **let** lpkVE = slpk(lskVE) **in out**(c , lpkVE);
9. **let** lpkCA = slpk(lskCA) **in out**(c , lpkCA);
10. **let** lpkRSU = slpk(lskRSU) **in out**(c , lpkRSU);
11. ((!procVE(lpkRSU, lskVE, skVE, pkCA, lpkCA)) |
12. (!procCA(skCA, lpkVE, lskCA, pkVE)) |
13. (!procMRSU(lskRSU, lpkCA)) | (!procRSU(lskRSU)))

The keyword *process* denotes the main process (i.e., the protocol itself) and each line of the code can be seen as a (sub-)process. Lines 1-10 define the initialization of the long-term keys at the beginning, which will be used throughout the protocol. Specifically, line 1 (similarly, lines 2-5) defines the process that creates the data with the name *skVE* and of the type decryption private key (*skey*). Line 6 (similarly, 7-10) defines the process that computes the public key *pkVE* from the secret key *skVE* using the function pk(skey), and sends it out on the channel *c*. Finally, lines 11-13 define the four processes *procVE*, *procCA*, *procMRSU*, *procRSU* running in parallel (denoted by "|"). The syntax element "!" means that each process is allowed to replicate its operation infinitely in order to capture parallel protocol sessions. We define the long-term and the short-term signing and encryption keys, and each process has parameters of the keys that it uses during the operation. The short-term signature keys of the vehicles (*procVE*) are generated during the protocol run (not at the initialization) inside the process *procVE*.

In ProVerif, the participants communicate via channels. We define a channel c, namely, "***free*** *c :* ***channel***.", allowing the attacker to overhear and intercept the communication. We define message types, such as the type for data (*bitstring*), type for long-term signature private and public keys (*lskey*, *lpkey*, respectively), and short-term signature private and public keys (*sskey*, *spkey*, respectively), the time (*time*) and road information (*RoadInfo*).

Then, we define cryptographic functions and primitives, such as the function for generating a public key from the corresponding private key, asymmetric key encryption (***fun*** *aenc(bitstring, pkey) : bitstring.*) and decryption, digital signature generation with a short-term and long-term keys (***fun*** *ssign(bitstring, sskey) : bitstring.*, and ***fun*** *sign(bitstring, lskey) : bitstring.*) and verification.

The security properties are defined as follows in the ProVerif syntax. For the **key secrecy** properties:

1. To verify that the attacker cannot obtain the long-term private keys SK_{CA}^{enc} and SK_{VE}^{enc} for decryption, we specify two queries, namely "***query secret*** *skCA.*" and "***query secret*** *skVE.*".

2. For the short-term signature private key SK_{VE}^{S} of a vehicle: "***query secret*** *sskVE.*".
3. For the long-term signature private keys SK_{VE}^{L}, SK_{RSU}^{L}, SK_{mRSU}^{L}, SK_{CA}^{L} of VE, RSU, mRSU, CA, respectively: "***query secret*** *lskVE.*", "***query secret*** *lskRSU.*", "***query secret*** *lskMRSU.*", and "***query secret*** *lskCA.*".
4. For the long-term ID of VE (ID_{VE}): "**query** attacker(IDve)"[7].

With these queries we can check if the attacker can obtain the keys and the long-term ID of VE. After running the verification, as result we got "*RESULT not attacker(IDve[]) is true*" for IDve, and "*RESULT secret x is true*" for all the six private keys, where $x \in \{$*skCA[]*, *skVE[]*, *sskVE[]*, *lskVE[]*, *lskRSU[]*, *lskCA[]*$\}$. This formally proves that the attacker cannot obtain any of these keys and the long-term ID of VE during the protocol runs.

For the **authenticity** properties, in ProVerif events and the so-called *correspondence assertions* are used (see sections 3.2.2-3.2.3 of the manual [41]).

For property **Auth1**, we define the following events:

1. **event** acceptsVECA(spkey,id).
2. **event** termVECA(spkey,id).
3. **event** acceptsCA(spkey,id)
4. **event** termCA(spkey,id).

The first event captures that a vehicle VE accepted running a protocol session with the CA and with the short-term public key to be certified (of type *spkey*) and with the long-term ID of VE (of type id). The second captures that VE believes that it terminated the protocol with the CA with the key of type *spkey* and the long-term ID of type id. The third and fourth events specify that the CA accepted, finished the protocol with the key of type *spkey* and with the long-term ID of type id, respectively. The correspondence assertions are:

1. **query** x : spkey, y : id;
   inj-event(termVECA(x,y)) ==> **inj-event**(acceptsCA(x,y)).
2. **query** x : spkey, z : id;
   inj-event(termCA(x,z)) ==> **inj-event**(acceptsVECA(x,z)).

The first query instructs ProVerif to check if whenever a vehicle VE believes that it terminated a protocol session with the CA with a short-term signature public key (x) and the long-term ID of VE (y), then previously at some point, the CA accepted running this protocol session with a vehicle using the same short-term public key (x) and the same long-term ID (y). Here, we rely on the so-called injective correspondence (inj-event, see section 3.2.3 of the manual [41]), in order to check the possibility of someone replaying the messages of the CA to VE. Similarly, the second query verifies that whenever the CA finished its session and sent back the certified short-term public key x to a vehicle of

[7]Note that this property at the same time also proves the secrecy of keys SK_{CA}^{enc} and SK_{VE}^{enc} as they are used for decrypting the message containing IDve (lines 6-9 of Alg. 4, and lines 7-10 of Alg. 5).

long-term ID y, there was a vehicle with y that accepted running this session with the CA, and requested the certification for the key x.

For property **Auth2**, we define the following events:

1. **event** receivedVERSU(id, time, roadinfo). 2. sentRSU(id, time, roadinfo).

The first event captures that the vehicle VE received a broadcast containing an ID (of type id) of the RSU who sent it, a timestamp (of type time) and the record RoadInfo (of type roadinfo). The second event captures when a RSU sent the broadcast message that contains its ID (of type id), the current time (of type time) and a RoadInfo record. The correspondence assertions are:

1. **query** x : id, y : time, z : roadinfo;
   event(acceptedVERSU(x,y,z)) ==> **event**(sentRSU(x,y,z)).

This query checks the authentication of a RSU towards a vehicle VE, namely, whenever the vehicle VE received a broadcast message that contains a RSU ID x, a timestamp y and a RoadInfo record z, then this RSU really sent the broadcast message with the same x, y and z.

For property **Auth3**, we define the following events:

1. **event** sentVEmRSU(spkey, id, lanesinfo, time).
2. **event** acceptedMRSU(spkey, id, lanesinfo, time).

The first event captures that a vehicle VE sent a message containing its short-term public key (of type spkey), the ID of the mRSU (of type id), the lane information (of type lanesinfo) and the timestamp (of type time).[8] Then, the query to verify is:

1. **query** x : spkey, y : id, z : laneid, w : time;
   event(acceptedMRSU(x,y,z,w)) ==> **event**(sentVEmRSU(x,y,z,w)).

This query checks if whenever a mRSU accepted the message that contains a short-term public key (x), the ID of the mRSU (y), lane information (z), and a timestamp (w), then a vehicle really sent this freshly with x, y, z and w.

To verify the **integrity** properties, we define an attacker process called *procAttInt* for verifying the effect of message modification. For the property **Int1**, we specify a vehicle process, *procVE*, such that when a vehicle receives a broadcast message from a RSU, and all the verification steps are successful, a piece of data named VERIFICATIONOK will be sent out on the channel c, otherwise, nothing will be sent, namely: **free** VERIFICATIONOK [**private**]. Here, VERIFICATIONOK is declared as a piece of data that is not available to the attackers at the beginning ([**private**]), but they can intercept it later.

Next, we specified the process *procAttInt* such that a modified version of the original broadcast message will be sent via the channel c. We examined different

[8] Note that the message sent by VE contains more elements, but only these four are used in the events as they are sufficient to uniquely identify the context of the communication.

versions of message modifications, e.g., the same broadcast message signed with a different key, or when an element of the message is modified.

let procAttInt = modify1 | modify2
let modify1 = **out** (c, sign((IDrsu, IDmRSU, RoadInfo, curr_time), skAtt))
let modify2 = **out** (c, sign((IDAtt, IDmRSU, RoadInfo, curr_time), lskRSU)).

The case of property **Int2** is similar to **Int1**, except that this time the process mRSU, *procMRSU*, sends out the "flag" VERIFICATIONOK on the channel c when the verification of the message sent by a vehicle (*procVE*) is successful. The process *procAttInt* this time sends different modifications of the original/correct traffic message on the channel c.

Finally, for the property **Int3**, both the processes *procVE* and *procCA* send out VERIFICATIONOK on the channel c when the verification of their received messages are successful, and nothing otherwise. The process *procAttInt* sends different modifications of the two original messages, namely, the certificate request message from *procVE*, and the certificate reply from *procCA*.

In order to show that our protocol is capable of detecting the message modifications, we specified the following query in all three cases.

query attacker(VERIFICATIONOK).

This query checks if the attacker will be able to intercept the flag VERIFICATIONOK. As a result, in case of the original messages, ProVerif returned "*RESULT not attacker(VERIFICATIONOK[]) is false*", which means that the attacker was able to intercept the flag during the protocol, while for the modified versions of the original messages ProVerif returned "*RESULT not attacker(VERIFICATIONOK[]) is true*", as the verification of the messages was unsuccessful, hence, the flag was not sent out. We got the same result in cases when the other elements of the message were modified. Therefore, our proposed protocol is capable of distinguishing between a correct and a modified message.

5.2. The security against insider attackers

We assume that the mRSUs are well protected and therefore they are less vulnerable to attacks. Since in our approach, mRSUs are equipped with powerful CPU and resources, it can be assumed that intrusion detection systems, firewalls and anti-malware software are installed in them, and that they are physically well protected. Hence, in our analysis, compromised mRSUs are not considered.

5.2.1. Compromised vehicles

In the following, let us consider the settings in lines 21-23 of Algorithm 8. In order to generate a fake traffic congestion in a lane segment, say L_1, an attacker has to affect the function *increaseValue*(CV(L_1)), namely, increasing the congestion value of L_1 that ultimately leads to increasing the duration for the green light incorrectly. To do this, the attacker needs to compromise a certain number of vehicles (e.g., by infecting them with malware) and instruct them to send "Got Stuck" messages at a certain point of L_1, even if they do not

get stuck. Each compromised vehicle is instructed to generate and send a "Got Stuck" message in line 2 of Algorithm 3, and set their distance value (*dist*) to adjust them into the same point of the lane segment. Since the certificate and the signature will be valid, these messages will be accepted by the corresponding mRSU. However, because of the anomaly detection defined in Algorithm 9, the attacker can only be successful if the number of benign vehicles sending "Not Stuck" messages (in that area and time) is less than the threshold V_{ano}^{thres}. Otherwise, the congestion value will not be increased. Hence, by setting the threshold to a small value we can limit the chance of the attacker.

Continuing the previous argument and scenario, let us assume that the number of benign vehicles is less than V_{ano}^{thres}, so the attacker successfully increases the congestion value of L_1. The attacker still cannot be sure that the mRSU will set the traffic light program to give longer green time for L_1, as this would also depend on the congestion values of the other lane segments. The attacker can achieve its goal if $\sum_k^m \text{CV}(L_j) < \sum_1^n \text{CV}(L_j)$ (line 30, Alg. 8), and this can be the case when either the total congestion value of the crossing lanes of L_1 is very low or the attacker manages to increase the congestion value of L_1 greatly (e.g. when there are only compromised vehicles on L_1 and there is no congestion on the crossing lanes). For this, the attackers need to control a huge amount of vehicles over a long time, which is costly and difficult.

5.2.2. Compromised RSUs

A compromised RSU can broadcast incorrect information about a road segment for the arriving vehicles. It can also remain silent not broadcasting anything, so the vehicles would not know which lane they arrived at, and hence the corresponding mRSU will not get any "Got Stuck" message from the vehicles[9].

1. In case a compromised RSU refuses to broadcast any message, the vehicles report to the corresponding mRSU the fact that it is not receiving any information (line 39 of Alg. 4), so an engineer or system administrator can check and fix this RSU. However, this would require the availability of the satellite signal.
2. A compromised RSU can send incorrect lane information (*LanesInfo* in RoadInfo) with the result that the vehicles would include incorrect lane IDs into their messages. In order to mitigate the impact of this attack, a vehicle sends a message containing these incorrect lanes to the mRSU for verification purposes (line 2 and 10 of Alg. 3), which the mRSU verifies in lines 12-13 of Alg. 7, respectively. In case of inconsistency, the mRSU alerts the appropriate authority who will then fix the RSU. To check this in ProVerif, we declare the data FakeLanesInfo with the type lanesinfo, and define the process *procComprRSU1* as follows.

[9]Vehicles could still send the "Got Stuck" messages but without the accurate information about the lane and location of the stuck vehicles, as we assumed that the SatNav system in the vehicles cannot identify accurately the lanes.

1. **free** FakeLanesInfo : lanesinfo.
2. **let** procComprRSU1(lskRSU : lskey) =
3. **let** RoadInfo = (SatNvArea, NumLanes, FakeLanesInfo, DistInterval) **in**
4. **out** (c, sign((IDrsu, IDmRSU, RoadInfo, curr_time), lskRSU)).

The record RoadInfo will be sent out on the channel c with FakeLanesInfo (line 4). Then, we define the event *acceptsMRSUFakeLanes(lanesinfo)*, which captures that the *mRSU* accepts the message containing a data of type lanesinfo. Finally, the ProVerif query to check if the event when *mRSU* accepts *FakeLanesInfo* would occur during the protocol run:

query event(acceptsMRSUFakeLanes(FakeLanesInfo)).

"*RESULT not event(acceptsMRSUFakeLanes(FakeLanesInfo[])) is true*" was got as a result, which basically means that the mRSU never accepts the fake lane information by a compromised RSU. The reason is because the mRSU will carry out the verification on the correctness of the lane information (lines 12-13 and lines 20-21 Alg. 7).

3. Finally, a compromised RSU can broadcast a message with an incorrect mRSU ID, so the vehicles will send their messages to this incorrect mRSU. This attack can be prevented by the vehicles when they double check the received mRSUid against their MRSUrecord (see line 10 in Algorithm 2). In ProVerif, for the verification of this attack, we declare a fake mRSU ID called *FakeIDmRSU*, and define a process, *procComprRSU2*, which sends out a message containing this fake ID.

1. **free** FakeIDmRSU:id.
2. **let** procComprRSU2(lskRSU : lskey) =
3. **let** RoadInfo = (SatNvArea, NumLanes, LanesInfo) **in**
4. **out** (c, sign((IDrsu, FakeIDmRSU, RoadInfo, curr_time), lskRSU)).

The query for this case is "***query event****(acceptsVEFakeMRSUID(id))*", which captures that VE accepts the fake mRSU ID of type id. As a result, we got that this event only happens besides the attacker process *procComprRSU2* when there is no SatNav signal in the area of the vehicle.

6. Simulation

We demonstrate the efficiency of our proposed approach compared to the traditional systems which rely on induction loops, cameras or purely static lights.

6.1. Simulation Environment

We use the Veins simulation framework [24], which is an open source framework for vehicular networks and V2X simulations. Veins is built upon the SUMO simulator and the OMNET++ discrete event simulator. For the simulation, Veins-5.0, OMNET++ 5.4.1 and SUMO 1.2.0 were used. We run the simulation on a selected area of the city of Preston, UK, which is exported from OpenStreetMap (see Fig. 5)[10]. As depicted in Fig. 5, we set two routes for the cars (only cars were considered) to travel at the same time.

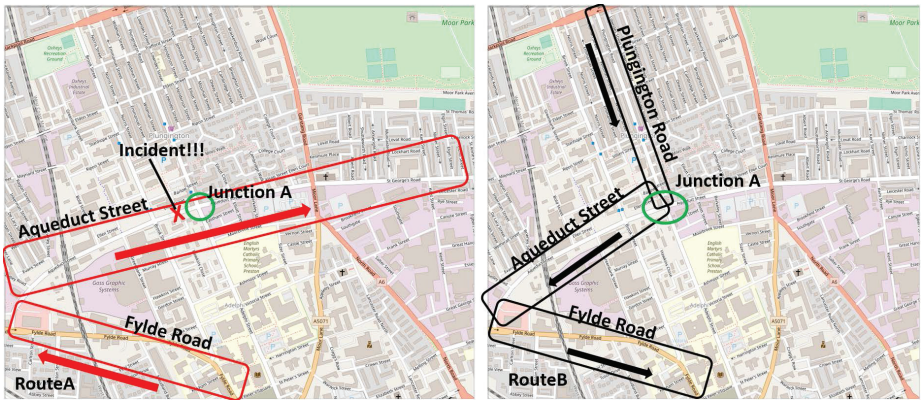

Figure 5: *The simulation area, part of the city of Preston, UK, PR1 2TY. The two routes cross each other at junction A. RouteA starts from Fylde Road through Aqueduct street to the end, while on the right, RouteB starts from Plungington Road, through Aqueduct street and Fylde Road in the reverse direction compared to RouteA.*

Aqueduct street has only two lanes in reverse directions, and in real life, a high number of cars or an incident on that road easily causes traffic congestion. In our simulation, we set an incident on *RouteA* (the car at the beginning of the flow breaks down) right before the junction A (illustrated by the red cross in Fig. 5). Since at junction A, the cars from *RouteB* will turn into the road that has the incident, we expect a high delay for the cars on *RouteA* because they have to wait until the flow from *RouteB* passes or until the traffic light at the junction A is green for *RouteA* in order to overtake the car broken down on the opposite lane. This situation is depicted in Fig. 6.

We compare four different approaches based on the total travel time of all the cars on *RouteA* and *RouteB* from the start to the end of the simulation: The first approach is our proposed method based on V2I communication. The second approach is based only on an adaptive (actuated) traffic light concept with the car detectors installed beneath the roads. In this case, the traffic lights are able to prolong the green time favouring the road with a high number of

[10]We choose Preston because it is the home of the first author's institution.

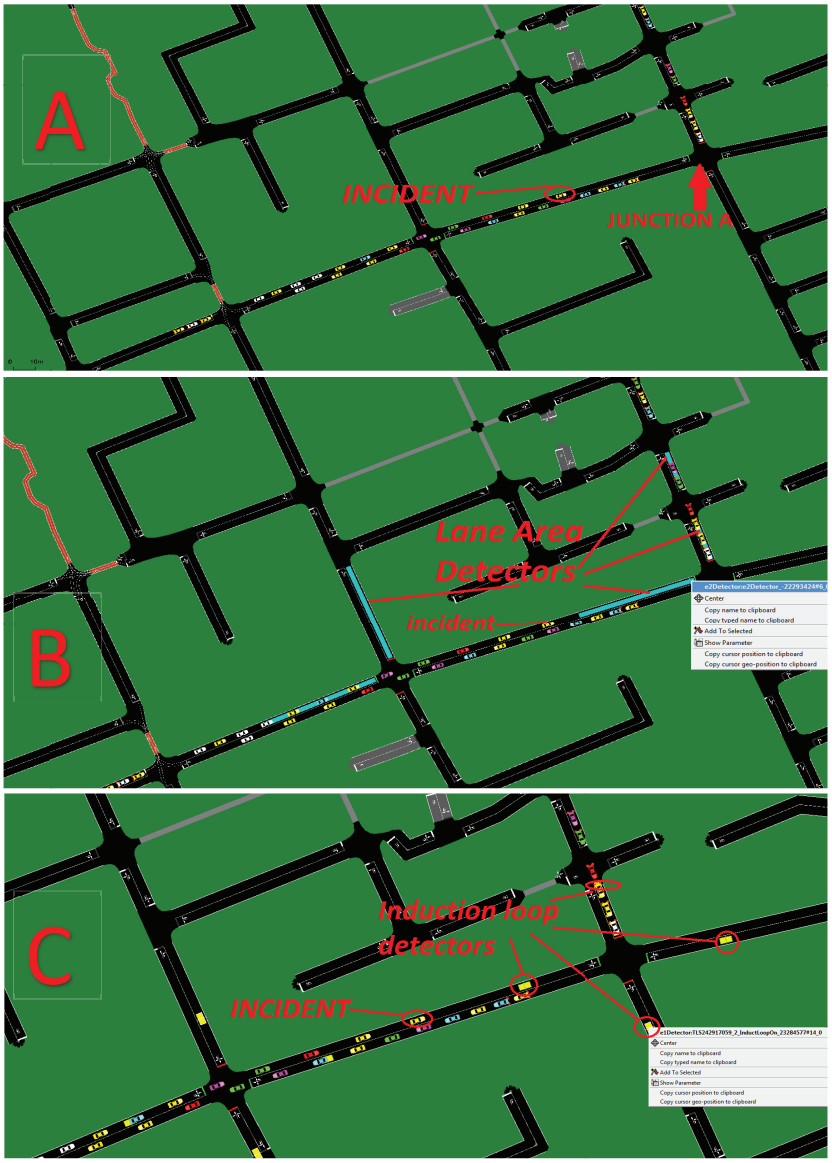

Figure 6: *The simulation in Veins (SUMO) shows the car at the beginning of the flow on RouteA is broken down. Note that the cars travel on the left-side of the roads following the UK rules (a view from SUMO gui. A: Static, B:Lane area detector, C: Actuated).*

cars following each other closely. The third is based on lane area detectors that follows the concept of camcorders installed at the junctions to record and count the vehicles within a certain distance. The fourth approach is when only traffic

lights with static program are used. Note that V2I communication is not applied in the last three cases.

The common simulation configurations we set in all four cases are:

- The car, map and route parameters and the car numbers are the same for all four cases. We used the default car following model implemented in SUMO, which is a modified version of the Krauss model by Stefan Krauss [42]. The simulation time is set to 2500 sec, long enough for all the vehicles to be able to finish their journey.

- The first vehicle on *RouteA* is set to be broken down (speed is zero) at the middle of the road segment before junction A. This happens at time 165s during the simulation of the V2I case, and 185s-190s during the simulation of the other three cases.

- *RouteA* and *RouteB* include five traffic light systems in total, each at a junction. Cars are allowed to overtake in the opposite lane. This makes it possible for the cars stuck behind the incident to overtake the car broken down on their opposite lane when there is a chance.

In the (purely) static lights case, the phases of the lights were set to 42s (for the red-green light state) and 3s (for red-yellow), which is the default setting of the map imported from OpenStreetMap.

In the actuated lights case, an induction detector loop is installed close to the junctions (Fig. 6/c) measuring the vehicle flow and extend the green time accordingly. We set the minimum duration of the red-green phase to 42s that can be prolonged to the maximum duration of 63s (i.e. 63s = 3/2 * 42s).

In case of the approach using lane area detector or camcorders installed at the junctions (Fig. 6/b) we set the length of the lane area detector such that the car broke down outside the sensing/detector area, but its sensing range is longer than a half of the road segments (enough to capture at least seven cars). In this case, camcorders cannot detect the cars that get stuck behind the incident.

In Veins, the V2X communication simulation is based on the 802.11p standard. We choose the following simulation configuration:

- The parameters of the 802.11p NIC include *nic.mac1609_4.txPower = 20mW* for the transmission power, *nic.mac1609_4.bitrate = 6Mbps* bitrate level. The transmission power is set to strong enough for a car to be able to communicate with the mRSUs. Since we could not set the signal strength to accurately limit the communication to just one specific area, the mRSUs are programmed to ignore the messages sent from other areas.

- All the five traffic light systems on *RouteA* and *RouteB* are defined to be able to receive instructions from the mRSUs about the congestion status and change the light status accordingly. We also define five mRSUs to handle congestion status messages from vehicles in five different segments of *RouteA* and *RouteB*.

- A vehicle will send a "Got Stuck" message to a corresponding mRSU if its speed drops to less than 1 for 21s (half of the length of the default light period, 42s). In order to reduce the amount of message lost due to channel congestion, each vehicle sets a random delay (with the uniform distribution) between 0 and 2s before sending the message. Further, each vehicle also re-sends the same message (after the original one was already sent) once after a random delay between 0 and 2s. This delay period is small enough to avoid any change in the stuck status of a vehicle, and at the same time, can mitigate message lost during the simulation.

- For each junction with traffic lights in *RouteA*/*RouteB* (there are five of them in the simulation), the mRSUs check the congestion status on the crossing roads in its area after each second. If the number of cars that got stuck (i.e., $speed < 1$) on one road (say, Road1) outweighs the number of the stuck cars on the crossing road (say, Road2), then the light system will be programmed to switch to green for Road1, red for Road2, and vice versa. Otherwise, if there is an equal number of stuck cars on the two crossing roads then the light system switches back to a default static program (i.e., 42s for green-red, and 3s for yellow-red).

- A re-sent "Got Stuck"/"Go Again" message by a car will be ignored by an mRSU unless it has not received the original message due to packet lost.

- When the speed of a stuck car becomes greater than 1 for at least 20s (or a car moves into the area of a different mRSU), it will send a "Go Again" status message to the mRSU (that controls the area where it got stuck) with a random (uniform) delay between 0 and 2. This is then re-sent once with a random delay between 0 and 2s (after the original was sent).

Finally, the configuration for the lane area detection method is as follows:

- Since there are only cars on *RouteA* and *RouteB*, only the lane area detectors on those roads are defined. The lane area detectors are set to be longer than a half of the road segments, and be able to detect at least seven cars in its area (this is equivalent to a camcorder that is able to capture around half of a road segment).

- Static traffic lights are used by default with the same setting as in the other cases, namely, 42s for green-red, and 3s for yellow-red. Five mRSUs are defined to collect the information from the lane area detectors and change the traffic lights.

- Let *RoadA* and *RoadB* be two crossing roads at a junction: the traffic lights are set to green for *RoadA* and red for *RoadB* if the number of the vehicles detected by the lane area detector on *RoadA* is (at least) seven, and greater than the number of cars detected on *RoadB* for at least 20s, and vice versa. Otherwise, the traffic light is set to the default static phases.

Table 8: TOTAL TRAVEL TIME COMPARISON 1

Cars RouteA	Cars RouteB	Incident Duration	V2I (Our Approach)	Lane Area Detector	Actuated Lights	Static Lights
12	14	280s	7690	8571	8969	9045
24	14	280s	11709	13759	13922	14666
36	14	280s	15732	19964	20442	21384
48	14	280s	20522	26103	26376	27725
60	14	280s	25692	31289	31917	33938

Table 9: TOTAL TRAVEL TIME COMPARISON 2

Cars RouteA	Cars RouteB	Incident Duration	V2I (Our Approach)	Lane Area Detector	Actuated Lights	Static Lights
60	14	280s	25692	31289	31917	33938
60	28	280s	35851	38460	40074	40715
60	42	280s	43942	44392	46491	46778
60	56	280s	50458	50744	52603	52778
60	70	280s	56871	57251	59065	59283

Table 10: TOTAL TRAVEL TIME COMPARISON 3

Cars RouteA	Cars RouteB	Incident Duration	V2I (Our Approach)	Lane Area Detector	Actuated Lights	Static Lights
60	70	280s	56871	57251	59065	59283
60	70	380s	57659	60630	62482	64785
60	70	480s	58748	64122	65391	67586
60	70	580s	59090	65970	68769	69434
60	70	680s	59616	67468	69215	71077

6.2. Simulation Results

We run the simulation with the above configurations besides different numbers of cars on *RouteA* and *RouteB*. In order to see how the number of the cars behind the incident affects the total travel time, in Table 8 we increase the number of the cars on *RouteA* from 12 to 60, while the number of cars on *RouteB* is set to 14 in every case. We can see in Table 8 that the proposed V2I communication method has the lowest total travel time among the four methods. The reason is that once the car flow on *RouteB* passes the place of the incident, the mRSUs change all the five traffic light systems (at five junctions in RouteA/B) to green for the car flow on *RouteA*. The lane area detector method out performs the induction loop method and, unsurprisingly, the option with purely static lamps performs the worst. Finally, we can also see that by increasing the number of cars on *RouteA*, the time difference becomes greater.

In Table 9, we fix the number of cars on *RouteA* to 60, and change the cars on *RouteB* from 14 to 70. With this, we want to check how the cars on the crossing road slow down the traffic on *RouteA*, preventing the cars stuck behind the incident from overtaking in the opposite lane. We can still see that the proposed V2I method out performs the other three, but (unlike the first comparison) the more cars on *RouteB*, the smaller the difference. We believe that our V2I method achieves a lower travel time compared to the lane area detection method because the lane area detectors are not be able to detect the incident outside the range, and hence, there is a delay until at least seven cars

manage to overtake the broken down car and get into the range of the detectors due to the heavy traffic in the opposite lane. Finally, the reason the difference is decreasing with the increasing number of cars on *RouteB* is because the cars on *RouteA* and *RouteB* compete with each other by sending the "Got Stuck" messages to the mRSU. By increasing further the number of cars on *RouteB* the V2I remains the best among the four approaches. Eventually, Table 10 outlines the differences in case the incident duration is increased from 280s to 680s.

6.3. Simulation discussion

We have shown that besides the default parameters of the traffic lights programs downloaded from OpenStreetMap, there exists a configuration for our V2I solution that outperforms the other traditional solutions. Obviously, we could modify the light programs of the traditional methods, so that the actuated light can prolong the green time much longer than 63s, or use longer lane area detector to capture more cars. In this case, the parameter for the V2I approach could be adjusted accordingly, for instance, the time period for sending a "Got Stuck" message could be shortened. Nevertheless, since in the V2I approach the mRSU can get a full picture of the congestion in lane segments, a more informed decision can be made than with the traditional systems.

7. Message Overhead and Beacon Congestion

In the following, we refer the reader to different points of Table 3. The size of the "Got Stuck" message of a vehicle in points 12-13 is estimated to be around 189 bytes besides the ECDSA signature scheme (with a 28 bytes key), and a 6-lane road. Specifically, 189 bytes = *lanes_ info* (30 bytes)[11] + *loc_ of_ stuck* (6 bytes) + *curr_ time* (8 bytes) + *state* and *Vtype* (1 byte) + mRSUid (3 bytes) + ECDSA signature and certificate (140 bytes [31]).

The signed "Not Stuck" message (points 4-5) is estimated to be around 190 bytes for a 6-lane road, namely, 190 bytes = *lanes_ info* (30 bytes) + *curr_ time* (8 bytes) + *curr_ loc* (8 bytes) + state (1 byte) + mRSUid (3 bytes) + ECDSA signature and certificate (140 bytes).

Similarly, the estimated size of the signed "Go Again" (points 14-15) message is around 154 bytes (6+8+140). The encrypted certificate request message (point 22) is between 84 and 256 bytes[12], and finally, the signed "No Br" message (points 24-25) is estimated to be 153 bytes.

Eventually, the signed RSU broadcast in point 2 is around 163 bytes (where the plaintext is 23 bytes and the signature is 140 bytes). These values are in line with the literature (e.g. [31]) as they estimated the security messages to be around 200-400 bytes.

[11] In Table 2, *Pavements* is 1 byte, lane ID is 4 bytes, the direction information and lane width is 1 byte all together. Hence, the information for one lane is 5 bytes together.

[12] 84 bytes can be in case of elliptic curve integrated encryption scheme (ECIES), while 128/256 bytes in case of RSA.

Table 11: COMPARING AVG MAC BUSY TIME AND THE MESSAGES RECEIVED BY EACH mRSU (1 accident, 280s accident duration). The percentage shows the amount of messages received by a mRSU out of the total messages addressed to it. This rate includes the re-sent "Got Stuck" and "Go Again" messages. The packet size was set to 200bytes.

MAC busy time AVG	$mRSU_0$ recvd	$mRSU_1$ recvd	$mRSU_2$ recvd	$mRSU_3$ recvd	$mRSU_4$ recvd	Cars RouteA	Cars RouteB
0.193	72.5%	80.6%	85.7%	100%	83.3%	50	50
0.181	72.3%	79.7%	87%	100%	84%	50	42
0.144	74%	78.4%	90.3%	100%	81.4%	50	28
0.097	78.4%	79.8%	89.4%	100%	80.6%	50	14

To calculate the worst-case transfer rate, let us distinguish the cases with congestion and without congestion. In the first case, let us assume that 50 cars got stuck on RoadA and 50 cars on RoadB at juction A at the same time (we recall Fig. 6 and assume that a road segment can accommodate 50 cars). Unlike other studies (e.g. [31]), in our case, a vehicle does not process any message from other vehicles but a mRSU. Hence, we focus on the messages sent to a mRSU by the vehicles. Suppose that a vehicle sends ten "Got Stuck"/"Go Again" messages per second, in the worst-case scenario the corresponding mRSU would receive 1000 messages (10 messages x 100 cars). Based on the message overhead above, 189 bytes, this results in a transfer rate of (1000 x 189byte)/sec, which is around 1.51Mbps. The rate would change to 1.6Mbps in case of 200byte messages, and 0.15Mbps in case each vehicle only sends one message in one second.

In case there is no congestion on RoadA and RoadB at all, each vehicle only sends one "Not Stuck" message of 190 bytes within a road segment, which in the worst-case would require 100(cars) x 190byte, around 0.15Mbps.

These are within 6Mbps, the (default) transfer rate in DSRC [43]. Obviously, the actual rate would be less in practice as depending on when the cars got stuck, messages are sent at different times, and the "Not Stuck" messages are sent at different times as well. The simulations show the average MAC busy time in the first column of Table 11.

Regarding channel congestion, during the simulation we adapted the IEEE 1609.4 multi-channel operation. In our case, however, there are no WSA (Wave Service Announcement) beacons, as there is only one service. Both the RSUs and the vehicles only send WSM (Wave Short Messages) on the control channel (CCH). Based on the scenario and the settings given in Section 6 and packet size of 200bytes, Table 11 depicts the comparison of the average MAC busy time (for all cars in the simulation), and the amount of "Not Stuck", "Got Stuck" and "Go Again" messages received by each mRSU. The percentage in the case of $mRSU_0$ is the lowest because it is in the area of the incident, hence, directly affected by the congestion. $mRSU_1$ and $mRSU_2$ receive the messages from the road segments next to the one with the incident, while in the area of $mRSU_3$ there is no road congestion at all, hence, each car only sends one "Not Stuck" message (with a random delay), resulting in no message lost. Finally, $mRSU_4$ covers the entire area on the right of junction A (Fig. 5).

Note that this percentage includes the re-sent "Got Stuck" and "Go Again"

messages. In the simulation, after one re-transmission with a random delay, eventually all the 'Got Stuck" and "Go Again" messages arrived at the mRSUs.

Moreover, in our system the vehicles only start sending traffic status messages to the mRSUs after it has arrived at the new road segment and already received the RSU broadcast. This limits the chance of collision between messages sent by a vehicle and a RSU. The RSUs in the neighbouring road segments are set to start broadcasting at different times to reduce message collision.

Finally, the authors in [44] showed that the delivery delay for safety messages increases linearly with the speed, both in case of DSRC (from 90ms/20mph to 130ms/50mph) and LTE (2690ms/20mph to 3000ms/50mph). While we use different message contents, the result in this paper can be used as a reference.

8. Discussion

Security, privacy and accountability: Our approach also provides protection against physical attacks, such as intrusion into the on board devices of a vehicle or a RSU/mRSU to extract the private keys. The hardware security modules (HSMs) or the trusted platformed modules (TPM) are designed to physically protect the private keys. Alternatively, we can use the on-demand secure key generation approach in [45]. Denial of Services (DoS) attacks can be launched against our system, when the attackers send a huge amount of messages to the mRSUs. Although these messages are invalid, they consume the computing resource of the mRSUs. Known methods against DoS attacks [46, 47] can be implemented at the mRSUs to mitigate their impact. Finally, a jamming attack is always a potential threat in wireless communication, where the attacker prevents the RSUs, vehicles and mRSUs from receiving any messages. Different anti-jamming techniques [48] could be adopted to mitigate the effect of jamming.

From a privacy perspective, we assume that the vehicles regularly change their short-term public and private key pairs before arriving at a new road segment, and no future key pairs can be linked to the old ones (except for the trusted CA). This makes it difficult for an attacker or a system administrator/technician who eavesdrops the wireless communication to track the vehicles in the long-term. It is, however, still possible for someone to track the vehicles within a single road segment. Note that we adapted the concept of the pseudonyms because it provides a sensible balance between anonymity and accountability. However, our concept can be easily incorporated with different privacy enhancing approaches, e.g. group signature [49]. As for accountability, whenever there is an incident that requires an investigation (e.g. police or insurance company), our approach allows the CA to assist the investigation (under legislation) by revealing the link between the short-term public keys and the long-term ID of a vehicle.

The installation cost: Since the main RSUs require powerful computing and memory resources, their installation can be expensive. However, this can be adjusted by choosing how large a region a mRSU can control. The larger the region, the less mRSUs are required as only one mRSU is required for a region. In the case of the RSUs, since their numbers are much higher than the mRSUs,

it is important to keep their cost reasonable. Each RSU only stores a minimal amount of data, and carries out two types of operations, signature generations and broadcasting. To moderate the cost of a RSU, a TPM is used to handle the cryptographic operations, which are much cheaper than using HSMs[13].

HSMs are powerful cryptoprocessors that perform cryptographic operations very effectively, and provide superior physical protection with tamper-resistant, tamper-proof properties. The private keys are generated inside the HSMs, and never leave it. Whilst very secure, HSMs are expensive and widely used by banks and financial institutions. Example HSMs include the IBM 4768 and 4769 PCIe [14]. TPMs are micro-controllers that stores keys, and digital certificates. The early version 1.2 of TPMs was typically integrated to the motherboard of a PC with cryptographic operations. Recently, TMP 2.0 has been improved to support a more extensive set of cryptographic operations including digital signature [50]. Compared to HSMs which cost thousands of Euros, TMPs are much cheaper (normally, less than 100 Euros).

V2V versus V2I concept: An alternative smart traffic control system could rely (entirely) on vehicle-to-vehicle (V2V) communication as well, where vehicles will send traffic information to each other and a vehicle is able to set the traffic lights. While this concept can be cheaper (a vehicle would take over the tasks of our mRSU), it also poses a higher security risk, because vehicles can be more easily compromised than the mRSUs in our approach. Besides, we get back to the question of an accurate satellite navigation system, as if a concept is reliant only on V2V communication, then each vehicle needs to be able to accurately determine the lanes. Channel congestion is a well-known problem of V2V.

Lane identification: Identifying lanes is relevant for lane level traffic management, for example, traffic diversion could be done on time to avoid congestion or accidents on the motorways. Approaches to identifying lanes include camera based, e.g. [51], precise point positioning (PPP), e.g. [52], and GNSS shadow matching based methods, e.g. [36]. The main challenge faced by camera based methods is their effectiveness in bad weather or at night, while the latter two approaches require the availability of the satellite signals (from several satellites at the same time), as well as depending on the surrounding obstacles such as buildings, mountains, trees etc.

5G technology: In this paper, we mainly focused on V2I communications (e.g., DSRC/802.11p). Our concept could be adapted in a 5G network, where instead of contacting the mRSUs the vehicles can follow the cellular V2X standard to communicate with the base station instead. Since 5G allows a much bigger message size, more information (e.g., pictures, videos) could be shared among the participants. RSUs could broadcast a set of pre-taken pictures about the road segments with the lane information and lane lines in different weather

[13]Note that it would be cheaper if the RSUs broadcast unsigned messages. However, anyone would be able to send a fake *RoadInfo* to the vehicles, and this cannot be detected in the areas where the satellite signal is permanently weak/unavailable.

[14]https://www.ibm.com/security/cryptocards/hsms. Accessed 01/07/2020.

conditions and time to help vehicles identify the lanes better. Our message contents could be adapted in the cellular V2X standard. It is a potential future work to explore this in more detail.

9. Conclusion and Future Work

In this paper, we have provided an automated traffic congestion detection and notification concept based on V2I communication, including a system architecture and communication protocols. Unlike the previous studies and projects in a similar area, our solution is specifically designed for traffic congestion detection, hence it is different from those studies. The security properties of the new system were formally proved using the ProVerif tool. The security verification showed that our method is secure against external attackers, and in case of sufficient satellite signal, it is also secure against the compromised vehicles or RSUs. Finally, we demonstrate the performance of our concept through simulations with the Veins framework. Results have shown that besides the default settings OpenStreetMap, the total travel time of the traditional traffic control solutions is higher than the total travel time in our method (besides a reasonably chosen set of parameters and roads as an example).

10. Acknowledgement

This work was supported by the Ariel Cyber Innovation Center in conjunction with the Israel National Cyber directorate in the Prime Minister's Office.

References

[1] B. De Schutter, B. De Moor, Optimal traffic light control for a single intersection, European Journal of Control 4 (3) (1998) 260–276.

[2] B. Zhou, J. Cao, X. Zeng, H. Wu, Adaptive traffic light control in wireless sensor network-based intelligent transportation system, in: IEEE 72nd Vehicular Technology Conference (VTC 2010-Fall), 2010, pp. 1–5.

[3] F. Mehboob, M. Abbas, R. Jiang, R. A., A. S. Khan, S. Rehman, Trajectory based vehicle counting and anomalous event visualization in smart cities, Cluster Computer 21 (2018) 443–452.

[4] C. Osorio, K. Nanduri, Urban transportation emissions mitigation: Coupling high-resolution vehicular emissions and traffic models for traffic signal optimization, Transportation Research Part B: Methodological 81 (P2) (2015) 520–538.

[5] S. Djahel, M. Salehie, I. Tal, P. Jamshidi, Adaptive traffic management for secure and efficient emergency services in smart cities, in: The IEEE Pervasive Computing and Communication (PerCom) conference (WiP track), San Diego, California, USA, 2013.

[6] I. Leontiadis, G. Marfia, D. Mack, G. Pau, C. Mascolo, M. Gerla, On the effectiveness of an opportunistic traffic management system for vehicular networks, IEEE Transactions on Intelligent Transportation Systems 12 (4) (2011) 1537–1548.

[7] D. Curiac, C. Volosencu, Urban traffic control system architecture based on wireless sensor-actuator networks, in: 2nd International Conference on Manufacturing Engineering, Quality and Production Systems, 2010, pp. 259–263.

[8] Scats (sydney coordinated adaptive traffic system), `http://www.scats.com.au/`, accessed: 23/03/2020.

[9] D. I. Robertson, R. D. Bretherton, Optimizing networks of traffic signals in real time-the scoot method, IEEE Transactions on Vehicular Technology 40 (1) (1991) 11–15.

[10] R. Chandra, C. Gregory, Insync adaptive traffic signal technology: Real-time artificial intelligence delivering real-world results, InSync White paper, accessed: 23/03/2020 (3 2012).

[11] I. Khan, K. Umar, H. Muhammmad, S. Bilal, S. B. Shah, Traffic density based light control system, Journal of American Academic Research 5 (2017) 90–94.

[12] M. J. Deshmukh, C. N. Modi, Designing an adaptive vehicular density-based traffic signal controlling system, in: P. K. Sa, S. Bakshi, I. K. Hatzilygeroudis, M. N. Sahoo (Eds.), Recent Findings in Intelligent Computing Techniques, Springer Singapore, Singapore, 2018, pp. 107–115.

[13] A distributed image-retrieval method in multi-camera system of smart city based on cloud computing, Future Generation Computer Systems 81 (2018) 244 – 251.

[14] J. Barthélemy, N. Verstaevel, H. Forehead, P. Perez, Edge-computing video analytics for real-time traffic monitoring in a smart city, Sensors 19 (9) (2019) (2019) 1 – 23.

[15] L. Caltagirone, M. Bellone, L. Svensson, M. Wahde, Lidar–camera fusion for road detection using fully convolutional neural networks, Robotics and Autonomous Systems 111 (2019) 125 – 131.

[16] C. Häne, L. Heng, G. H. Lee, F. Fraundorfer, P. Furgale, T. Sattler, M. Pollefeys, 3d visual perception for self-driving cars using a multi-camera system: Calibration, mapping, localization, and obstacle detection, Image and Vision Computing 68 (2017) 14 – 27.

[17] T. Jeske, Floating car data from smartphones : What google and waze know about you and how hackers can control traffic, 2013.

[18] K. Zetter, Hackers can mess with traffic lights to jam roads and reroute cars, accessed: 23/03/2020 (2014).
URL https://www.wired.com/2014/04/traffic-lights-hacking/

[19] S. Grad, Engineers who hacked into l.a. traffic signal computer, accessed: 23/03/2020 (2009).
URL LosAngelesTimes

[20] B. Ghena, W. Beyer, A. Hillaker, J. Pevarnek, J. A. Halderman, Green lights forever: Analyzing the security of traffic infrastructure, in: Proceedings of the 8th USENIX Conference on Offensive Technologies, WOOT'14, USENIX Association, Berkeley, CA, USA, 2014, pp. 7–17.

[21] M. B. Sinai, N. Partush, S. Yadid, E. Yahav, Exploiting social navigation, CoRR abs/1410.0151 (2014).

[22] F. Kargl, P. Papadimitratos, L. Buttyan, M. Müter, E. Schoch, B. Wiedersheim, T. Thong, G. Calandriello, A. Held, A. Kung, J. Hubaux, Secure vehicular communication systems: implementation, performance, and research challenges, IEEE Communications Magazine 46 (11) (2008) 110–118.

[23] Proverif: Cryptographic protocol verifier in the formal model, http://prosecco.gforge.inria.fr/personal/bblanche/proverif/, accessed: 23/03/2020.

[24] C. Sommer, R. German, F. Dressler, Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis, IEEE Transactions on Mobile Computing 10 (1) (2011) 3–15. doi:10.1109/TMC.2010.133.

[25] V. Ta, A. Dvir, Y. Arie, Securing road traffic congestion detection by incorporating v2i communications, in: 2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM), 2018, pp. 1–6.

[26] A. Laszka, B. Potteiger, Y. Vorobeychik, S. Amin, X. D. Koutsoukos, Vulnerability of transportation networks to traffic-signal tampering, 2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS) (2016) 1–10.

[27] Q. A. Chen, Y. Yin, Y. Feng, Z. Mao, H. Liu, Exposing congestion attack on emerging connected vehicle based traffic signal control, in: Network and Distributed Systems Security (NDSS), 2018, pp. 1–15.

[28] A. Burg, A. Chattopadhyay, K.-Y. Lam, Wireless communication and security issues for cyber–physical systems and the internet-of-things, Proceedings of the IEEE 106 (2018) 38–60.

[29] F. Harrou, A. Zeroual, Y. Sun, Traffic congestion monitoring using an improved knn strategy, Measurement 156 (2020) 107534.

[30] P. Xie, T. Li, J. Liu, S. Du, X. Yang, J. Zhang, Urban flow prediction from spatio-temporal data using machine learning: A survey, Information Fusion 59 (2020) 1 – 12.

[31] M. Raya, J.-P. Hubaux, Securing vehicular ad hoc networks, Journal of Computer Security 15 (1) (2007) 39–68.

[32] J. R. Douceur, The sybil attack, in: Revised Papers from the First International Workshop on Peer-to-Peer Systems, IPTPS'01, Springer-Verlag, Berlin, Heidelberg, 2002, pp. 251–260.

[33] P. Goodwin, The economic costs of road traffic congestion, University College London, The Rail Freight Group, accessed: 23/03/2020 (2004).

[34] F. Ran, Z. Jiang, M. Xu, Vision-based lane detection algorithm in urban traffic scenes, in: K. Li, Y. Xue, S. Cui, Q. Niu (Eds.), Intelligent Computing in Smart Grid and Electrical Vehicles, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 409–419.

[35] P. Papadimitratos, L. Buttyan, T. Holczer, E. Schoch, J. Freudiger M. Raya, Z. Ma, F. Kargl, A. Kung, J.-P. Hubaux, Secure Vehicular Communication Systems: Design and Architecture, IEEE Communcations Magazine 46 (11) (2008) 100–109.

[36] A. D. R. Yozevitch, B. Ben-Moshe, Gnss accuracy improvement using rapid shadow transitions, IEEE Transactions on Intelligent Transportation Systems 15 (3) (2014) 1113–1122.

[37] A. Boualouache, S. Senouci, S. Moussaoui, A survey on pseudonym changing strategies for vehicular ad-hoc networks, IEEE Communications Surveys Tutorials 20 (1) (2018) 770–790.

[38] M. Abadi, B. Blanchet, C. Fournet, Just fast keying in the pi calculus, ACM Transactions on Information and System Security (TISSEC) 10 (3) (2007) 1–59.

[39] B. Blanchet, Symbolic and computational mechanized verification of the ARINC823 avionic protocols, in: 30th IEEE Computer Security Foundations Symposium (CSF'17), IEEE, Santa Barbara, CA, USA, 2017, pp. 68–82.

[40] B. Blanchet, A. Chaudhuri, Automated formal analysis of a protocol for secure file sharing on untrusted storage, in: Proceedings of the 29th IEEE Symposium on Security and Privacy (SP'08), IEEE, 2008, pp. 417–431.

[41] B. Blanchet, B. Smyth, V. Cheval, M. Sylvestr, Proverif 2.00: Automatic cryptographic protocol verifier, user manual and tutorial, User Manual (5 2018).

[42] S. Krauß, Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics, Ph.D. thesis (1998).

[43] J. B. Kenney, Dedicated short-range communications (dsrc) standards in the united states, Proceedings of the IEEE 99 (7) (2011) 1162–1182.

[44] Vehicle-to-vehicle (v2v) and vehicle-to-infrastructure (v2i) communication in a heterogeneous wireless network - performance evaluation, Transportation Research Part C: Emerging Technologies 68 (2016) 168 – 184.

[45] S. N. Muthaiah, On-demand secure key generation in a vehicle-to-vehicle communication network, patent No. US8526606B2, Filed Dec. 20th., 2010, Issued Sept. 3rd., 2013 (September 2013).

[46] M. Mejri, J. Ben-Othman, Gdvan: A new greedy behavior attack detection algorithm for vanets, IEEE Transactions on Mobile Computing 16 (03) (2017) 759–771.

[47] D. Mansouri, L. Mokddad, J. Ben-othman, M. Ioualalen, Preventing denial of service attacks in wireless sensor networks, in: 2015 IEEE International Conference on Communications (ICC), 2015, pp. 3014–3019.

[48] A. Mpitziopoulos, D. Gavalas, C. Konstantopoulos, G. Pantziou, A survey on jamming attacks and countermeasures in wsns, IEEE Communications Surveys Tutorials 11 (4) (2009) 42–56.

[49] J. Guo, J. Baugh, S. Wang, A group signature based secure and privacy-preserving vehicular communication framework, 2007, pp. 103–108.

[50] T. C. Group, Tcg pc client platform tpm profile (ptp) specification, TCG Published, accessed: 12/03/2020 (2015).
URL https://www.trustedcomputinggroup.org/wp-content/uploads/PC-Client-Specific-Platform-TPM-Profile-for-TPM-2-0-v43-150126.pdf

[51] A. Gupta, A. Choudhary, A framework for camera-based real-time lane and road surface marking detection and recognition, IEEE Transactions on Intelligent Vehicles 3 (4) (2018) 476–485. doi:10.1109/TIV.2018.2873902.

[52] V. L. Knoop, P. F. de Bakker, C. C. J. M. Tiberius, B. van Arem, Lane determination with gps precise point positioning, IEEE Transactions on Intelligent Transportation Systems 18 (9) (2017) 2503–2513.

Appendix A. Auxiliary Algorithms

Algorithm 10 Auxiliary Algorithms

```
1: procedure updateRSUstate
2:     if the RSU is running then return "functioning"
3:     else return "shutdown"
4:     end if
5: end procedure
```

```
6: procedure enhancePics(pic_set, lanes_info)
7:     if lane lines are not visible in pic_set then
8:         enh_picset = drawLanes(pic_set, lanes_info)
9:         return enh_picset
10:    else
11:        return pic_set
12:    end if
13: end procedure
```

```
14: procedure updateSatNavLoc
15:    if satellite signal OK then return Current_loc
16:    else return (-1,-1)                          ▷ return an invalid coordinate
17:    end if
18: end procedure
```

```
19: procedure updatestate(loc)
20:    if the vehicle has parked then return "parked"
21:    else if loc shows the vehicle is at a junction then return "leave"
22:    else return "onroad"
23:    end if
24: end procedure
```

```
25: procedure updatemRSU(loc, MRSUrecord)
26:    if there exists entry in MRSUrecord, where
         entry = (mRSUid, Re_j) AND (loc is in Re_j) then
27:   return mRSUid
28:    else return -1
29:    end if
30: end procedure
```

```
31: procedure Cache((flag, time, loc, state, PK^S_VE), RecordSet)
32:    if  state == "onroad" then
33:        if Record(PK^S_VE) ∈ RecordSet then
34:            insert (flag, time, loc, state) in Record(PK^S_VE)
35:        else
36:            create Record(PK^S_VE)
37:            insert (flag, time, loc, state) in Record(PK^S_VE)
38:        end if
39:    else
40:        if (state == "leave") or (state == "parked") then
41:            delete Record(PK^S_VE)
42:        end if
43:    end if
44: end procedure
```