An Efficient Certificateless Authentication Protocol for the SAE J1939

Basker Palaniswamy

***Abstract*—Authentication continues to be a challenge for legacy real-time communications networks involving low-speed buses interconnecting resource-limited devices. A commercial vehicle network is such a network which does not change much over the years due to safety standards and regulations in the transportation domain. The SAE J1939 incorporating the ISO 11898-1 specification for the data link and physical layers of the standard CAN and CAN-flexible data rate (CAN-FD) handles communication among electronic control units (ECUs). The SAE J1939 is susceptible to attacks such as replay, masquerading and man-in-the-middle. This paper presents a formal analysis of the existing authentication protocols for the SAE J1939 and identifies limitation, especially man-in-the-middle attack. To mitigate the attack, we propose two new authentication protocols. One pass authentication protocol is proposed for computationally restricted nodes, and for the nodes that support public key operations, a certificateless signature-based authentication protocol is proposed which is based on certificateless key insulated manageable signature scheme (CL-KIMS). The security of the new protocol suite and the signature scheme is formally analysed in the random oracle model. We use the Tamarin tool to verify mutual authentication, session key security, known key secrecy and forward security of the proposed protocols. Performance comparison shows that compared with the existing protocol suite, the new protocol suite is computation and communication efficient with robust security. Our simulation study in Matlab 2018a reveals that the key exchange protocols in the new protocol suite are efficient regarding consumption of lesser total message delay than its counterpart.**

***Index Terms*—J1939 security, IVN security, security protocols, control system security, commercial vehicle cybersecurity, Truck security.**

I. Introduction

In the past, the safety of vehicles relied much on the reliability of its crucial mechanical parts, e.g. steering and brake, and communication protocols that were deployed to assist the mechanical parts inside the vehicles, e.g. the SAE J1939 [1], CAN 2.0A and CAN 2.0B [2]. Nonetheless, at present, the safety of vehicles have the dependability on the security of the information that are exchanged using the communication protocols. Moreover, advancements in the vehicular ad-hoc network (VANET) facilitate the vehicles to utilise different modes of communication, such as vehicle to infrastructure [3], vehicle to vehicle [4], vehicle to X communications [5]) to exchange information beyond its realm. But these communication modes help adversaries to infiltrate into the intra-vehicular network (IVN) of the vehicles. Thus, the information exchanged in the communication protocols are under threat for the adversarial attacks, namely replay, masquerading and man-in-the-middle (MITM) [6]. For instance, there is a chance for a highly likely situation of a deadlock condition to occur for the movement of vehicles on roads when a small portion of vehicles is targeted for the attacks [7].

As of 2015, there are 335 million commercial vehicles in operation across the world [8]. The SAE J1939 is the defacto communication protocol deployed in the IVN of commercial vehicles [9]. The commercial vehicles can be attacked by an adversary if the SAE J1939 communications are unprotected. It should be noted that a fair amount of steps that have been taken to safeguard the life on roads by the introduction of autonomous vehicles could go in vain as the autonomous driving is forecasted to cut down 5 million accidents and 2 million injuries occurring every year [10]. Authentication for SAEJ1939 can bring down the possibilities of the attacks in the commercial vehicles [9]; thence, establishing a secure and efficient authentication in SAE J1939 is an obligatory demand.

Generally, group key-based authentication schemes (GKAs) are efficient than the pair-wise key-based authentication schemes (PKAs) for IVNs. In principle, for a n number of nodes in IVN, group key schemes require n folded interactions, interactions containing a specified number of steps for exchanging a session key, by a trusted entity. Nevertheless, PKAs need $\frac{1}{2}\binom{n}{2}$ interactions for establishing a session key among n nodes. Researchers have conspicuously enumerated the pros and cons of the security strength of GKAs and PKAs for an IVN setting; they have also briefed out the historical timeline of the attempts taken for analysing and securing IVN [11].

Initially, the research on vehicular communication security has been focused on passenger cars. Charlie Miller and Chris Valasek have pointed the insecurity issues, analysing some of the passenger cars[12]. They have also shown the feasibility of obtaining secret information from cars, especially cryptographic keys[13]. Their research elucidates the necessity of an alternative mechanism to protect IVN without relying on the assumption of a tamper-proof modules (TPMs) [14] to secure the keys.

In contrast, very few attempts have been taken towards the analysis and design of security protocols for commercial vehicles, i.e., trucks and heavy-duty vehicles. Notably, Murvay et al. [9] have analysed the security issues in SAE J1939, and they have proposed an authentication protocol suite to thwart the identified issues, considering the computational capabilities of nodes in IVN. Albeit researchers have handled security problems of commercial vehicles and cars seperately hitherto, there is an union of the problems prevailing in IVN which has to be addressed. In essence, given an ”active adversary” with the capacity to compromise previous secret session keys and

long-term keys, a security protocol suite to withstand such an adversary has not been evident in the union to the best of our knowledge.

Following the convention in addressing the security problems of IVN, we enumerate the contributions:

- The protocol suite for SAE J1939 [9] is formally analysed against the active adversary using the Tamarin tool. A case study presenting an exploit of a MITM attack is reported for the protocols in the protocol suite.
- To prevent the identified MITM attack and other known attacks, namely, replay and masquerading, we present a new protocol suite for the SAE J1939. A round optimal authenticated key agreement protocol (AKEP) is designed for the resource-restricted nodes. For the resource unrestricted nodes, a certificateless AKEP is designed.
- To facilitate the certificateless AKEP, we design a certificateless key insulated manageable signature scheme (CL-KIMS).
- We analyze the security of the new protocol suite and signature scheme in the random oracle model [15].
- We deploy the Tamarin tool to formally verify the security goals of authenticated key exchange protocols; we evaluate the performance of the protocol suites in Matlab 2018a.

The rest of the paper is organized as follows. Section II discusses the related work. Section III presents preliminaries. Section IV presents the formal security analysis of the Murvay et al. protocol [9]. Section V describes the new protocol suite and our signature scheme. Section VI presents formal security analysis of the new protocol suite in the random oracle model. Section VII provides the formal verification of the new protocol suite using Tamarin tool. Section VIII compares performance of our protocol suite with the protocol suite of Murvay et al. [9]. Section IX evaluates performance of the protocol suites using Matlab 2018a. Section X concludes the work.

II. Related works

This section reviews related efforts taken towards securing CAN communication inclusive of J1939, and it also presents the works that are reported in designing key insulated signature schemes and its light-weight versions.

A. J1939 protocol

Because J1939 is a higher-layer protocol that supports CAN protocol in its lower layer, we review some of the authentication solutions available for the CAN protocol to understand the strategies of authentication. The significant steps taken so far is primarily to counteract any attempts made by adversaries to disrurpt normal communication. CANAuth has been proposed by Herrewege et al. [16]. CANAuth utilises hash-based message authentication code (HMAC) to authenticate nodes. To nullify the replay attacks, the researchers used counter values in the HMAC computation. Nonetheless, CANAuth depends on another protocol CAN+ to function [17].

Hartkopp et al. [18] have proposed MaCAN. Unlike CANAuth, it does not require CAN+ protocol to transmit HMAC tags. Since the data field is utilised for transmitting the tags, this protocol requires extra data frames to carry out authentication. MaCAN has been formally analysed by Bruni et al. [19]. Bruni et al. [19] have found two flaws in MaCAN. The first one allows an attacker to delete authentic messages during authentication. The second flaw permits an attacker to impersonate a legitimate node. In response, Bruni et al. [19] have proposed a modified MaCAN protocol that eliminates the attack feasibilities for an adversary.

LibraCAN has been proposed by Groza et al. [20]. LibraCAN achieves authentication by a mixed-mode approach. The protocol deploys the strategy of key splitting and MAC combining. LibraCAN leverages security by assuring a strong non-malleability property. But the protocol is vulnerable to the replay attack since the linear mixing procedure does not include temporal information.

In a sequel, an efficient protocol for the CAN has been proposed by Groza et al. [21]. The protocol is underpinned with the widely known TESLA [22]. But the protocol is designed on an assumption of a single secure node that acts as a cornerstone in IVN. It should be noded that an adversary can compromise a node completely [23].

Woo et al. [2] have demonstrated a practical remote attack using a vehicle, and in response, and they have designed a frame-level three-round authenticated key exchange protocol for CAN 2.0B. However, an authentication scenario for the nodes that are unrestricted in resources is not considered.

There are only few works that investigate security of the J1939 protocol. The work by Burokova et al. [1] describes practical attacks against the protocol used in trucks. They have found that instrument cluster unit, power train and engine brake can be controlled by spoofing and replaying messages. Mukherjee at al.[24] have demonstrated denial of service (DoS) attacks at the data link layer level of the SAE J1939. No practical solutions are provided for mitigating the attacks. Murvay et al. [9] have experimentally demonstrated DoS attacks and have proposed two authentication protocols to address the identified weaknesses. One is based on the symmetric-key cryptography for the resource restricted nodes; the other one is based on the public-key cryptography for the resource unrestrcited nodes. The security of the protocols has not been verified.

Overall, the security of all the protocols discussed above depends on the protection of secret keys in TPM.

B. Key Insulated Signature Scheme

Now, we review considerable works that are proposed to leverage the independence of TPM for securing secret keys.

In this regard, Dodis et al. [25] have introduced a key-insulated public-key cryptosystem, where the lifetime N of a secret key is split into t discrete-time durations. The cryptosystem assures two prominent properties, i.e., key insulation and random access key update. The key insulation property ensures the secrecy of secret keys for the time duration $(\ell, \ldots, t)$ even if the secret keys belong to the time duration $(1, \ldots, \ell-1)$ are compromised. The random access key update facilitates dynamic key update to any specific time duration in

the discrete-time duration from any given time duration. The key update from one period to another is accomplished by interacting with a trusted helper environment.

Based on this idea, researchers [26] have designed a strong key insulated signature scheme. For each key update, the signature leaks some information about the helper key. To address this problem, Hanaoka et al. [27] have proposed a parallel key insulated public key encryption. In their scheme, the decryption key is computed using two helper keys instead of one. This allows to maintain security even when one of the helper is compromised.

Computation restricted platforms (such as the internet of things and in-vehicle network) need lightweight cryptographic algorithms to achieve the intended security goals. Karati et al. [28] have proposed a certificateless signature scheme for the industrial internet of things. The scheme is based on bilinear pairing so it is quite computationally expensive. The scheme is analysed by Xiong et al. [29] who show that signatures can be forged. To overcome the weakness, Xiong et al. [29] have proposed a certificateless parallel key insulated signature scheme (CL-PKIS). CL-PKIS claims to provide strong key insulation property and light-weight computation cost. After a careful investigation, one can see that CL-PKIS does not offer key insulation property. This is to say that after compromising the secret key of $(\ell-1)^{th}$ period, the adversary can construct a valid secret key for ℓ^{th} period by choosing an arbitrary helper key. If an adversary signs any message in the ℓ^{th} period using the constructed key, the verifier returns *true* without differentiating a signature generated using the constructed key and the actual key. This is due to two reasons: CL-PKIS allows a signer to generate helper public keys without having a provision for the verifier to update the helper public keys, and the helper public key is made to be carried along with message-signature pair to the verifier for every message, which consumes additional bandwidth during communication.

In general, all the key insulated signature schemes discussed above cannot be straightforwardly adopted to the in-vehicle environment because they require the involvement of a user/driver to update the secret key from the helper environment.

III. Preliminaries

This section presents the SAE J1939 standard, network architecture, adversarial model, security requirements and the Tamarin tool to understand the background of this work.

A. The J1939 standard

Controller area network protocol is a broadcast communication protocol that supports a data rate up to 1 Mbps [30]. The initial version of can bus is CAN 2.0A or standard CAN. It supports four types of frames. They are data frame, remote frame, overload frame and error frame. The data frame and remote frame are used to send data and request data, respectively. Both frames have a 11-bit identifier field to resolve bus arbitration. That is when two or more frames are simultaneously transmitted, the frame with the highest priority wins the bus arbitration. In case of bus arbitration between data frame and remote frame as they have the same identifier, data frame wins the arbitration due to the dominant bit (logic 0) in RTR field. The next version of can bus is CAN 2.0B or extended CAN. The extended can has a 29-bit identifier field. The standard CAN and the extended CAN may exist on same bus, but the standard CAN has the highest priority. Another version of CAN bus is the CAN-flexible data rate (FD), CAN-FD. It supports three types of frames, viz: data, error and overload frame. This frame also has a 29-bit identifier field, but it supports variable data rate up to 8 Mbps. The size of the data field in the standard CAN and the extended CAN is 8 bytes, whereas the size of the data field in CAN-FD is 64 bytes.

Priority	Parameter Group Number (PGN)				Source
	EDP	DP	PDU Format	PDU Specific	Address
3 bit	1 bit	1 bit	8 bit	8 bit	8 bit
EDP-Extended Data Page		DP-Data Page			PDU-Protocol Data Unit

Fig. 1: J1939 identifier field

The SAE J1939 is a five-layered protocol (physical, data link, network transport and application). The physical and the datalink layer of the SAE J1939 includes the speccification of the ISO 118981. The SAE J1939 standard is built on the CAN protocol. This means that the lower layers of the SAE J1939 utilises CAN protocol [9]. The 29-bit identifier specifies that the SAE J1939 transmits frames as extended frames although standard CAN is supported. J1939 can support the standard CAN and CAN-FD in its lower layers. The SAE J1939 supports two data rates 250 Kbps and 500 Kbps. The identifier of the standard is broken down into three main fields: priority, parametric group number (PGN) and source address (ref Fig. 1). The priority field decides the priority of the frame during bus arbitration. The parametric group number is used to encapsulate signals that are with a common characteristic. The PGN is split into four fields: extended data page, data page, protocol data unit (PDU) format and PDU specific. The extended data page is always set to zero, and the data page is set based on PDU format and PDU specific. The PDU format denotes the type of PDU, i.e, if the PDU field is less than 240, it is treated as a destination address (PDU type 1). If the address is between 240 and 255, it is treated as a group extension (PDU type 2). If the PDU format is set to 255, it denotes a global reception address. This is separately specified since messages that do not belong to the nodes are discarded by the nodes. Each SAE J1939 message is broadcast to all ECU nodes. PDU specific is meant to handle specific purposes such as requesting information from other nodes and multi-frame messaging. The source address is used to include the address of the sender. The transport protocol of the standard utilises two modes to transfer information: connection mode data transfer and broadcast mode data transfer. More information on these modes can be referred in [9].

B. Network architecure

The network architecture of an in-vehicular network (IVN) is presented in Figure 2. It consists of three groups of ECUs, namely G_1, G_2 and G_3 clustered at the three levels:

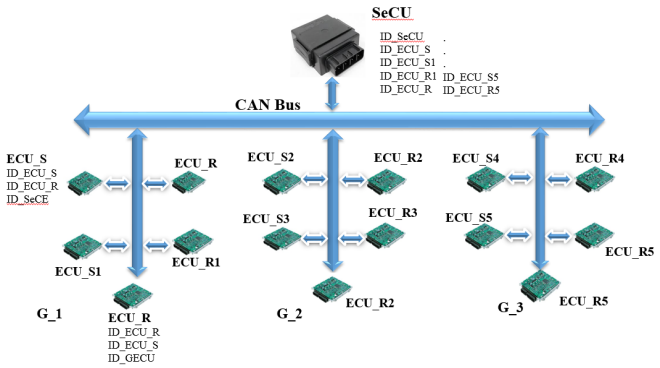

Fig. 2: J1939 based IVN

criticality high, moderate and low, respectively. The level of ECU can be determined using automotive safety integrity level (ASIL) requirements [31]. For example, ECUs controlling the windshield or wipers may be of low-level priority, whereas ECU of the antilock braking system or engine control unit are of high-level priority. SeCU is the central authority responsible for session keys management. SeCU maintains a list of identities (see Figure 2) and counters for all the ECUs. We write ECU_S and ECU_R to indicate the sender and receiver ECU, respectively. Note that an ECU can broadcast messages using multiple identities but we assume that ECUs broadcast messages on a single identity. Every ECU maintains a tuple which consists of identities and counters. Precisely, every ECU maintains its own ID and counter, ID and counter of its partners, ID and counter of $SeCU$ and a session counter.

C. Adversary Model

We assume the presence of an active probabilistic polynomial time adversary [32]. The adversary can inject, delete, modify and replay messages during a key exchange and normal communication. It can also acquire session keys and session specific secrets when past communication sessions are compromised. This may happen if an ECU is replaced for a repair or an ECU establishes communication with an external and insecure device.

For the signature scheme, we adopt the adversarial model introduced by Al-Riyami and Paterson in [33]. They define two types of adversaries: type-I and type-II adversaries ($\mathcal{A}_{\mathcal{I}}$ and $\mathcal{A}_{\mathcal{II}}$, respectively). For $\mathcal{A}_{\mathcal{I}}$, the adversary can be anyone except key generation centre (KGC). In case $\mathcal{A}_{\mathcal{II}}$, the adversary can be a malicious KGC. Consequently, $\mathcal{A}_{\mathcal{I}}$ can replace public keys but cannot acquire user secret keys. $\mathcal{A}_{\mathcal{II}}$ can acquire master secret key and partial secret key but it cannot replace the public key.

Now we specify the list of queries that can be made to ECU_i by the adversary.

$Extract - partial - private - key$: $\mathcal{A}$ asks this query by inputting $S.No$. Upon asking this query, the query returns a tuple $<sk_{S.No}, S.No>$ to adversary.

$Extract - public - key$: $\mathcal{A}$ asks this query by presenting $S.No$. This query returns P_{ECU_i} to $\mathcal{A}$.

$Extract - secret - key$: Upon asking this query with $S.No$, this query returns x_i to $\mathcal{A}$.

$Public - key - replace$: $\mathcal{A}$ asks this query by presenting $S.No$ and $P^*_{ECU_i}$. Upon querying, the oracle sets the public key of $S.No$ as $P^*_{ECU_i}$.

$Extract - signing - key$: $\mathcal{A}$ submits $S.No$ and t to this oracle. Upon submission, $\mathcal{A}$ is given with $sigk_{i,t}$.

$Extract - helper - key$: $\mathcal{A}$ asks this query by submitting $S.No$ and t. Upon asking this query, $\mathcal{A}$ is returned with $h_{i,t}$.

$Sign$: $\mathcal{A}$ asks this query by presenting $S.No^*$, t and M^*. The $Sign$ query returns signature G^*, $S.No^*$, t and M^*.

$\mathcal{A}$ plays the following four games $G_i \in \{1,2,3,4\}$ with a challenger $\mathcal{C}$.

$Game_1$:

$Setup$: $\mathcal{C}$ generates master secret key and public parameters by executing the setup algorithm. After generation of master secret key, $\mathcal{C}$ hides the secret and gives params to $\mathcal{A}_{\mathcal{I}}$.

$Query$: $\mathcal{A}_{\mathcal{I}}$ can access the above mentioned query adaptively for polynomial times.

$Forgery$: After finishing all the queries, $\mathcal{A}_{\mathcal{I}}$ randomly chooses a message M^* and outputs a signature G^* at time t. The adversary is said to win the game if the forgery meets the following conditions.

1) $\mathcal{A}_{\mathcal{I}}$ has never asked $S.No^*$ to the oracle $Extract - partial - private - key$, $Extract - signing - key$, $Extract - secret - key$ and $Extract - helper - key$.
2) $\mathcal{A}_{\mathcal{I}}$ has never asked $S.No^*$, M^* and t^* to the $sign$ oracle.
3) $True \leftarrow verify$ $(params, S.No^*, P_{ECU_i}, HP_i, M^*, G^*)$.

Definition 1: A CL-KIMS scheme is secure if there does not exist any probablistic polynomial time adversary $\mathcal{A}_{\mathcal{I}}$ that wins $Game_1$ with non-negligible advanatage.

$Game_2$:

$Setup$: $\mathcal{C}$ generates the params and master secret key. $\mathcal{C}$ issues params and the master secret key to the adversary $\mathcal{A}_{\mathcal{II}}$.

$Query$: $\mathcal{A}$ can ask the above mentioned queries adaptively.

$Forgery$: After collecting polynomial number of queries, $\mathcal{A}$ chooses an arbitrary message M^* and outputs the signature G^* at time t. $\mathcal{A}$ is supposed to win the game if the forgery meets the following condition:

1) $\mathcal{A}_{\mathcal{I}}$ has never asked $S.No^*$ to the oracle $Extract - secret - key$, $Extract - signing - key$ and $Extract - helper - key$.
2) $\mathcal{A}$ has never asked $S.No^*$, M^* and t to the $sign$ oracle.
3) $True \leftarrow verify$ $(params, S.No^*, P_{ECU_i}, HP_i, M^*, G^*)$.

Definition 2: If there does not exist an $\mathcal{A}_{\mathcal{II}}$ that wins the game G_2 with non-negligible advantage, then the CL-KIMS is a secure signature scheme.

$Game_3$:

$Setup$: $\mathcal{C}$ generates master secret key and public parameters by executing the setup algorithm. After generating master secret key, $\mathcal{C}$ gives params to $\mathcal{A}_{\mathcal{I}}$.

$Query$: $\mathcal{A}$ can access the above mentioned query adaptively for polynomial times.

$Forgery$: After finishing all the queries, $\mathcal{A}$ randomly chooses a message M^* and outputs a signature G^* at time t. The adversary is said to win the game if the forgery meets the following conditions.

1) $\mathcal{A}_{\mathcal{I}}$ has never asked $S.No^*$ to the oracle $Extract-partial-private-key$, $Extract-secret-key$, $Extract-helper-key$ at time t and $Extract-signing-key$ at time t.
2) $\mathcal{A}$ has never asked $S.No^*$, M^* and t^* to the $sign$ oracle.
3) $True \leftarrow verify$ $(params,$ $S.No^*,$ $P_{ECU_i},$ $HP_i,$ $M^*,G^*)$.

Definition 3: A CL-KIMS scheme is perfectly key insulated secure if there does not exist probablistic polynomial time adversary $\mathcal{A}_{\mathcal{I}}$ that wins $Game_3$ with non-negligible advanatage.
$Game_4$:

$Setup$: $\mathcal{C}$ generates the params and master secret key. $\mathcal{C}$ issues params and the master secret key to $\mathcal{A}_{\mathcal{II}}$.

$Query$: $\mathcal{A}_{\mathcal{II}}$ can ask the above mentioned queries adaptively.

$Forgery$: After collecting polynomial number of queries, $\mathcal{A}$ chooses an arbitrary message M^* and outputs the signature G^* at time t. $\mathcal{A}$ is supposed to win the game if the forgery meets the following condition:

1) $\mathcal{A}_{\mathcal{II}}$ has never asked $S.No^*$ to the oracle $Extract-secret-key$, $Extract-signing-key$ at time t and $Extract-helper-key$ at time t.
2) $\mathcal{A}$ has never asked $S.No^*$, M^* and t to the $sign$ oracle.
3) $True \leftarrow verify$ $(params,$ $S.No^*,$ $P_{ECU_i},$ $HP_i,$ $M^*,G^*)$.

Definition 4: If there does not exist an adversary $\mathcal{A}_{\mathcal{II}}$ that wins the game G_4 with non-negligible advantage, then the CL-KIMS is a strong key insulated secure signature.

D. Security requirements

The parties involved in the protocols are $SeCU$ and the ECUs. The following security goals should be achieved by an authenticated key exchange protocol (AKEP).

- *S1. Mutual entity authentication*, i.e. each of the communicating parties should be able to verify the identity of other party (through acquisition or collaborative evidence) [34].
- *S2. Key freshness* guarantees that the current session key is chosen independently and at random [35].
- *S3. Forward secrecy (FS)* prevents the adversary to compute session keys even if the long-term secrets of a party are compromised [36].
- *S4. Known-key secrecy (KKS)* assures the secrecy of current session key/secret when the old session secrets/keys are compromised [37].
- *S5. Session key secrecy* assures that the session key generated for every session is secret and it is not known to the adversary.

AKEP should resist the following attacks: replay, masquerading and MITM.

E. Tamarin

Tamarin is a "state-of-the-art" tool used for verifying the security properties claimed by cryptographic protocols. The tool accepts different adversary models. In particular, it can be deployed to verify protocols for the Dolev-Yao adversary [38] or user specified adversarial capabilities. The later noteworthy feature of this tool is a distinctive feature of this tool in comparison with the rest of the formal verification tools. The tool plays a priminant role for analysing group key schemes [39]. The complete description of the tool and its potential can be found in [40].

The mutual entity authentication can be verified using four properties: aliveness, weak agreement, non-injective agreement and injective-agreement. The properties follow a hierarchy. The top is the injective agreement [41]. It should be noted that the verification of the injective agreement also guarantees the resistance of the protocol towards the following attacks: replay, masquerading and MITM. The properties are normally verified using commit and running actions. Generally, using first-order logic, properties namely, the forward secrecy, known-key secrecy and session key secrecy can be verified.

TABLE I: Notations

Notation	Description
K_{ses}	The session secret key
K_{sync}	A secret key for time synchronisation
$K_{syncpair}$	A secret key for time synchronisation between a sender ECU and a receiver ECU
K_m	The secret master key
$Cert_{ECU_i}$	Certificate of i^{th} ECU
$Cert_{OEM}$	Certificate of the original eqipement manufacturer
$rand$	Random material
t_{SeCU}	Curent time in security designated ECU
$cntr_i$	Counter of i^{th} ECU
$cntr^{ses}$	Session counter
t_{snd}	Present counter value on the sender ECU
t_{rcv}	Present counter value on the receiver ECU
e	Public key encryption
$KD()$	Key derivation function
MAC	Message Authentication Code computed using HMAC
MAC_t	64-bit truncated message authentication code computed using HMAC
LGK	Long-term group secret key
K_i	Pair-wise i^{th} long-term secret key shared between i^{th} ECU and SeCU
$seed^i$	Seed value for i^{th} ECU
$cntr^{ses}$	Session counter

IV. Formal Analysis of Murvay et al. Protocols

We analyse the latest security protocol suite proposed by Murvay et al. for the SAE J1939. The suite consists of two key exchange protocols based on symmetric and asymmetric cryptography. Both protocols are followed by a runtime protocol for broadcasting messages among ECUs. We formally analyse the suite in Tamarin to determine its security weaknesses. For convenience, we refer to the symmetric key exchange protocol and asymmetric key exchange protocol as J1939-Sym and J1939-Asym, respectively. For a detailed description of the protocol, we refer the reader to the work [9].

A. Analysis of J1939-Sym

The purpose of this protocol to exchange session secrets from $SeCU$ to all ECUs. The protocol targets ECUs with computationally limited resources. The protocol does not provide secure mutual authentication because in the first protocol step, it does not require authentication of the random value ($rand_i$) obtained from ECU_i. Consequently, the injective agreement does not hold. Note that injective agreement requires a unique sender for each fresh commitment. This also means that the protocol accepts any replayed past communication. Figure 3 presents such case.

The weakness can be exploited in a man-in-the-middle attack. The steps given below explain the attack. The adversary replays message $rand^*$ generated by ECU_i (Step 1). The message is not authenticated by $SeCU$ so $SeCU$ responds according to the protocol and uses $rand^*$ in Step 3. Steps from 5 to 11 are executed by ECU_i and $SeCU$. Steps from 12 to 15 are run between the adversary and $SeCU$. Note that $SeCU$ is not able to authenticate random messages used in Steps 1, 5 and 12 and follows the protocol computing the key K_{sync}. This leads to MAC failure as ECU_i, and $SeCU$ are in out-of-sync with different keys.

$\mathcal{A}$
1) $Token_{ECU_i}\ rand^*(inject)$;
SeCU
2) *Generate* K_{ses} *and* K_{sync};
3) *Send* $e_{K_m}(K_{ses},\ K_{sync},\ rand^*)$;
$\mathcal{A}$/**ECU$_i$**
4) *Store /Discard*;
ECU$_i$
5) $Token_{ECU_i}\ rand$;
SeCU
6) *Take* K_{ses} *and generate* K^*_{sync};
7) *Send* $e_{K_m}(K_{ses},\ K_{sync},\ rand)$;
ECU$_i$
11) *Decrypt and store* K_{ses} *and* K_{sync};
$\mathcal{A}$
12) $Token_{ECU_i}\ rand^{**}(inject)$;
SeCU
13) *Take* K_{ses} *and generate* K^{**}_{sync};
14) *Send* $e_{K_m}(K_{ses},\ K^{**}_{sync},\ rand)$;
$\mathcal{A}$/**ECU$_i$**
15) *Store /Discard*;
ECU$_i$
16) *Send* $rand,\ MAC_{K_{sync}}(rand)$
SeCU
17) $MAC\ Verification\ fails\ (K_{sync} \neq K^{**}_{sync})$

B. Analysis of J1939-Asym

This protocol aims to establish session secrets among ECUs and $SeCU$. It applies public-key cryptography and needs a support of public key infrastructure (PKI) to obtain certificates of public keys. The protocol also lacks secure mutual authentication. The below steps present an MITM attack. It is easy to see that the random number $rand^*$ is not signed in Step 1. As the result, the protocol accepts past communication with a new random number. This is to say that $SeCU$ cannot tell apart forged requests (Steps 1 and 12) from a valid one (Step 5). The MITM attack was successfully implemented using the Matlab 2018 software.

$\mathcal{A}$
1) *Send* $Cert_{ECU_i} = \{g^x,\ Info,\ rand^*,\ Sig_{OEM}(g^x,\ Info)\}$;
SeCU
2) *Generate* K_{ses} *and* K_{sync};
3) *Send* $g^y\ e_K(Sig_{SeCU}(g^x,\ g^y,\ rand^*,\ K_{ses},$ $K_{sync},\ rand^*),\ K_{ses},\ K_{sync})$;
$\mathcal{A}$/**ECU$_i$**
4) *Store /Discard*;
ECU$_i$
5) *Send* $Cert_{ECU_i} = \{g^x,\ Info,\ rand,\ Sig_{OEM}(g^x,\ Info)\}$;
SeCU
6) *Generate* K_{ses} *and* K_{sync};
7) *Send* $g^y\ e_K(Sig_{SeCU}(g^x,\ g^y,\ rand,\ K_{ses},$ $K_{sync},\ rand),\ K_{ses},\ K_{sync})$;
ECU$_i$
11) *Decrypt and store* K_{ses} *and* K_{sync};
$\mathcal{A}$
12) *Send* $Cert_{ECU_i} = \{g^x,\ Info,\ rand^{**},\ Sig_{OEM}(g^x,\ Info)\}$;
SeCU
13) *Generate* K_{ses} *and* K_{sync};
14) *Send* $g^y\ e_K(Sig_{SeCU}(g^x,\ g^y,\ rand^{**},\ K_{ses},$ $K_{sync},\ rand^{**}),\ K_{ses},\ K_{sync})$;
$\mathcal{A}$/**ECU$_i$**
15) *Store /Discard*;
ECU$_i$
16) *Send* $rand,\ MAC_{K_{sync}}(rand)$
SeCU
17) $MAC\ Verification\ fails\ (K_{sync} \neq K^{**}_{sync})$

The protocol lacks forward security. When the private key x of ECU_i is compromised, then the encryption key K=g^{xy} can be computed by the adversary. As K is known, the keys K_{ses} and k_{sync} can be calculated.

V. New Protocol Suite

We propose a new protocol suite for J1939 to remove the identified vulnerabilities. It is composed of two amalgamated protocols for computationally restricted and non-restricted nodes. Each amalgamated protocol consists of an authenticated key exchange protocol (AKEP) and a protocol for time synchronization. Symmetric key AKEP (J1939-symmetric) based on hash functions only is proposed for the computationally restricted nodes. We remove the identified vulnerabilities in J1939-Sym as follows. MAC computations in J1939-symmetric includes all the random numbers. All the messages in J1939-symmetric are accompanied by their MAC tags. Certificateless AKEP (J1939-PKI) is proposed for the nodes that support PKI. Challenge-response pair is properly included in the signature computation to remove the identified vulnerability in J1939-Asym.

A. Registration & Protocols

All parties involved in the protocol must be registered by a trusted authority. In our case, the trusted authority is a vehicle manufacturer.

J1939-symmetric-Registration

The steps shown below present the registration procedure of vehicle manufacturer for J1939-symmetric involving $SeCU$ and ECU_i. The communication is done via a secure channel.

```
lemma Injectiveagreement_ECU:
  all-traces
  "∀ rand ECU SeCU #i.
    (Commit_strongA( ECU, SeCU, rand ) @ #i) ⇒
    (∃ #j.
      ((Running_strongA( SeCU, ECU, rand ) @ #j) ∧ (#j < #i)) ∧
      (¬(∃ ECU2 SeCU2 #i2.
          (Commit_strongA( ECU2, SeCU2, rand ) @ #i2) ∧ (¬(#i2 = #i))))))"
/*
guarded formula characterizing all counter-examples:
"∃ rand ECU SeCU #i.
  (Commit_strongA( ECU, SeCU, rand ) @ #i)
 ∧
  ∀ #j.
   (Running_strongA( SeCU, ECU, rand ) @ #j)
  ⇒
   ((¬(#j < #i)) ∨
    (∃ ECU2 SeCU2 #i2.
      (Commit_strongA( ECU2, SeCU2, rand ) @ #i2) ∧ ¬(#i2 = #i)))"
*/
```

Fig. 3: J1939-Sym injective agreement

$\mathbf{ECU_i}$
1) *Send ID*;
SeCU
2) *Generate* K_i;
3) *Send* K_i;
$\mathbf{ECU_i}$
5) *Store* K_i *in TPM*

J1939-PKI-Registration

For J1939-PKI registration, we assume the vehicle manufacturer and key generation centre KGC are same. Since J1939-PKI is based on certificateless signature scheme, we present the construction of CL-KIMS.

Construction of CL-KIMS

CL-KIMS is composed of the following nine algorithms.

1) Setup: KGC executes the algorithm. After recieving the security parameter n, KGC produces the master secret key and the public parameters $params$.

(i) KGC picks up an additive cyclic group G whose order is p. KGC randomly chooses a generator $P \in G$.
(ii) KGC selects the master secret key as $k \in_R Z_p$ and computes the public key of KGC as P_{KGC}=$k.P$.
(iii) KGC advertises five hash functions: H_1:$\{0,1\}^* \times G \rightarrow Z_p$, H_2:$Z_p \times \{0,1\}^{64} \times Z_p \rightarrow Z_p$, H_3:$Z_p \times Z_p \rightarrow Z_p$, H_4:$\{0,1\}^* \times G \times G \times G \rightarrow Z_p$ and H_5:$\{0,1\}^* \times \{0,1\}^* \times G \times G \times G \times \{0,1\}^* \times \{0,1\}^{64} \rightarrow Z_p$.
(iv) KGC finally outputs the public parameters as *params*=$\{G, P, P_{KGC}, H_1, H_2, H_3, H_4, H_5\}$.

2) Partial private key: After getting $params$, ECU_i inputs its serial number $S.No$ = $\{0,1\}^*$. KGC generates the partial private key as follows:

(i) KGC picks up a random number $r \in Z_p$ and computes $R_{S.No}$=$r.P$ and m=$H_1(S.No \parallel R_{S.No})$.
(ii) KGC calculates $sk_{S.No}$=r+$m.k \ mod \ p$ and returns ($sk_{S.No}$, $R_{S.No}$) to ECU_i.

3) Set secret key: Upon receiving $params$, $S.No$ and time period t, ECU_i generates its secret, helper key and initial sigining key as follows:

(i) ECU_i picks $x_i \in_R Z_p$ and computes its public key as P_{ECU_i}=$x_i.P$.

4) Set helper key: $SeCU$ sets the initial helper key for ECU_i.

(i) $SeCU$ picks $seed^i \in_R Z_p$ as the initial seed for ECU_i and computes helper value as h_1=$H_2(seed^i \parallel LGK)$. Correspondingly, the initial helper public key can be computed as HP_1=$h_1.P$.

5) Set signing key: ECU_i sets the signing key as follows: $sigk_i$=$sk_{S.No}$-h_i+$l.x$. Prior to setting the signing key, ECU_i computes $k = H_4(S.No \parallel P_{ECU_i} \parallel P_{KGC} \parallel R_{S.No})$ and l=$H_3(k \parallel t)$.

6) Update helper key: $SeCU$ and ECU_i execute this algorithm to update helper keys.

(i) In general, i^{th} helper value can be computed as h_i=$H_3(h_{i-1} \parallel LGK)$. $SeCU$ computes the i^{th} public terms of the helper values as HP_i=$h_i.P$.
In contrast to Xiong et al. [29] signature scheme, in this scheme the helper key is generated using a hash chain h_{i-1} and long-term group secret key LGK. The helper keys can be updated at specified time interval. The key update can be done with the help of session counter $cntr^{ses}$. The automatic key update assures self-healing ability to the signature scheme.

7) Update signing key: ECU_i updates the signing key as follows: $sigk_{i+1}$=$sk_{S.No}$-h_{i+1}+$l.x$

8) Set signing: Given a message $M \in \{0,1\}^* \in Z_p$ and period i, ECU_i signs the message as follows:

(i) ECU_i picks up $j \in Z_p$ and computes J=$j.P$.
(ii) ECU_i computes u=$H_5(M, S.No, R_{S.No}, P_{ECU_i}, J, P_{KGC}, t)$ G=j+ $u.sigk_i$ mod p.

9) Verify: The verification algorithm is loaded with $params$, $S.No$, P_{ECU_i}, HP_i and message signature pair (M, G). Then, it verifies whether the

following equation holds: $G.P$=J+u.($R_{S.No}$+$H(S.No \parallel R_{S.No}).P_{KGC} - HP_i$+$l.P_{ECU_i}$). If the equation holds, then the verifier returns $true$; otherwise, the verifier returns $false$. Secure channel is maintained during registration.

B. J1939-symmetric

J1939-symmetric suite consists of two protocols: key exchange and time synchronization. J1939-symmetric is presented below. The first seven steps present the key exchange protocol executed between ECU_i and $SeCU$. ECU_i sends a tag along with a random number. $SeCU$ verifies the authenticity of ECU and picks up a random token. The random token is fixed for the ECU group and it is changed, when $SeCU$ receives a request from another ECU group. The random token is forwarded to ECU_i after deriving session and synchronous keys. ECU_i verifies the authenticity of $SeCU$ and computes the keys by using the token. The protocol for time synchronisation consists of Steps 1.8 to 1.11 and is executed between ECU_i and $SeCU$. It is designed using symmetric cryptography (only MAC). All the tag computations in the key exchange protocol and the protocol for time synchronisation should include all the fields of a J1939 frame except the fields, where the tag is fed.

In order to broadcast messages using the standard CAN frame, messages to be broadcasted should be mapped to the standard CAN frame. To present the frame allocation in the standard CAN, we assume the following: size of a random number, token, key, truncated MAC tag and counter is 64 bits, 64 bits, 128 bits, 64 bits and 32 bits, respectively. Accordingly, the key exchange protocol needs two frames each for Step 1.1 and 1.5. In total, the key exchange protocol demands four frames.

ECU_i
1.1)$rand^* \in Z_p,\ Send\ rand^*,\ MAC_t(rand^* \parallel K_i)$;
SeCU
1.2) $Verify\ MAC_t(rand^* \parallel K_i)$;
1.3) $Generate\ token^*_{G_i} \in \{0,1\}^{64}$
1.4) $KD(token^*_{G_i} \parallel LGK) = K_{ses},\ K_{sync},\ K_{syncpair}$;
1.5) $Send\ token^*_{G_i},\ MAC_t(token^*_{G_i} \parallel rand^* \parallel K_i)$;
ECU_i
1.6) $Verify,\ MAC_t(token^*_{G_i} \parallel rand \parallel K_i)$;
1.7) $KD(token^*_{G_i} \parallel LGK) = K_{ses},\ K_{sync},\ K_{syncpair}$;
ECU_i
1.8)$rand^{**} \in Z_p,\ Send\ rand^{**},\ MAC_{./t}(rand^{**} \parallel K_{sync})$;
SeCU
1.9) $Verify\ MAC_{./t}(rand^{**} \parallel K_{sync})$;
1.10) $Send\ t_{SeCU},\ MAC_{./t}(t_{SeCU \parallel rand^{**} \parallel K_{sync}})$;
ECU_i
1.11) $Verify\ MAC_{./t}(t_{SeCU \parallel rand^{**} \parallel K_{sync}})$;

The last protocol of J1939-symmetric is the protocol for synchronising time among ECUs. The below steps presents the synchronising steps between a sender and a receiver ECUs.

ECU_s
2.1)$rand \in Z_p,\ Send\ rand,\ t_{snd},\ MAC(t_{snd} \parallel rand \parallel K_{syncpair})$;
ECU_r
2.2) $Verify\ MAC(t_{snd} \parallel rand \parallel K_{syncpair})$;
2.3) $Accept\ t_{snd}$;
4) $Send\ rand,\ t_{rcv},\ MAC(t_{rcv} \parallel rand \parallel K_{syncpair})$;
ECU_s
2.5) $Verify\ \ MAC(t_{rcv} \parallel rand \parallel K_{syncpair})$;
2.6) $Accept\ t_{rcv}$;

C. J1939-PKI

The below steps present J1939-PKI. ECU_i generates and forwards $x.P$ along with a signature to SeCU. SeCU verifies the signature and computes a session key K. The encrypted secrets C, $y.P$ and a signature are sent to ECU_i. ECU_i verifies the signature and decrypts the cipher text C. The signature computation in Step 3.2 and Step 3.6 includes all the fields of a J1939 frame. The frame allocation requires 2 frames for the messages at Step 3.2 and Step 3.6.

ECU_i
3.1) $Choose\ x\ \in_R\ Z_p,\ x.P,\ Sig_{ECU_i}(x.P)$;
3.2) $Send\ x.P,\ Sig_{ECU_i}(x.P)$;
SeCU
3.3) $Verify\ Sig_{ECU_i}(x.P),\ token^{**}_{G_i} \in_R Z_p$;
3.4) $KD(token^{**}_{G_i} \parallel LGK) = K_{ses},\ K_{sync},\ K_{syncpair}$;
3.5) $Choose\ y \in_R Z_p,\ y.P,\ K = x.y.P,\ C = \{K_{ses},\ K_{sync},\ K_{syncpair}\}K$;
3.6) $Send\ C,\ y.P,\ Sig_{SeCU}(y.P \parallel C \parallel x.P)$;
ECU_i
3.7) $Verify\ \ Sig_{SeCU}(y.P \parallel C \parallel x.P),\ K = x.y.P$;
3.8) $Decrypt\ C$;

VI. Security analysis

A formal analysis of J1939-symmetric and J1939-PKI is presented using the radom oracle model [42] to prove the security of the session secrets. Informal analysis of J1939-symmetric and J1939-PKI is done to prove the security of the protocols against the security goals and the attacks.

A. Informal analysis

Proposition 1: *The protocols J1939-symmetric and J1939-PKI provide session key freshness.*

In J1939-symmetric, the session keys, namely K_{ses}, K_{sync} and $K_{syncpair}$ are generated using a randomly generated token ($token^*_G$) for every session by the key derivation function KD (ref Step 1.7 in J1939-symmetric). Hence, J1939-symmetric assures the session key freshness.

Similar to J1939-symmetric, in J1939-PKI the session keys, namely K_{ses}, K_{sync} and $K_{syncpair}$ are generated using a randomly generated token ($token^{**}_G$) for every session by the key derivation function KD (ref Step 3.4 in J1939-PKI). Therefore, J1939-PKI ensures the session key freshness.

Proposition 2: *For the protocols J1939-symmetric and J1939-PKI, any replay of messages will be detected and discarded by the protocol parties.*

In J1939-symmetric protocol, the challenge-response pair is properly established. That is, $rand^*$ in the step 1.1 (challenge) is included in the MAC of Step 1.5. Therefore, any attempt of replay will not yield for the adversary. In J1939-PKI protocol, challenge message (xP in Step 3.2) is included in the response message (the signature in Step 3.6). Since the challenge-response pair is properly included in the protocol, the replay attack will not work on this protocol.

Proposition 3: *Given the protocols J1939-symmetric and J1939-PKI, any attempt of masquerading attack will fail with non-negligible probability.*

In J1939-symmetric protocol, the adversary has the masquerading possibility in the steps 1.1, 1.8 and 1.10. In all

the aforementioned steps, a MAC is included which require a session-specific secret or long-term secret. In particular, the steps 1.1 and 1.8 need the long-term secret K_i, and the step 1.10 requires session-specific secret k_{sync}. Hence, without the knowledge of either long-term or session-specific secret, the adversary cannot masquerade with high probability.

Similarly, in J1939-PKI, for the adversary to masquerade with non-negligible probability, the knowledge of long-term secret key is needed to construct a valid message specified in Step 3.2 and Step 3.6. Therefore, without knowing the long-term secrets of parties in the protocol, the adversary cannot masquerade with non-negligible probability.

Proposition 4: *Given the protocols J1939-symmetric and J1939-PKI, any attempt of MITM attack will suceed with negligible probability.*

In J1939-symmetric, we have argued that the adversary can succeed either the replay or the masquerading attack in the steps 1.1, 1.5, 1.8 and 1.10 with negligible probability. Hence, it is infeasible for the adversary to mount a MITM attack with non-negligible probability.

In J1939-PKI, to successfully mount a MITM attack with non-negligible likelihood, the adversary should be able to replay and masquerade messages. Since we have argued that neither of the attacks can succeed with non-negligible likelihood, it is infeasible for the adversary to mount a MITM attack with non-negligible likelihood.

B. Formal Analysis

Now, we present a formal analysis of J1939-symmetric and J1939-PKI. J1939-symmetric uses long-term secrets (K_i and LGK) to establish session secrets among ECUs that does not support PKI. J1939-PKI utilises long-term secrets $sk(ECU_i)$ and $sk(SeCU)$to establish session secrets among ECUs that support PKI.

A generic party $\mathcal{P}$ is introduced, which can be either one of ECU_i or *SeCU*. An instance can be defined as an execution of a protocol ($\mathcal{T}$). For example, an instance of $\mathcal{T}$ run by $\mathcal{P}$ is denoted by $\mathcal{T}_{\mathcal{P}}^{(i)}$, where i is the i-th run of the protocol. An oracle is defined as a collection of protocol instances that are run by the same parties $\mathcal{O}_{\mathcal{P}}^{\mathcal{T}} = \{\mathcal{T}_{\mathcal{P}}^{(i)} | i = 1, 2, \ldots\}$. A sequence of games is defined for the protocol to prove its security. The first game defines the attack against real protocol, and the final game defines the attack against the ideal protocol, where the advantage of adversary $\mathcal{A}$ is negligible. Since an advantage between any two consecutive games is negligible, we can conclude that $\mathcal{A}$ has a negligible advantage of winning the game against the real protocol.

$\mathcal{A}$ can make the following queries:

—$Send(\mathcal{T}_{\mathcal{P}}^{(i)}, M)$: The adversary ability to send messages is modelled using this query.

—$Execute(\mathcal{T}_{\mathcal{P}}^{(i)})$: $\mathcal{A}$ can overhear any communication in insecure channel that is modelled using this query.

—$Hash(\mathcal{T}_{\mathcal{P}}^{(i)})$: The hash oracle may be queried by an adversary. When the hash oracle receives a query, it looks at the hash table ($\mathcal{H}$), which stores the message and hash in the format (M, H).If the queried message (M) and is stored already in $\mathcal{H}$, it will return the stored hash (H). Otherwise, it will return a random number (H') for the inquired M and stores the pair (M, H') in $\mathcal{H}$.

Similarly, —$Sig(\mathcal{T}_{\mathcal{P}}^{(i)})$ & —$Enc(\mathcal{T}_{\mathcal{P}}^{(i)})$ queries model signature and encryption queries made by $\mathcal{A}$, respectively. The signature and the encryption oracles use tables $\mathcal{E}$ and $\mathcal{S}$, respectively to store message and cipher text pair, and message and signature pair in the form (M, C) and (M, S), respectively. When $\mathcal{E}$ and $\mathcal{S}$ receive cipher text or a signature that is not in $\mathcal{E}$ and $\mathcal{S}$, respectively, $\mathcal{E}$ and $\mathcal{S}$ return a random cipher text and a random signature as S' and C', respectively, and the tables make an appropriate record of the returned values.

—$Session - key\ reveal(\mathcal{T}_{\mathcal{P}}^{(i)})$: The ability of learning session keys by $\mathcal{A}$ is modelled using this query.

—$Session - state\ reveal(\mathcal{T}_{\mathcal{P}}^{(i)})$: $\mathcal{A}$ uses this query to learn session specific ephemeral secrets.

—$Corrupt(\mathcal{P}_{\mathcal{G}}^{(i)})$: Long-term secrets of parties can be acquired by $\mathcal{A}$ if $\mathcal{A}$ uses this query.

—$Test(\mathcal{P}_{\mathcal{G}}^{(i)})$: This query can be asked by $\mathcal{A}$ once only at the very end of the game. The asked party $\mathcal{G}$, tosses an unbiased coin b. If $b = 1$, then it returns the actual session key. Otherwise, it returns a random number. $\mathcal{A}$ wins the game if it is able to identify b.

Remark 1: The hash $H(\cdot)$, encryption $Enc(\cdot)$ and signature $Sig(\cdot)$ oracles are instances of the random oracle (RO).

Now the definitions that are needed for our analysis is introduced. The definition for partnership, and accepted and sid can be referred in [43] and [44], respectively.

Definition 5: (Freshness) The i-th session is *fresh* if $\mathcal{P}_{\mathcal{G}}^{(i)}$ is in accept state and the following queries are not asked, $Session - state\ reveal(\mathcal{T}_{\mathcal{P}}^{(i)})$, $Session - key\ reveal(\mathcal{T}_{\mathcal{P}}^{(i)})$ and $Corrupt(\mathcal{T}_{\mathcal{P}}^{(i)})$.

Definition 6: (Negligible function [45]) A function m:$\mathbb{N}\rightarrow\mathbb{R}$ is negligible if it approaches zero fater than the reciprocal of any polynomial. That is, for every $c\in\mathbb{N}$ there exist an integer n_c, such that $m(n)\leq\frac{1}{n^c}$ for all $n\geq n_c$.

Definition 7: (Elliptic curve discrete logarithm problem (ECDLP) assumption) For the given two points P and xP, it is assumed that finding x from the points is computationally intractable or any probabilistic polynomial time algorithm $\mathcal{A}$ has a negligible advantage in finding the integer x $Adv_{\mathcal{A}}^{ECDLP}(n) = Pr[\mathcal{A}(1^n, P, xP)] = x] \leq m(n)$.

Definition 8: (Computational elliptic curve Diffie-Hellman (CECDH) assumption) Given three points P, xP, yP, it is assumed that finding the point xyP from the points is computationally intractable or any probabilistic polynomial time algorithm $\mathcal{A}$ has a negligible advantage in finding xyP $Adv_{\mathcal{A}}^{ECCDH}(n) = Pr[\mathcal{A}(1^n, P, xP, yP)] = xyP] \leq m(n)$.

Definition 9: (Semantic security) The protocol is semantically secure if the adversarial advantage is negigible in guessing a secret bit for a test query issued to a fresh session or $Adv_{\mathcal{A}}^{SS}(\mathcal{T}, n) = Pr[\mathcal{A}(\mathcal{T}, n) = b] - \frac{1}{2} \leq m(n)$.

Difference lemma [46] Let A, B and F be events defined in some probability distribution, and suppose that $A \wedge \neg F \equiv B \wedge \neg F$. Then, $Pr[A] - Pr[B] \leq Pr[F]$.

Theorem 1: For the j1939-symmetric protocol (further called $\mathcal{T}$) and the following assumptions: a long term secret key can be guessed with an advantage $\frac{1}{2^l}$, the hash function

generates MACs with ℓ bits, $\mathcal{A}$ has access to all public oracles and can issue a polynomial number of queries. Then, semantic security of $\mathcal{T}$ can be broken by $\mathcal{A}$ with the advantage $Adv_{\mathcal{P}}^{SS} \leq \frac{q_{hash}^2}{2^\ell} + \frac{1}{2^\ell}$,

where q_{hash} stands for the number of queries to the hash oracle.

Proof of Theorem 1: Three games G_0, G_1 and G_2 are defined. The event in which $\mathcal{A}$ wins the game G_i, where $i = 0, 1, 2$ is E_i.

Game G_0: The real-world attack against semantic security is specified in this game. The adversary issues a $Test(\mathcal{P})$ query to an arbitrary session which is in the accept state and the freshness holds. $\mathcal{A}$ wins the game if it can guess the bit b. According to the definition of semantic security, $\mathcal{A}$ wins with an advantage:

$Adv_{\mathcal{T}}^{EE} = |Pr(E_0) - \frac{1}{2}|$.

Game G_1: This game and the previous game are identical unless the collision is found for MACs, i.e., MAC in the steps 1.1 and 1.5 of J1939-symmetric. If the collision occurs (in the hash table) for the MACs in the steps 1.1 and 1.5, $\mathcal{A}$ can modify the random integer and the token, respectively. As a result, $\mathcal{A}$ can inject tokens from the past communications, which will make the protocol to loose the session key freshness. According to the birthday paradox, the number of queries to the hash table need to be at least $\frac{q_{hash}^2}{2^\ell}$. We use the difference lemma to show that

$| Pr[E_1] - Pr[E_0] | \leq \frac{q_{hash}^2}{2^\ell}$.

Game G_2: This game presents the known key secrecy of $\mathcal{T}$. In this game, $\mathcal{A}$ asks $Session - key\ reveal(\mathcal{T})$ without losing the freshness of the session. That is, the queries are issued to the remaining sessions apart from the test session to get the session keys (K_ses, K_sync and $K_syncpair$). Despite possessing the remaining session keys, $\mathcal{A}$ requires the long-term secret key LGK to compute the present session keys. The advantage of $\mathcal{A}$ for attaining the LGK is $\frac{1}{2^\ell}$. The advantage of $\mathcal{A}$ in this game is:

$| Pr[E_2] - Pr[E_1] | \leq \frac{1}{2^\ell}$.

The game G_2 is the final one, where the probability of $\mathcal{A}$ in winning the event is, $Pr[E_2] = \frac{1}{2}$. By the triangular inequality, we have:

$$| Pr[E_0] - Pr[E_2] | = | Pr[E_0] - Pr[E_1] + Pr[E_1] - Pr[E_2] |$$
$$\leq | Pr[E_0] - Pr[E_1] | + | Pr[E_1] - Pr[E_2] | = \frac{q_{hash}^2}{2^\ell} + \frac{1}{2^l}.$$

This means that

$$| Pr[E_0] - Pr[E_2] | = | Pr[E_0] - \frac{1}{2} |$$

$$Adv_{\mathcal{T}}^{EE} \leq \frac{q_{hash}^2}{2^\ell} + \frac{1}{2^l}.$$

This concludes our proof. □

Theorem 2: For the J1939-PKI protocol (referred as $\mathcal{T}$) and assuming that the signature function generates digests with k bits of security, the encryption function produces cryptogram with l bits of security, the ECDLP and CECDH problem can be solved with an advantage Adv_{DLP} and Adv_{CECDH}, respectively the random numbers of m bits are used, $\mathcal{A}$ has access to all public oracles and can issue a polynomial number of queries. Then, semantic security of $\mathcal{T}$ can be broken by $\mathcal{A}$ with the advantage $Adv_{\mathcal{T}}^{EE} \leq max(Adv_{ECDLP}, Adv_{ECDLP}^2) + \frac{q_{enc}^2}{2^l} + Adv_{ECDLP} + max(Adv_{ECDLP}, Adv_{ECDHP})$ where q_{enc} stands for the number of queries to the encryption oracle.

Proof of Theorem 2: Three games are defined (G_0 to G_2). E_i be the event in which adversary wins the game G_i.

Game G_0: This game presents the real-world attack against $\mathcal{T}$. After collecting polynomial number of queries, $\mathcal{A}$ issues $Test(\mathcal{T})$ query to arbitrary session which is in accept state and freshness holds. The test query returns a bit to $\mathcal{A}$. $\mathcal{A}$ wins the game and gets session secrets if $\mathcal{A}$ guesses the bit correctly; otherwise, $\mathcal{A}$ gets random strings for the session secrets. As per the definition of semantic security, the advantage of $\mathcal{A}$ is

$Adv_{\mathcal{T}}^{EE} = |Pr(E_0) - \frac{1}{2}|$.

Game G_1: This game is identical to previous game except that $\mathcal{A}$ finds collision in the signature table. If $\mathcal{A}$ can forge signatures in Step 2 and Step 6, then it can forge ECU_i to $SeCU$ and inject fake session secrets to ECU_i by forging $SeCU$, respectively. The forgery can be done by two types of activities, which gives two cases.

Case i: Outsider adversary which can replace the public key of ECU_i but it cannot access the master secret key. For the replaced public key, PK_1 cannot be constructed as s is unknown. $\mathcal{A}$ can acquire s from PK_1 by employing a ECDLP solver. The advantage attained by $\mathcal{A}$ for employing the solver is Adv_{ECDLP}.

Case ii: Insider adversary can acquire the pubic key as a honest ECU_i, but the adversary cannot change the public key in the public directory. For replacing the public key from the public directory, PK_1 and PK_2 must be changed. The advantage attained by $\mathcal{A}$ for replacing the public key is Adv_{ECDLP}^2. The overall advantage acquried by $\mathcal{A}$ is

$|Pr(E_1) - Pr(E_0)| \leq max(Adv_{ECDLP}, Adv_{ECDLP}^2)$.

Game G_2: This game and the previous game are same except that $\mathcal{A}$ finds collision in the encryption table. If $\mathcal{A}$ can find collision in the encrytion table, then it can inject fake session secrets by replacing C. The advatage gained by $\mathcal{A}$ as per the difference lemma is

$|Pr(E_2) - Pr(E_1)| \leq \frac{q_{enc}^2}{2^l}$.

Game G_3: Known key secrecy is presented in this game. $\mathcal{A}$ issues $Session - key\ reveal(\mathcal{T}_{\mathcal{ECU}_\rangle}^{(\neg i)})$ and $Session - state\ reveal(\mathcal{T}_{\mathcal{ECU}_\rangle}^{(\neg i)})$ queries to the remaining session to acquire remaining session secrets and ephemeral secrets, respectively. Although $\mathcal{A}$ can acquire the remaining session secrets, K of the i^{th} session cannot be computed as ECDHP holds. So to compute K, $\mathcal{A}$ employs a ECDHP solver and feeds $x.P$ and $y.P$. The advantage attained by $\mathcal{A}$ for the employment is Adv_{ECDHP}. Then, according to the difference lemma

$|Pr(E_3) - Pr(E_2)| \leq Adv_{ECDLP}$.

Game G_4: This game presents the forward security of the protocol, and it is similar to the previous game. $\mathcal{A}$ issues $Corrupt(\mathcal{P}_{\mathcal{ECU}_\rangle}^{(i)})$ finally to acquire the long-term secret r of ECU_i. Although $\mathcal{A}$ knows r, it cannot decrypt C. So to decrypt C $\mathcal{A}$ must be knowing either x or K.$\mathcal{A}$ can employ two seperate solvers to acquire x and xy from $x.P$ and K, respectively. The advantage of $\mathcal{A}$ in the employment

is Adv_{ECDLP} and Adv_{ECDHP}. Then, as per the difference lemma

$$|Pr(E_4) - Pr(E_3)| \leq max(Adv_{ECDLP}, Adv_{ECDHP}).$$

In this final game, the probability of winning the event E_4 for $\mathcal{A}$ is $\frac{1}{2}$. by the triangular inequality, we have:

$$\begin{aligned} \mid Pr[E_0] - Pr[E_4] \mid = \mid Pr[E_0] - Pr[E_1] + Pr[E_1] - \\ Pr[E_2] + Pr[E_2] - Pr[E_3] + Pr[E_3] - Pr[E_4] \mid \\ \leq \mid Pr[E_0] - Pr[E_1] \mid + \mid Pr[E_1] - Pr[E_2] \mid \\ + \mid Pr[E_2] - Pr[E_3] \mid + \mid Pr[E_3] - Pr[E_4] \mid \\ = max(Adv_{ECDLP}, Adv^2_{ECDLP}) + \frac{q^2_{enc}}{2^l} + \\ Adv_{ECDLP} + max(Adv_{ECDLP}, Adv_{ECDHP}). \end{aligned}$$

This means that

$$\mid Pr[E_0] - Pr[E_4] \mid = \mid Pr[E_0] - \frac{1}{2} \mid$$

$$\begin{aligned} Adv^{EE}_{\mathcal{T}} \leq max(Adv_{ECDLP}, Adv^2_{ECDLP}) + \frac{q^2_{enc}}{2^l} + \\ Adv_{ECDLP} + max(Adv_{ECDLP}, Adv_{ECDHP}). \end{aligned}$$

This concludes our proof. □

Theorem 3: Suppose that if the discrete logarithmic problem is intractable in G, then the CL-KIMS is secure under the random oracle model.

The above theorem can be proved using the following four lemmata.

Lemma 4: If an adversary $\mathcal{A}_{\mathcal{I}}$ can break the CL-KIMS scheme against replacing public key attack with non-negligible advanatge $\epsilon^{\mathcal{A}_{\mathcal{I}}}_{success}$ and makes the following queries q_{SSK}, q_{SS}, q_{PPK}, q_{SH} and q_{H_1} to the oracles set secret key, set signing key, set partial private key, set helper key and hash oracle respectively, then there exist a challanger $\mathcal{C}$ that can solve the discrete logarithmic problem with the following advantage:

$$\epsilon^{\mathcal{C}}_{Success} \geq (1 - \frac{q_{PPK} + q_{SSK} + q_{SH} + q_{SS}}{p}).\frac{1}{q_{H_1}}.\epsilon^{\mathcal{A}_{\mathcal{I}}}_{success}$$

$T^{'}_m \approx \mathcal{O}(T_m.(q_{SS}+q_{PPK}+q_{SSK}+q_{SH}))$.

Proof: Given a random instance (P, Q) of a discrete logarithmic problem such that Q=$a.P$, where $a \in Z_p$, the goal of the challanger $\mathcal{C}$ is to compute a. To compute a, the challanger $\mathcal{C}$ simulates oracles and interact with $\mathcal{A}_{\mathcal{I}}$.

Setup: Before interacting with $\mathcal{A}_{\mathcal{I}}$, $\mathcal{C}$ initialises $\mathcal{A}_{\mathcal{I}}$ and itself with the following. At first, $\mathcal{C}$ sets $\mathcal{A}_{\mathcal{I}}$ with P_{KGC}=Q and sends (P, P_{KGC}) to $\mathcal{A}_{\mathcal{I}}$. Further, $\mathcal{C}$ randomly chooses $S.No^* \in \{0,1\}^*$ as the target $S.No$ for $\mathcal{A}$. To consistently reply to $\mathcal{A}_{\mathcal{I}}$'s query, $\mathcal{C}$ maintains six lists. They are L_0 =$\{S.No, sk_{S.No}, R_{S.No}, x_i, P_{ECU_i}, G, J\}$, L_1=$\{S.No, R_{S.No}\}$, L_2=$\{seed^i_1, cntr^{GECU}_1, GK\}$, L_3=$\{h_{i-1}, GK\}$, L_4=$\{S.No, R_{S.No}, P_{ECU_i}, P_{KGC}\}$, L_5=$\{M, S.No, R_{S.No}, P_{ECU_i}, J, P_{KGC}, i\}$.

$H_1 - Query$: $\mathcal{C}$ maintains a list called L_1. If $\mathcal{A}_{\mathcal{I}}$ queries $\mathcal{C}$ by inputting $S.No$. $\mathcal{C}$ performs the following:

1) If $S.No$ is already in the list, then $\mathcal{C}$ outputs the tuple $< S.No, R_{S.No} >$ to $\mathcal{A}_{\mathcal{I}}$.
2) Otherwise, $\mathcal{C}$ chooses a random number for $r* \in Z_p$ and computes $R^*_{S.No}$ as $r.P^*$.
3) $\mathcal{C}$ updates the list L_0 with the tuple $< S.No, R^*_{S.No} >$.

$H_2 - Query$: $\mathcal{C}$ maintains a list called L_2. If $\mathcal{A}_{\mathcal{I}}$ queries $\mathcal{C}$ by inputting $seed^i_1$ and $Cntr^{GECU}_1$, then $\mathcal{C}$ performs the following:

1) If $seed^i_1$ and $Cntr^{GECU}_1$ are in the list L_2, then $\mathcal{C}$ outputs the tuple $< seed^i_1, Cntr^{GECU}_1, GK >$ to $\mathcal{A}_{\mathcal{I}}$.
2) Otherwise, $\mathcal{C}$ chooses a random number $GK^* \in Z_p$ for GK and computes $< seed^i_1, Cntr^{GECU}_1, GK^* >$
3) $\mathcal{C}$ returns $< seed^i_1, Cntr^{GECU}_1, GK^* >$ to $\mathcal{A}_{\mathcal{I}}$ and updates the list with $< seed^i_1, Cntr^{GECU}_1, GK^* >$.

$H_3 - Query$: $\mathcal{C}$ maintains a list called L_3. If $\mathcal{A}_{\mathcal{I}}$ queries $\mathcal{C}$ by inputting $S.No$, then $\mathcal{C}$ performs the following:

1) If $H(h_{i-1} \parallel GK)$ is in the list L_2, then $\mathcal{C}$ outputs $H(h_{i-1} \parallel GK)$ to $\mathcal{A}_{\mathcal{I}}$.
2) Otherwise, $\mathcal{C}$ chooses two random numbers $GK^* \in Z_p$ for GK and $h^*_{i-1} \in Z_p$ for h_{i-1} and computes $H(h^*_{i-1}, GK^*)$.
3) $\mathcal{C}$ returns $H(h^*_{i-1} \parallel GK^*)$ to $\mathcal{A}_{\mathcal{I}}$ and updates the list L_2 with $H(h^*_{i-1} \parallel GK^*)$.

$H_4 - Query$: $\mathcal{C}$ maintains a list called L_4. If $\mathcal{A}_{\mathcal{I}}$ queries $\mathcal{C}$ by inputting $S.No$, $R_{S.No}$, P_{ECU_i}, P_{KGC}, then $\mathcal{C}$ performs the following:

1) If $H(S.No \parallel R_{S.No} \parallel P_{ECU_i} \parallel P_{KGC})$ is in the list L_4, then $\mathcal{C}$ outputs $H(S.No \parallel R_{S.No} \parallel P_{ECU_i} \parallel P_{KGC})$ to $\mathcal{A}_{\mathcal{I}}$.
2) Otherwise, $\mathcal{C}$ chooses random numbers for all the elements in H_4 and computes $H_4(S.No^* \parallel R^*_{S.No} \parallel P^*_{ECU_i} \parallel P^*_{KGC})$.
3) $\mathcal{C}$ returns $H_4(S.No^* \parallel R^*_{S.No} \parallel P^*_{ECU_i} \parallel P^*_{KGC})$ to $\mathcal{A}_{\mathcal{I}}$ and updates the list L_4 with $H_4(S.No^* \parallel R^*_{S.No} \parallel P^*_{ECU_i} \parallel P^*_{KGC})$.

$H_5 - Query$: $\mathcal{C}$ maintains a list called L_5. If $\mathcal{A}_{\mathcal{I}}$ queries $\mathcal{C}$ by inputting $S.No$ and M, then $\mathcal{C}$ performs the following:

1) If $H(M \parallel S.No \parallel R_{S.No} \parallel P_{ECU_i} \parallel J \parallel P_{KGC} \parallel i)$ is in the list L_4, then $\mathcal{C}$ outputs $H(M \parallel S.No \parallel R_{S.No} \parallel P_{ECU_i} \parallel J \parallel P_{KGC} \parallel i)$ to $\mathcal{A}_{\mathcal{I}}$.
2) Otherwise, $\mathcal{C}$ chooses random numbers for all the elements in H_5 and computes $H_5(M^* \parallel S.No^* \parallel R^*_{S.No} \parallel P^*_{ECU_i} \parallel J^* \parallel P^*_{KGC} \parallel i^*)$.
3) $\mathcal{C}$ returns $H_5(M^* \parallel S.No^* \parallel R^*_{S.No} \parallel P^*_{ECU_i} \parallel J^* \parallel P^*_{KGC} \parallel i^*)$ to $\mathcal{A}_{\mathcal{I}}$ and updates the list L_5 with $H_5(M^* \parallel S.No^* \parallel R^*_{S.No} \parallel P^*_{ECU_i} \parallel J^* \parallel P^*_{KGC} \parallel i^*)$.

$Partial\ private\ key\ extract - Query$: A list is maintained by $\mathcal{C}$ called PPK^{list}. $\mathcal{A}_{\mathcal{I}}$ queries the list.

1) If $< sk_{S.No}, R_{S.No} >$ is in PPK^{list}, then $\mathcal{C}$ outputs $< sk_{S.No}, R_{S.No} >$ to $\mathcal{A}_{\mathcal{I}}$.
2) Otherwise, $\mathcal{C}$ chooses two random numbers from Z_p, r^* and k^* and computes $R^*_{S.No}$=$r^*.P$ and $sk^*_{S.No}$=$R^*_{S.No}$+ $r^* + m.k^*$.
3) $\mathcal{C}$ returns $< R^*_{S.No}, sk^*_{S.No} >$ to $\mathcal{A}_{\mathcal{I}}$ and updates the list with $< R^*_{S.No}, sk^*_{S.No} >$.

$Secret\ key\ extract - Query$: A list is maintained by $\mathcal{C}$ called SK^{list}. $\mathcal{A}_{\mathcal{I}}$ queries the list.

1) If $< x_i, P_{ECU_i} >$ is in SK^{list}, then $\mathcal{C}$ outputs $< x_i, P_{ECU_i} >$ to $\mathcal{A}_{\mathcal{I}}$.
2) Otherwise, $\mathcal{C}$ chooses a random $x_i^* \in Z_p$ and computes $P_{ECU_i}^*$
3) $\mathcal{C}$ returns $< x_i^*, P_{ECU_i}^* >$ to $\mathcal{A}_{\mathcal{I}}$ and updates the list with $< x_i^*, P_{ECU_i}^* >$.

Signing key extract $-$ *Query* : A list is maintained by $\mathcal{C}$ called SG^{list}. $\mathcal{A}_{\mathcal{I}}$ queries the list by inputting i.

1) If $< sk_{S.No}, h_i, x.k >$ is in SG^{list}, then $\mathcal{C}$ outputs $< sk_{S.No}, h_i, x.k >$ to $\mathcal{A}_{\mathcal{I}}$.
2) Otherwise, $\mathcal{C}$ chooses three random numbers ($< sk_{S.No}^*, h_i^*, x.k^* >$) from Z_p.
3) $\mathcal{C}$ returns $< sk_{S.No}^*, h_i^*, x.k^* >$ to $\mathcal{A}_{\mathcal{I}}$ and updates the list with $< sk_{S.No}^*, h_i^*, x.k^* >$.

Public key replace $-$ *Query* : A list is maintained by $\mathcal{C}$ called L_0. $\mathcal{A}_{\mathcal{I}}$ queries the list by inputting $P_{ECU_i}^*$.

1) If L_0 =$\{S.No, sk_{S.No}, R_{S.No}, x_i, P_{ECU_i}^*\}$ is in L_0, then $\mathcal{C}$ outputs $\{S.No, sk_{S.No}, R_{S.No}, x_i, P_{ECU_i}^*\}$ to $\mathcal{A}_{\mathcal{I}}$.
2) Otherwise, $\mathcal{C}$ chooses a random number ($< x_i^{**} >$) from Z_p and computes $P_{ECU_i}^{**}$=$x_i^{**}.P$.
3) $\mathcal{C}$ returns $\{S.No, sk_{S.No}, R_{S.No}, x_i^{**}, P_{ECU_i}^{**}\}$ to $\mathcal{A}_{\mathcal{I}}$ and updates the list with $\{S.No, sk_{S.No}, R_{S.No}, x_i^{**}, P_{ECU_i}^{**}\}$.

Helper key extract $-$ *Query* : A list is maintained by $\mathcal{C}$ called HP^{list}. $\mathcal{A}_{\mathcal{I}}$ queries the list by inputting t.

1) If $< H_3(h_{i-1}, GK) >$ is in HP^{list}, then $\mathcal{C}$ outputs $< H_3(h_{i-1}, GK) >$ to $\mathcal{A}_{\mathcal{I}}$.
2) Otherwise, $\mathcal{C}$ chooses two random numbers GK^*, $h_{i-1} \in Z_p$ and computes $< H_3(h_{i-1}^*, GK^*) >$.
3) $\mathcal{C}$ returns $< H_3(h_{i-1}^*, GK^*) >$ to $\mathcal{A}_{\mathcal{I}}$ and updates the list with $< H_3(h_{i-1}^*, GK^*) >$.

Sign $-$ *Query* : When the tuple $< S.No^*, M^*, t^* >$ is delivered to the oracle, $\mathcal{C}$ chooses the list L_5 and reply to $\mathcal{A}_{\mathcal{I}}$ as follows:

1) If $< S.No^*, M^*, t^* >$ is in L_5=$\{M^*, S.No^*, R_{S.No}, P_{ECU_i}, J, P_{KGC}, i^*\}$, then $\mathcal{C}$ outputs G^*=j+$l^*.sigk_i$ mod p and J to $\mathcal{A}_{\mathcal{I}}$.
2) Otherwise, $\mathcal{C}$ chooses two random numbers $< j^{**}, sigk_i^{**} >$ from Z_p and computes G^{**}=j^{**}+$l^{**}.sigk_i^{**}$ mod p and J^{**}. $\mathcal{C}$ updates L_5=$\{M^*, S.No^*, R_{S.No}, P_{ECU_i}, J^{**}, P_{KGC}, i^*\}$.
3) $\mathcal{C}$ returns $\{G^{**}, J^{**}\}$ to $\mathcal{A}_{\mathcal{I}}$ and updates the list L_0 =$\{S.No, sk_{S.No}, R_{S.No}, x_i, P_{ECU_i}, G^{**}, J^{**}\}$.

Forgery : After finishing the oracle queries, $\mathcal{A}_{\mathcal{I}}$ outputs a forged signature (G^*, J^*, i^*) for M^* and $S.No^*$. $\mathcal{A}_{\mathcal{I}}$ computes G^*. The G^* can be verified as follows: $G^*.P$=J+$l.(R_{S.No} + H_1(S.No \parallel R_{S.No}).P_{KGC} - H_2(seed_1^i \parallel cntr_1^{GECU} \parallel GK).P_{ECU_i}$ According to forking lemma [47], if the challanger replays the oracle queries with same random tape for a different random function, then the challanger can produce five different valid signatures G_1^*, G_2^*, G_3^*, G_4^* and G_5^*. From the signature G^*, $\mathcal{C}$ has an opportunity to compute the discrete logarthim for the following: J=$j.P$, $R_{S.No}$=$r.P$, P_{KGC}=$k.P$, $H(h_{i-1} \parallel GK).P$ and P_{ECU_i}=$x.P$.

Now, we analyse the the probability that $\mathcal{C}$ solves the discrete logarthimic problem. We define three events where $\mathcal{C}$ invovles in solving the discrete logarthimic problem.

1) S_1: The event in which $\mathcal{C}$ does not abort the simulation while interacting with $\mathcal{A}_{\mathcal{I}}$.
2) S_2:The event in which $\mathcal{A}_{\mathcal{I}}$ generates a valid signature forgery for the message M^* and $S.No^*$ at time t.
3) S_3: The event in which $\mathcal{A}_{\mathcal{I}}$ satisfies the generated signature for $S.No^*$=$S.No$.

The success probability of $\mathcal{C}$ is defined by the following equation.

$$\begin{aligned} Pr[success] &= \epsilon_{Success}^{\mathcal{C}} \\ &= Pr[S_1 \wedge S_2 \wedge S_3] \\ &= Pr[S_1].Pr[S_2 \mid S_1].Pr[S_3 \mid S1 \wedge S2] \end{aligned}$$

The probability that $\mathcal{C}$ does not abort the simulation when $\mathcal{A}_{\mathcal{I}}$ makes at most queries to partial private key is $(1 - \frac{1}{p})^{q_{PPK}}$. The probability that $\mathcal{C}$ does not abort the simulation when $\mathcal{A}_{\mathcal{I}}$ makes at most queries to set secret key is $(1 - \frac{1}{p})^{q_{SSK}}$. The probability that $\mathcal{C}$ does not abort the simulation when $\mathcal{A}_{\mathcal{I}}$ makes at most queries to set helper key is $(1 - \frac{1}{p})^{q_{SH}}$.The probability that $\mathcal{C}$ does not abort the simulation when $\mathcal{A}_{\mathcal{I}}$ makes at most queries to set signing is $(1 - \frac{1}{p})^{q_{SS}}$. That gives $Pr[S_1] \geq (1 - \frac{q_{PPK}+q_{SSK}+q_{SH}+q_{SS}}{p})$. The probability that $\mathcal{C}$ outputs a valid signature is $Pr[S_2 \mid S_1] \geq \epsilon_{success}^{\mathcal{A}_{\mathcal{I}}}$. The probability that $\mathcal{A}_{\mathcal{I}}$ produces a valid signature by satisfying $S.No$=$S.No^*$ without aborting the simulation is given by $Pr[S_3 \mid S_2 \wedge S_3] \geq \frac{1}{q_{H_1}}$. Hence, the probability of solving the discrete logarthimic problem is given by

$$\epsilon_{Success}^{\mathcal{C}} \geq (1 - \frac{q_{PPK} + q_{SSK} + q_{SH} + q_{SS}}{p}).\frac{1}{q_{H_1}}.\epsilon_{success}^{\mathcal{A}_{\mathcal{I}}}$$

Analysis of running time : As the computation time for scalar multiplication is dominant than the computation time of other primitives, the running time complexity is evaluated for the operations involving point multiplication alone. $T_m^{'} \approx \mathcal{O}(T_m.(q_{SS}+q_{PPK}+q_{SSK}+q_{SH}))$

Lemma 5: If an adversary $\mathcal{A}_{\mathcal{II}}$ who knows master secret (k) and partial private key ($sk_{S.No}$) breaks the CL-KIMS with non-negligible advanatge $\epsilon_{success}^{A_{II}}$ and makes the following queries q_{SSK}, q_{SS}, q_{SH} and q_{H_1} to the oracles set secret key, set signing key, set helper key and hash oracle respectively, then there exist a challanger $\mathcal{C}$ that can solve the discrete logarithmic problem with the following advantage:

$$\epsilon_{Success}^{\mathcal{C}} \geq (1 - \frac{q_{SSK} + q_{SH} + q_{SS}}{p}).\frac{1}{q_{H_1}}.\epsilon_{success}^{\mathcal{A}_{\mathcal{II}}}$$

$T_m^{'} \approx \mathcal{O}(T_m.(q_{SS}+q_{SSK}))$.

The proof of this lemma is exactly same as that of proof of lemma 4. So we skip the proof of this lemma here. The only difference is that partial private key query does not involve.

Lemma 6: If an adversary $\mathcal{A}_{\mathcal{I}}$ who knows the helper keys and signing keys for $(t-1)$ period and tries to alter the public key of ECU_i at t can break the CL-KIMS with non-negligible advantage $\epsilon_{success}^{\mathcal{A}_{\mathcal{I}}}$ which makes the following queries q_{SSK}, q_{SS}, q_{PPK}, q_{SH} and q_{H_1} to the oracles set secret key, set signing key, set partial private key, set helper key and hash oracle respectively, then there exists a challenger $\mathcal{C}$ that can

solve the discrete logarithmic problem with the following advantage:

$$\epsilon^{\mathcal{C}}_{Success} \geq (1 - \frac{q_{PPK} + q_{SSK} + q_{SH} + q_{SS} - 2(t+1)}{p}). \frac{1}{q_{H_1}}.\epsilon^{\mathcal{A}_{\mathcal{I}}}_{success}$$

The proof of this lemma is exactly same as that of proof of lemma 4. So we skip the proof of this lemma here.

Lemma 7: If an adversary $\mathcal{A}_{\mathcal{II}}$ who knows the helper keys and signing keys for $(t-1)$ period ECU_i can break the CL-KIMS with non-negligible advantage $\epsilon^{\mathcal{A}_{\mathcal{I}}}_{success}$ which makes the following queries q_{SSK}, q_{SS}, q_{SH} and q_{H_1} to the oracles set secret key, set signing key, set helper key and hash oracle respectively, then there exists a challenger $\mathcal{C}$ that can solve the discrete logarithmic problem with the following advantage:

$$\epsilon^{\mathcal{C}}_{Success} \geq (1 - \frac{q_{SSK} + q_{SH} + q_{SS} - 2(t+1)}{p}). \frac{1}{q_{H_1}}.\epsilon^{\mathcal{A}_{\mathcal{II}}}_{success}$$

$T'_m \approx \mathcal{O}(T_m.(q_{SSK}+q_{SH}))$.

The proof of this lemma is exactly same as that of proof of lemma 4. So we skip the proof of this lemma here.

VII. Formal Verification

In this section, formal verification of J1939-symmetric and J1939-PKI protocols is provided to ensure that the proposed protocols satisfy the intended security attributes. The entire scripts of the rewritten protocol can be found in [48].

A. Verification of J1939 symmetric protocol

Here, we present the security properties that are verified in the Tamarin. We present the lemmata that are used to verify the security properties. The following lemmata in the first order logic verifies the security properties:

1) `executable`,
2) `Injectiveagreement_ECU`, `Injectiveagreement_SeCU`,
3) `ECU_secrecy`, `SeCU_secrecy` and
4) `ECUi_KKS`.

```
1) lemma executable: exists-trace "(Ex
 ECU SeCU m #i #j. Send(<ECU,m>)@i &
 Receive(<SeCU,m>) @ j & i <j) <=> T"
```

This lemma states that there exist `ECU` that sends message `m` at time `i`, and there exists `GECU` that recieves the `m` at time `j`. This lemma is essential to verify the reachability of states.

```
2) lemma Injectiveagreement_ECU:" All
 rand ECU SeCU #i.Commit_strongA(ECU,
 SeCU, rand)@ i==>(Ex #j . Running_
 strongA(SeCU,ECU,rand) @ j & j < i
 & not (Ex ECU2 SeCU2 #i2. Commit_
 strongA(ECU2,SeCU2,rand) @ i2 & not
 (#i2 = #i)))"
 lemma Injectiveagreement_SeCU:" All
 rand ECU SeCU #i.Commit_strongB(SeCU,
 ECU,rand)@ i ==>(Ex #j . Running_
 strongB(ECU,SeCU,rand) @ j & j < i
 & not (Ex ECU2 SeCU2 #i2. Commit_
strongB(SeCU2,ECU2,rand) @ i2 & not
(#i2 = #i)))"
```

The above two lemmata verify the injective agreement property. The lemmata verify that there uniquely exists a running partner for each commit by a party. It is noted that the injective agreement alone is verified as it sits on the top of the hierarchy of authentication claims.

```
3)   lemma ECU_secrecy:" not(Ex k_ses
 k_sync k_pair #r.!KU(<k_ses,k_sync,
 k_pair>)@r)"
lemma SeCU_secrecy: exists-trace" not
( Ex k_ses k_sync k_pair #r.!KU(<k_ses,
 k_sync,k_pair>)@r)"
```

The above two lemmata state that it cannot be the case that the knowledge of the adversary includes k_{ses}, k_{sync} and k_{pair}.

```
4) lemma ECUi_KKS:exists-trace" not (
 Ex k_ses k_sync k_pair #r.K(<k_ses,
k_sync,k_pair>)@r) & Ex ECU #y.Sesre-
veal(ECU)@ y "
```

The above lemma states that the knowledge of the adversary does not include any session secrets even upon sesssion state and session key reveal is done.

Mapping security goals: The verification of injective agreement (lemmata 2) satisfies the mutual entity authentication ($S1$) and resistance of the protocol towards the following attacks: replay, masquerading and MITM. The verification of lemmata 3 satisfies the session key secrecy ($S5$). The verification of lemma 4 satisfies the security goal ($S4$), which is the known key secrecy.

B. Verification of J1939 PKI protocol

The security verification of J1939 PKI protocol is similar to the verification of the J1939 symmetric protocol. The verification consist of five lemmata. They are

1) executable,
2) Injectiveagreement_ECU, Injectiveagreement_SeCU,
3) ECU_secrecy, SeCU_secrecy,
4) ECUi_KKS and
5) forward secrecy.

As the first four lemmata is similar to the lemmata specified in verification of J1939 symmetric protocol, the purpose of the lemmata alone is discussed here. The executable lemmata guarantees the reachability of states in the protocol. The injective agreement of ECU and SeCU states that for commited variables xp and yp, there uniquely exists a running partner. The secrecy lemma states that the knowledge of the adversary does not include any session secrets. The known key secrecy lemma specifies that the knowledge of the adversary does not include any session secrets even upon session key and session secret are revealed. The forward secrecy lemma is specified below.

```
5) lemma FS:" All lecu #y.K(<lecu>) @y
  ==>not(Ex k_ses k_sync k_syncpair #r
 .K(<k_ses,k_sync,k_syncpair>)@r)"
```

This lemma states that it cannot be the case that the knowledge of the adversary does not include any session secrets even up on revealing the long-term secret of ECU_i. Table II consolidates the verified lemmata of the two protocols.

Mapping security goals: The verification of injective agreement satisfies mutual entity authentication ($S1$) and resistance of the protocol towards the following attacks: replay, masquerading and MITM.The verification lemma 3 satisfies the security goal session key secrecy ($S5$). The verification of lemma 4 satisfies the security goal ($S4$) KKS. The verification of lemma 5 ensures the satisfaction of the security goal FS ($S3$).

TABLE II: Verified lemmata for the new protocol suite

Protocol \ Lemmata	Injective agreement	Session key security	KKS	FS
J1939-symmetric& syn	✓	✓	✓	NA
J1939-PKI & syn	✓	✓	✓	✓

VIII. Performace Comparison

In this section, we compare the performace in regards to computation and communication efficiencies of the new protocol suite with the protocol suite of Murvay et al. [9] initially. Then, we compare the performance of our signature scheme with related signature schemes [29] and [28].

The TEPLA library [49] is used to evaluate the various elliptic curve operations. From the library, BN-254 curve with the base field size of 512 bits is chosen for the evaluation. The inputs for the evaluation are fed as per the specifications in the library. For the evaluation of AES-128 and SHA-256, the pycrypto library [50] is utilised. All the operations are iterated for 1000 iterations, and the average value is taken as the computation time (C_T) of the operations. Correspondingly, CPU cycles per operation (CPO) is measured.

The evaluation is done on Ubuntu (V 16.04) with 64 bit Intel core I7 central processing unit (CPU) at 2.6 GHz operating frequency. The average computation time and CPO of the operations on Ubuntu is presented in Table III. We define the following notations. T_A, T_M, T_P, T_{Ex}, T_E and T_H-time required for a point addition on the curve, for a point multiplication on the curve, for a pairing operation on the curve, for an exponentiation operation on the curve, for performing an encryption operation and for performing a hash operation, respectively. It should be noted that the time duration for exponentiation operation (T_{Ex}) and the point multiplication are approximately equal [51]. To present a fair comparison at protocol level, we assume that a signature computation, signature verification, public-key encryption/decryption and certificate verification need a pairing operation in Murvay et al. [9] protocol and a point multiplication in J1939 PKI as the protocol suite of Murvay et al. [9] is based on conventional elliptic curve based PKI. To compare the communication cost involved in protocols, we assume the following: the size of a token is 64 bits, the size of a random number is 64 bits, the size of a symmetric encryption is 128 bits, the size of a MAC/MAC_t is 256/64 bits, the size of a signature is 512 bits, the size of a certificate is 512 bits, the size of a public key is 512 bits and the size of an elliptic curve point is 256 bits.

TABLE III: Average compuation time and CPO of the cryptographic operations in the Ubuntu OS

Notations	T_A	T_M	T_P	T_E	T_H
C_T	0.2 μs	1.38 ms	13.44 ms	0.22 ms	55.11 μs
CPO	5.18×10^4	3.6×10^6	41.78×10^6	5.46×10^5	14.61×10^4

TABLE IV: Security properties governed in J1939-symmetric protocol

Comparison of various security attributes		
Security properties	Murvay et al.[9]	J1939-symmetric
Mutual authentication	No	Yes
KKS	Yes	Yes
Session key security	No	Yes
Session key freshness	Yes	Yes
Resistance to impersonation attack	No	Yes
Resistance to replay attack	No	Yes
Resistance to MITM	No	Yes
DoS	No	Yes
Provable security	No	Yes
Formal verification	No	Yes

For J1939-symmetric protocol, the total computation time required is 6 T_H which is 330.66 μs and 87.66 $\times$ 10^4 CPU cycles, whereas Murvay et al. [9] protocol requires T_E which is 0.22 ms and 5.46 $\times$ 10^5 CPU cycles. It is apparent that J1939-symmetric requires $\approx$ 99 $\%$ lesser CPU cycles than the protocol suite of Murvay et al. [9]. The communication cost involved in J1939-symmetric is 256 bits that accounts for four standard CAN frames, whereas the communication cost involved in J1939-Sym [9] is 384 bits that accounts for six standard CAN frames. It is conspicuous that the key exchange protocol in J1939-symmetric consumes 20 $\%$ lesser number

TABLE V: Security properties governed in J1939-PKI protocol

Comparison of various security attributes		
Security properties	Murvay et al.[9]	J1939-PKI
Mutual authentication	No	Yes
KKS	Yes	Yes
Session key security	Yes	Yes
Forward secrecy	No	Yes
Session key freshness	Yes	Yes
Resistance to impersonation attack	No	Yes
Resistance to replay attack	No	Yes
Resistance to MITM	No	Yes
DoS	No	Yes
Provable security	No	Yes
Formal verification	No	Yes

of bits than J1939-Sym [9]. Table IV presents the security and functional attributes of J1939-symmetric in comparison with the protocol of Murvay et al. [9]. Therefore, J-1939-symmetric is computation and communication efficient with robust security.

The computation cost involved in J1939-PKI protocol is 4 T_M+2T_{Ex}+T_E, which is 8.5 ms and 221.46 $\times$ 10^5 CPU cycles. But J1939-Asym [9] needs 4T_P+T_{exp}+2T_H, which is 55.58 ms and 171.01 $\times$ 10^6 CPU cycles. It is clear that J1939-PKI consumes $\approx$ 60 $\%$ lesser CPU cycles than J1939-Asym [9]. Regarding communication cost, J1939-symmetric consumes 2176 bits that accounts for five CAN-FD frames, whereas J1939–Asym [9] needs 2368 bits that accounts for 10 CAN-FD frames. It is apparent that J1939-PKI consumes 3 $\%$ lesser bits than J1939-Asym [9]. In terms of security and functional attributes, Table IV presents the comparison result. Hence, J1939-PKI is computation and communication efficient and robust in security.

In table IV, V and VI yes denote the property is satisfied. No denotes the property is not satisfied, and NA denotes not applicable. In table VI, DL denotes discrete logarithm. EBSDH and BSDH denote extended bilinear strong Diffie- Hellman and Bilinear strong Diffie-Hellman problems, respectively [28]. Table VI and VII show the security property satisfied and performance comparison of the protocol, respectively. From Table VII, it is evident that CL-KIMS and Xiong et al. [29] signature scheme posses lesser computation time and CPU cycles than the signature scheme of Karati et al. [28]. Despite having the same computation time and CPU cycles, CL-KIMS satisfies important security attributes (ref Table VI) than the signature scheme of Xiong et al. [29]. Hence, CL-KIMS preserves important security attributes of a key insulated signature scheme, namely strong key insulation and secure random access key update and additionally offers properties namely, self-healing ability and independence of user involvement during key update.

IX. Simulation Performance

In this section, we simulate the behaviour of key exchange in Matlab 2018a using the vehicular network toolbox to analyse the simulation performance of the proposed protocol suite in comparison with the protocol suite of Murvay et al.

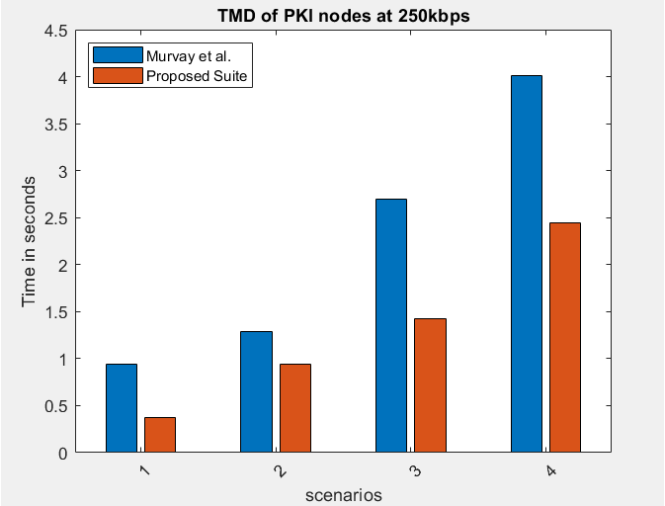

Fig. 4: TMD of PKI nodes at 250 kbps

[9]. We measure the communication response time regarding total message delay for both protocol suites. At first, we present the experimental scenario, and then we present the evaluation of total message delay (TMD). Before presenting the experimental scenario and TMD, the vehicular network toolbox in Matlab 2018a is introduced.

The vehicular network toolbox allows a user to simulate in-vehicle communication protocols, namely CAN, CAN-FD, J1939 etc. using Matlab functions and Simulink block sets. The toolbox has an industrial standard CAN database files which can be used to relay messages by choosing any protocol supported by the tool. Specifically, choosing a protocol supported by the tool, replay and injection of messages can be performed. Further analysis on the relayed messages can be performed by exploring the log file. The tool also facilitates visualisation of the signals with the help of scope in the toolbox.

A. Experimental Scenario

We consider four experimental scenarios in which the number of ECUs chosen are 25, 50, 75 and 100, respectively. All the experimental scenarios are simulated for 250 kbps and 500 kbp. Virtual channel 1 is chosen as the transmitter and receiver channel. To relay the frames, single frame transmission is chosen over the multi-frame transmission. Time taken to complete frame transmission is measured using tic-toc command, and the time taken is measured by average elapsed time to relay the frames. It should be noted that the tic-toc command in the Matlab behaves like a stop-watch timer. When a tic is encountered in the code, the timer in the Matlab starts. When toc is encountered in the code, the timer stops. Then, the difference in time between toc and toc is displayed as elapsed time for relaying frames.

1) Total message delay: Figure 4 and 5 present total message delay of proposed protocol suite for PKI nodes and Murvay et al. protocol suite at 250 kbps and 500 kbps, respectively. In the figures 3 and 4, it is apparent that the TMD of the proposed protocol suite is lesser than the TMD

TABLE VI: Security properties governed in proposed signature scheme

Comparison of various security attributes			
Security properties	Karati et al.[28]	Xiong et al.[29]	CL-KIMS
Security against type-I adversary	No	Yes	Yes
Security against type-II adversary	No	Yes	Yes
Strong key insulation	NA	No	Yes
Independence of user involvement during key update	NA	No	Yes
Secure random access key update	NA	No	Yes
Self-healing ability	NA	No	Yes
Security assumption	EBSDH & BSDH	DL	DL

TABLE VII: Performance comparison of the signature scheme

Item	Karati et al.[28]	Xiong et al.[29]	CL-KIMS
Signature length	$2 \mid G \mid$	$2 \mid G \mid + \mid Z_p \mid$	$2 \mid G \mid + \mid Z_p \mid$
Signing cost (S)	$3T_{exp}$	$2T_M$	$2T_M$
Verification cost (V)	$2T_{exp}$+T_P	$6T_M$	$6T_M$
Total cost (S+V)	20.34ms	11.04 ms	11.04 ms
CPU cycles	115.99$\times 10^6$	30.04$\times 10^6$	30.40$\times 10^6$

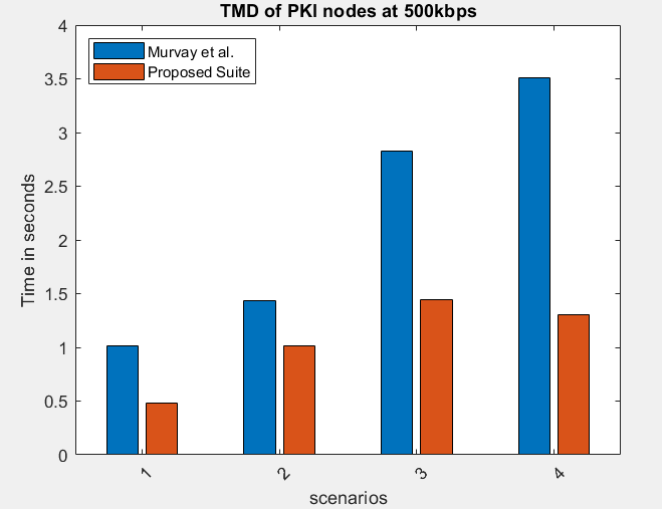

Fig. 5: TMD of PKI nodes at 500 kbps

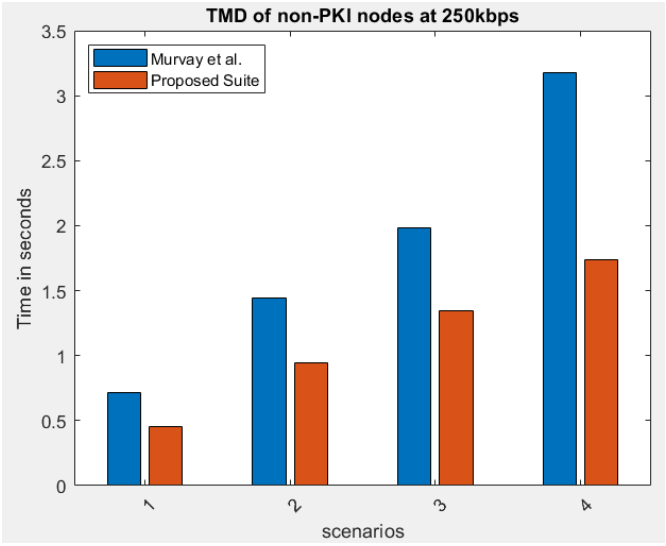

Fig. 6: TMD of non-PKI nodes at 250 kbps

of Murvay et al. protocol suite. Based on the scenarios 1, 3 and 5 in figure 4 and 5, the TMD of the proposed protocol suite is approximately 50 % lesser than the TMD of Murvay et al. protocol suite. This is due to the fact that the lesser number of frames are relayed because of not using certificates. In case of Murvay et al. protocol, the elliptic curve implementation requires 10 frames of CAN-FD in total for a given session, whereas the proposed protocol suite require only 5 frames of CAN-FD in a given session.

Figure 6 and 7 present TMD of proposed protocol suite for non-PKI nodes and Murvay et al. protocol suite at 250 kbps and 500 kbps. From both figures 6 and 7, it is clear that TMD of the proposed protocol suite for non-PKI nodes is lower than the TMD of Murvay et al. protocol suite. It is due to the lesser number frames consumed in Step 1.1 and Step 1.5 of J1939-symmetric. In case of Murvay et al. protocol suite, the total standard CAN frames required is 6 frames in a given session, whereas in the proposed protocol suite, just 4 standard CAN frames are sufficient.

Based on the simulation, it is evident that the key exchange protocols in the new protocol suite outperform the key exchange protocols of Murvay et al. [9].

X. Conclusion

In this paper, the protocol suite of Murvay et al. is formally analysed. We describe the vulnerability of insecure mutual authentication that leads to MITM attack using the Tamarin tool. In response, a new protocol suite is proposed. The new protocol suite has key exchange and time synchronisation protocols and a signature scheme. The key exchange protocol

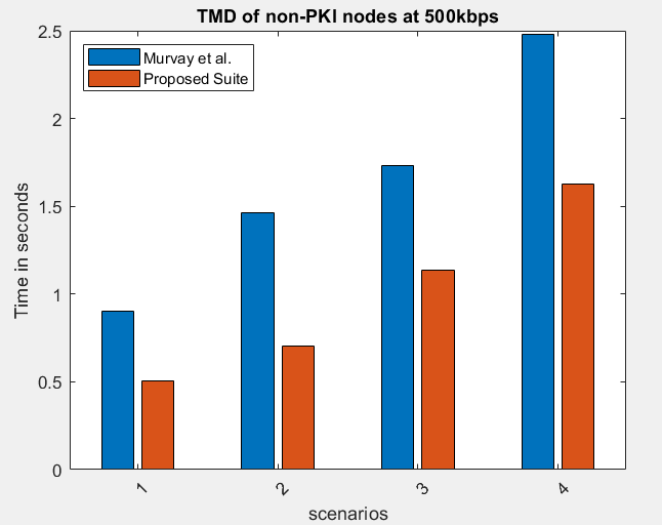

Fig. 7: TMD of non-PKI nodes at 500 kbps

for computationally restricted node assures one pass authentication, and the protocol for the nodes that support PKI is based on certificateless signature scheme. Both the protocols assure secure mutual authentication, and the later one assures forward secrecy. CL-KIMS assures important properties, namely strong key insulation, secure key update and self-healing ability. The protocol suite and the signature scheme are formally analysed in the random oracle model. The protocol suite is formally verified using Tamarin tool for mutual authentication, session secrecy, resistance towards the attacks such as replay, masquerading and MITM and other security properties. The performance comparison shows that the new protocol suite is computation and communication efficient than the protocol suite of Murvay et al. with robust security; the simulation performance shows that the key exchange protocols in the new protocol suite hold lesser total message delay than the protocol suite of Murvay et al.

REFERENCES

[1] Y. Burakova, B. Hass, L. Millar, and A. Weimerskirch, "Truck hacking: An experimental analysis of the sae j1939 standard," in *10th USENIX Workshop on Offensive Technologies*, 2016.

[2] S. Woo, H. J. Jo, and D. H. Lee, "A practical wireless attack on the connected car and security protocol for in-vehicle can," *IEEE Transactions on intelligent transportation systems*, vol. 16, no. 2, pp. 993–1006, 2015.

[3] Y. Toor, P. Muhlethaler, A. Laouiti, and A. De La Fortelle, "Vehicle ad hoc networks: Applications and related technical issues," *IEEE communications surveys & tutorials*, vol. 10, no. 3, pp. 74–88, 2008.

[4] H. Hartenstein and L. Laberteaux, "A tutorial survey on vehicular ad hoc networks," *IEEE Communications magazine*, vol. 46, no. 6, pp. 164–171, 2008.

[5] S. Djahel, R. Doolan, G.-M. Muntean, and J. Murphy, "A communications-oriented perspective on traffic management systems for smart cities: Challenges and innovative approaches," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 125–151, 2014.

[6] S. Eberz, M. Strohmeier, M. Wilhelm, and I. Martinovic, "A practical man-in-the-middle attack on signal-based key generation protocols," in *European Symposium on Research in Computer Security*. Springer, 2012, pp. 235–252.

[7] https://spectrum.ieee.org/cars-that-think/transportation/safety/hacking-gridlock.

[8] [Online]. Available: https://www.statista.com/statistics/281134/number-of-vehicles-in-use-worldwide/

[9] P.-S. Murvay and B. Groza, "Security shortcomings and countermeasures for the sae j1939 commercial vehicle bus protocol," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 4325–4339, 2018.

[10] N. A. Greenblatt, "Self-driving cars and the law," *IEEE spectrum*, vol. 53, no. 2, pp. 46–51, 2016.

[11] B. Groza and P.-S. Murvay, "Security solutions for the controller area network: Bringing authentication to in-vehicle networks," *ieee vehicular technology magazine*, vol. 13, no. 1, pp. 40–47, 2018.

[12] C. Miller and C. Valasek, "Can message injection: Og dynamite edition," Technical Report. http://illmatics. com/canmessageinjection. pdf, Tech. Rep., 2016.

[13] ——, "A survey of remote automotive attack surfaces," *black hat USA*, vol. 2014, p. 94, 2014.

[14] D. A. Fisher, J. M. McCune, and A. D. Andrews, "Trust and trusted computing platforms," CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, Tech. Rep., 2011.

[15] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proceedings of the 1st ACM conference on Computer and communications security*. ACM, 1993, pp. 62–73.

[16] A. Van Herrewege, D. Singelee, and I. Verbauwhede, "Canauth-a simple, backward compatible broadcast authentication protocol for can bus," in *ECRYPT Workshop on Lightweight Cryptography*, vol. 2011, 2011.

[17] T. Ziermann, S. Wildermann, and J. Teich, "Can+: A new backward-compatible controller area network (can) protocol with up to 16x higher data rates," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 1088–1093.

[18] O. Hartkopp and R. M. SCHILLING, "Message authenticated can," in *Escar Conference, Berlin, Germany*, 2012.

[19] A. Bruni, M. Sojka, F. Nielson, and H. R. Nielson, "Formal security analysis of the macan protocol," in *International Conference on Integrated Formal Methods*. Springer, 2014, pp. 241–255.

[20] B. Groza, S. Murvay, A. V. Herrewege, and I. Verbauwhede, "Libracan: Lightweight broadcast authentication for controller area networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 3, p. 90, 2017.

[21] B. Groza and S. Murvay, "Efficient protocols for secure broadcast in controller area networks," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2034–2042, 2013.

[22] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "The tesla broadcast authentication protocol," *Rsa Cryptobytes*, vol. 5, no. 2, pp. 2–13, 2002.

[23] C. Miller and C. Valasek, "Adventures in automotive networks and control units," *Def Con*, vol. 21, pp. 260–264, 2013.

[24] S. Mukherjee, H. Shirazi, I. Ray, J. Daily, and R. Gamble, "Practical dos attacks on embedded networks in commercial vehicles," in *International Conference on Information Systems Security*. Springer, 2016, pp. 23–42.

[25] Y. Dodis, J. Katz, S. Xu, and M. Yung, "Key-insulated public key cryptosystems," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2002, pp. 65–82.

[26] ——, "Strong key-insulated signature schemes," in *International Workshop on Public Key Cryptography*. Springer, 2003, pp. 130–144.

[27] G. Hanaoka, Y. Hanaoka, and H. Imai, "Parallel key-insulated public key encryption," in *International Workshop on Public Key Cryptography*. Springer, 2006, pp. 105–122.

[28] A. Karati, S. H. Islam, and M. Karuppiah, "Provably secure and lightweight certificateless signature scheme for iiot environments," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3701–3711, 2018.

[29] H. Xiong, Q. Mei, and Y. Zhao, "Efficient and provably secure certificateless parallel key-insulated signature without pairing for iiot environments," *IEEE Systems Journal*, 2019.

[30] S. Corrigan, "Introduction to the controller area network (can)," pp. 1–17, 2002.

[31] S. Woo, H. J. Jo, I. S. Kim, and D. H. Lee, "A practical security architecture for in-vehicle can-fd," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 8, pp. 2248–2261, 2016.

[32] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2001, pp. 453–474.

[33] S. S. Al-Riyami and K. G. Paterson, "Certificateless public key cryptography," in *International conference on the theory and application of cryptology and information security*. Springer, 2003, pp. 452–473.

[34] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *International conference on the theory and applications of cryptographic techniques*. Springer, 2000, pp. 139–155.

[35] L. Harn and C. Lin, "Authenticated group key transfer protocol based on secret sharing," *IEEE transactions on computers*, vol. 59, no. 6, pp. 842–846, 2010.

[36] A. K. Awasthi and S. Lal, "A remote user authentication scheme using smart cards with forward secrecy," *IEEE Transactions on Consumer Electronics*, vol. 49, no. 4, pp. 1246–1248, 2003.

[37] D. Choi, H. Kim, D. Won, and S. Kim, "Advanced key-management architecture for secure scada communications," *IEEE Transactions on Power Delivery*, vol. 24, no. 3, pp. 1154–1163, 2009.

[38] I. Cervesato, "The dolev-yao intruder is the most powerful attacker," in *16th Annual Symposium on Logic in Computer Science—LICS*, vol. 1, 2001.

[39] B. Schmidt, R. Sasse, C. Cremers, and D. Basin, "Automated verification of group key agreement protocols," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 179–194.

[40] [Online]. Available: https://tamarin-prover.github.io/manual/

[41] G. Lowe, "A hierarchy of authentication specifications," in *Proceedings 10th Computer Security Foundations Workshop*. IEEE, 1997, pp. 31–43.

[42] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proceedings of the 1st ACM conference on Computer and communications security*. ACM, 1993, pp. 62–73.

[43] M. Abdalla and D. Pointcheval, "Interactive diffie-hellman assumptions with applications to password-based authentication," in *International Conference on Financial Cryptography and Data Security*. Springer, 2005, pp. 341–356.

[44] M. Abdalla, P.-A. Fouque, and D. Pointcheval, "Password-based authenticated key exchange in the three-party setting," in *International Workshop on Public Key Cryptography*. Springer, 2005, pp. 65–84.

[45] M. Bellare, "A note on negligible functions." *Journal of Cryptology*, vol. 15, no. 4, 2002.

[46] V. Shoup, "Sequences of games: a tool for taming complexity in security proofs." *IACR Cryptology ePrint Archive*, vol. 2004, p. 332, 2004.

[47] D. Pointcheval and J. Stern, "Security proofs for signature schemes," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1996, pp. 387–398.

[48] [Online]. Available: https://github.com/FUllSCIPTS/paper3

[49] [Online]. Available: http://www.cipher.risk.tsukuba.ac.jp/tepla/

[50] https://pypi.org/project/pycrypto/.

[51] M. Azees, P. Vijayakumar, and L. J. Deboarh, "Eaap: Efficient anonymous authentication with conditional privacy-preserving scheme for vehicular ad hoc networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 9, pp. 2467–2476, 2017.