Cryptographic Key Distribution Protocol with Trusted Platform Module for Securing In-vehicle Communications

Béla Genge, Piroska Haller

'George Emil Palade' University of Medicine, Pharmacy, Science and Technology of Târgu Mureş, Gh. Marinescu 38, Târgu Mureş, Romania
bela.genge@umfst.ro, piroska.haller@umfst.ro

Abstract. The modern car comprises dozens of Electronic Control Units (ECUs) running several hundred MBs of code, alongside sophisticated dashboards with integrated wireless communications. While this has brought upon a wide range of advantages and integrated features, it also exposed modern vehicles to significant threats. As a response, we developed a generic scheme for distributing keys between ECUs. The approach leverages the Trusted Platform Module (TPM) 2.0 standard to provide a tamper-proof cryptographic co-processors capable of executing cryptographic operations in a secure manner. The scheme aims at empowering ECUs to leverage state of the art cryptographic techniques by adopting recent technological advances such as the TPM. Implementation details and analysis demonstrate the feasible application of the approach in the context of the modern vehicle.

Keywords: Automotive systems, cryptographic protocols, trusted platform module

1 Introduction

The modern car has experienced profound changes in terms of its internal technological ecosystem. Nowadays, we find dozens of Electronic Control Units (ECUs) running several hundred MBs of code, alongside sophisticated dashboards with integrated wireless communications [1].

While this technological advancement has brought upon a wide range of advantages and integrated features, it also exposed modern vehicles to significant threats. As demonstrated by many recent threats [2], vehicle digital communication systems can be exploited in a way that alters the vehicle's behavior in order to gain certain advantages (e.g., financial, performance), or, in more extreme cases, to cause physical damage. Subsequently, changes in the vehicle's parameters, the connection of sensor emulators (e.g., AdBlue emulators), can significantly alter the operation of the vehicle's internal subsystems. Such changes can deactivate critical systems such as the Selective Catalytic Reduction (SCR) dosing system.

Consequently, it is imperative to ensure that ECUs are able to securely exchange data, but most importantly, it is necessary that ECUs are empowered with the ability to verify the authenticity of critical frames. To this end, we find several techniques to ensure the integrity and authenticity of data transfers within vehicles [3, 4]. On the other hand, only few studies addressed the issue of key distribution in these systems. The main rationale is the high computational requirements of modern asymmetric algorithms, which can significantly affect the normal operation of in-vehicle controllers.

To address this issue, this paper documents a generic scheme for distributing keys between ECUs found in modern vehicles. The approach leverages the Trusted Platform Module (TPM) 2.0 standard [5] developed by the Trusted Computing Group (TCG) to provide a tamper-proof cryptographic co-processors capable of executing cryptographic operations in a secure manner. The approach is aimed at empowering ECUs to leverage state of the art cryptographic techniques by adopting recent technological advances such as the TPM. Implementation details and analysis demonstrate the feasible application of the approach in the context of the modern vehicle.

The remainder of this work is structured as follows. Section 2 provides the background terminology and an overview of related studies. Then, Section 3 details the developed scheme, while Section 4 provides an analysis of concurrency and security. This is followed by Section 5, which provides incipient implementation details and an analysis on the possible integration impact. The paper concludes with Section 6.

2 Background and Related Work

2.1 Internal Architecture of the Modern Car

From an architectural point of view, the modern car comprises dozens of embedded devices, also known as Electronic Control Units (ECUs), communicating with each other, as well as with digital and/or analog sensors. Here, we encounter a diverse ecosystem of control units (e.g., Engine Control Unit, Communication Control Unit) and sensors (e.g., Nox Sensor), together with communication protocols that range from Controller Area Network (CAN), to SENT (Single Edge Nibble Transmission) or MOST (Media Oriented Systems Transport).

In today's modern vehicles the "backbone" communication is provided by the Controller Area Network (CAN). Initially developed by Robert Bosch GmbH in 1986 for connecting electronic devices in motor vehicles, CAN is no longer restricted to automotive applications. Initially standardized by the International Standardization Organization (ISO) in 1993 as ISO 11898, and later restructured in three parts, the CAN standard specifications cover the data link layer and CAN physical layer for high-speed and low-speed CAN.

The original CAN specifications (Versions 1.0, 1.2 and 2.0A) specify an 11 bit message identifier. This is known as "Standard CAN". Specification V2.0A has been updated (to version 2.0B), to remove the message number limitation

and meet the SAE J1939 standard for the use of CAN in heavy duty vehicles. Version 2.0B CAN, referred to as "Extended CAN", contains a 29-bit identifier in the arbitration field, which allows over 536 Million message identifiers.

Given the limitations of the "classical" CAN protocol in terms of bandwidth (up to 1Mbit/s) and payload size (each CAN data frame can hold up to 8 data bytes), recently, two main improved communication infrastructures have been proposed. The CAN+ protocol was proposed by Ziermann, *et al.* in 2009 [6], and it exploits the time between transmissions to send additional data. More recently, in 2012, Robert Bosch GmbH developed the CAN with flexible data-rate protocol (CAN-FD version 1.0) [7], which later became part of the ISO 11898-1:2015 standard. This specification allows for increased data lengths and, once the arbitration is decided, a higher bit rate can be configured for the data phase. Up to 8Mbit/s data rates and up to 64 data bytes are now possible with CAN-FD.

The AUTomotive Open System ARchitecture (AUTOSAR) Specification of Secure Onboard Communication standard (AUTOSAR SecOC) [4] provides the principles for achieving authentication and integrity protection of sensitive data exchanged communication peers in an automotive network. The approach is generic, and it supports both symmetric and asymmetric methods. It recommends the use of Message Authentication Code (MAC) for symmetric, and public key-based digital signatures for asymmetric applications. The specification recommends that each Protocol Data unit – PDU (e.g., a CAN frame) is extended with an authentication component (e.g., the MAC) and a freshness value (e.g., freshness counter, timestamp). In addition, for certain protocols, such as the CAN protocol, the size of the MAC tag can be truncated down to 64 bits in order to fit into a single frame.

2.2 Trusted Platform Module

Each component found within the modern car, depending on its hardware configuration, presents different requirements in terms of security objectives. As an example, ECUs require a broader range of features in comparison to a digital sensor, even if the same communication medium is used. Typically, security objectives are implemented within the existing system and are executed on the same processing unit. This affects the computational load on the processing unit. In the case of environmentally restricted units, such an overload can lead to delays that may alter the real-time operation of the CAN ecosystem.

In order to enable security features in environmentally restricted units, the use of security controllers is imperative. To this end, the Trusted Platform Module (TPM) 2.0 standard [5] developed by the Trusted Computing Group (TCG), offers the description for compatible security controllers. TPMs represent tamper-proof cryptographic co-processors capable of executing cryptographic operations in a secure manner, isolated from the main processing unit. In the remainder of this document the TPM abbreviation is used to denote a security controller compatible with the TPM 2.0 specification.

The TPM presents several core capabilities based on which complex security mechanisms can be built. Serving as a Root of Trust (RoT), it offers secure key storage in volatile and non-volatile memory, an Endorsement key (Ek) sealed and only available to the TPM itself, and a set of Platform Configurable Registers (PCRs) capable of performing local integrity measurements by leveraging hash based functions. For persistent storage, usually TPMs contain a non-volatile memory which, in certain cases, given it's modest dimension can't facilitate the storage of a high number of cryptographic keys. As a solution to this problem, the TPM presents two additional methods of processing keys: sealing, which securely stores the cryptographic keys on disk by encrypting the private part of the key pair by leveraging the Ek, and key hierarchies, which allow deterministic key derivation starting from a parent key stored in the TPM. The keys obtained after derivation at run-time follow a tree-like graph structure, where a node can represent a parent or child key. To store the key generated from a key hierarchy, TPM uses the sealing method.

2.3 Related Work

Several techniques have been developed in order to secure CAN communications. Starting with the work of Herrewege, *et al.* [3], it was shown that, while the CAN bus has significant limitations in terms of payload and bandwidth, data authentication is still achievable via Message Authentication Codes (MAC), and more specifically the HMAC standard, alongside a counter in order to ensure freshness and resistance against replay attacks. The approach entitled *CANAuth* presumes the presence of the CAN+ protocol, which permits out-of-band transmissions [6].

With respect to techniques targeting CAN-FD, which is within the main scope of the developed approach, we mention the work of Woo, *et al.* [8], where a secure communication protocol and security architecture were developed for CAN-FD-enabled in-vehicle systems. The approach embraces two distinct layers. A first security layer, where regular ECUs communicate securely by leveraging symmetric cryptographic constructions (e.g., symmetric encryption, and MAC). This is reinforced by a second security layer consisting of an advanced gateway capable of signing and verifying signatures via asymmetric cryptographic computations.

Wang and Liu [9] acknowledged the importance of securing gateway ECUs (i.e., ECUs enabling external communications), which provide various communication interfaces, including the support for Vehicle to Internet communications. These benefit from significant computational power and communication bandwidth and enable the realisation of new infrastructural paradigms such as the Internet of Vehicles. Wang's approach focused on the secure distribution of cryptographic keys in and outside the vehicle. In a similar fashion, we find other approaches, where the emphasis was placed on the distribution of cryptographic keys [10]. Groza and Murvay [10] evaluated several techniques for distributing keys in CAN-FD, including Diffie-Hellman and identity-based cryptography. Compared to this approach, we present a generic set of protocols that distinguish between long-term and short-term keys in order to reduce the number of

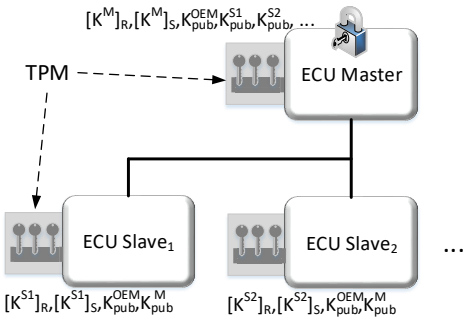

Fig. 1. Bootstrapping.

messages and, overall, the impact on communications. Furthermore, we leverage the capabilities of TPM to reduce the computational overhead.

3 Cryptographic Key Distribution Scheme

Typically, security features are implemented within the existing system and are executed on the same processing unit, which affects the computational load on the processing unit. In the case of environmentally restricted units, such an overload can lead to delays that may alter the real-time operation of the CAN ecosystem. In order to enable security features in environmentally restricted units, the use of security controllers is imperative. We realize that not all components found within the vehicle can be equipped with a TPM. Furthermore, in the case of data exchanges with digital sensors, complex cryptographic operations (e.g., digital signatures) may not be supported. Therefore, this paper does not focus on the key distribution concerning components not supporting TPM extensions.

3.1 Setup and Bootstrapping

Figure 1 denotes the main entities present in the key distribution scheme. Here, we observe two main entities: "Master" and "Slave". The Master is the one that triggers the generation and distribution of a new cryptographic key, while the Slaves are the recipients of fresh cryptographic keys. The developed key distribution scheme leverages public key cryptography, and several types of symmetric keys.

In terms of cryptographic keys, each ECU, that is, each TPM associated to an ECU, is bootstrapped with several types of keys. More specifically, the following are the relevant keys at bootstrapping in the context of an ECU/TPM:

– Storage Root Key (SRK): A pair of special, public-private keys that never leave the TPM. The SRK is used to encrypt the keys that are stored outside the TPM.

- Key Signing Key (KSK): A pair of public-private keys used in the key distribution procedure, as described later, in order to enforce the non-repudiation property.
- OEM Public key (OemPK): The public key of the manufacturer of the ECU firmware.
- Slave Public Keys (SlavePKs): The public keys of slave ECUs, in the case of the Master ECU.
- Master Public Key (MasterPK): The public key of the master ECU, in the case of Slave ECUs.

To provide a more formal notation of the previous key types, several symbols and notations are introduced. The following notation denotes a public-private key pair:

$$[K^X]_Y = (K^X_{pub}, K^Y_{prv}), X \in \{M, OEM, S_1, S_2, ...\}, Y \in \{R, S\}. \quad (1)$$

In the notation above, $[K^X]_Y$ denotes a pair of public-private key pairs, such that K^X_{pub} is the public component, and K^Y_{prv} is the private component. In the same definition, the value of X is replaced in particular cases with the appropriate symbol from the $\{M, OEM, S_1, S_2, ...\}$ set, while the value of Y is replaced with a symbol from set $\{R, S\}$.

According to this definition, the ECU Master is bootstrapped with the following keys: $[K^M]_R$ – the Master's SRK; $[K^M]_S$ – the Master's KSK; K^{OEM}_{pub} – the Master OEM's public key; $K^{S_1}_{pub}, K^{S_1}_{pub}$ – one or more SlavePKs according to the number of Slave ECUs associated to a Master ECU. Similarly, a Slave ECU is bootstrapped with the following keys: $[K^{S_1}]_R$ – the Slave's SRK; $[K^{S_1}]_S$ – the Slave's KSK; K^{OEM}_{pub} – the Slave OEM's public key; K^M_{pub} – the Master ECU's public key.

3.2 Secure Key Generation and Storage

Once the bootstrapping is completed, Master ECUs are the ones that generate the long-term and the short-term keys. The two key types are distinguished according to their cryptoperiod. The cryptoperiod denotes the time span during which a cryptographic key is authorized for use. According to NIST's "Recommendation For Key Management", long-term keys should be changed once every few weeks, while short-term keys (sometimes referred to as session keys), should be changed once every few days.

Keys need to be generated in a hardware-protected area that cannot be tampered. As it turns out, the TPM addresses this requirement, since it provides secure generation and storage of cryptographic keys. The cryptographic keys generated for this purpose are symmetric keys, that is, the keys used to provide data confidentiality, integrity, and authenticity with the help of encryption algorithms. It is imperative that the generated keys do not leave the TPM unencrypted. For this purpose, TPMs have a particular feature known as sealing. Sealing denotes the procedure of encrypting data (e.g., session keys) with the

TPM's SRK. Since the private part of SRK never leaves the TPM, it is practically impossible for someone to decrypt the session key without knowing the SRK's private key. As a result, the TPM can generate a large number of cryptographic keys that can be applied in data authentication, data integrity, and a wide variety of other scenarios. In the case a particular key is needed, the TPM can load, unseal the key, and apply it to enforce certain security properties.

3.3 Long-term Key Distribution Protocol

In order to reduce the overhead on the CAN-FD bus, the long-term key distribution protocol is defined as a single message protocol. The message is sent by the Master ECU to each Slave ECU. We call this protocol the Long-term Key Distribution Protocol and denote it by `Proto-LTK`:

$$\begin{aligned} ECU_{Master} \rightarrow ECU_Y : pid_{LTK}, kid_{LTK}, N, \{K\}_{K^Y_{pub}}, \\ \{pid_{LTK}, kid_{LTK}, N, \{K\}_{K^Y_{pub}}\}_{K^M_{prv}}, \end{aligned} \tag{2}$$

where $Y \in \{S_1, S_2, ...\}$. In `Proto-LTK`, pid_{LTK} denotes the protocol identifier, which needs to be unique for each type of key. This term is particularly useful in distinguishing between several different use cases, and key distribution scenarios, and, ultimately, to avoid the so-called multi-protocol attacks, where cryptographic terms from one protocol are replayed to other protocols. The second term kid_{LTK} is the key identifier, which can be implemented as a counter. The next term N is a random number used to enforce the freshness of the message, which is in accordance with AUTOSAR SecOc's recommendations regarding message freshness. Next, the term $\{K\}_{K^Y_{pub}}$ is the actual transmission of a freshly generated long-term key K encrypted with each of the target slave's public key. The last term is the digital signature of all prior terms, computed with the Master's private key K^M_{prv}. The purpose of this term is to enforce the authenticity (and integrity via authenticity) and non-repudiation security properties.

In case the message is missed by a target slave, the protocol `Proto-LTK` is extended with a challenge-response sequence, which ensures the explicit interrogation of the currently used long-term key:

$$\begin{aligned} ECU_{S_i} \rightarrow ECU_M : \ pid_{LTK}, N_S \\ ECU_M \rightarrow ECU_{S_i} : \ pid_{LTK}, kid_{LTK}, N_S, \\ \{K\}_{K^Y_{pub}}, \{pid_{LTK}, kid_{LTK}, N_S, \{K\}_{K^Y_{pub}}\}_{K^M_{prv}}. \end{aligned} \tag{3}$$

In this case, the slave ECU (ECU_{S_i}, where $i \in \{1, 2, ...\}$) initiates the execution of the protocol by sending a protocol identifier pid_{LTK}, and a freshly generated random number N_S. As a response to this request, the Master issues the encrypted and signed long-term key K.

3.4 Short-term Key Distribution Protocol

Once the long-term key (LTK) is installed, it is used to periodically distribute a new short-term key (STK). We call this protocol the Short-term Key Distribu-

tion Protocol (`Proto-STK`). For this purpose, a protocol similar to the one above is used:

$$ECU_{Master} \rightarrow Broadcast : pid_{STK}, kid_{STK}, N, \{k\}_K, \\ \{pid_{STK}, kid_{STK}, N, \{k\}_K\}_{K^M_{prv}}, \quad (4)$$

where pid_{STK} is the protocol identifier, k_{STK} is the key identifier, N is a freshness counter, and k is the newly generated short-term key. Compared to `Proto-STK`, the protocol exhibits two main differences: (i) `Proto-STK` uses a broadcast message, which means that a single message is issued for distributing the new short-term key k; and (ii) `Proto-STK` digitally signs the newly issued STK, which empowers Slave ECUs with the ability to verify the authenticity of the newly generated key. Once again, the protocol is extended with a challenge-response scheme in order to ensure the subsequent request of the current key:

$$ECU_{S_i} \rightarrow ECU_M : \ pid_{LTK}, N_S \\ ECU_M \rightarrow ECU_{S_i} : \ pid_{LTK}, kid_{LTK}, N_S, \\ \{K\}_{K^Y_{pub}}, \{pid_{LTK}, kid_{LTK}, N_S, \{K\}_{K^Y_{pub}}\}_{K^M_{prv}}. \quad (5)$$

4 Concurrency and Security Analysis

4.1 Concurrency Analysis

In order to assess the possible concurrency issues introduced by the developed scheme we use the TLA+ language [12]. TLA+ (Temporal Logic of Actions) is a formal specification language developed by Leslie Lamport. It is a language for constructing specifications, which can then be analyzed by model checking tools. The model checker can then explore all possible behavior, and find violations of desired properties. A language that builds on TLA+ is PlusCal, a formal specification language that can be translated to TLA+ specification. PlusCal was developed by Leslie Lamport to aid the specification of algorithms. While PlusCal resembles more of an imperative language, its expressions use the syntax of TLA+. There is also a translator available that translates PlusCal specifications to TLA+.

PlusCal adopts a wide range of statements from the 'C' programming language, making its learning curve less steep. We modeled the developed key distribution protocol by leveraging PlusCal, and we translated the specification to TLA+ by using the TLA+ toolbox. The specification presumes two entities, a Master and a Slave. The Master periodically issues a new session key, and uses the session key to periodically broadcast authenticated messages. The Slave updates its key, and uses the key with the identifier it receives in order to verify the authentication tag. As it turns out, TLA+ detected miss-synchronization between nodes. According to TLA+ output, certain nodes may miss a specific key update, and would therefore be unable to authenticate subsequent frames. Therefore, it is imperative that nodes have the ability to buffer a sequence of frames and leverage the challenge-response variants of the protocols mentioned earlier in order to synchronize keys.

4.2 Security Analysis

We proceed to the formal analysis of the proposed key distribution scheme with the help of the Scyther [11] tool. Scyther is a model checking tool designed under the perfect encryption assumption, which means that an adversary can only learn an encrypted value if he knows the decryption key. In Scyther, the adversary is assumed to have full control over the underlying communication network. It can eavesdrop messages, it can replay, split and concatenate messages, however, it can correctly encrypt and decrypt a message only if it is in the possession of the right key. Throughout our assessment we used *claim* events to analyze the *aliveness*, *non-injective agreement*, *non-injective synchronization*, and *commitment* security properties:

- *Aliveness*: is a weak form of authentication and it ensures to the protocol initiator I the aliveness of the protocol respondent R, if, after a complete run of the protocol by I, R also executed some protocol events (not necessarily in a recent time).
- *Non-injective agreement (Niagree)*: ensures to the protocol initiator I a non-injective agreement with the respondent R on free variables appearing in the protocol's specification, if whenever I completes a protocol run, then R has also been involved in executing protocol events with I. However, this property does not guarantee a one-to-one mapping of runs. As a result, participant I may complete two runs, while participant R may have taken part only in a single protocol run.
- *Non-injective synchronization (Nisynch)*: is a stronger form of authentication, which means that all messages sent/received by the initiator I have been indeed received/sent by the respondent R.
- *Commitment*: ensures that if whenever I completes a protocol run with a set of terms, then the respondent R also completed the protocol run with the same terms. The property is also useful when verifying a protocol against impersonation attacks.

As part of Scyther's modeling language, protocol participants are defined according to their roles (i.e., initiator, respondent). Message exchanges are modeled with *send* and *recv* events. There are predefined types for generating random numbers (i.e., a *Nonce* - Number once used), while time stamps and session keys are modeled as user defined types. The complete protocol model is illustrated in Figure 2. Here, the Master was modeled as the protocol's initiator, while a regular node was modeled as the protocol's respondent.

By leveraging the Scyther tool, the developed protocol was proven to be correct within the Dolev-Yao attacker model. More specifically, it was proven that the protocol ensures: (i) the synchronized agreement of roles with respect to runs; and (iii) the secrecy of the LTK K. Similarly to this protocol, the other variants have also been modeled with the help of the Scyther tool, and were found to be correct within the Dolev-Yao attacker model.

```
usertype ProtocolID;
usertype KeyID;
usertype SessionKey;
const pid: ProtocolID;
protocol DIAS-KEYDISTRO(I,R) {
        role I {
                fresh kid: KeyID;
                fresh Ni: Nonce;
                fresh K: SessionKey;
                send_1(I,R, pid, kid, Ni, {K}pk(R),
{pid, kid, Ni, {K}pk(R)}sk(I));
        }
        role R {
                var kid: KeyID;
                var Ni: Nonce;
                var K: SessionKey;
                recv_1(I,R, pid, kid, Ni, {K}pk(R),
{pid, kid, Ni, {K}pk(R)}sk(I));
                claim_R1(R,Secret,K);
                claim_R2(R,Nisynch);
        }
}
```

Fig. 2. The Key Distribution Protocol's description in Scyther's modeling language.

5 Implementation Details

A prototype of the proposed scheme has been implemented in the context of a laboratory testbed. The testbed comprised of an ECU implemented with the help of Raspberry Pi model 3B+ running the Automotive Grade Linux operating system. The ECU was equipped with an An MCP2515 CAN controller, with a TJA1050 CAN transceiver, and an Infineon OPTIGA SLB 9670 Trusted Platform Module. The TPM supports a wide range of cryptographic applications, while providing both symmetric and asymmetric cryptographic primitives. A wide variety of algorithms are supported, including Elliptic Curve Cryptography (ECC), EC Diffie-Helman, 1024 and 2046-bit RSA. The primitives for building the key distribution protocols and for communicating with the TPM were implemented in Python 3.5.

For the following analysis, we used 2048-bit RSA and 256-bit SHA, thus the most resource-intensive algorithms supported by the TPM. Accordingly, we tested the time for generating the message of `Proto-LTK`. As it turns out, the measured time for generating a new symmetric key K of 32 bytes, the sealing of the message, including generating signature was of 2.11 seconds. Indeed, this is an intensive operation, however, this is also a key advantage of offloading these time-intensive operations to a cryptographic co-processor. An important observation is that, in the case of `Proto-LTK`, a new message needs to be generated for each ECU involved in the communication. However, since this needs to be performed only once every few weeks (as mentioned earlier), we consider that the added value of leveraging state of the art cryptographic constructions outweigh the computational downside. Conversely, `Proto-STK` broadcasts a single

message to all involved ECUs, which yields a computational effort for signing only one message. In terms of message dimension, the following implementation variant is proposed for `Proto-LTK`. Since pid_{LTK} could be used to distinguish between different groups of ECUs using the same protocol, a single byte would be sufficient for this element. kid_{LTK} and N can be used together as freshness as well as denoting the identifier of the newly issued key. Accordingly, two bytes for kid_{LTK} would permit releasing a new LTK every week for more than one thousand years. Subsequently, the value of N needs to be sized according to the number of ECUs. Considering the number of ECUs in today's and future vehicles, the size of N can be of one byte. In case more than 256 ECUs are present, we assume there would be certain partitioning of ECUs in several networks/groups, where pid_{LTK} could be used to distinguish between different groups. The signature with RSA-2048 is of 256 bytes, while the encryption is also of 256 bytes.

Overall, the number of bytes that need to be sent by one run of `Proto-LTK` is of 516 bytes. This value is identical to the number of bytes that are generated by `Proto-STK` (assuming the same ciphers are used). This translates to 9 frames to be sent on CAN-FD. We observe, however, that the maximum supported data rate of CAN-FD is of 8Mbit/s. Therefore, a single run of `Proto-LTK` would consume aprox. 4Kbits, that is aprox. 0.05% of the available bandwidth. Considering that the modern vehicle contains dozens of ECUs, for example in case of 100 ECUs, each ECU receiving a new STK once every day, this would overall consume 5% of the vehicle's bandwidth once every day. Therefore, the overall impact on the vehicle's communication can be seen as negligible, at least from a daily consumption point of view.

Table 1. Implementation size for RSA 2048.

	pid	kid	N	$\{K\}_{K_{pub}}$	$Signature$
Byte count	1	2	1	256	256
TOTAL	516 Bytes				

6 Conclusions

We developed a novel scheme for distributing cryptographic keys in modern vehicles. The scheme includes two protocols, namely: `Proto-LTK` for distributing long-term keys, and `Proto-STK` for distributing short-term keys. Both protocols are supoprted by synchronization constructions such that ECUs may synchronize and actively interrogate the current key. In order to ensure feasibility of integration, the scheme adopted the TPM standard. By using an external cryptographic co-processor, several advantages are provided: (i) the TPM provides a tamper-proof approach for storing and manipulating cryptographic keys; and (ii) the TPM can off-load the computation-intensive operations, and thus empower

ECUs with the possibility to use modern cryptographic algorithms. Implementation details and analysis have shown that the impact on the used bandwidth, in the case of CAN-FD is of less than 5%. As future work we intend to develop a full working prototype and assess the performance of the protocol in the context of inter-ECU communications.

Acknowledgement

This work was funded by the European Union's Horizon 2020 Research and Innovation Programme through DIAS project (*https://dias-project.com/*) under Grant Agreement No. 814951. This document reflects only the author's view and the Agency is not responsible for any use that may be made of the information it contains.

References

1. Coppola, R., Morisio, M.: Connected car: technologies, issues, future trends. ACM Comput. Surv. 49, no. 3, 46:1–46:36 (2016).
2. Urquhart, C., Bellekens, X., Tachtatzis, C., Atkinson, R., Hindy, H., Seeam, A.: Cyber-security internals of a Skoda Octavia vRS: a hands on approach. IEEE Access 7, 146057-146069 (2019). doi:10.1109/ACCESS.2019.2943837
3. Van Herrewege, A., Singelee, D., Verbauwhede, I.: CANAuth - A Simple, Backward Compatible Broadcast Authentication Protocol for CAN bus. ECRYPT Workshop on Lightweight Cryptography 2011, 1–7 (2011).
4. AUTOSAR: Specification of Secure Onboard Communication AUTOSAR CP Release 4.3.1. AUTOSAR (2017).
5. Trusted Computing Group: Trusted Platform Module Library Specification, Family "2.0", Level 00, Revision 01.59 (2019).
6. Ziermann, T., Wildermann, S., Teich, J.: CAN+: A new backward-compatible Controller Area Network (CAN) protocol with up to 16× higher data rates. Design, Automation Test in Europe Conference Exhibition, 1088–1093 (2009). doi: 10.1109/DATE.2009.5090826
7. Robert Bosch Gmbh: CAN with flexible data-rate. Vector CANtech, Inc., MI, USA, Specification Version 1.0 (2012).
8. Woo, S., Jo, H.J., Kim, I.S., Lee, D.H.: A practical security architecture for in-vehicle CAN-FD. IEEE Transactions on Intelligent Transportation Systems 17, no. 8, 2248–2261 (2016). doi:10.1109/TITS.2016.2519464
9. Wang, L., Liu, X.: NOTSA: Novel OBU With three-level security architecture for internet of vehicles. IEEE Internet of Things Journal 5, no. 5, 3548–3558 (2018). doi:10.1109/JIOT.2018.2800281
10. Groza, B., Murvay, P.S.: Identity-based key exchange on in-vehicle networks: CAN-FD & FlexRay. Sensors 19, no. 22 (2019). doi:10.3390/s19224919
11. Cremers, C.J.F.: The Scyther Tool: verification, falsification, and analysis of security protocols. Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, USA, Proc., 414–418 (2008).
12. Lamport, L.: Specifying Concurrent Systems with TLA+, 1999.