Lightweight Authentication for Secure Automotive Networks

Philipp Mundhenk[1], Sebastian Steinhorst[1], Martin Lukasiewycz[1],
Suhaib A. Fahmy[2], Samarjit Chakraborty[3]

[1] TUM CREATE, Singapore, Email: <firstname.lastname>@tum-create.edu.sg
[2] School of Computer Engineering, Nanyang Technological University, Singapore, Email: sfahmy@ntu.edu.sg
[3] TU Munich, Germany, Email: samarjit@tum.de

***Abstract*—We propose a framework to bridge the gap between secure authentication in automotive networks and on the internet. Our proposed framework allows runtime key exchanges with minimal overhead for resource-constrained in-vehicle networks. It combines symmetric and asymmetric cryptography to establish secure communication and enable secure updates of keys and software throughout the lifetime of the vehicle. For this purpose, we tailor authentication protocols for devices and authorization protocols for streams to the automotive domain. As a result, our framework natively supports multicast and broadcast communication. We show that our lightweight framework is able to initiate secure message streams fast enough to meet the real-time requirements of automotive networks.**

I. Introduction and Related Work

The rapidly increasing connectedness of modern cars leads to new challenges in the security of inter- and intra-vehicle communication. As increasing numbers of vehicles are being connected to the outside world, the exposure risk of safety-critical systems rises significantly. With the multitude of communication interfaces, it is very difficult, or even impossible, to reliably control entry points into the vehicle or shield the vehicular network with firewalls. Hence, it is important that internal communication within a vehicle is secured. The impact of a malicious attack on automotive safety-critical systems can be devastating, including high financial damages and even loss of life. To mitigate the effects of such attacks, protection mechanisms have been developed to limit unauthorized access to in-vehicle communication and minimize the number of attack vectors.

Currently, most internal communication in vehicles is insecure. Encryption is rare, and if available often uses similar keys across a series of vehicles and *Electronic Control Units (ECUs)*. The first extensive overviews of security in modern networked vehicles were presented in [1] and [2], where it was shown that ECUs could be attacked and reprogrammed directly, by obtaining pre-programmed security keys from the car tuning community.

Initial efforts to introduce encryption into real-time vehicular networks have been presented in the literature. To unify protection efforts, the Hersteller Initiative Software (HIS) has specified a cryptographic accelerator, called *Secure Hardware Extension (SHE)* for use in vehicles [3]. To limit the latency impact on the real-time communication, symmetric cryptography has been chosen in SHE and elsewhere in the literature, since it is computationally simpler. Symmetric encryption requires that secret keys are known to all participants of a protected communication. This opens a new attack vector, as keys are often pre-programmed into ECUs and valid for the lifetime of the ECU [1].

One approach to employing symmetric cryptography and limiting the overhead of additional security in legacy communication systems is to use *Message Authentication Codes (MACs)*. In [4], MACs are introduced to *Controller Area Networks (CANs)* and safety and security are considered in an integrated functional model mapped to a CAN architecture. In [5], an approach to introduce MACs into FlexRay is presented. The *Timed Efficient Stream Loss-Tolerant Authentication (TESLA)* protocol is employed for time-delayed release of keys in the time-triggered segment of FlexRay. While TESLA supports asymmetric behavior with symmetric mechanisms, it does not authenticate communication participants or authorize communication streams. In [6], a *Cipher-based Message Authentication Code (CMAC)* is utilized and integrated with the Cyclic Redundancy Check (CRC) of the underlying communication system, reducing the overhead and achieving the required sender authentication and message integrity checks. To be able to ensure real-time behavior, all of these approaches employ symmetric cryptography, requiring a pre-shared key.

This work was financially supported in part by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) programme.

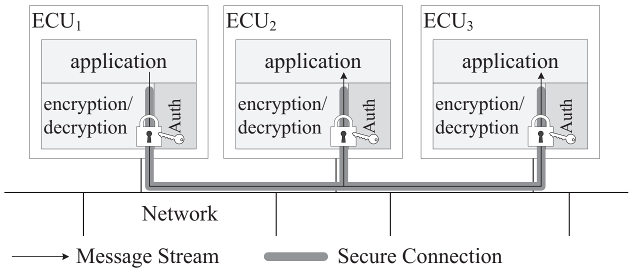

***Fig. 1:** The proposed approach combines symmetric and asymmetric cryptography efficiently. Initially, a secure communication channel is established, relying on an authentication mechanism with asymmetric encryption for key exchange. The real-time messages are then transferred efficiently, using symmetric encryption with the shared keys.*

To reduce the risk from pre-shared keys, authentication frameworks have been developed in other domains, such as corporate networks and the internet, allowing the exchange of keys without prior interaction of the communication participants. Examples include *Kerberos* [7] and the widely used *Secure Sockets Layer (SSL)/Transport Layer Security (TLS)* framework [8].

Contributions. We propose a lightweight authentication framework bridging the gap between secure authentication in automotive networks and on the internet. With our proposed framework, the keys required for secure communication with symmetric cryptography, such as the *Advanced Encryption Standard (AES)* or CMACs, can be exchanged between ECUs (see Fig. 1). To ensure the security of our framework, we base it on proven frameworks and adapt these to the specific requirements of the automotive domain (see Section II). Our framework consists of two phases: (1) ECU authentication and (2) stream authorization. In the first phase, each ECU authenticates against a central security module. This is performed when the vehicle is not operating and real-time behavior is of limited importance. In the second phase, during vehicle operation, every message stream is authorized and symmetric keys for stream access are distributed to ECUs.

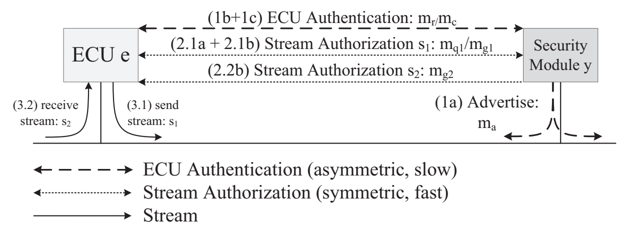
(1b+1c) ECU Authentication: m_r/m_c
ECU e
(2.1a + 2.1b) Stream Authorization s_1: m_{q1}/m_{g1}
(2.2b) Stream Authorization s_2: m_{g2}
Security Module y
(3.2) receive stream: s_2
(3.1) send stream: s_1
(1a) Advertise: m_a
ECU Authentication (asymmetric, slow)
Stream Authorization (symmetric, fast)
Stream

Fig. 2: *In our framework, every ECU is authenticated against the security module (1a-1c). Subsequently, the ECU can request the keys for a message stream (2.1a-2.1b) and start transmitting (3.1). If the ECU is to start receiving a message stream, it is notified by the security module with the message stream key (2.2b) before it receives the stream (3.2).*

Our lightweight framework allows us to combine the flexibility of asymmetric security for ECU authentication with the speed of symmetric frameworks for message stream authorization. A central security module controls access to the network and message streams for all ECUs on the network. As the security module administers all message stream keys in the network, multicast and broadcast communication can be established easily. The detailed design of our lightweight authentication framework is presented in Section III. We evaluate the performance of our framework in Section IV.

II. Authentication Framework

This section details the requirements of authentication for in-vehicle networks and evaluates existing frameworks, before proposing a lightweight framework for automotive real-time systems. While our proposed framework is optimized for the automotive domain, it can be adapted to other distributed real-time systems, too.

Scenario and existing authentication mechanisms. Most ECUs in vehicles have very limited memory and computational power. To allow these devices to communicate securely, fast and efficient algorithms are required. Symmetric cryptography is far less computationally demanding than asymmetric cryptography, and should hence be the method of choice. However, these algorithms have the downside of requiring pre-shared keys. For most approaches in the literature and industry, these keys are programmed into ECUs during manufacturing. This often leads to identical keys being used across a series of ECUs and vehicles [1]. This is a significant risk, as once a key is known, a whole series of vehicles might be susceptible to attacks. Thus, pre-programmed keys offer only limited security.

Multiple key exchange mechanisms have been proposed, optimized for different domains. To remedy the trade-off between symmetric and asymmetric cryptography, TESLA was introduced to mimic asymmetric behavior with symmetric cryptography [9]. In contrast to TESLA, *Transport Layer Security (TLS)* allows the sender and receiver to be authenticated using asymmetric cryptography. *Kerberos* is an authentication framework built on symmetric cryptography [7].

Proposed Approach. We propose to use a simplified version of the two-phase setup in Kerberos adapted to the automotive context [10]. Kerberos is a highly flexible protocol. By limiting this flexibility to a minimal set for real-time systems and adjusting the underlying structure to the capabilities of automotive networks, we can drastically reduce the amount of messages and data to be transmitted. We can reduce message exchanges by combining the *Authentication Server (AS)* and *Ticket Granting Server (TGS)* into one system: the security module.

We use the two-phase authentication of Kerberos as shown in Fig. 2:

1) ECU authentication (asymmetric)
2) Stream authorization (symmetric)

We propose to employ the asymmetric phase of the framework for ECU authentication. This can be performed periodically, while the vehicle is not in operation. The interval depends on the strength of the algorithm used for symmetric cryptography. The symmetric phase of our framework is used for stream authorization and is performed as required. By receiving the key to a message stream from the security module, an ECU is granted the right to access the stream and communicate with other participants in the network. Depending on the real-time requirements of the stream, this step can be performed either at vehicle start-up, or when the setup of a message stream is required.

III. Design of the Lightweight Authentication Framework

In this section, we introduce the ECU authentication and stream authorization approaches in detail and analyze the latencies introduced by incorporating them in a vehicle network.

Terminology. We denote an ECU as e, sending a set of streams S_e to a set of receiving ECUs R_e. In turn, a stream $s \in S_e$ is identified by the sending ECU e, a set of receiving ECUs R_s and a set of message instances M_s: $s = (e, R_s, M_s)$. A message instance $m \in M_s$ is assigned to a stream s and carries a payload o. The security module of the system is denoted as y. A timestamp on device d is denoted as ω_d and a random number as ρ. A nonce n is a unique random number ρ which has not been used before. A key is denoted as k and identified by its value v and its length l. We further define a hashing function for message m on device d as $h_d(m)$. A parameter to determine if the system is running is defined as $\theta \in \{0, 1\}$ and a function $\alpha(e, s) \in \{0, 1\}$ determines if an ECU e has access to the stream s (1) or not (0). The time required to execute a function x is defined as τ_x. An action z is triggered, if a condition v is fulfilled: $v \rightarrow z$.

A. ECU Authentication

First, we define the authentication mechanism for ECUs. This mechanism facilitates the initial key exchange between ECU and the security module, and is based on asymmetric cryptography.

Authentication Mechanism. To describe ECU authentication, we define a set of keys for device d, consisting of a public key $k_{d,\text{pub}}$ and a private key $k_{d,\text{priv}}$. We further define an asymmetric encryption function ϵ^a for device d, that translates clear text t into cipher text c with key k, such that $c = \epsilon^a_d(t, k)$. Accordingly, we define an asymmetric decryption function δ^a for device d, that translates cipher text c into clear text t with key k, such that $t = \epsilon^a_d(c, k)$. For authentication, we further require the certificate f_d for device d and a function $\phi_{d_1}(f_{d_2}) = \{0, 1\}$ to verify the certificate f_{d_2} on device d_1.

As shown in Fig. 2, three steps are required to authenticate an ECU:

(a) Advertisement by the security module (Fig. 2 (1a)): The security module y advertises its certificate to every ECU on the bus at startup with message m_a:

$$\theta \rightarrow m_a \text{ with } m_a \in M^a_s,\ s = (y, \{e\}, M^a_s),\ o = f_y \tag{1}$$

(b) Authentication of security module (Fig. 2 (1b)): The authentication of the security module y is achieved by presenting a valid security certificate f_y to ECU e. The certificate (including the public-key of the security module) is broadcast on the network unencrypted. Each ECU has a pre-programmed list with the certificates of the intermediary *Certificate Authorities (CAs)* and the root CA. With this data, the certificate of the security module can be verified. It is to be noted that the programmed list should be regularly updated with a *Certificate Revocation List (CRL)* (see Section V). On successful verification of the security module ($\phi_e(f_y) = 1$), the ECU transmits its registration message m_r, including its own certificate f_e and the ECU key $k_{e,\text{sym}}$. The key, a timestamp ω_e and a nonce n are encrypted with the public key of the security module $k_{y,\text{pub}}$:

$$\begin{aligned} &\phi_e(f_y) \rightarrow m_r \\ &\text{with } m_r \in M^r_s,\ s = (e, \{y\}, M^r_s), \\ &\qquad o = (\epsilon^a_e((k_{e,\text{sym}}, n, \omega_e), k_{y,\text{pub}}), f_e) \end{aligned} \tag{2}$$

(c) Authentication of ECU (Fig. 2 (1c)): Upon receiving the registration message, the security module verifies the ECU certificate f_e. In the case of successful verification ($\phi_y(f_e) = 1$), the security module saves the ECU key and sends a confirmation message m_c

to the ECU. This message contains the ECU identifier, encrypted with the ECU key:

$$\phi_y(f_e) \rightarrow m_c \tag{3}$$
$$\text{with } m_c \in M_s^c,\ s=(y,\{e\},M_s^c),\ o=\epsilon_y^s((e,n,\omega_y),k_{e,\text{sym}})$$

Latency. The latency τ_{setup} introduced by ECU authentication is only of limited relevance, as this is performed at times when the car is not in operation, such as in a factory or workshop. However, for completeness, we can define the latency based on the time τ_x required for the functions in Equations (1), (2) and (3), summing up to the full latency of:

$$\tau_{\text{setup}} = \tau_{\phi_e} + \tau_n + \tau_{\epsilon_e^a(t,k_{\text{pub}})} + \tau_{\phi_y} + 3\cdot\tau_{tx} \tag{4}$$

This setup time omits minor latencies, such as message passing within the ECU and security module.

B. Stream Authorization

After ECUs authenticate themselves with the security module and set symmetric ECU keys, the ECUs can request initiation of message streams. All message streams and stream keys in the system are administered by the security module. Sending and receiving ECUs need to request the key for each message stream from the security module. This module authorizes access to message streams; authentication between sending and receiving ECUs for a stream is not required.

Authorization Mechanism. For stream authorization, we define a symmetric key k_{sym} and two functions ϵ^s and δ^s for encryption and decryption, such that $c=\epsilon_d^s(t,k_{\text{sym}})$ and $t=\delta_d^s(c,k_{\text{sym}})$ for clear text t and cipher text c on device d.

The stream can only be established after the ECU receives the confirmation message m_c for ECU authentication (see Fig. 2 (1c)). It is usually established when a message m is requested to be sent. The sending ECU e can then request a key from the security module y, allowing access to the stream. The request message m_q contains the identifier of the requested stream s_1, the identifier of the requesting ECU e, as well as a timestamp ω_e and a nonce n to protect against retransmissions. The content of the message m_q is encrypted with the ECU key $k_{e,\text{sym}}$:

$$\begin{aligned} & m_c \wedge m \rightarrow m_q \\ & \text{with } m_q \in M_s^q,\ s=(e,\{y\},M_s^q), \\ & \quad o=\epsilon_e^s((e,s_1,n,\omega_e),k_{e,\text{sym}}) \end{aligned} \tag{5}$$

If the received request message m_q can be successfully decrypted with the ECU key $k_{e,\text{sym}}$ and the requesting ECU e is allowed access to the stream s_1 ($\alpha(e,s_1)=1$), the security module y assigns a key $k_{s_1,\text{sym}}$ to the stream. This key is forwarded to all receiving ECUs $\tilde{e}\in R_{s_1}$, as well as the requesting sending ECU e. This way, it is ensured that all receivers have access to the data as soon as it is sent:

$$\begin{aligned} & m_q \wedge (\delta_y^s(m_q,k_{e,\text{sym}})=(e,s_1,\omega_e,n)) \\ & \wedge(\omega_e \le \omega_y)\wedge\alpha(e,s_1)\rightarrow \forall\tilde{e}\in\{R_{s_1},e\}: m_g \\ & \text{with } m_g\in M_s^g,\ s=(y,\tilde{e},M_s^g), \\ & \quad o=\epsilon_y^s(\tilde{e},s_1,k_{s1,\text{sym}},n_i),k_{\tilde{e},\text{sym}}), \\ & n_i=\rho:\rho\notin\bigcup_l \rho_l, l=\{0..i-1\} \end{aligned} \tag{6}$$

Latency. Similarly to the latency for ECU authentication, we can calculate the upper bound of the latency for stream authorization by adding the latencies of all functions required in Equations (5) and (6). In the case of x ECUs participating in the stream, the latency can be defined as:

$$\begin{aligned} \tau_{\text{stream}} = {} & (x+1)\cdot\tau_{\epsilon_e^s(t,k_{\text{sym}})} + 2\cdot\tau_{\delta_e^s(t,k_{\text{sym}})} \\ & +(x+1)\cdot\tau_n + (x+1)\cdot\tau_{tx} \end{aligned} \tag{7}$$

All values are assumed to be equal across all ECUs and the security module, as hardware support for random number generators and symmetric encryption/decryption is assumed.

IV. Experimental Results

In this section, we analyze and evaluate the characteristics and performance of our proposed approach. The performance comparisons in this section are based on the maximum latencies

key length (Bits)	public key operations $\tau_{\epsilon^a(t,k_{pub})}$, $\tau_{\delta^a(t,k_{pub})}$	private key operations $\tau_{\delta^a(t,k_{priv})}$, $\tau_{\epsilon^a(t,k_{priv})}$
512	0.206s	0.886s
1024	0.709s	4.977s
2048	2.626s	33.181s

TABLE I: *Latencies of RSA encryption and decryption in software for different RSA key lengths, measured on an STM32 microcontroller.*

introduced by adding cryptography into the network, as described in Section III. We show that our approach can be efficiently implemented on embedded systems, through measuring the times required for symmetric and asymmetric cryptography on a microcontroller.

The performance of our approach is influenced by many factors, such as:

- cryptographic algorithms for ECU authentication and stream authorization,
- implementation of cryptographic algorithms (in software or hardware),
- key lengths for ECU authentication and stream authorization,
- frequency of ECU key updates,
- latency of ECU authentication and stream authorization.

In a first evaluation, we will approximate the latencies for ECU authentication and stream authorization based on measurements obtained for cryptographic performance. We fix the cryptographic algorithms to *RSA* for ECU authentication (in software) and *AES Cipher-Block Chaining (CBC)* (in software and hardware) for stream authorization. We evaluate these algorithms with multiple different key lengths.

Platform. The bottleneck in our proposed framework is the implementation of the relatively demanding cryptographic algorithms on the ECUs. For our evaluations, we use an *STMicroelectronics STM32F415RG* controller with a 32 bit ARM Cortex-M4 core clocked at 16 MHz, referred to as STM32. This controller contains a cryptographic accelerator and is similar to those commonly found in vehicles.

Random numbers have been generated in hardware in all testcases, using the built-in Random Number Generator (RNG) of the STM32.

A. ECU Authentication

The mechanism for authentication of ECUs against the security module is performed using asymmetric cryptography. We measure the required time for asymmetric encryption and decryption on the microcontroller to approximate the time required for initial ECU authentication, according to our proposed solution. As hardware support for asymmetric cryptography cannot generally be assumed for smaller microcontrollers, as is the case for the STM32, we limit our evaluation to software implementations. The *CyaSSL* library is used to encrypt and decrypt messages of different sizes (1 byte up to the length of the used key) with keys of different lengths in RSA. The results are shown in Table I. Note that RSA is typically used to encrypt messages up to the size of one key length. As used in our proposed approach, this data is often a symmetric key (see Equation (2)), which is in turn used to encrypt the actual data to be transmitted. Due to the nature of the RSA algorithm, the length of the input data has no influence on computation time, and it is thus omitted in Table I. To ensure reasonable security, a key length of at least 1024 Bytes is required. A key length of 2048 Bytes or longer is preferable, shorter keys can not be considered secure and are only given for reference. As shown in Table I, reasonable key lengths require encryption/decryption times at in the seconds when executed in software on low performance ECUs.

Based on the measurements in Table I and Equation (4), we can define $\tau_{\phi_e}\approx 2.626$s, $\tau_n = 0.83\mu$s and $\tau_{\epsilon_e^a(t,k_{\text{pub}})} = 2.626$s and approximate the setup time for a system with 2048 Bit key to be: $\tau_{\text{setup}}\approx 5.3$s. As hardware encryption is assumed in the security module, the verification of a certificate and the encryption on the ECU require significantly more time than all other parameters

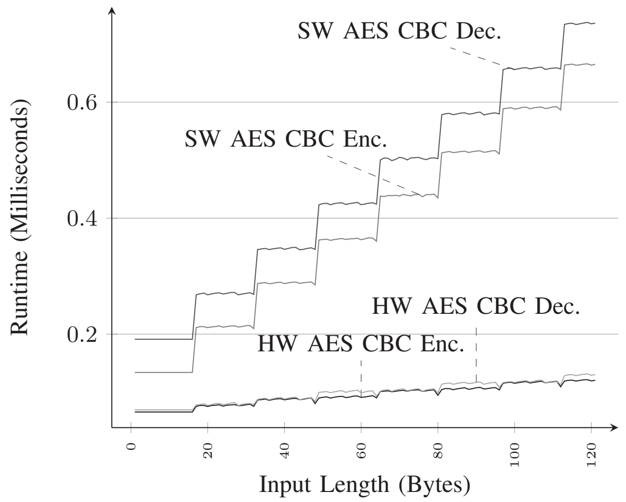

Fig. 3: *AES performance for software implementations (SW) and with hardware support (HW). The measurements of encryptions (Enc.) and decryptions (Dec.) have been performed on an STM32 microcontroller with AES in Cipher-Block Chaining (AES CBC) mode.*

$\tau_{h(f)}, \tau_{comp}, \tau_{\phi_{sm}}, 3 \cdot \tau_{tx}$. It is further assumed that the CRLs are locally available on the ECU and do not need to be fetched, and so, these values have been omitted.

As expected, this is a considerable latency introduced into the communication. Thus, we conclude that it would not be feasible to introduce asymmetric cryptography mechanisms in the message stream itself, but rather in the first phase of authentication. Since ECU authentication is only performed when the vehicle is not operating and no real-time performance is required, this latency is acceptable.

B. Stream Authorization

As described in Section II, stream authorization is performed with symmetric keys. To approximate the performance of the proposed approach, we measure the performance of symmetric encryption on the STM32 microcontroller. We use the *STM32 Cryptographic Library*.

The results of our measurements are shown in Fig. 3 for AES CBC in hardware and software. It is important to note that due to the structure of AES, the length of the key does not have any mentionable influence on the duration of the encryption/decryption. From Fig. 3, it is clear that hardware performance clearly dominates software performance, even for short messages. Consequently, to ensure minimal latencies, hardware encryption is required for stream setup and message encryption.

In the following, we approximate the latency introduced by stream authorization, based on Equation (5) and Fig. 3. As the latencies are highly dependent on the throughput of the underlying communication system, we approximate the latency for different communication systems in Table II. Note that these approximations are based on the net data rate of the different bus systems and the medium access schemes can influence this timing drastically if the required messages have to be queued.

As shown in Table II, the available data rate of the bus influences the latency of stream authorization significantly. With faster buses, such as Automotive Ethernet, the impact of our scheme is reduced to cryptographic latency. The cryptographic latency in our case is 0.35 ms. Even for high-speed CAN, the impact of our approach is small, because we have minimized the required communication. This is especially beneficial in low-cost networks. Low-speed CAN may not be suitable for secured traffic, but high-speed CAN or CAN FD can tolerate our scheme well. For all communication systems from high-speed CAN onwards, it is possible to use our lightweight authentication framework and set up real-time message streams with minimal latency.

Communication System	**gross data rate (MBit/s)**	**net data rate (MBit/s)**	**stream auth. lat. $\tau_{\mathtt{stream}}$ (ms)**	**crypto/network lat. ratio (%)**
CAN Low-Speed (LS)	0.1	≤ 0.06	≥ 2.43	14.4
CAN High-Speed (HS)	0.5	≤ 0.29	≥ 0.77	45.8
CAN FD (automotive)	2.5	≤ 2.37	≥ 0.41	86.4
FlexRay	10	≤ 9.69	≥ 0.37	96.5
Automotive Ethernet	100	≤ 97.5	≥ 0.35	99.4

TABLE II: *Comparison of latencies for stream authorization on different communication systems, based on 2 ECUs with HW support for symmetric cryptography and AES-256 encryption/decryption.*

C. Discussion

By separating the ECU authentication and stream authorization and thus the asymmetric and symmetric authentication steps, we are able to achieve a significant speed-up, allowing the negotiation of keys in a highly efficient manner. Our proposed approach also works efficiently for bus systems such as high-speed CAN. By reducing the cryptographic latency down to 0.35 ms (see Table II), we can initiate message streams with very short latency, allowing applications to exchange symmetric keys and secure their entire communication efficiently.

V. Conclusion

We have proposed a lightweight authentication framework for real-time automotive systems, enabling the secure and efficient distribution of symmetric cryptographic keys among ECUs without pre-shared secrets. Our framework combines symmetric and asymmetric cryptographic methods to implement a two-phase authentication of ECUs and message streams. While our framework is optimized for automotive networks, it can be applied to any real-time system requiring security. We have evaluated the performance based on measurements of the underlying cryptographic methods on cryptographically accelerated hardware and compared the performance of our approach to existing authentication frameworks. Our framework proves to be highly efficient with the ability to set up real-time message streams.

Future work. Our future work includes investigations into the handling of CRLs and the scheduling of message stream authorization in high-utility situations, such as the start of vehicle operation, among others.

References

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental security analysis of a modern automobile. In *Proc. of the 31st IEEE Symposium on Security and Privacy (SP)*, 2010.

[2] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *Proc. of the 20th USENIX conference on Security*, 2011.

[3] R. Escherich, I. Ledendecker, C. Schmal, B. Kuhls, C. Grothe, and F. Scharberth. *SHE – Secure Hardware Extension Functional Specification*. 2009.

[4] C.-W. Lin, Q. Zhu, C. Phung, and A. Sangiovanni-Vincentelli. Security-aware mapping for CAN-based real-time distributed automotive systems. In *Proc. of ICCAD*, 2013.

[5] G. Han, H. Zeng, Y. Li, and W. Dou. SAFE: Security-Aware FlexRay scheduling engine. In *Proc. of DATE*, 2014.

[6] R. Zalman and A. Mayer. A secure but still safe and low cost automotive communication technique. In *Proc. of DAC*, 2014.

[7] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. *The Kerberos Network Authentication Service (V5)*. Number 4120 in Request for Comments. IETF, 2005.

[8] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. Number 5246 in Request for Comments. IETF, 2008.

[9] A. Perrig, D. Song, R. Canetti, J. D. Tygar, and B. Briscoe. *Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction*. Number 4082 in Request for Comments. IETF, 2005.

[10] L. Zhu and B. Tung. *Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)*. Number 4556 in Request for Comments. IETF, 2006.