An Improved Needham-Schroeder Session Key Distribution Protocol for In-Vehicle CAN Network

Yin Long, Jian Xu(✉), Chen Wang, and Zihao Wang

Software College, Northeastern University, Shenyang, China
xuj@mail.neu.edu.cn

Abstract. With the rapid development of automobile technology, the internal network of automobiles is facing more and more security problems. Many CAN-based in-vehicle applications lack security mechanisms for data confidentiality and secure session key distribution. To address the above problems, we propose an improved Needham-Schroeder session key distribution protocol suitable for in-vehicle CAN network, using message authentication code (MAC), key derivation function, digital signature, and timestamping mechanisms to resolve the defect that NSSK lacks resistance to Denning-Sacco attack. We use a random oracle model to conduct a formal security analysis of the proposed protocol, then we use Tamarin-Prover to verify the security properties of the protocol, the result indicates that the protocol is secure and applicable for in-vehicle CAN communication.

Keywords: CAN security · Security protocols · Vehicle cybersecurity · The NSSK protocol

1 Introduction

With the vigorous development of the Internet of Things, edge computing, and other contemporary technologies, more and more devices in traditional local area networks are beginning to connect to the mobile Internet. Many tasks that originally need to be performed by a single device can be deployed on the cloud now. This trend has gradually spread to the field of automobile manufacturing. The integration of technologies such as V2V and autonomous driving has driven the development of in-vehicle networks and smart car communication technologies. However, as the number of smart cars connected to the Internet increases, a series of problems related to the security of in-vehicle communication have arisen. For example, when a car system connects to the Internet via an insecure channel and communicates with other compromised cars or roadside units, a potential attacker will launch an attack on it through an exposed service of the system. If the attacker can find the vulnerabilities of the system and implant a backdoor program to monitor and analyze the data frames sent to the car's in-vehicle network (most of these data frames are non-encrypted), then he can guess the content of the protocol between the ECUs such as the engine speed or the direction of driving, then he forges

W. Shi et al. (Eds.): SPNCE 2021, LNICST 423, pp. 35–52, 2022.
https://doi.org/10.1007/978-3-030-96791-8_3

the malicious control messages and executes the attack by sending these messages to the in-vehicle network, which may lead to very dangerous driving behavior and even threaten the personal safety of passengers.

The data transmission protocols used for the in-vehicle network such as CAN and LIN have many security problems. Mainly include but are not limited to the following points: 1. Lack of access control. Since most ECUs use fixed keys provided by vehicle manufacturers for encryption or authentication, it is easy to be intercepted by malicious attackers, and illegal access requests are initiated to read the internal storage of the ECU, and the communication interface between the vehicle and external devices is used as a springboard to launch attacks on externally connected devices. 2. Broadcast communication mechanism. CAN bus communication is based on the broadcast mechanism, and any ECU in the vehicle network can send and receive data frames. Therefore, the attacker can analyze the data frame type and content by eavesdropping on the data frame message in the channel and achieve the purpose of the attack by sending the tampered data frames to the CAN bus network. 3. Lack of an effective authentication mechanism. There is no authentication mechanism in the CAN bus protocol data frame structure. A malicious node can pretend to be a legitimate node to send an attack message, while a node receiving a spoofed message cannot distinguish whether the message is from the legitimate node or the attacker. 4. For non-encrypted data frames, ECUs in the car communicate in plain text, which lacks information confidentiality protection, and it is difficult to avoid information leakage.

1.1 Related Works

To improve the security of the CAN network, some researchers have proposed many CAN-based secure communication schemes and key agreement protocols. Woo et al. [1] have shown that attacks may be launched externally by using a wireless channel to penetrate an IVN. Palaniswamy B et al. [2] provided a protocol suite for entity authentication, key management, a secure message flow for remote transmission request frames and session key update to be applied for vehicle connection with external devices. Wu et al. [3] proposed a key management scheme for in-vehicle multi-layer electronic control units (ECUs). Pullen D [4] uses implicit certificates to derive authenticated message-based group keys for ECUs. Pan Q [5] proposed a dynamic key generation algorithm to ensure the safety of CAN bus transmission and reduce the complexity of the system. Youn T Y [6] suggested an efficient key management scheme causing no communication overhead in a session key update process. Jain S [7] utilized the physical properties of the CAN bus to generate group keys. Lee H [8] showed the possibilities of triggering unprotected remote frames in CAN 2.0B, which results in a DoS attack. X Ying [9, 10] deployed an intrusion detection system TACAN a transmitter authentication in CAN by exploiting the covert channels in IVN and proposed a new masquerading attack called cloaking attack and formal analysis for clock skew based intrusion detection system. Wang Q [11] proposes a golden entropy model based on CAN ID entropy to detect various injection attacks on the CAN bus. King Z [12] used the HMAC algorithm and timestamp to generate digest and encrypt data, which is effective against denial of service attacks or repeated attacks. Fassak S [13] used Elliptic Curve Cryptography (ECC) to

establish a session key in automotive CAN networks, which is then used to derive symmetric keys for authenticating CAN frames. S. Nürnberger [14] proposed a framework for embedded controllers connected to the CAN bus, which allows both senders and receivers to authenticate exchanged data. A. I. Radu [15] proposed Leia, a lightweight authentication protocol for the Controller Area Network (CAN), which allows critical vehicle ECUs to authenticate each other providing compartmentalization and preventing some attacks.

1.2 Our Goal and Contribution

Owing to the communication in the CAN network has the characteristic of broadcasting, and the CAN protocol pays more attention to the demand for real-time communication. Therefore, it is necessary to design a lightweight key distribution protocol. some protocols [4, 5, 13] introduced above use asymmetric cryptographic algorithms in the key generation and distribution phase. It may add to the computational overhead for many ECUs with low power consumption requirements. The NSSK protocol [16] is a classic session key distribution protocol based on symmetric encryption, but it cannot identify and resist the known session key attack. Moreover, The NSSK protocol is a simple three-party protocol, which cannot fit the key distribution of other ECUs in a group composed of more than two ECUs in the actual scenes. Therefore, we propose an improved NSSK protocol by adding MAC authentication, digital signature, and time-stamping mechanisms. And we design a two-stage key distribution protocol to ensure that all ECUs in a group composed of more than two ECUs can securely get the same session key.

Our main contributions are as follows:

(1) We propose an improved session key distribution scheme based on classic Needham-Schroeder SKDS, which solves the original scheme's lack of authentication and recognition of replay attacks such as the Denning-Sacco attack, and ensures the security and verifiability of the key distribution process.
(2) The two-stage key distribution protocol we designed is suitable for the session key distribution scenario in the CAN network. To securely distribute the session key generated within a pair of ECUs to other ECUs with the same CAN-ID, we designed the Second SKDS.
(3) We analyze the security of the new protocol suite by formal analysis performed in the random oracle model and confirm our security goals using a formal verification tool named Tamarin-Prover. We finally give a comparison of the new protocol and the original protocol on security issues.

The rest of the paper is organized as follows. Section 2 introduces the preliminaries of the CAN frames and NSSK. Section 3 introduces the new protocol suite we designed. Section 4 presents the formal security analysis in the random oracle model. Section 5 discusses the protocol formal verification by using Tamarin-Prover. Section 6 concludes the work.

2 Preliminaries

2.1 CAN Frame

CAN is one of the most representative protocols often used for in-vehicle networks. In the CAN protocol, each ECU transfers information to other ECUs using a data frame. The sender ECU sends the data frames with an ID. The receiver ECU retrieves the data frame and examines the ID to identify whether the data frame is sent by the right sender ECU. The CAN protocol has two types, CAN 2.0A and CAN 2.0B, The main difference between them is that there are more 18 bits in the ID field of CAN 2.0B than which in CAN 2.0A. The data composition of the CAN 2.0B data frame is shown in Fig. 1. we give a brief introduction for these segments in the CAN data frame. SOT is a bit that represents the beginning of the data frame. The arbitration field represents the priority of the frame. Its ID field is divided into two parts: Base Identifier (11 bits) and Extended Identifier (18bits). The IDE field determines whether the Extended Identifier is used. The Control field indicates the number of bytes of data and the segment of reserved bits. Data field is a maximum of 8 bytes of content that contains the data to be transmitted to receiver ECU. CRC field is responsible to detect the transmission error segment of the frame. ACK filed represents the segment that is confirmed to be received normally.

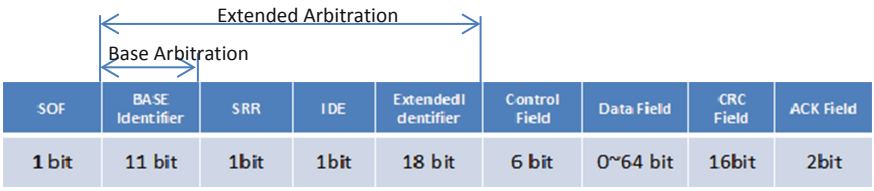

Fig. 1. The data composition of CAN 2.0B frame

2.2 System Architecture

The system architecture of the in-vehicle network is shown in Fig. 2. The ECUs are deployed in different subnets. The data bus in each subnet is indirectly connected with the central gateway. Establish communication with different subnets. The gateway can be regarded as an ECU node that provides secure communication and authentication management. We call the gateway GECU. In this scenario, GECU is responsible for session key distribution with each ECU and provides identity authentication for each ECU node.

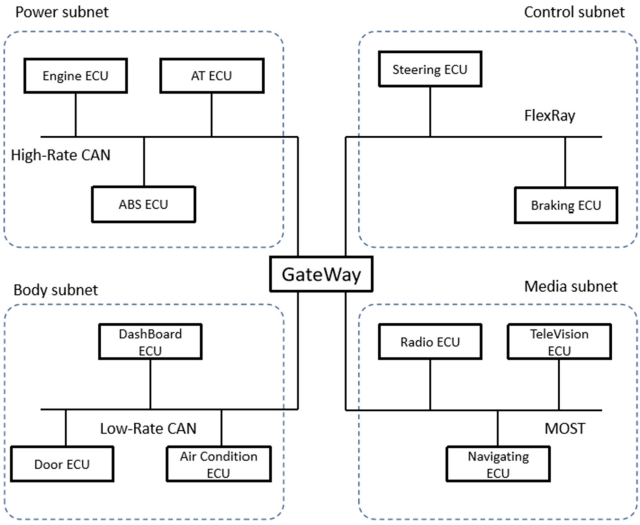

Fig. 2. The structure of the in-vehicle network

2.3 The Classic NSSK Protocol and Denning-Sacco Attack

The classic Needham-Schroeder SKDS scheme [16], which is shown in Fig. 3, GECU is responsible for generating the secret session key K_{ab}, encrypting the $K_{ab}||ID_{ECU_A}$ with the Long-term key K_{bs} of the GECU and the ECU_B to obtain the ciphertext $C_1 = \{K_{ab}, ID_{ECU_A}\}K_{bs}$, and then using the GECU and the long-term key K_{as} of ECU_A encrypts $N_a||ID_{ECU_B}||K_{ab}||C_1$ (N_a is a random number generated by ECU_A) to obtain the ciphertext $C_2 = \{N_a, ID_{ECU_B}, K_{ab}, \{K_{ab}, ID_{ECU_A}\}K_{bs}\}K_{as}$ and sends it to the sender ECU_A. After decryption by the ECU_A, the secret session key K_{ab} and C_1 are obtained. The ECU_A then sends C_1 to the ECU_B, and the ECU_B obtains the secret session key K_{ab} by decryption, and both parties use K_{ab} to encrypt communication data.

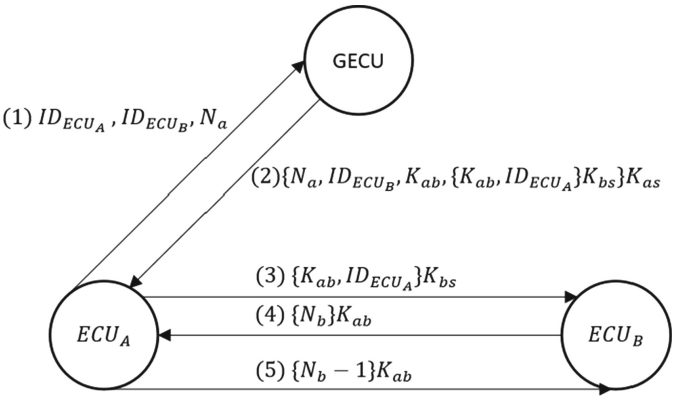

Fig. 3. The NSSK protocol

However, the classic NSSK protocol has an attack called the Denning-Sacco attack. If an adversary A records a session S between the two parties during the key distribution process of the two parties and obtains the session key K_{ab} of S. Then the adversary A initiates a new NS session S′ with ECU_B, starting from step (3) of session S′, and sends the previously used ticket $t = \{K_{ab}, ID_{ECU_A}\}K_{bs}$ to ECU_B. When ECU_B receives t because it cannot identify whether t comes from the real ECU_A, it will still decrypt t and accept the session key K_{ab}, and use K_{ab} to encrypt the random number N_b and send it to adversary A. After receiving N_b, the adversary A continues to forge the confirmation message $\{N_b - 1\}K_{ab}$ and send it to ECU_B. Therefore, if ECU_B continues to use K_{ab} to encrypt data, it may cause information leakage. Attack as follows, I(ECU_A) denotes the adversary A who impersonates the identity of ECU_A:

1. $\mathrm{ECU_A} \rightarrow$ GECU: $\mathrm{ID_{ECU_A}}, \mathrm{ID_{ECU_B}}, \mathrm{N_A}$
2. GECU $\rightarrow \mathrm{ECU_A}$: $\{\mathrm{N_a}, \mathrm{ID_{ECU_B}}, \mathrm{K_{ab}}, \{\mathrm{K_{ab}}, \mathrm{ID_{ECU_A}}\}\mathrm{K_{bs}}\}\mathrm{K_{as}}$
3. $\mathrm{I(ECU_A)} \rightarrow \mathrm{ECU_B}$: $\{\mathrm{K_{ab}}, \mathrm{ID_{ECU_A}}\}\mathrm{K_{bs}}$
4. $\mathrm{ECU_B} \rightarrow \mathrm{I(ECU_A)}$: $\{\mathrm{N_b}\}\mathrm{K_{ab}}$
5. $\mathrm{I(ECU_A)} \rightarrow \mathrm{ECU_B}$: $\{\mathrm{N_b} - 1\}\mathrm{K_{ab}}$

There is a defect in classic NSSK protocol that receiver ECU_B is unable to recognize the replay message is sent by a legitimate sender ECU_A or an impersonation attacker. The protocol lacks identity verification for the sender ECU_A and time deviations as well.

3 New Protocol Suite

To solve the defects of the original NSSK, we make several modifications to the original NSSK protocol to resist the Denning-Sacco attack:

(1) Each ECU participating in key distribution is required to provide MAC verification to ensure message integrity and ensure that the message is generated by the legitimate ECU.
(2) The delivery of the session key between ECU_A and ECU_B is changed to the delivery of a secret seed. After receiving the seed, the agreed key derivation function is used to derive the pair session key (E_K, A_K). E_K for encryption and A_K for message authentication. When ECU_B receives the correct session key, it calculates the MAC and cipher-text respectively and sends them to ECU_A for session key confirmation.
(3) When ECU_A sends ciphertext data containing secret seed to ECU_B, it requires ECU_A to provide the current timestamp. To prevent ECU_B from receiving a replayed message at a certain time in the future because it cannot recognize whether it is a transmitted message and mistakenly accept it.
(4) It is required that ECU_A use the private key to sign the ciphertext containing the seed and the timestamp, and only after ECU_B can successfully use the corresponding public key to verify the validity of the signature and check the freshness of the timestamp, it is allowed to accept the secret seed to derive the session key. Otherwise ECU_B may accept a forgery session key while the attacker executes the Denning-Sacco attack by replaying the transmitted data in the past executed protocol.

The key distribution protocol in the initial session abbreviated as Initial SKDP is shown in Fig. 4. The declaration we used in the protocol is shown in Table 1.

Table 1. Notations of some Cryptography terms.

Notation	Description
ECU_i	i^{th} electronic control unit
ID_i	Identity of ECU_i
GECU	Gateway ECU
CTR_{ECU_i}	ECU_i data frame counter
$Seed_k$	Seed value of k^{th} session
Ek_k	Encryption key of k^{th} session
Ak_k	Authentication key of k^{th} session
C	Ciphertext
M	Plaintext
$K_{A,i}$	Long-term symmetric key:GECU and ECU_A, i = 1 for encryption and i = 2 for authentication
$KDF_X()$	Keyed one-way function used for key derivation
H(.)	Keyed-hash Message Authentication Code (HMAC)

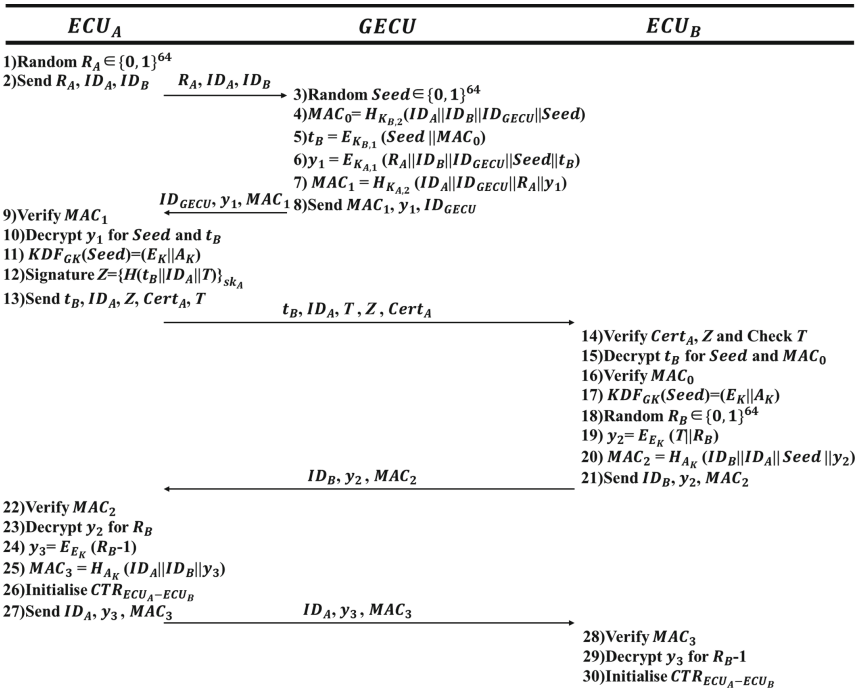

Fig. 4. The initial SKDP

The steps in the Initial SKDP are listed as follows.

(1) ECU_A selects a random number R_A.
(2) ECU_A sends the random number R_A, and its identification ID_A, and ECU_B's identification ID_B to $GECU$.
(3) $GECU$ chooses a random number as the secret seed which is denoted by $Seed$.
(4) $GECU$ calculates the MAC of data $ID_A||ID_B||ID_{GECU}||Seed$ with the authentication key $K_{B,2}$ of ECU_B, and the result is denoted by MAC_0.
(5) $GECU$ uses the encryption key $K_{B,1}$ of ECU_B to encrypt the data $Seed||MAC_0$, and the result is represented by t_B.
(6) $GECU$ uses the encryption key $K_{A,1}$ of ECU_A to encrypt the data $R_A||ID_B||ID_{GECU}||Seed||t_B$, and the result is denoted by y_1.
(7) $GECU$ uses the authentication key $K_{A,2}$ of ECU_A to calculate the MAC of the data $ID_A||ID_{GECU}||R_A||y_1$, and the result is represented by MAC_1.
(8) $GECU$ sends MAC_1, y_1 and identification ID_{GECU} to ECU_A.
(9) ECU_A verifies MAC_1 after receiving the data.
(10) ECU_A decrypts y_1 to obtain the $Seed$ and t_B.
(11) ECU_A calculates the session key E_k and A_k by a predefined key derivation function of $KDF_{GK} = E_k||A_k$.
(12) ECU_A uses the private key sk_A to calculate the digital signature $Z = \{H(t_B||ID_A||T)\}sk_A$, where T is the current timestamp.
(13) ECU_A sends data t_B, ID_A, Z, T and certificate $Cert$ containing the public key Pk_A of ECU_A to ECU_B.
(14) After receiving the data, the ECU_B verifies the signature Z and the certificate $Cert$, and checks whether the timestamp T is fresher than the timestamps it has previously received.
(15) After the ECU_B check is passed, decrypt t_B to obtain the $Seed$ and MAC_0.
(16) ECU_B verify MAC_0.
(17) ECU_B calculates the session key E_k and A_k by a predefined function of $KDF_{GK} = E_k||A_k$.
(18) ECU_B selects a random number R_B.
(19) ECU_B encrypts the data $T||R_B$ with the session encryption key E_k, and the result is denoted by y_2.
(20) The ECU_B uses the session authentication key A_k to calculate the MAC of the data $ID_B||ID_A||Seed||y_2$, and the result is denoted by MAC_2.
(21) ECU_B sends the data ID_B, y_2 and MAC_2 to ECU_A.
(22) ECU_A verifies MAC_2.
(23) ECU_A decrypts y_2 and gets the random number R_B.
(24) ECU_A encrypts data R_B-1 with the session encryption keyE_k, and the result is represented by y_3.
(25) ECU_A uses the session authentication key A_k to calculate the MAC of the data $ID_A||ID_B||y_3$, and the result is represented by MAC_3.
(26) ECU_A resets the counter $CTR_{ECU_A-ECU_B}$.
(27) ECU_A sends data ID_A, y_3 and MAC_3 to ECU_B.
(28) ECU_B verifies MAC_3 after receiving the data.
(29) ECU_B decrypts y_3 to obtain $R_B - 1$, confirming that R_B is received correctly by ECU_A.
(30) ECU_B resets the counter $CTR_{ECU_A-ECU_B}$.

When there are more receiver ECUs, the sender ECU needs to negotiate with the GECU to obtain the seed and continue to send it to other ECUs. The protocol in the subsequent session abbreviated as Second SKDP is shown in Fig. 5.

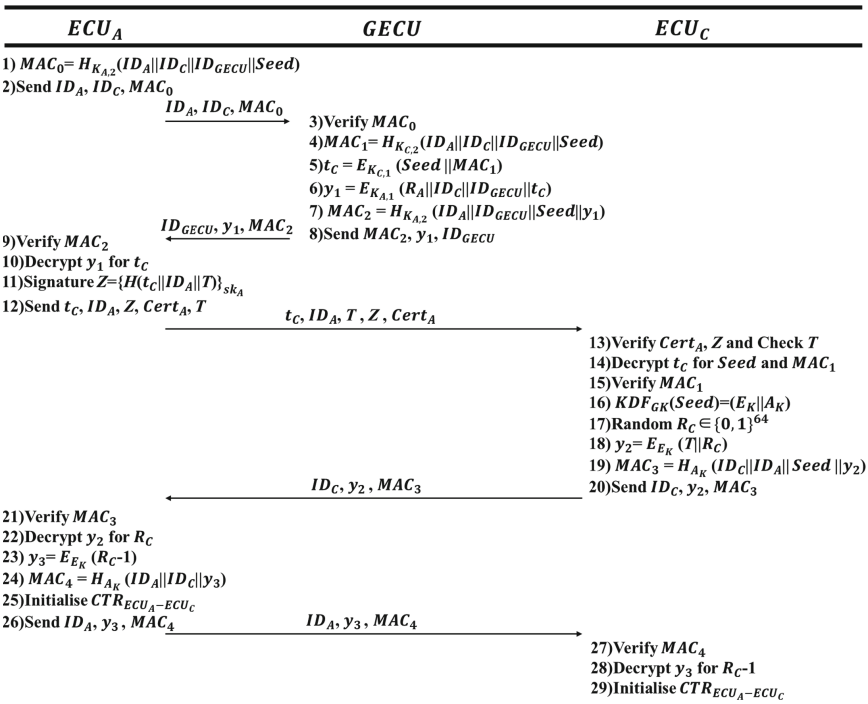

Fig. 5. The second SKDP

The steps in the Second SKDP are listed as follows.

(1) ECU_A uses its authentication key $K_{A,2}$ to calculate the MAC of the data $ID_A||ID_C||ID_{GECU}||Seed$ and the result is represented by MAC_0.
(2) ECU_A sends ID_A, ID_C, MAC_0 to $GECU$.
(3) $GECU$ verifies MAC_0.
(4) $GECU$ uses the authentication key $K_{C,2}$ of ECU_C to calculate the MAC of the data $ID_A||ID_C||ID_{GECU}||Seed$ and the result is represented by MAC_1.
(5) $GECU$ encrypts the data $Seed||MAC_1$ with the encryption key $K_{C,1}$ of ECU_C, and the result is represented by t_C.
(6) $GECU$ uses the encryption key $K_{A,1}$ of ECU_A to encrypt data $R_A||ID_C||ID_{GECU}||t_C$, and the result is denoted by y_1.
(7) $GECU$ uses ECU_A's authentication key $K_{A,2}$ to calculate the MAC of the data $ID_A||ID_{GECU}||Seed||y_1$, and the result is represented by MAC_2.

(8) $GECU$ sends data MAC_2, y_1, ID_{GECU} to ECU_A.
(9) ECU_A receives data to verify MAC_2.
(10) ECU_A decrypts the data y_1 to obtain t_C.
(11) ECU_A uses the private key sk_A to calculate the digital signature $Z = \{H(t_C||ID_A||T)\}sk_A$, where T is the current timestamp.
(12) ECU_A sends data t_C, ID_A, Z, T and certificates $Cert$ containing the public key Pk_A of ECU_A to ECU_B.
(13) After receiving the data, the ECU_B verifies the signature Z and the certificate $Cert$, and checks whether the timestamp T is fresher than the timestamps it has previously received.

(14)–(29) The process is similar to (15)–(30) in Fig. 4.

4 Security Analysis

4.1 Formal Security Analysis of Initial SKDP and Second SKDP

Before discussing the formal analysis of protocol security, we introduce some symbolic definitions and theorems about the formal description.

Definition 1: (Negligible function): A function $\mu(n)$ is negligible, if and only if it exists $n_0 \in Z^+$ for any $c > 0$, it has $|\mu(n)| \leq \frac{1}{n^c}$ for all $n > n_0$.

Definition 2: (Semantic Security) For a given protocol P, let adversary A initiates a new round of conversational interaction. In the conversation, adversary A initiates a query and guesses the coin (a random number b), then the protocol P is semantically secure. Only if the advantage of A is negligible or $Adv_A^{SS}(P, n) = Pr[A(P, n) = b] - \frac{1}{2} \leq \mu(n)$.

Definition 3: (Difference lemma) Use M, N, and F defined in some probability distribution. If $M \wedge \neg F \equiv N \wedge \neg F$, then $Pr[M] - Pr[N] \leq Pr[F]$.

Theorem: For a given protocol Initial SKDP or Second SKDP, there are the following assumptions: a long-term key can be guessed with the advantage Adv_P^{SS}, the length of the MAC value generated by the hash function is l bits, and the signature generated by the digital signature function is m bits, The encryption function can ensure k bits security. Then the semantic security of the protocol can be broken by adversary A with the advantage $Adv_P^{SS} \leq \frac{q_{hash}^2}{2^l} + Adv_{sk} + \frac{q_{sig}^2}{2^m} + \frac{q_{Enc}^2}{2^k}$, whose q_{hash} represents the number of times the hash function oracle is queried, q_{sig} and q_{Enc} also represent the number of times that the signature function oracle is queried and the encryption function oracle, respectively.

Before starting to prove the theorem, we define three random oracle model query operations: $Hash(\cdot)$, $Sig(\cdot)$, $Enc(\cdot)$.

Hash(·): The adversary A initiates a hash query corresponding to the plain text M or the ciphertext C. If the corresponding record has been stored in the oracle's hash table H. according to the format (M, f) or (C, g), the hash oracle will The records f and g stored in the hash table H will be returned to the adversary A. Otherwise, the oracle returns the generated random numbers f' or g' to the adversary A, and stores (M, f') and (C, g') in the hash table H.

Enc(·): The adversary A initiates a query for the encryption result corresponding to the plain text M. If the corresponding record has been stored in the ciphertext table E of the oracle according to the format (M, C), the ciphertext oracle will store it in the ciphertext table E The record C is returned to the adversary A. Otherwise, the oracle returns the generated random number C' to the adversary A, and stores (M, C') in the ciphertext table E.

Sig(·): The adversary A initiates a query for the signature result corresponding to message M. If the corresponding record should be stored in the signature table S of the oracle according to the format (M, S), the signature oracle will return the record S stored in the signature table S To the adversary A. Otherwise, the oracle returns the generated random number S' to the adversary, and stores (M, S') in the signature table S.

In addition to the above-defined oracle model query operations, we also define the following query operations:

Send(·): Simulate the query sent by the adversary A.

Execute(·): Query that simulates an adversary's passive attack.

RevealSessKey(·): Simulate the query that the adversary A forces the protocol entity to reveal the session key.

Corrupt(·): Simulate the query of the adversary A to obtain the long-term key held by the protocol entity.

Test(·): Initiated by adversary A at the end of the game. The queried entity returns an unbiased coin value b. If $b = 1$, it returns the real session key, otherwise it returns a random number. If the opponent can distinguish b correctly, the game is won.

The following is the proof process of the agreement Initial SKDP and Second SKDP:

Proof: We define five games G_0, G_1, G_2, G_3, and G_4. S_i is the event when adversary A wins the game G_i, i = 0, 1, 2, 3, 4.

Game G_0: Describes a real attack on the semantic security of protocol P. Adversary A first sends the activation protocol and then uses it to monitor the protocol information. After collecting information through a probabilistic polynomial-time running and waiting for the query, adversary A initiates the query. The condition for adversary A to win the game is that it can successfully guess the bit b. According to the definition of semantic security, the advantage of adversary A for winning the game in this game is $Adv_P^{SS} = |Pr(S_0) - \frac{1}{2}|$.

Game G_1: The only difference between this game and the previous game G_0 is the addition of a collision with MACs. If a MAC collision occurs at a certain stage in the protocol, adversary A can modify the secret seed and force ECUs to accept forged data in the explicit key confirmation stage. According to the birthday paradox, the advantage of querying the hash table is at least approximately $\frac{q_{hash}^2}{2^l}$, Use the difference lemma to express the advantage $|Pr(S_1) - Pr(S_0)| \leq \frac{q_{hash}^2}{2^l}$.

Game G_2: This game is aimed at the key secrecy of the protocol. In this game, adversary A initiates a query for the expired session key by asking. But even with an expired session key, the adversary still needs a long-term key to calculate the current session key. The advantage Adv_{sk} of the adversary A to obtain the long-term key is used to indicate that the advantage of adversary A replacing the long-term key by using a random number is Adv_{sk}, so the advantage difference between the two games is $|Pr(S_2) - Pr(S_1)| \leq Adv_{sk}$.

Game G_3: This game is different from the previous game G_2 in which, adversary A tries to find the signature collision. If a signature collision occurs, adversary A can make the participants accept a forged ciphertext and timestamp. In the end, adversary A can successfully cheat the participants to confirm the wrong session key that they have received. This advantage is that $|Pr(S_3) - Pr(S_2)| \leq \frac{q_{sig}^2}{2^l}$.

Game G_4: This game is different from the previous game G_3 in which, adversary A tries to look up the hash table to find the collision of the ciphertext. If adversary A finds a collision, it can forge a fake secret seed and send it. The advantage of adversary A in this game by querying the hash table to try to collide is $\frac{q_{Enc}^2}{2^k}$, so the advantage of the game is $|Pr(S_4) - Pr(S_3)| \leq \frac{q_{Enc}^2}{2^k}$.

Based on the above proofs, the advantage that adversary A can win the game is calculated according to the triangle inequality as follows:

$$
\begin{aligned}
&\mathrm{Adv}_{\mathrm{P}}^{\mathrm{SS}} = |\mathrm{Pr}(\mathrm{S}_4) - \mathrm{Pr}(\mathrm{S}_0)| \\
&= |\mathrm{Pr}(\mathrm{S}_4) - \mathrm{Pr}(\mathrm{S}_3) + \mathrm{Pr}(\mathrm{S}_3) - \mathrm{Pr}(\mathrm{S}_2) + \mathrm{Pr}(\mathrm{S}_2) - \mathrm{Pr}(\mathrm{S}_1) + \mathrm{Pr}(\mathrm{S}_1) - \mathrm{Pr}(\mathrm{S}_0)| \\
&\leq |\mathrm{Pr}(\mathrm{S}_4) - \mathrm{Pr}(\mathrm{S}_3)| + |\mathrm{Pr}(\mathrm{S}_3) - \mathrm{Pr}(\mathrm{S}_2)| + |\mathrm{Pr}(\mathrm{S}_2) - \mathrm{Pr}(\mathrm{S}_1)| + |\mathrm{Pr}(\mathrm{S}_1) - \mathrm{Pr}(\mathrm{S}_0)| \\
&\leq \frac{\mathrm{q}_{\mathrm{hash}}^2}{2^{\mathrm{l}}} + \mathrm{Adv}_{\mathrm{sk}} + \frac{\mathrm{q}_{\mathrm{sig}}^2}{2^{\mathrm{l}}} + \frac{\mathrm{q}_{\mathrm{Enc}}^2}{2^{\mathrm{k}}}
\end{aligned}
$$

Since the advantage between each game is negligible. Therefore, for a probabilistic polynomial-time adversary A, the advantage of looking for an attack on the protocol model is negligible. According to the semantic security theorem, if the long-term key is not leaked, the protocol is semantic security.

4.2 Informal Security Analysis of Initial SKDP and Second SKDP

Proposition1: The protocols Initial SKDP and Second SKDP provide session key freshness. In the protocols, Initial SKDP and Second SKDP, the freshness of the session key is guaranteed by *Seed*, a fresh random number to generate the session key through the derivation function KDF.

Proposition2: For a given protocol Initial SKDP and Second SKDP, any replay of the message will be detected and rejected by the receiver. For example, in Initial SKDP, when an attacker impersonates ECU_A and replays data and signatures with timestamp T and ciphertext t_B in step 13 to ECU_B, ECU_B will detect the timestamp T contained in the data is the same as the previously received one, so the received data will be

discarded. In other processes of sending and receiving data, the receiver will use the same authentication key as the sender to verify the MAC information in the message after receiving the data. If the verification fails and the received data is discarded. The ciphertext data received from step 21 is checked by ECU_A. If it does not contain the timestamp T sent in step 13, it will be rejected. If the ciphertext data which is sent from step 27 and is received by ECU_B is not $R_B - 1$ after decryption, the message will be rejected. The proof for Second SKDP is similar, so the protocol can resist replay attacks.

Proposition3: For a given protocol Initial SKDP and Second SKDP, any forgery attack will fail with an overwhelming probability. Because the adversary wants to attack (win the game) successfully, he must obtain the long-term keys on ECU_A and ECU_B to successfully solve the seed used to derive the session key. And the adversary also needs to be able to provide a valid signature that can be verified. This requires the adversary to know the private key of the signing party to successfully forge the signature. The difficulty of guessing the signature private key is a discrete logarithmic problem, which is difficult to solve in probabilistic polynomial time.

Proposition4: The given protocols Initial SKDP and Second SKDP can resist man-in-the-middle attacks. The adversary may control the communication channel and use the information exchanged between communication entities to carry out the attack. The adversary uses the existing communication records or the forged information items to communicate with the two parties respectively, that is, a combination of replay attacks and forgery attacks to carry out man-in-the-middle attacks. According to the results discussed above, these attack methods will be an overwhelming probability of failure.

5 Security Verification

In this section, We use Tamarin-prover to perform a security validation of our proposed protocol. For simplicity, we only present a partial script of Initial SKDP to introduce the lemmas of security goals.

5.1 Tamarin Overview

The Tamarin-prover is a powerful tool for symbolic modeling and analysis of security protocols. It takes a security protocol model as input, specifying the actions taken by agents running the protocol in different roles (e.g., the protocol initiator, the protocol responder, and the trusted key server), to demonstrate that even when arbitrarily many instances of the protocol's roles are interleaved in parallel, together with the actions of the adversary, the protocol fulfills its specified properties, and we can visualize the conditions under which the lemma proves successful or fails through the interactive mode.

Tamarin uses multiset rewriting to rewrite the rules of the protocol and lemmas. The description of a secure protocol consists of three parts: terms, facts, and rules.

Terms, consisting of variables, constants, and functional symbols that can represent the knowledge of the protocol participants, the adversary can acquire knowledge through the public terms of the protocol participants.

Facts represent the type of protocol variables. There are three types of special facts are built-in Tamarin-prover, In(), Out(), and Fr(). Respectively to simulate interactions with untrusted networks and simulate the generation of unique fresh values.

Rules declared as [L]-[A]->[R], represent the premises, action facts, and conclusion of the Protocol. The action facts can be used to express the interaction of the protocol, such as receiving events, and security properties for variables in subsequent lemmas such as interaction authentication and key privacy. Below are several important properties in protocol security validation.

(1) Aliveness is a basic authentication that ensures that an expected communication party performs certain events, namely that this communication object is present.
(2) Non-injective agreement requires both parties to agree on the transmission of data.
(3) Injective agreement needs to meet two conditions: a source and an intended destination agree on variables, and for every committed variable by a source, there uniquely exists an intended destination that accepts the variable.

In addition to the attributes listed above, the validation of the security protocol requires paying attention to data secrecy. That means a particular message content cannot be known by the adversary unless an honest entity exchanges confidential information with the captured entity. When verifying the above security properties, a lemma called the existence trace is often added to verify that the state in the protocol specification is accessible.

5.2 Verification of the Initial SKDP

We verify the security properties of the initial session key distribution protocol by clarifying the following lemma, involving the security attributes required to be verified: executable, the secrecy of session key, non-injective agreement, and injective consistent injective agreement.

```
lemma executability:
    exists-trace
    " Ex A B Rb kab #i #j #k.
      Commit_B(A,B,<'A', 'B', dec(Rb),kab>)@#i
     & Commit_A(A,B,<'A', 'B', Rb,kab>)@#j & #j<#i
     & Running_A(A,B,<'A', 'B', dec(Rb),kab>)@#j
     & Running_B(A,B,<'A', 'B', Rb,kab>)@#k & #k<#j
     & not(Ex #r1. Reveal(A)@#r1)
     & not(Ex #r2. Reveal(B)@#r2) "
```

The above lemma requires that when ECU_B receives the session key *kab*, at time *k* by executing Running_B to encrypt the random number *Nb* with *kab*, there is always ECU_A receiving the message at time *j* by executing Commit_A and $k < j$. Similarly, when ECU_A receives *Nb* and decrypts it, at the same time *j* runs Running_A to encrypt $Nb - 1$ with *kab* and sends it out. At this time, there must be ECU_B that receives the message by executing Commit_B at the time *i* and $j < i$. According to this lemma, the executability between states can be proved.

```
lemma Secrecy:
  " not( Ex A B m #i .
      Secret(A, B, m)@ #i
      & (Ex #r. K(m) @ #r)
      & not(Ex #r. Reveal(B) @ #r)
      & not(Ex #r. Reveal(A) @ #r)
    ) "
```

The above lemma describes the secrecy of the distributed session key between ECU_A and ECU_B, that no session key *m* between ECU_A and ECU_B is known by the adversary at time *r*, if not the long-term key used in ECU_A or ECU_B is leaked.

```
lemma agreement_ECUA:
 "All A B t #i.
   Commit_A(A,B,t) @i
   ==> (Ex #j. Running_B(A,B,t) @j  & j < i)
          | (Ex #r. Reveal(A)@r)
          | (Ex #r. Reveal(B)@r)"
```

The above lemma describes the non-injective agreement of ECU_A, that is when ECU_A receives a message at the time *i*, there must be ECU_B sending the message at time *j* and both ECU_A and ECU_B are honest, Similarly, the non-injective agreement lemma agreement_ECUB on ECU_B can be listed by imitating it.

```
lemma injectiveagreement_ECUB:
 "All A B t #i.
  Commit_B(A,B,t) @i
  ==> (Ex #j. Running_A(A,B,t) @j
     & j < i
     & not (Ex A2 B2 #i2. Commit_B(A2,B2,t) @i2
                 & not (#i2 = #i)))
        | (Ex #r. Reveal(A)@r)
        | (Ex #r. Reveal(B)@r)"
```

The above lemma describes the injective agreement of ECU_B, that is when ECU_B receives a message at the time i, it must be sent by ECU_A at time j, and there is no other time in the protocol execution round that ECU_B can receive the message, that is, sending and receiving is injective, and both ECU_A and ECU_B are honest. In the same way, the injective agreement lemma injectiveagreement_ECUA on ECU_A can be listed by imitating it.

5.3 Verification of Second SKDP

We verify that the Second SKDP is similar to those in Sect. 5.2, so we omit the verification details and only give a conclusion.

For each protocol, we list four lemmas for verifying security properties. They are executable, non-injective agreement, injective agreement, and session key secrecy. As stated in the lemma mentioned in the previous section. The executable lemma verifies the reachability of the protocol state. The non-injective agreement lemma verifies that there is a running entity corresponding to each commit operation. The injective agreement lemma verifies that the messaging between the two parties is sequential and injective. Session key confidentiality verifies that the adversary's knowledge does not contain the session key.

Table 2 provides the verification results for all protocols. It can be seen that all the protocols proposed in this article have passed the security verification of the protocol, that is, the key management is secure in the scenario proposed in this article.

Table 2. Verified lemma for the new protocol suite.

	Executable	Non-injective agreement	Injective agreement	Session key secrecy
Initial SKDP	√	√	√	√
Second SKDP	√	√	√	√

We compare the new protocol with the classic NSSK protocol and an improved NSSK protocol in [17] with several security attribute levels, and the results are shown in

Table 3. It can be seen that the new key distribution protocol we designed considers more security elements than others. It's more suitable for fast and efficient key distribution among ECUs that require exclusive session keys in the in-vehicle network.

Table 3. Comparison of new-protocol suite and NSSK.

	NSSK	Improved NSSK in [17]	Initial SKDP & Second SKDP
Key authentication	Implicit	Implicit	Explicit
Mutual entity authentication	No	No	Yes
Session key freshness	Yes	Yes	Yes
Resistance to Denning-Sacco attack	No	Yes	Yes
Resistance to replay attack	No	Yes	Yes
Resistance to MITM	Yes	Yes	Yes
Provable security	No	Yes	Yes
Key synchronization within the group	No	No	Yes

6 Summary

We propose an improved key update scheme based on Needham-Schroeder SKDP to distribute session keys, solve the shortcomings of the original scheme's lack of resistance to the Denning-Sacco attack, and ensure the security and verifiability of the key distribution process. It can resist known attacks such as replay attacks and man-in-the-middle attacks. Next, we execute formal analysis on the new protocol by using a random oracle model to simulate cryptographic games, the result proves the semantic security of the protocol. Finally, by using the Tamarin tool to verify the security properties of the protocol, it proves that the protocol has security objectives such as mutual entity authentication, explicit key confirmation, key authentication, and session key confidentiality. Compared with the scheme of updating the session keys of all ECU nodes as a whole, our scheme is more suitable for fast and efficient key distribution of single-point or multipoint CAN communication networks.

Acknowledgements. This work was supported in part by the National Natural Science Foundation of China under Grant 61872069, 62072090, and in part by the Fundamental Research Funds for the Central Universities under Grant N2017012.

References

1. Woo, S., Jo, H.J., Kim, I.S., et al.: A practical security architecture for in-vehicle CAN-FD. IEEE Transactions on Intelligent Transportation Systems **17**, 2248–2261 (2016)

2. Palaniswamy, B., et al.: An efficient authentication scheme for intra-vehicular controller area network. IEEE Trans. Inf. Forensics Secur. **PP**(99), 1 (2020)
3. Wu, Z., Zhao, J., Zhu, Y., et al.: Research on in-vehicle key management system under upcoming vehicle network architecture. Electronics **8**(9), 1026 (2019)
4. Pullen, D., Anagnostopoulos, N.A., Arul, T., et al.: Using implicit certification to efficiently establish authenticated group keys for in-vehicle networks. In: 2019 IEEE Vehicular Networking Conference (VNC). IEEE (2019)
5. Pan, Q., Tan, J.: A dynamic key generation scheme based on CAN bus. In: 2019 10th International Conference on Information Technology in Medicine and Education (ITME) (2019)
6. Youn, T.Y., Lee, Y., Woo, S.: Practical sender authentication scheme for in-vehicle CAN with efficient key management. IEEE Access **PP**(99), 1 (2020)
7. Jain, S., Guajardo, J.: Physical layer group key agreement for automotive controller area networks. In: Gierlichs, B., Poschmann, A. (eds.) CHES 2016. LNCS, vol. 9813, pp. 85–105. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_5
8. Lee, H., Jeong, S.H., Kim, H.K.: OTIDS: a novel intrusion detection system for in-vehicle network by using remote frame. In: 2017 15th Annual Conference on Privacy, Security and Trust (PST). IEEE (2018)
9. Ying, X., Bernieri, G., Conti, M., Poovendran, R.: TACAN: transmitter authentication through covert channels in controller area networks. arXiv:1903.05231 (2019)
10. Ying, X., Sagong, S.U., Clark, A., Bushnell, L., Poovendran, R.: Shape of the cloak: formal analysis of clock skew-based intrusion detection system in controller area networks. IEEE Trans. Inf. Forensics Secur. **14**(9), 2300–2314 (2019)
11. Wang, Q., Lu, Z., Qu, G.: An entropy analysis based intrusion detection system for controller area network in vehicles. In: 2018 31st IEEE International System-on-Chip Conference (SOCC), pp. 90–95. IEEE (2018)
12. King, Z., Yu, S.: Investigating and securing communications in the controller area network (CAN). In: 2017 International Conference on Computing, Networking and Communications (ICNC), pp. 814–818. IEEE (2017)
13. Fassak, S., Idrissi, Y.E.H.E., Zahid, N., Jedra, M.: A secure protocol for session keys establishment between ECUs in the CAN bus. In: Proceedings of the International Conference on Wireless Networks and Mobile Communications (WINCOM), Rabat, Morocco, 1–4 November 2017, pp. 37–42 (2017)
14. Nürnberger, S., Rossow, C.: vatiCAN–vetted, authenticated CAN bus. In: Gierlichs, B., Poschmann, A. (eds.) CHES 2016. LNCS, vol. 9813, pp. 106–124. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_6
15. Radu, A.I., Garcia, F.D.: LeiA: a lightweight authentication protocol for CAN. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016. LNCS, vol. 9879, pp. 283–300. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45741-3_15
16. Needham, R., Schroeder, M.: Using encryption for authentication in large networks of computers. Commun. ACM **21**(12), 993–999 (1978)
17. Jin-Gang, Y., Zhi-Gang, Z.: An improved NSSK authentication protocol and its formal analysis. In: 2017 10th International Conference on Intelligent Computation Technology and Automation (ICICTA) (2017)