Lightweight TLS 1.3 Handshake for C-ITS Systems

Danylo Goncharskyi*, Sung Yong Kim*, Pengwenlong Gu*, Ahmed Serhrouchni*,
Rida Khatoun* and Farid Nait-Abdesselam†

* LTCI, Telecom Paris, Institut Polytechnique de Paris, France
† University of Missouri Kansas City, USA

***Abstract*—Cooperative Intelligent Transport Systems (C-ITS) Deployment Platform is considered the newest version of vehicular communication systems, which enables the cooperation between two or more ITS sub-systems to provide enhanced services. With the expanded communication range and system complexity, ensuring the credibility of access nodes and protecting users from being monitored has become a difficult problem in network security, especially the services provided by remote servers like navigation. Transport Layer Security (TLS) is widely used for user authentication and encrypted data transmission in all networks. However, although the TLS handshake complexity is significantly reduced in TLS 1.3 the transmission of a full certificate chain during the handshake is still costly, especially for high-mobility vehicles. In this paper, we propose an optional extension named Certificate Get to reduce the TLS handshake overhead in C-ITS. Specifically, with our proposed extension, the revisiting client transmits a hash value of the certificate chain corresponding to a certain server in the *ClientHello* message, which can reduce the transmission payload of the certificate chain from an average of 4874 bytes to 68 bytes. Simulation results show that our proposed scheme achieves a significant performance gain by greatly reducing the certificate transmission delay by 50% for both TLS 1.3 and TLS 1.2.**

***Index Terms*—Transport Layer Security, Vehicular Communication, Handshake overhead, TLS hello extension**

I. Introduction

Cooperative Intelligent Transport Systems (C-ITS) Deployment Platform [1], organised by the European Commission is considered the newest version of vehicular communication systems, which enables the cooperation between two or more ITS sub-systems to provide enhanced services. In C-ITS systems, besides the mature ad-hoc short-range communication techniques, wide-area wireless communication techniques (like 5G) are also supported [2] that allow vehicles to communicate with other vehicles and roadside infrastructure within different ranges. With the expanded communication range and system complexity, ensuring the credibility of access nodes and protecting users from being monitored has become a difficult problem in network security, especially the services provided by remote servers like navigation. If road users connect to untrusted servers, it may cause system failures, mislead road users or even claim peoples' lives.

Transport Layer Security (TLS) is widely used for user authentication and encrypted data transmission in all kinds of networks in the past decades. TLS 1.3 [3] is the newly published version, in which the encryption algorithm is simplified, the handshake complexity is reduced from 2-RTT to 1-RTT and a 0-RTT handshake protocol is also proposed for session resumptions in low delay networks. Thus, compared with the previous version, TLS 1.3 is a suitable choice for the C-ITS system. However, although the TLS handshake complexity is greatly reduced, the certificate chains are still used to check that the public key and other data contained in an end-entity certificate effectively belong to its subject. It means that in each connection, the server needs to transmit its full certificate chain to the user, which makes the handshake procedure still costly, especially for High mobility vehicles.

In order to reduce the TLS handshake complexity, many methods have been proposed in the past few years. In [4], for low-power IoT systems, the authors proposed to analyse the cryptographic computations of the TLS protocol together with the memory and IP package handling in the edge device to identify and optimise limiting factors regarding the latency and energy demand. In [5], a new handshake protocol with unbalanced cost is proposed for wireless updating, in which the computation task is transferred from IoT devices to the powerful centre. In [6], a lightweight TLS protocol named iTLS is proposed, which can dynamically generates identity-based early keys before receiving a server response, allowing clients to send the encrypted data without additional round trips. From the TLS 1.2, hello extensions are also authorised to optimise the TLS handshake protocol. In [7] and in [8], two extensions are proposed for IoTs to reduce encryption energy consumption and traffic overhead respectively. Besides the handshake protocol optimisation, in [9] and [10] , the authors proposed two physical layer schemes to achieve better energy efficiency in TLS handshakes. Other than the TLS protocol, the authors proposed in [11] to offload computations and storage of security parameters to fog nodes in the vicinity and a symmetric-key payload encryption scheme to minimise the overhead of message communication in IoT environments.

In this paper, we propose an optional extension named Certificate Get to reduce the TLS handshake overhead in C-ITS. Based on our knowledge, due to its high correlation with usage scenarios, C-ITS services always have a high rate of repeated user visits rate, which motivates us to reduce the certificate chain transmission in handshake processes. Specifically, with our proposed extension, the revisiting client transmits a hash value of the certificate chain corresponding to a certain server in the *ClientHello* message. Then, the server will check if the received hash value matches well the hash of its own certificate chain. If matching, the server will not re-transmit the certificate chain in the *ServerHello* message. In

this way, a large number of repeated certificate transmissions can be avoided and does not impact the security of the TLS protocol. Simulation results show that our proposed scheme achieves a significant performance gain by greatly reducing the certificate transmission delay by 50% for both TLS 1.3 and TLS 1.2.

II. System Model

A. C-ITS Systems

In the C-ITS system, due to the increased support for wide-area wireless communication techniques like 5G, the entire system has become a heterogeneous structure. In Fig. 1, a simplified structure is illustrated. It can be observed that it consists of a set of base stations (BSs) and road side units (RSUs) deployed at the road sides communicating with the vehicles moving along the road. The whole area is divided into several macro cells by 5G BSs and each macro cell is also covered by one BS. BSs are connected to the gateway via S_1 link and each two adjacent BSs are connected by X_2 links. In each macro cell, several RSUs are located, connected to the BSs via wireless links. Each RSU can communicate with multiple base stations simultaneously. In this way, cooperation among multiple ITS subsystems can be achieved.

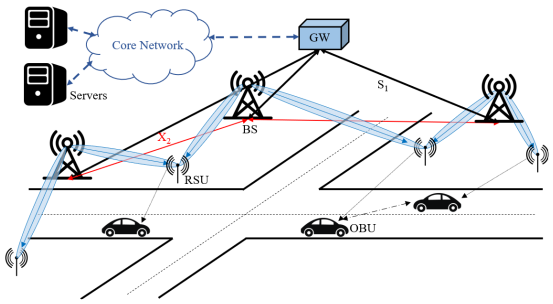


Fig. 1. Infrastructure of a C-ITS system.

The C-ITS system provides multiple services aims to improve safety, efficiency, comfort and sustainability of transportation. Some services like navigation and multimedia need to rely on remote servers to provide. This makes the communication between the vehicle and the server need to traverse the core network [12]. In this case, the TLS protocol is required for user authentication [13] and encrypted data transmission between the vehicle and the server.

B. Certificate Chain

The certificate chain or chain of trust is defined in RFC 5280 [14] to address the issue that a public key user is only initialised with a limited number of assured Certificate authority (CA) public keys. The proposed CA hierarchies is given in Fig. 2, which is consists of two layers of CAs, one root is at the top of the CA hierarchy whose certificate is a self-signed certificate and a layer of several intermediate CAs, which have CA certificates that are signed by the root CA and issue certificates for certain servers.

At the user side, for an received TLS certificate to be trusted, it must have been issued by a certificate authority (CA) which

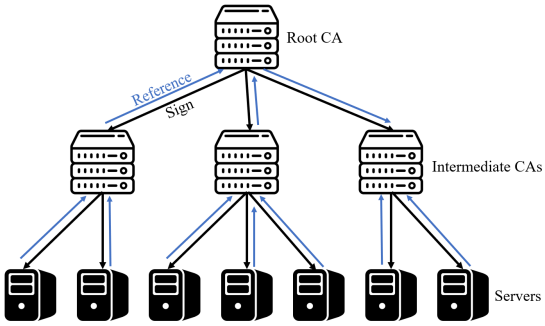


Fig. 2. Proposed CA hierarchies in RFC 5280.

is included in the trusted store the user is connecting. If the certificate was not issued by a trusted CA, the user will check to see if the certificate of the issuing CA was issued by another trusted CA. It continues this checking process until a trusted CA is found. Note that the lifespan of a new TLS certificate various from days to several months.

III. Certificate Get Extension

In this section, we present our proposed Certificate Get extension. In a C-ITS system, our proposed extension can simplify the certificate chain transmission between vehicles and servers, which will let vehicles benefit lower handshake delays and even lower energy consumption. In standard RFC 8446, two handshake processes are proposed: a full 1-RTT handshake procedure and a pre-shared key based 0-RTT handshake procedure. In this paper, we propose our extension for the 1-RTT handshake mode.

A. TLS 1.3 1-RTT Handshake

TLS handshake protocol is designed to authenticate devices and to negotiate security parameters in their connections like the version of the TLS protocol, encryption and signature algorithms, and the master keys.

A full handshake of TLS 1.3 is illustrated in Fig. 3. As always, the handshake process is kicked off by sending a *ClientHello* from the client to server. After received the *ClientHello*, the server reply the client a *ServerHello*, which includes the chosen key agreement protocol, the server's key share as well as the certificate chain. After the exchange of keys, both sides have the following information: Client Random, Server Random, Client Params and Server Params. Then both sides can compute the "Master Secret" using the DH and HKDF algorithms separately. Afterwards, the server sends a *EncrpytedExtensions* message to establish the server parameters. Then, the server send its *Certificate*, a *CertificateVerify*, which is a signature over the entire handshake using the private key corresponding to the public key in the *Certificate* message, and a *Finished* message which provides key confirmation and binds the endpoint's indentity to the exchanged key. Finally, the client replies server a *Finished* messages to finish the handshake process.

The structure of a extended *ClientHello* defined in RFC 4366 is given in Fig. 4. It can be observed that normally the following information should be provided by the client in a

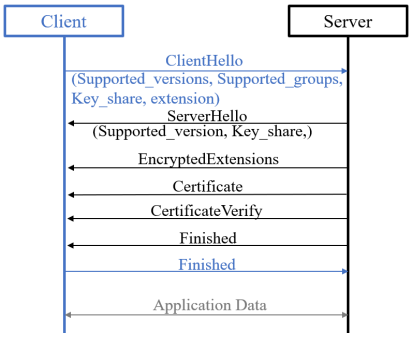


Fig. 3. 1-RTT handshake in TLS 1.3.

ClientHello: protocol version, client random data (used later in the handshake), an optional session id to resume, a list of cipher suites, a list of compression methods and a list of extensions.

```
struct {
        ProtocolVersion legacy_version = 0x0303;
        Random random;
        opaque legacy_session_id<0..32>;
        CipherSuite cipher_suites<2..2^16-2>;
        opaque legacy_compression_methods<1..2^8-1>;
        Extension extensions<8..2^16-1>;
} ClientHello;
```

Fig. 4. Structure of a *ClientHello* message.

Note that since TLS 1.2, the extension field is available for clients to request custom TLS functionalities from the server [15]. All extensions are standardised by the Internet Assigned Numbers Authority (IANA). Each of them is identified by a 2-bytes value. With the help of extensions, the encryption algorithm is simplified in TLS 1.3 handshakes compared to TLS 1.2 [16]. The key exchange and signature algorithms can be negotiated by using extensions like ”supported groups”, ”key share” and ”signature algorithms” in the first flight message of both sides. This enables TLS 1.3 to start encrypted communication after only one round trip, which not only helped reduce the handshake process from 2-RTT to 1-RTT but also greatly improves the security of the handshake process.

B. 0-RTT Handshake and Security Issues

Besides the improvements in standard handshake process, TLS 1.3 also support 0-rtt handshake for session resumptions or pre-shared key (PSK) based connections. Normally, a full 1-RTT handshake is used in the first authentication between a server and a client and both sides can derive a resumption secret. Then, the following session resumptions can be done in 0-RTT mode, there are two standard ways that the server can verify the resumption secret provided by a client.

- Session Caches: In this way, the server stores all resumption secrets of recent session and issues each client a unique identity. Whenever a user requests to reconnect, the server compares the identity and resumption secret provided by the user in the 0-RTT messages with the information stored in its own database. If the verification is passed, the user is allowed to reconnect. One major drawback of this method is that for massive access scenarios, the server needs to store a large amount of user information, which leads to high storage cost and high look up delay.
- Session ticket: Using session ticket, the server does not need to cache the user information. Instead, at the beginning of each new session, the server issues a ticket to the client which contains the resumption secret and encrypt the ticket with a session ticket encrypt key (STEK)[1]. The encrypted ticket is stored by the client but cannot decrypt it. Thus, for session resumption, the client only need to send back the session ticket to the server in its 0-RTT messages, which allows the server to recover the resumption key and verify the identity of the user.

However, due to the simplified process of the 0-rtt handshake protocol, it also faces some serious security challenges in actual deployment. For the above two schemes, the common safety hazard is that they are neither forward secure nor replay resistant, especially in wireless scenarios like the C-ITS since wireless channels are open to all users.

In TLS 1.3, the forward secrecy can be guaranteed by using the Elliptic-curve Diffie–Hellman (ECDH) protocol, in which the session key generated upon the negotiated DH parameters is unique for one session and does not rely on the server's long-term keys. In 1-RTT mode, even the private key of client or server is leaked or compromised by an adversary, it is not possible to decrypt the previously sniffed traffic data. However, in 0-RTT mode, the early data is encrypted by the pre-shared resumption secret which can only achieves a delayed forward secrecy. Besides, the replay attack remains another huge security threat to the 0-RTT handshake. If the resumption secret, the session ticket, or simply a *ClientHello* containing Early Data is captured by a malicious node, the adversary can launch large numbers of retries or duplicates them on multiple connections, which may lead to leakage of user information or denial of legitimate services [17].

C. Proposed Extension

In order to improve the efficiency of the TLS protocol handshake while ensuring security, we propose an extension for the 1-RTT handshake. With our proposed extension, other than its first connection, the client transmits a hash value of the certificate chain corresponding to a certain server in the *ClientHello* message. Then, the server will check if the received hash value matches well the hash of its own certificate chain. If matching, the server will not re-transmit the certificate chain in the *ServerHello* message. In this way, in large-scale wireless networks like the C-ITS, a large number of repeated certificate transmissions can be avoided, thereby greatly reducing energy consumption and wireless channel resource occupation.

The format of hello extensions defined in RFC 5246 [18] is given in Fig. 5. Following this format, one can propose a particular extension type. It should be noted that every

[1]The same STEK is used for many sessions and clients

extension included in the *ServerHello* should be included already in *ClientHello*. And the extended *ClientHello* may be used in starting a new session or in session resumption. Moreover, even if an optional extension is not supported by the server, the session should be established in a mode without this extension.

```
struct {
        ExtensionType           extension_type;
        opaque                  extension_data<0..2^16-1>;
} Extension;
```

Fig. 5. Format of a hello extension.

Following the format given in Fig. 5, we propose an optional extension named Certificate Get. Its structure is illustrated in Fig. 6. In this extension, we defined two extension types: the Certificate_Chain allows the client to communicate the server that the hash value transmitted is about a full certificate chain or just the certificate of the server. And the Hash_Identifier is used to specify the hash algorithm that used by the client. Currently we have specified two algorithms sha_256 and sha_512, which can be enriched in the future.

```
struct {
        Hash_Identifier         hash_type;
        Cerificate_Chain        certficate_chain_type;
        opaque                  hash_Certificate <0..2^16-1>;
} Certificate_Get;

enum {
        hash_certificate_srver_with_Root_CA (0);
        hash_certificat_server (1);
} Cerificate_Chain;

enum {
        sha_256 (0);
        sha_512 (1);
        .......
} Hash_Identifier;
```

Fig. 6. Structure of the Certificate Get extension.

During a handshake, the client initials a extended *ClientHello* message with a payload of 68 bytes. This 68-byte payload contains: a 2-byte extension identifier which is not distributed by the IANA yet, one byte to identify the hash algorithm, another byte to notify its a hash of a full certificate chain or a certificate of the server. And followed by a 64-byte hash value. After received the *ClientHello* message, the server firstly verifies if this extension is supported, if not, the server will reply an empty extension and transmits its certificate chain to the client. Otherwise, the sever hashes the certificate or the certificate chain using the hash algorithm selected by the client and compare these two hash values, if matching, the server replies the client the same hash value, if not, the server replies the client an empty extension and its full certificate chain.

D. Security and Performance Gains

By adopting our proposed certificate hash extension in TLS handshakes, both security and performance gains can be obtained. In terms of security, handshakes with our proposed extension can achieve the same security level as the original 1-RTT handshake protocol since our proposed extension is optional. The worst case would be the server does not support this extension or the hash value received by the server does not match the hash value of its own certificate chain. In this way, the server would ignore the extension request in *ClientHello* message and transmit its full certificate chain to the client with a *ServerHello* message, which would not cause handshake failure or impact the security level of the handshake protocol. Compared to the PSK based session resumption, our proposed extension achieves perfect forward secrecy and is more resistant to replay attacks.

In terms of performance, our proposed extension helps to reduce the certificate chain transmission payload to 68 bytes, which can greatly reduce the transmission latency and energy consumption. Moreover, it should be noted that even using the PSK based session resumption, the client needs to transmit a *Session Ticket* to the server, the transmission payload is much heavier than transmitting a fixed-length hash value.

IV. Simulation Results

In this section, the performance of our proposed extension will be tested via multiple simulations. In order to simulate the process of handshakes between the vehicle and the remote server and accurately evaluate the transmission overhead of the certificate chain in this process, we design the following two-part simulation experiments. In the first part, we visit the world's top 100 websites to test and analyse the overhead of the handshake process and the proportion of the certificate chain in it, to confirm whether replacing the transmission of the entire certificate chain with a 64-byte hash value can effectively reduce the handshake protocol overhead. In the second part, we set up a server in Thionville (France) using Openssl 3.0.6 and we make multiple accesses from Singapore to simulate the procedure that a vehicle accesses a remote server. In this way, the ability of our proposed extension to reduce handshake latency can be tested in a real network environment. Note that the Openssl server allows only one session at a time. Other parameters used in the simulation are summarized in Table I:

TABLE I
PARAMETERS USED IN SIMULATION

	Server	Client
Location	Thionville (France)	Singapore
Up-link Speed	45.79 Mbps	97.73 Mbps
Down-link Speed	42.96 Mbps	98.34 Mbps
Ping delay	9 ms	3.4 ms

A. Certificate Chain used in Top Websites

To better understand the use of the certificate chain in real websites, we first test the certificate size of the 100 most visited websites. To get a reliable website ranking, we use the kaggle data-set[2] in our tests to locate the top 100 websites. Then we connected each website several times to collect the handshake data and analyse them with Wireshark. The test

[2]https://www.kaggle.com/datasets/cheedcheed/top1m

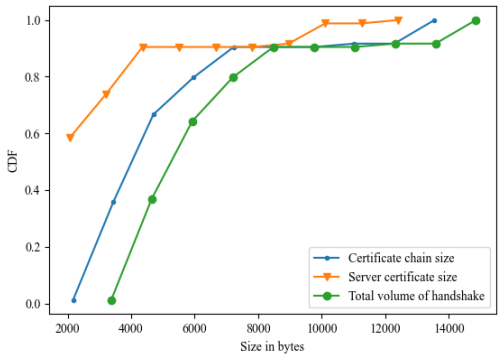


Fig. 7. CDF of server certificate size, certificate chain size and the total volume of handshake data over the 100 most visited websites.

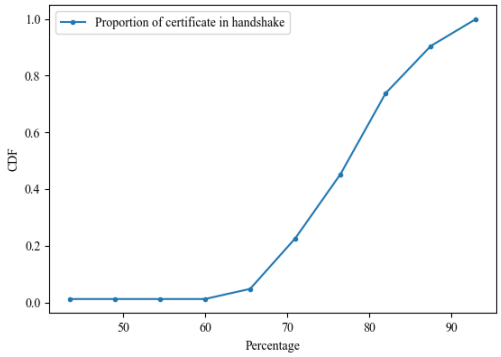


Fig. 8. CDF of the proportion of certificate chain in total handshake data volume for the 100 most visited websites.

results show that over the top 100 websites, the average size of their certificate chain is 4847 bytes and their average server certificate size is 3058 bytes.

In Fig. 7, the CDF of server certificate size, certificate chain size and the total volume of handshake data over the 100 most visited websites are given. It can be observed that for different websites, the server certificate size various from around 200 bytes to almost 12000 bytes, which hugely impacted the volume of handshake data. In order to show precisely this impact, a CDF of the proportion of certificate chain in total handshake data volume is given in Fig. 8. It can be observed that for more than 80% of the top websites, the certificate chain transmission takes more than 70% of the total handshake volume, which confirmed our guess that the transmission of certificate chain is costly in almost all scenarios. Besides the certificate chain transmission, the rest of handshake data volume is around 1000 to 1500 bytes.

Moreover, we also tested the support of the 0-RTT session resumption over these 100 websites. Surprisingly, the test results show that only 23 of them support it. It clearly shows that Service providers still have doubts about the security and secrecy of the 0-RTT session resumption protocol, especially its resistance to replay attacks. This also keeps a lot of repeated certificate chain transfers going on during day-to-day use.

B. Performance of Certificate Get Extension

In this part, we test the performance of our proposed Certificate Get extension. The development of our proposed extension is realised by using Openssl 3.0.6, which supports both TLS1.2 and TLS1.3 full handshakes. Based on the CA hierarchies presented in section II, we set up a certificate chain consisting of one root certificate, one intermediate certificate and one server certificate. The length of the certificate chain was set based on our test results given in the last subsection over the top 100 websites, which is 6672 bytes. And it should be noted that we choose the hash function SHA-256 to hash the certificate in our proposed extension, which gives a 64 bytes fixed-length hash value. Moreover, in order to make a fair comparison, besides the original full handshake protocols, we also considered the PSK based session resumption protocol as a benchmark scheme, in both TLS 1.3 and TLS 1.2 scenarios.

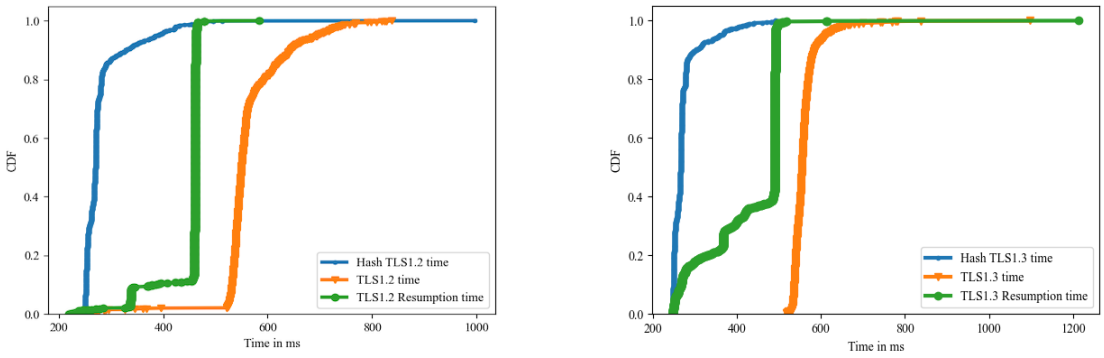
In our tests, since the client is enough far away from the server, the certificate transmission delay is an effective norm to evaluate the performance of our proposed extension. In full handshake modes (both TLS 1.3 and TLS 1.2), the certificate transmission delay can be considered as the time from the *ClientHello* sent until the full certificate chain or its hash value is received by client. And in session resumption modes, the certificate transmission delay can be recognised as the time difference between the *ClientHello* sent and the *ServerHello* received. Note that in order not to lose focus, early data is not allowed in our tests. We launched 1000 handshakes each for all three modes: full handshake, full handshake with our proposed extension and PSK based session resumption.

The CDF of the certificate transmission delay in all these three modes is illustrated in Fig. 9. It can be observed that the full handshake with our proposed extension outperformed the other two benchmark schemes. Only less than 20% of the PSK based session resumptions under TLS 1.2 and around 40% of the session resumption under TLS 1.3 can achieve the same performance level as our proposed scheme. This also proves that the PSK based session resumption mechanism in both TLS 1.2 and TLS 1.3 does reduce the complexity of the handshake process. However, since the session resumption in TLS requires the transmission of a *Session Ticket* from the client to the server, it is still more costly than our proposed scheme.

The mean certificate transmission delay for these three modes in both TLS 1.3 and TLS 1.2 are given in Table II. Each value in this table is obtained based on 1000 times simulations. It should be noted that although the full handshake in TLS 1.2 is 2-RTT, the certificate chain is transmitted with the *ServerHello* in the first round. Thus, in full handshake modes, the certificate transmission delay in TLS 1.3 is slightly shorter

TABLE II
MEAN CERTIFICATE TRANSMISSION DELAY

	TLS1.2	TLS1.3
Certificate transmission	565	559
Hashed Certificate transmission	281	273
PSK based session resumption	446	429



(a) Certificate transmission delay in TLS1.2.

(b) Certificate transmission delay in TLS1.3.

Fig. 9. CDF of the certificate transmission delay for full handshake modes, full handshake with our proposed extension and reduced round session resumption.

than in TLS 1.2. With our proposed scheme, a significant performance gain can be observed. Compared with the original certificate exchange, the certificate transmission delays are greatly reduced by 50% for both TLS 1.3 and TLS 1.2. And even compared with the PSK based session resumption, the certificate transmission delays are also reduced by 40%. Combined with the results given in Fig. 8, it confirms that the TLS handshake delays are mainly caused by the transmission of the certificate chain. By adopting our proposed scheme, for revisit users, this problem can be effectively addressed.

V. Conclusion

In this paper, an optional extension named Certificate Get is proposed to reduce the TLS handshake overhead in C-ITS systems. Specifically, with our proposed extension, the revisiting client transmits a hash value of the certificate chain corresponding to a certain server in the *ClientHello* message. Then, the server will check if the received hash value matches well the hash of its certificate chain. If matching, the server will not re-transmit the certificate chain in the *ServerHello* message instead of its full certificate chain. In this way, a large number of repeated certificate transmissions can be avoided and does not impact the security of the TLS protocol. Moreover, since our proposed extension is optional, it does not reduce the security of the original handshake protocol. Simulation results show that our proposed scheme achieves a significant performance gain by greatly reducing the certificate transmission delay by 50% for both TLS 1.3 and TLS 1.2.

For future work, since energy consumption is a hot topic, we will continue to discover the impact this extension may cause on energy consumption on both the client and the server sides.

References

[1] M. Lu, O. Turetken, O. E. Adali, J. Castells, R. Blokpoel, and P. Grefen, "C-ITS (cooperative intelligent transport systems) deployment in Europe: challenges and key findings," in *25th ITS World Congress, Copenhagen, Denmark*, 2018, pp. 17–21.

[2] P. Gu, D. Zhong, C. Hua, F. Nait-Abdesselam, A. Serhrouchni, and R. Khatoun, "Scaling a blockchain system for 5g-based vehicular networks using heuristic sharding," in *2021 IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 1–6.

[3] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018.

[4] F. Lauer, C. C. Rheinländer, C. Kestel, and N. Wehn, "Analysis and Optimization of TLS-based Security Mechanisms for Low Power IoT Systems," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, 2020, pp. 775–780.

[5] J. Cai, X. Huang, J. Zhang, J. Zhao, Y. Lei, D. Liu, and X. Ma, "A Handshake Protocol With Unbalanced Cost for Wireless Updating," *IEEE Access*, vol. 6, pp. 18 570–18 581, 2018.

[6] P. Li, J. Su, and X. Wang, "iTLS: Lightweight Transport-Layer Security Protocol for IoT With Minimal Latency and Perfect Forward Secrecy," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6828–6841, 2020.

[7] Q. Varo, W. Lardier, and J. Yan, "Dynamic Reduced-Round TLS Extension for Secure and Energy-Saving Communication of IoT Devices," *IEEE Internet of Things Journal*, pp. 1–1, 2022.

[8] P. Li, J. Su, and X. Wang, "ITLS/IDTLS: Lightweight End-to-End Security Protocol for IoT Through Minimal Latency," in *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, ser. SIGCOMM Posters and Demos '19, 2019, p. 166–168.

[9] U. Banerjee and A. P. Chandrakasan, "Efficient Post-Quantum TLS Handshakes using Identity-Based Key Exchange from Lattices," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[10] J. Mades, G. Ebelt, B. Janjic, F. Lauer, C. C. Rheinländer, and N. Wehn, "TLS-Level Security for Low Power Industrial IoT Network Infrastructures," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 1720–1721.

[11] A. Diro, H. Reda, N. Chilamkurti, A. Mahmood, N. Zaman, and Y. Nam, "Lightweight authenticated-encryption scheme for internet of things based on publish-subscribe communication," *IEEE Access*, vol. 8, pp. 60 539–60 551, 2020.

[12] M. H. Sherif, A. Serhrouchni, A. Y. Gaid, and F. Farazmandnia, "Set and ssl: Electronic payments on the internet," in *Proceedings Third IEEE Symposium on Computers and Communications. ISCC'98.(Cat. No. 98EX166)*. IEEE, 1998, pp. 353–358.

[13] M. T. Hammi, E. Livolant, P. Bellot, A. Serhrouchni, and P. Minet, "A lightweight mutual authentication protocol for the iot," in *International Conference on Mobile and Wireless Technology*. Springer, Singapore, 2017, pp. 3–12.

[14] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, May 2008.

[15] D. E. E. 3rd, "Transport Layer Security (TLS) Extensions: Extension Definitions," RFC 6066, Jan. 2011.

[16] D. Goncharskyi, S. Y. Kim, A. Serhrouchni, P. Gu, R. Khatoun, and J. Hachem, "Delay measurement of 0-rtt transport layer security (tls) handshake protocol," in *2022 8th International Conference on Control, Decision and Information Technologies (CoDIT)*, vol. 1, 2022, pp. 1450–1454.

[17] M. Fischlin and F. Günther, "Replay attacks on zero round-trip time: The case of the tls 1.3 handshake candidates," in *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, 2017, pp. 60–75.

[18] E. Rescorla and T. Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, Aug. 2008.