TLS Early Data Resistance to Replay Attacks in Wireless Internet of Things

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

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

***Abstract*—Transport Layer Security (TLS) is widely used for user authentication and encrypted data transmission in all kinds of networks. In its newly published version, TLS 1.3, a 0-RTT handshake protocol is proposed for session resumptions in low delay networks, which makes it possible to secure the data transmission and protect users from being monitored in wireless Internet of Things (IoTs). However, the 0-RTT TLS handshake protocol is vulnerable to the replay attack. In this paper, we propose a Time-Based One-Time Password (TOTP) empowered TLS encryption algorithm to resist replay attacks during the handshake process, in which we propose to integrate the TOTP into the encryption process of the *EarlyData*. It can significantly improve the forward secrecy of the 0-RTT handshake protocol and its capacity to resist the replay attack. On the other hand, we make no changes to the interaction process of the standardized 0-RTT handshake protocol to guarantee the compatibility of our proposed scheme, which makes our proposed scheme suitable for large area wireless IoTs. Simulation results show that under the premise of choosing an appropriate TOTP update rate, our proposed scheme can effectively resist replay attacks while ensuring the processing efficiency of the system.**

***Index Terms*—Transport Layer Security, 0-rtt handshake protocol, replay-attack resistant, forward secrecy**

I. Introduction

In the 5G era, due to the increase in network capacity and the greatly reduced transmission delay, the wireless Internet of Things (IoTs) will play an increasingly important role in industry, transportation, and smart medical care. However, in the scenario of open wireless channels, how to ensure the credibility of access nodes and protect users from being monitored has become a difficult problem in network security. In the past decades, Transport Layer Security (TLS) is widely used for user authentication and encrypted data transmission in all kinds of networks. In its newly published version, TLS 1.3 [1], a 0-RTT handshake protocol is also proposed for session resumptions in low delay networks, which is a suitable choice for wireless IoTs.

However, due to the simplified process of the 0-RTT handshake protocol, it also faces some serious security challenges in actual deployment, the replay attack is one of them. As a type of man-in-the-middle attacks, the replay attack can be launched by a malicious node who sniffs data packets and then replays them multiple times. In 0-RTT handshake mode of TLS 1.3, the *EarlyData* is encrypted by the pre-shared key and guarantees only a delayed forward secrecy. In this way, if this pre-shared key or the session ticket is captured by the malicious node, it 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 [2].

To make the 0-RTT handshake resilient against the replay attack, different solutions have been proposed in the past few years. In [3], the authors proposed to wait a short time (200 ms for example) to verify that the packet isn't created by communicating with the server several times. In [4], Li *et. al.* proposed an "identity share" extension. Thus, the shared key can be dynamically created based on the identity. In [5], the authors proposed a dynamic hash chain based authentication scheme for smart grid networks, wherein the key of each device is renewed based on hash-chain concept at every data collection round. And in [6], the authors proposed a PPRFs based forward security and replay resilience protocol, which can be directly applied in TLS 1.3 0-RTT, without requiring any changes to clients or the protocol. Thus, it can be observed that the most effective way to resist replay attacks during the 0-rtt handshake process is to constantly change the user's authentication information so that the previously leaked information cannot be repeatedly accepted by the server. However, due to the limited computing power of wireless IoT devices, the high complexity algorithms such as PPRFs can hardly be implemented in large scale wireless IoT systems. Therefore, how to balance the security and the complexity has become a major issue that must be addressed while designing a 0-rtt session resumption protocol for wireless IoTs.

In order to address this issue, we propose in this paper a Time-Based One-Time Password (TOTP) [7] empowered TLS encryption algorithm for wireless IoTs, in which the compatibility and the security of the system are jointly considered. Specifically, we propose to integrate the TOTP into the encryption process of the *EarlyData*, which significantly improved the forward secrecy of the 0-RTT handshake protocol and its capacity to resist the replay attack. On the other hand, to guarantee its compatibility, we make no changes to the interaction process of the standardized 0-RTT handshake protocol, which makes our proposed scheme suitable for large area wireless IoTs. Simulation results show that under the

978-1-6654-3540-6/22

premise of choosing an appropriate TOTP update rate, our proposed scheme can effectively resist replay attacks while ensuring the processing efficiency of the system.

II. 0-RTT Handshake and Security Issues

A. TLS 1.3 Handshakes Protocols

In standard RFC 8446 [1], two handshake protocols are proposed: a 1-RTT full handshake protocol and a simplified 0-RTT protocol. A TLS 1.3 full handshake is given in Fig. 1. As always, a *ClientHello* is sent to the server to kick off the handshake process. Note that in TLS 1.3, using the extensions "supported groups" and "key share", the client sends the list of supported cipher suites and its key share in the *ClientHello*, which allows to reduce the handshake process from 2-RTT to 1-RTT. After received the *ClientHello*, the server reply the client a *ServerHello*, in which the chosen key agreement protocol and the server's key share are comprised. 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 *ChangeCipherSpec* message to the client, and the following messages like certificate will all be encrypted using keys derived from a *server_handshake_traffic_secret* ({} in Fig. 1). Then, the server sends its finish message to the client, which is a hash of the entire handshake up to this point. Finally, the client replies with its *ChangeCipherSpec* and finish messages to finish the handshake process, which are encrypted using keys derived from a *client_handshake_traffic_secret* ({} in Fig. 1).

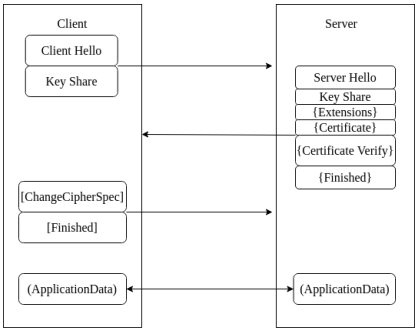


Fig. 1. 1-RTT handshanke of TLS 1.3.

Note that the key exchange and signature algorithms in TLS 1.3 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 speeds up connections but also greatly improves the security of the handshake process.

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. In order to activate the 0-RTT handshake, the following three conditions must be satisfied:

- The session ticket received by the client in the previous handshake has "max early data size" extension.
- In the process of PSK session resumption, the "early data" extension is configured in the *ClientHello*.
- The "early data" extension is configured in the server's *EncryptedExtensions* message.

If all these three conditions cannot be satisfied at the same time, session resumption will be done in 1-RTT mode.

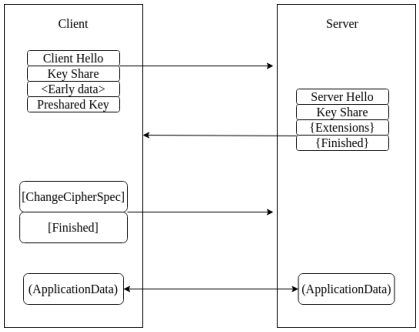


Fig. 2. 0-RTT handshanke of TLS 1.3.

The 0-RTT handshake process is illustrated in Fig. 2. During this process, the client firstly sends the early data along with the *ClientHello* message which contains the PSK identity information to resume the connection. The earky data is encrypted using keys derived from *client_early_traffic_secret* (<> in Fig 2). Then the server verifies the PSK information (eg. SNI value, ticket age and cipher suites). If verified as correct, the server replies a *EncryptedExtensions* message with acceptance of early data connection. Note that in 0-RTT mode, the server can either request or not request for client certificate. If not requested, the server would drop "Certificate" and "CertificateVerify" in its finished message. At this stage, the 0-RTT handshake has been completed.

B. Security issues of 0-RTT handshake

Due to the simplified process of the 0-RTT handshake protocol, it also faces some serious security challenges in actual deployment. 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, for session resumption, there are two standard ways that the server can verify the resumption secret provided by a client.

- Session Caches: It is the same method as for TLS1.2: the server stores the session. 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].

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

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.

For the above two schemes, the common safety hazard is that they are neither forward secure nor replay resistant, especially in wireless scenarios where channels are open to all users.

The forward secrecy issue can be addressed 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 this way, 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. Even in 0-RTT mode, only the early data is encrypted by the resumption secret which guarantees a delayed forward secrecy. However, the replay attack remains a huge security threat to the 0-RTT handshake.

III. TOTP Based Encryption Key

As aforementioned, one major way to defend against the replay attacks is to constantly change the clients' authentication messages, let the leaked messages cannot be reused for authentication. One time password (OTP) is a powerful technique to address this issue. In our previous work [8], we proposed an extension for TLS 1.2 based on the HMAC-based One-time Password (HOTP) algorithm, which allows secured client authenticate and client identity protection. In all types of OTP algorithms, a moving factor must be synchronised between the client and the server, which in HOTP algorithm is an event counter. However, in wireless IoTs, frequent factor sharing between the server and clients may bring information leakage and out of synchronization risks, which will seriously affect the effectiveness and security of the whole system. In order to address this issue, we propose in this paper using TOTP algorithm instead of HOTP algorithm.

A. TOTP Algorithm

The TOTP algorithm is standardised in RFC 6238 [7] as an extension of HOTP algorithm to support the time-based moving factor. Compared to the HOTP algorithm, TOTP algorithm is much less vulnerable to brute-force attacks [9] and other ways to guess the next OTP since the password guessing is highly cost and each TOTP password have a limited lifespan.

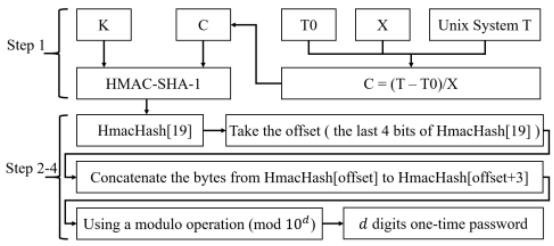


Fig. 3. Generation of a TOTP.

The generation procedure of a one time password using TOTP algorithm is given in Fig. 3, which can be mainly divided into four steps. In the first step, based a static shared secret key K and a dynamicly changing counter C, a 20 byte long HmacHash can be calculated using the HMAC-SHA-1 algorithm. Note that the counter C in TOTP algorithm can be calculated as follows:

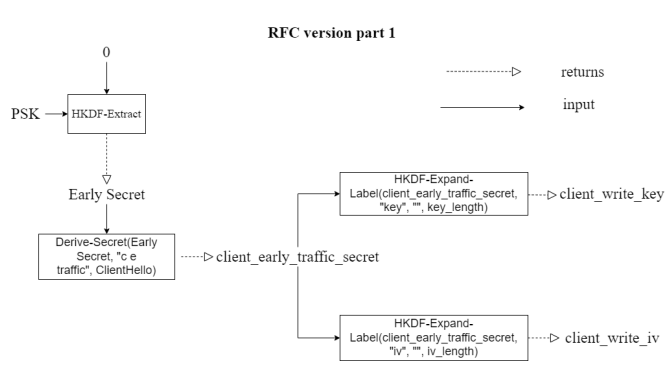
$$C = (T - T_0)/X, \tag{1}$$

where T is the actual time (Unix system time is used in most cases), T_0 is initial time of the session and X represents the time step. The second to fourth steps are the same as the HOTP algorithm [10], wherein the system take the last 4 bits of the HmacHash as an offset, then concatenate the bytes from HmacHash[offset] to HmacHash[offset+3] and covert it to decimal. In the last step, using a modulo operation ($mod\ 10^d$), a required d digits one-time password can be obtained. Using this algorithm, both the client and the server can obtain a consistent TOTP according to the synchronised Unix time.

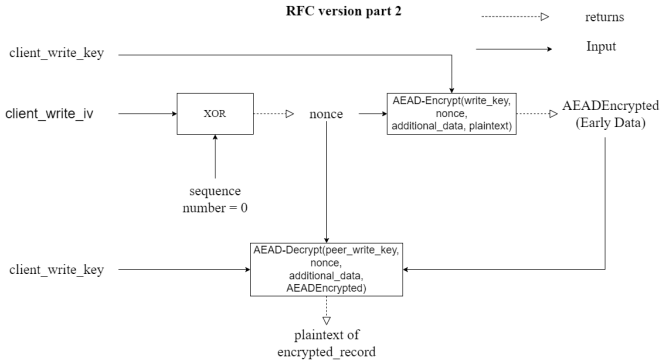
B. Key generation chain for the 0-RTT handshake

After getting the one-time password, we discuss how to incorporate it into the encryption key derivation process to make the 0-RTT TLS handshake resistant to replay attacks. In order to facilitate the reader's understanding of the key derivation and *EarlyData* encryption processes defined in [1], we prepared Fig. 4 to illustrate them. Specifically, the key derivation process is given Fig. 4(a), it can be observed that using the functions HKDF-Extract and HKDF-Expand [11] together with the PSK, the *client_early_traffic_secret* can be generated. Then by using the function HKDF-Expand-Label, two keys *client_write_key* and *client_write_iv* can be obtained. The encryption process of the *EarlyData* is given in Fig. 4(b), wherein the obtained *client_write_iv* XORs with a sequence number to get a *nonce*. Then, using the obtained *nonce* and the *client_early_traffic_secret*, the *EarlyData* can be encrypted through a HMAC based function AEAD-Encrypt. Moreover, the same key pair is used in the decryption process through the function AEAD-Decrypt.

In our proposed scheme, to guarantee its compatibility, we try to minimize the changes to the existing standard. Therefore, we propose to integrate the TOTP into the encryption process of the *EarlyData*, as shown in Fig. 5. Specifically, we propose to generate the TOTP using the *client_early_traffic_secret* and then XORs with the *client_write_iv* to generate a *pre-nonce*. In the next step, this *pre-nonce* XORs the sequence number to get the *nonce* that used in *EarlyData* encryption and decryption. In this way, since in different time steps the generated *nonce*s are different, the replayed information that does not arrive in the same time steps will be directly rejected by the server because it cannot be decoded. Thus, the risk of the server being attacked by replay attacks can be significantly reduced. In addition, the TOTP is only used in the encryption and decryption of the *EarlyData* and does not affect the information exchange in the handshake process, which can ensure good compatibility of our proposed scheme.



(a) Write_key and write_iv generation process.



(b) Encryption and decryption processes using Write_key and write_iv.

Fig. 4. *EarlyData* encryption and decryption processes defined in TLS 1.3.

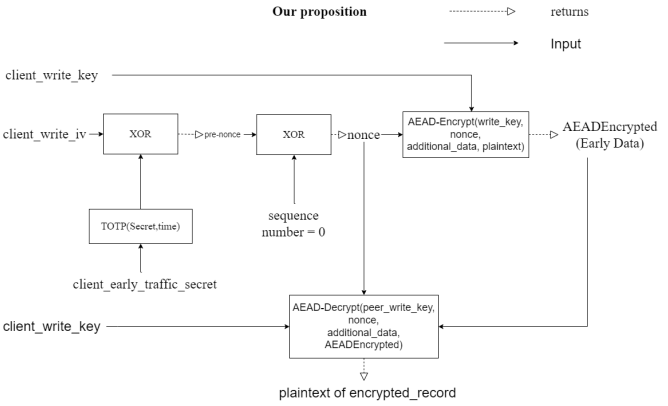


Fig. 5. Integration of TOTP in EarlyData encryption.

C. Security analysis

In scheme design, we try to improve the forward security and the ability to resist replay attacks of our scheme on the premise of ensuring good compatibility. Its good compatibility, including that the client and the server can independently generate consistent TOTPs and no changes have been made to the interaction process of the standardized 0-RTT handshake protocol, makes our proposed scheme suitable for large area systems, especially in unstable wireless scenarios.

For forward secrecy, as aforementioned, the Standardised protocol guarantees a delayed forward secrecy since the *EarlyData* is encrypted by the resumption secret. But with our proposed scheme, even the pre-shared key is leaked, it is still insufficient to decrypt the *EarlyData*, the malicious node also has to guess the expedition time of the packet and the TOTP time step, which guarantees strong forward secrecy.

For defence against replay attacks, since the TOTP algorithm uses time as the only variable, the uniqueness and consistency of key generation between the user and the server can be ensured. So that the *EarlyData* packets arriving within the same TOTP valid window can be decoded and accepted by the server. On the other hand, due to the higher transmission delay, the packets intercepted and replayed by malicious nodes are likely to arrive after the TOTP update and thus be denied by the server since the server cannot decode them. In this way, most of the replay attacks will be prevented due to the effective update of TOTP.

It is worth noting that the efficiency of our proposed scheme for replay attack defense depends on the server-side choice of TOTP update frequency. If the server-side TOTP update frequency is higher than the average transmission delay of the data packets, a large number of normal data packets will be rejected because they cannot be decoded by the server. However, if the update frequency of TOTP is much lower than the average transmission delay of data packets, the risk of system replay attack will be greatly increased. We will analyze and discuss the effect of TOTP update frequency on the system's defense against replay attacks in the next section combined with the simulation results.

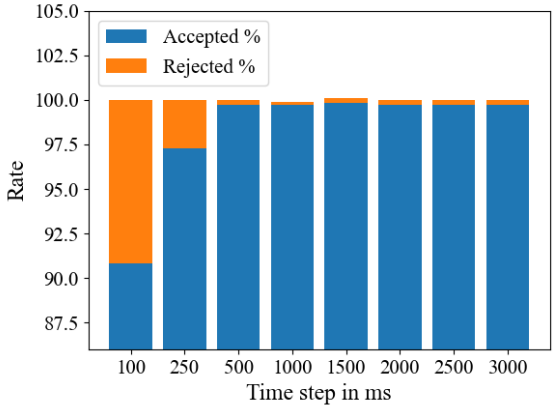
IV. Simulation Results

Since the TOTP algorithm is used in our proposed scheme, its update rate and the average delay of the network have a great influence on the simulation results. For a fair comparison, we set up two servers, one local in Paris and one in Lyon. Then, a client in Paris initiates 0-RTT handshake requests to both servers through OpenSSL [12]. Note that the uplink and downlink transmission rates in the network environment where the user is located are 134.989 Mbps and 129.80 Mbps respectively. Other parameters used in the simulation are summarized in Table I:

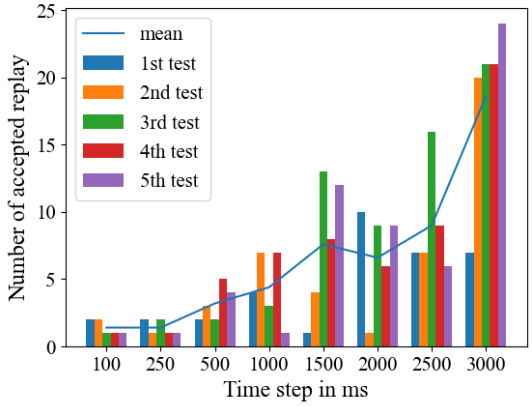
TABLE I
PARAMETERS USED IN SIMULATION

	Server 1	Sever 2
Location	Paris	Lyon
Up-link Speed	450 Mbps	390.55 Mbps
Down-link Speed	450 Mbps	519.55 Mbps
Ping delay	5 ms	35 ms

First, to verify the impact of TOTP update rate on system availability and resistance to replay attacks, we choose 8 different update rates : 100ms, 250ms, 500ms, 1000ms, 1500ms, 2000ms, 2500ms and 3000ms. For each update rate we launch

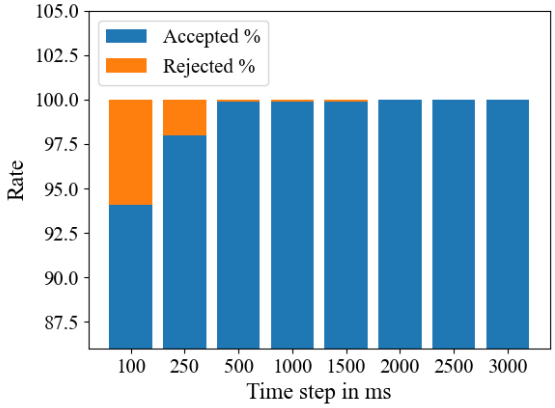


(a) *EarlyData* acceptance rate.

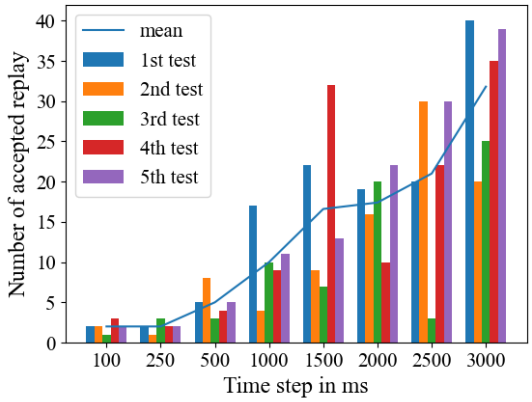


(b) Number of replayed *EarlyData* accepted by server within one TOTP update time window.

Fig. 6. Remote server replay resistant test results.



(a) *EarlyData* acceptance rate.



(b) Number of replayed *EarlyData* accepted by server within one TOTP update time window.

Fig. 7. Local server replay resistant test results.

5 sets of tests, in each set, 1000 legitimate *ClientHello* messages containing *EarlyData* are sent. Test results on the remote server and on the local server are illustrated in Fig. 6(a) and Fig. 7(a) respectively. It can be observed that as the TOTP update rate decreases, the acceptance rate of *EarlyData* continues to increase. After the TOTP update rate is reduced to 1000ms, more than 99% of *EarlyData* are accepted. At the same time, it can be observed that when the TOTP update rate is high, the transmission delay has a great influence on the acceptance rate of *EarlyData*. When the TOTP update rate is 100ms, the acceptance rate of *EarlyData* by the local server is higher than that of the remote server.

Using the same TOTP update rate, we test the resistance of our proposed scheme to replay attacks under different network delays. In the attack model of replay attack, one method is that the malicious node exhausts the server resources by sending a large number of intercepted data packets to the server, thereby making other benign nodes unable to access the server. Thus, after adopting the TOTP algorithm, we need to test the number of replayed packets that the server can accept within the time step of one TOTP. We also launch 5 sets of tests, in each set, 1000 times of replay packets are sent nonstop after the legitimate *ClientHello* message sent to the server. As illustrated in Fig. 6(b) and Fig. 7(b) respectively, it can be observed that for both local and remote servers reducing the TOTP update rate will increase the probability of a successful replay attack. And since the transmission delay to the local server is lower, it is relatively more vulnerable to replay attacks. However, for all TOTP update rates, only a small portion of replayed packets can be accepted by the server even with a 3000 ms TOTP time step(less than 40/1000 packets), which suggests that frequently changed TOTP can effectively resist replay attacks, since the replay packets arriving outside the same TOTP window will be rejected by the server because they cannot be decoded.

Finally, we test the ability of our proposed scheme to defend

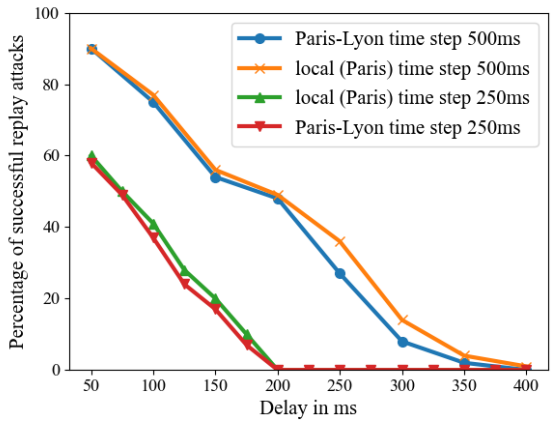


Fig. 8. Replayed *EarlyData* acceptance rate at servers with different transmission delays between the legitimate node and the malicious node.

against replay attacks in a relatively real network environment. In wireless IoTs, since the relative position between the legitimate node and the malicious node changes all the time and the fading of the wireless channel is random, which causes the delay of the replayed *ClientHello* is also random. We choose 8 different delays in the range of 50ms to 400ms to simulate the randomness of delays and test their success rate of replay attacks under two different TOTP update rates of 250ms and 500ms. The results are given in Fig. 8, each point represents a mean success rate of 100 times replay attacks. It can be observed that replayed *ClientHello* with a lower delay (50ms) leads to a higher attack success rate in all four cases. And as the delay increases, the successful rate of the replay attack gradually decreases. In comparison, servers with a lower TOTP update rate (500ms) or servers close to the malicious node (local) are more vulnerable to replay attacks, which suggests that in TOTP based schemes, a high TOTP update rate is the key to replay attack defending, especially for local servers.

Through the above three sets of tests, we can conclude that the efficiency of resisting replay attacks through TOTP is greatly affected by the TOTP update rate. Especially in the wireless IoT environment with unstable channel conditions and strong node mobility. Therefore, in order to improve the resistance to replay attacks, servers also need to continuously measure the communication delay of legitimates and malicious nodes to determine the optimal TOTP update rate to balance the availability and security of the system.

V. Conclusion

In this paper, we proposed a TOTP empowered TLS encryption algorithm for wireless IoTs, which can resist replay attacks and enhance the forward secrecy of the 0-RTT handshake. Specifically, the forward secrecy is significantly improved since to decrypt the captured *EarlyData* packet, beside the pre-shard key, the malicious node also have to guess the expedition time of the packet and the TOTP update time step; And for defence against replay attacks, the TOTP changes periodically on both server and client side, which ensures that only *EarlyData* packets arriving within the same TOTP valid window can be decoded and accepted by the server. In this way, due to the higher transmission delay, the packets intercepted and replayed by malicious nodes are likely to arrive after the TOTP update and thus be denied by the server since the server cannot decode them. Thus, most of the replay attacks will be prevented due to the effective update of TOTP. Moreover, in algorithm design, we made no changes to the interaction process of the standardized 0-RTT handshake protocol to guarantee the compatibility of our proposed scheme, which makes our proposed scheme suitable for large area wireless IoTs. Simulation results show that under the premise of choosing an appropriate TOTP update rate, our proposed scheme can effectively resist replay attacks while ensuring the processing efficiency of the system.

For future work, to adapt to different IoT applications, we will continue to work on an algorithm which can learn the communication delay of legitimate nodes and malicious nodes. And then can dynamically predict an optimal update rate of TOTP on the server-side to better improve the resistance to replay attacks while ensuring the availability of system.

References

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

[2] 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.

[3] X. Cao, S. Zhao, and Y. Zhang, "0-rtt attack and defense of quic protocol," in *2019 IEEE Globecom Workshops (GC Wkshps)*, 2019, pp. 1–6.

[4] 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. New York, NY, USA: Association for Computing Machinery, 2019, p. 166–168.

[5] M. Cebe and K. Akkaya, "A replay attack-resistant 0-rtt key management scheme for low-bandwidth smart grid communications," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[6] N. Aviram, K. Gellert, and T. Jager, "Session resumption protocols and efficient forward security for tls 1.3 0-rtt," in *Advances in Cryptology – EUROCRYPT 2019*, Y. Ishai and V. Rijmen, Eds. Cham: Springer International Publishing, 2019, pp. 117–150.

[7] M. View, J. Rydell, M. Pei, and S. Machani, "TOTP: Time-Based One-Time Password Algorithm," RFC 6238, May 2011.

[8] B. Hamdane, A. Serhrouchni, A. Montfaucon, and S. Guemara, "Using the hmac-based one-time password algorithm for tls authentication," in *2011 Conference on Network and Information Systems Security*, 2011, pp. 1–8.

[9] L. R. Knudsen and M. J. B. Robshaw, *Brute Force Attacks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 95–108.

[10] M. View, D. M'Raihi, F. Hoornaert, D. Naccache, M. Bellare, and O. Ranen, "HOTP: An HMAC-Based One-Time Password Algorithm," RFC 4226, Dec. 2005.

[11] D. H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," RFC 5869, May 2010.

[12] I. Ristic, *Openssl cookbook: A guide to the most frequently used openssl features and commands*. Feisty Duck, 2013.