Replay Attack in TLS 1.3 0-RTT Handshake: Countermeasure Techniques

1st M.E Abdelhafez
School of Science and Engineering
Malaysia University of Science and Technology (MUST)
Selangor, Malaysia
https://orcid.org/0000-0002-7112-6706

2nd Sureswaran Ramadass
School of Science and Engineering
Malaysia University of Science and Technology (MUST)
Selangor, Malaysia
sures@nav6.org

3rd Mohammed S. M. Gismallab
Center for Communication Systems and Sensing
King Fahd University of Petroleum and Minerals (KFUPM)
Dhahran, Saudi Arabia
https://orcid.org/0000-0001-8743-6859

2023 IEEE 6th International Conference on Electrical, Electronics and System Engineering (ICEESE) | 978-1-6654-5371-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICEESE56169.2023.10278190

Abstract— **Transport Layer Security (TLS) is an Internet Engineering Task Force (IETF) standard protocol commonly used on the Internet nowadays. TLS 1.3 the latest version published in August 2018 introduced a new mode to perform the handshake in zero round-trip time (0-RTT) which led to a significant improvement in the protocol performance. 0-RTT handshake mode allows the client to establish the connection and send the application data in the first message (ClientHello) without waiting for the server to send a response. But it also opens a window for replay attacks. The attack occurs when a valid 0-RTT handshake message is captured and sent again to the server causing the server to perform the same process again according to the application data carried in the message. Several anti-replay protection techniques were introduced to prevent or mitigate the replay attack. This paper sheds light on replay attacks countermeasures for TLS 1.3 0-RTT handshake mode and the most recent proposed methods. The strength and limitations of these countermeasures are discussed. In addition, the paper deliberates on the challenges such countermeasures impose in a distributed environment, for instance, multiple servers in a cloud computing environment or a content delivery network (CDN).**

Keywords— TLS 1.3, replay attack, 0-RTT, handshake

I. Introduction

Transport Layer Security (TLS) protocol is used commonly in web applications and several other applications such as DNS and email systems. TLS 1.3 is the latest version standardized by the Internet Engineering Task Force (IETF) in RFC 8446 [1]. A notable improvement was introduced in this version, enhancing the protocol performance significantly. A new feature was adopted in the protocol called zero round-trip time (0-RTT) handshake mode to allow the client to send the application data within the handshake phase carried in the ClientHello message. 0-RTT reduces the necessary time to establish a secure connection and send the application data without waiting for an acknowledgement from the server side. An experiment conducted in a real environment showed the time reduced by 44.7% [2] compared to the normal handshake which uses 1 round trip time (1-RTT) considering the network interface buffer speed and the processing time.

The significance of the latency in TLS emerged in the associated cost in some applications, taking into account there is a substantial percentage of resumed connections estimated by 40% [3]. On the other hand, amazon's revenue was reduced by 1% based on a study [4] due to a 100ms delay in the main page only. Accordingly, mitigating the security issues with the usage of 0-RTT is crucial and it is indeed important for time-sensitive applications such as self-driving cars and some IoT applications [5]–[7].

Security flaws were found in TLS 1.3 associated with the use of 0-RTT handshake mode. The aims of this paper are to review the anti-replay protection techniques that were used to prevent or mitigate replay attacks. It also investigates the replay attacks countermeasures and discusses their proposed solutions. The rest of the paper sections are organized as follows: Section II describes the handshake and record layer in TLS 1.3. Section III discusses replay attacks. Section IV presents the TLS 1.3 0-RTT handshake mode replay attack countermeasures. Section V shows the conclusion of the paper.

II. Handshake and Record Layer in TLS 1.3

The handshake phase is responsible for negotiating and exchanging the necessary information to establish an encrypted channel between the communication peers, and the record layer (also known as the record protocol), employs the previously exchanged encryption information to encrypt the actual transmitting data following structured encryption suites. Figure 1 shows the phases of establishing an encrypted communication channel and Figure 2 illustrates the standard encryption suites for TLS 1.3 respectively.

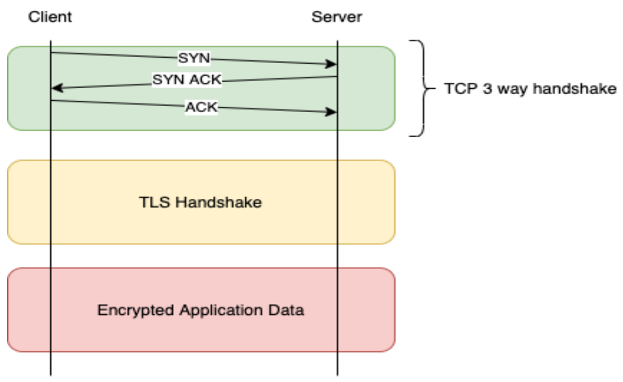

Fig. 1. TLS Connection Phases

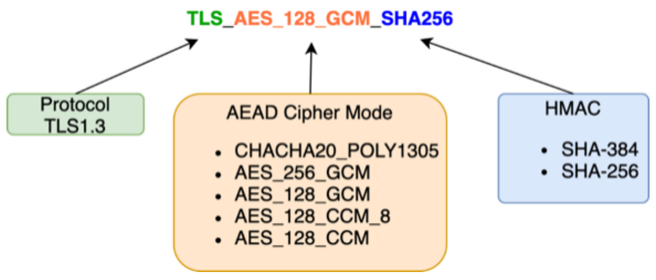

Fig. 2. TLS 1.3 Encryption Suites

The fresh handshake (also known as an initial handshake) is performed for the first time when the client sends a ClientHello message to the server carrying the necessary information and identifies which encryption suite is going to be used specifically which cipher mode and which hash function going to be used for authentication, the server in the other side will send back a ServerHello message to complete the negotiations and confirm the proposed encryption suite. The fresh handshake will require 1-RTT to complete the negotiation and start sending encrypted application data.

In TLS 1.3 there are two modes to resume the connection. (1) full resumption handshake which will require 1-RTT to resume the connection and start sending encrypted application data. (2) 0-RTT resumption handshake requires zero round trip time to resume the connection and send the encrypted application data at the same time. In both modes, the server will recognize the connection when the client provides a value called “Session-Ticket key” within the ClientHello message during the resumption. That is because the session-ticket key is associated with a private key on the server side that will be used to decrypt the provided ClientHello message by the client during the resumption. The session-ticket key and its private key on the server side will be created during the initial handshake and the server will share the session-ticket key with the client to be used in the coming resumption when the session is dead or going to time-out. Figure 3 (A) and Figure 3 (B) explain the full resumption HS and 0-RTT resumption HS respectively [8] .

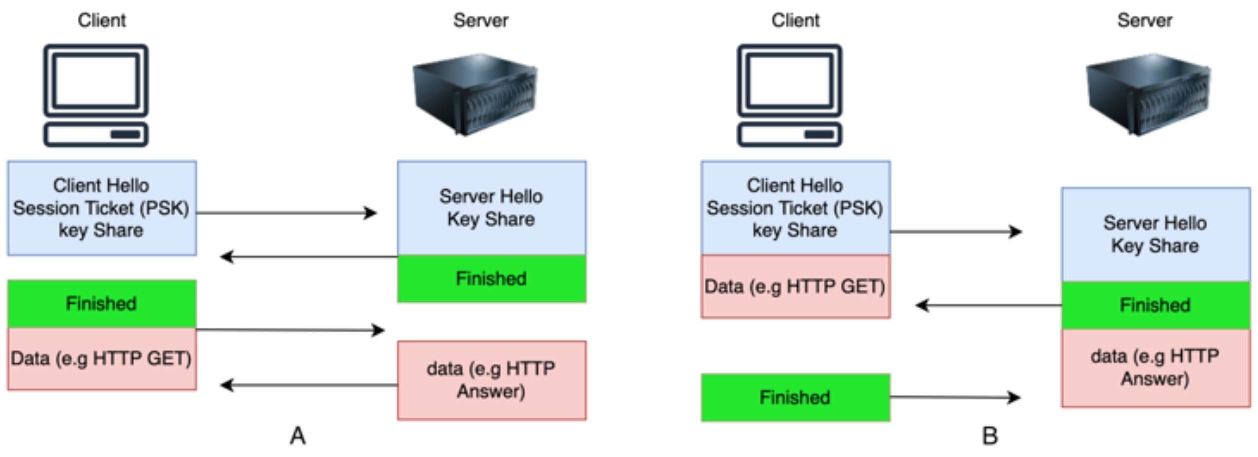

Fig. 3. Resumption Handshake Modes

III. Replay Attack

A replay attack occurs when an attacker intercepts a valid message sent by a client and replays it to the server at a later time, causing the server to perform the same action again as much as the server receives the message [9]. In the context of TLS 1.3 0-RTT handshake mode, a replay attack can occur if an attacker intercepts and stores the ClientHello message sent by the client, and then replays it to the server in a new connection attempt later. The replayed message is always valid because it was encrypted using the original key which is recognizable by the server. Figure 4 below illustrates the replay attack in TLS 1.3 0-RTT handshake mode.

A real attack conducted for experiment purposes [10] uses a real server and client setup proving that the replay attack in 0-RTT may result in a manipulation of a money transaction system where the transaction can be executed successfully many times.

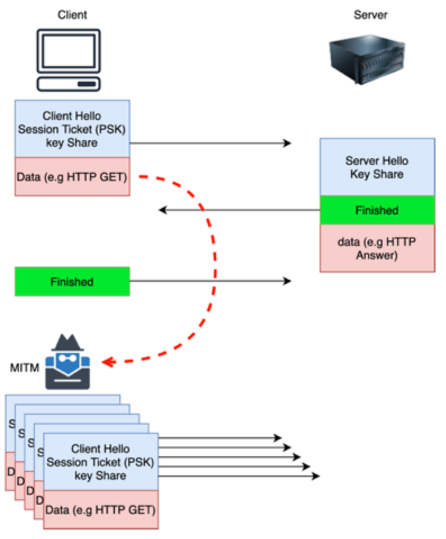

Fig. 4. Replay Attack TLS 1.3 0-RTT Resumption Handshake Mode, HTTP GET protocol function is an example of application data.

IV. TLS 1.3 0-RTT Handshake Mode Replay Attack Countermeasures

The replay attack in TLS 1.3 0-RTT handshake mode has been studied, and several anti-replay mechanisms have been proposed. Some mechanisms are practically implemented and proven, others are theoretical. in this section, we will discuss the recent work and its challenges.

A. *Freshness checks*

This technique employs the timestamp as an extended value to be sent by the client included in the first flight message (ClientHello message during the resumption handshake) represents when the message was sent, so the server will recognize the replayed message by looking at the provided timestamp and reject any message that is not within specific time frame window [1], [10], [11]. Figure 5 illustrates the freshness checks mechanism.

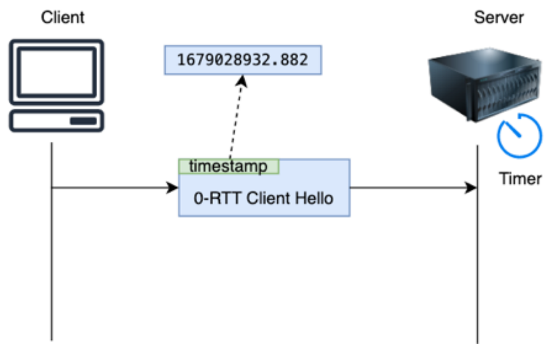

Fig. 5. Freshness Checks Mechanism

It is a straightforward technique that requires implementation on the server and the client-side, and it can reduce the possibility for the attack to happen. However, this time frame window is enough to perform a replay attack.

B. *Client-Hello Recording*

In this technique, the server will keep all received ClientHello messages, in a way when the server receives the first resumed 0-RTT ClientHello message, it will be listed and stored. in case of a replay attack, the server will look up the recorded messages list and rejects those matches or found in the list [1], [10], [11].

To implement this technique, the changes need to be made only on the server side. However, storing messages will increase memory utilization incrementally with the number of sessions and it will be very complicated to synchronize the list of the recorded ‘ClientHello’ message in a complex and distributed environment.

C. *Single-Use Tickets*

This mechanism takes advantage of the session ticket as it is associated with a private key called "session ticket key" in the server used to decrypt the application data that is carried in the ClientHello message. the server will delete the session ticket key immediately after the handshake which makes the replayed message fail to complete the handshake successfully because the server is unable to decrypt the message [1], [11] Figure 6 explains the session ticket mechanism.

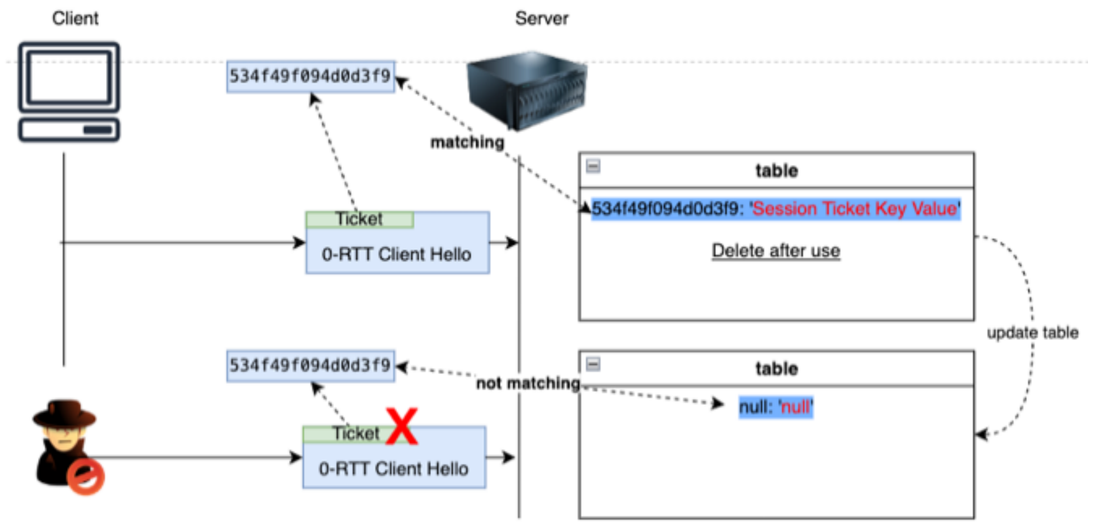

Fig. 6. Single-Use Ticket Mechanism

This technique requires synchronization in order to run in a complex environment such as a distributed environment where there are multiple servers in a cloud computing environment or a content delivery network. Besides consistent storage is required for this technique; all this could open a potential window for a replay attack specifically in a distributed environment [12], [13].

D. *Application Profile*

This technique is based on a profile defining the use of 0-RTT by the application. Every application protocol needs to use 0-RTT must have a profile that defines which functionality and which part of the application protocol should be exposed over 0-RTT. For example, in HTTP a “GET Request: idempotent” [14] can be used, which means if the client sends the request twice, the server will not do anything different than replay once [1].

This method is based on a different layer (application layer) to achieve the anti-replay, while TLS (transport layer) remains as it is. This technique does not support distributed environment.

E. Separate API

This mechanism can be implemented on the server or the client side. The API will guarantee that the application data (functions) such as HTTPS will only send the messages in a safe method, however, this technique relies on the application layer protocols to tolerate the replay attack impact when 0-RTT mode has been utilized, and the client behavior cannot be controlled. This technique does not support distributed environment [1].

F. Puncture Pseudorandom Function (PPRF)

It allows the server to decrypt the data in the first flight (0-RTT message) only once, it is automatically considered as protection from the replay attack, because the server can process the message once, but it was proposed to achieve forward secrecy for the data in the 0-RTT resumption handshake message, and it was proposed first time by (Green & Miers, 2015) to utilize puncturable encryption. However, their method showed long processing times for their construction. [11] have proposed the same scenario, and his implementation showed unsuitable for high traffic network. [15] has proposed four solutions based on (Session Tickets, Session Caches, sRSA-based PPRF, Tree-based PPRF) to provide forward secrecy. In general, PPRF does not support a distributed environment it requires a synchronization technique to run in a complex environment [16]–[18].

G. Universal SSL

Introduced by Cloudflare company to allow session resumption in multiple hosts, it supports two types of standard session resumption based on the RFCs: Session IDs RFC 5246 [19] and Session Tickets RFC 5077 [20] which are Obsoleted by RFC 8446 [1] This technique employs a "Memcached" cluster to cache all negotiated sessions by inserting it into Memcached indexed (distributed memory) by the sessionID [21], [22].

Memcached is a general-purpose distributed memory-caching system. It caches data in the RAM to accelerate data reading and is considered as a distributed database system. However, multiple Memcached servers are unaware of each other, there is no crosstalk, no synchronization or replication, and Universal SSL does not support TLS 1.3 [23], [24].

H. Just-in-Time Shared Keys (JIT-SK)

It employs the synchronized pseudorandom number generator (PRNG) to prevent replay attacks and provide forward secrecy by generating dynamically changing keys for each session to secure 0-RTT messages. Cannot reuse these keys multiple times. Accordingly, it will prevent replay and guarantee forward secrecy [25].

The author's proof of concept implementation uses PRNG, named Quantum Entropy Expansion and Propagation (QEEP), and WolfSSL libraries [26] for TLS 1.3. However, JIT-SK has no feature to run or to prevent the replay attack in a distributed environment.

Table 1 below illustrates the studied anti-replay mechanisms in this paper showing the features and the limitations of each.

TABLE I. ANTI-REPLAY MECHANISMS FOR 0-RTT HANDSHAKE MODE IN TLS 1.3

Anti-replay	***Protection type***	***Operation Layer***	***Operation side***	***Implementation in TLS 1.3***	***High network traffic***	***Distributed Environment Support***	***Fault tolerance***
Separate API	Prevention	Application + TLS	TLS 1.3 Protocol (Server or client side)	no	yes	no	no
Application Profile	Prevention	Application + TLS	TLS 1.3 Protocol (server side)	no	yes	no	no
Single-Use Tickets	Prevention	TLS	TLS 1.3 Protocol (server side)	yes	yes	no	no
Client-Hello Recording	Prevention	TLS	TLS 1.3 Protocol (server side)	yes	yes	no	no
Freshness checks	Prevention	TLS	TLS 1.3 Protocol (server side)	no	yes	no	no
PPRF	Prevention	TLS	TLS 1.3 Protocol (server side)	no	no	no	no
Universal SSL	Prevention	TLS 1.3 not supported	TLS 1.2 (Memcached Server)	no	yes	yes	no
JIT-SK	Prevention	TLS	Server and Client side	yes	yes	no	no

V. Conclusion

Enterprises that use TLS to provide integrity and confidentiality to their applications are highly concerned about the number of sessions that can be handled by the protocol, in the big scale applications the sessions can go up to millions of sessions per second. accordingly, supporting high traffic is a crucial factor in the proposed anti-replay mechanisms for a 0-RTT handshake in TLS 1.3. In this context, the JIT-SK mechanism is more promising to support high traffic networks, however, it might not be interoperable with the existing systems that use TLS 1.3. An increase in the computing resource utilization associated with these mechanisms is expected subsequently which will result in a cost, for instance in a form of power consumption. Research is required in the area of resource optimization and supporting high traffic networks in order to obey the demand of industries to fit their businesses.

References

[1] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," *(No. rfc8446)*, Aug. 2018. https://www.rfc-editor.org/rfc/rfc8446 (accessed Jan. 15, 2023).

[2] M. E. Abdelhafez, S. Ramadass, M. Abdelwahab, and B. G. D. A. Madhusanka, *A Study of Connection Speeds in Transport Layer Security Version 1.3 (TLS 1.3) Using Different Handshake Modes*, vol. 357 LNNS. 2022. doi: 10.1007/978-3-030-91738-8_47.

[3] S. Nick, "Introducing Zero Round Trip Time Resumption (0-RTT)," *Cloudflare*, Mar. 15, 2017. https://blog.cloudflare.com/introducing-0-rtt/ (accessed Apr. 12, 2023).

[4] N. Shalom and Y. Einav, "Amazon found every 100ms of latency cost them 1% in sales," *Gigaspaces*, 2019. https://www.gigaspaces.com/blog/amazon-found-every-100ms-of-latency-cost-them-1-in-sales (accessed Sep. 30, 2022).

[5] D. Marchsreiter and J. Sepulveda, "Hybrid Post-Quantum Enhanced TLS 1.3 on Embedded Devices," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, IEEE, Aug. 2022, pp. 905–912. doi: 10.1109/DSD57027.2022.00127.

[6] V. Dimov, E. Kirdan, and M.-O. Pahl, "Resource tradeoffs for TLS-secured MQTT-based IoT Management," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, IEEE, Apr. 2022, pp. 1–6. doi: 10.1109/NOMS54207.2022.9789806.

[7] G. Restuccia, H. Tschofenig, and E. Baccelli, "Low-Power IoT Communication Security: On the Performance of DTLS and TLS 1.3," in *2020 9th IFIP International Conference on Performance Evaluation and Modeling in Wireless Networks (PEMWN)*, IEEE, Dec. 2020, pp. 1–6. doi: 10.23919/PEMWN50727.2020.9293085.

[8] M. Fischlin, "Stealth Key Exchange and Confined Access to the Record Protocol Data in TLS 1.3," *Cryptology ePrint Archive*, 2023.

[9] H. Xue, Y. Yang, J. Liu, Z. Xu, and K. A. Dankanti, "Reverse fast replay attack tunnel lighting system based on CAN bus," in *Second International Conference on Electronic Information Engineering, Big Data, and Computer Technology (EIBDCT 2023)*, X. Kong, Ed., SPIE, May 2023, p. 24. doi: 10.1117/12.2674736.

[10] A. García Alguacil and A. Murillo Moya, "Playback: A TLS 1.3 Story," Aug. 2018. Accessed: Jan. 17, 2023. [Online]. Available: https://labs.portcullis.co.uk/blog/playback-a-tls-1-3-story/

[11] M. Fischlin and F. Günther, "Replay Attacks on Zero Round-Trip Time: The Case of the TLS 1.3 Handshake Candidates," *Proceedings - 2nd IEEE European Symposium on Security and Privacy, EuroS and P 2017*, pp. 60–75, 2017, doi: 10.1109/EuroSP.2017.18.

[12] M. Scott, "On TLS for the Internet of Things, in a Post Quantum world," *Cryptology ePrint Archive*, 2023.

[13] S. Hebrok *et al.*, "We Really Need to Talk About Session Tickets: A Large-Scale Analysis of Cryptographic Dangers with TLS Session Tickets," *Paderborn University*, 2023.

[14] R. T. Fielding and J. F. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," *Internet Engineering Task Force (IETF) (No. rfc7231)*, pp. 1–101, 2014, [Online]. Available: http://tools.ietf.org/html/rfc7231

[15] N. Aviram, K. Gellert, and T. Jager, "Session Resumption Protocols and Efficient Forward Security for TLS 1.3 0-RTT," *Journal of Cryptology*, vol. 34, no. 3, p. 20, Jul. 2021, doi: 10.1007/s00145-021-09385-0.

[16] S. Agrawal, E. Alpírez Bock, Y. Chen, and G. Watson, "White-Box Cryptography with Global Device Binding from Message-Recoverable Signatures and Token-Based Obfuscation," 2023, pp. 241–261. doi: 10.1007/978-3-031-29497-6_12.

[17] A. L. Putterman and E. Pyne, "Pseudorandom Linear Codes are List Decodable to Capacity," Mar. 2023.

[18] R. Yang, "Privately Puncturing PRFs from Lattices: Adaptive Security and Collusion Resistant Pseudorandomness," 2023, pp. 163–193. doi: 10.1007/978-3-031-30620-4_6.

[19] T. Dierks and E. Rescorla, "The transport layer security (TLS) protocol version 1.2," *(No. rfc5246)*, Aug. 2008. https://www.rfc-editor.org/rfc/rfc5246 (accessed Jan. 17, 2023).

[20] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State," *(No.*

rfc5077), 2008. https://www.rfc-editor.org/rfc/pdfrfc/rfc5077.txt.pdf (accessed Jan. 17, 2023).

[21] S. Nick, “Universal SSL: How It Scales,” *Cloudflare*, Oct. 2014. https://blog.cloudflare.com/universal-ssl-how-it-scales/ (accessed Jan. 17, 2023).

[22] L. Zi, “TLS Session Resumption: Full-speed and Secure,” Feb. 24, 2015. https://blog.cloudflare.com/tls-session-resumption-full-speed-and-secure/ (accessed Jan. 17, 2023).

[23] G. Lee, B. J. Kim, and E.-Y. Chung, “Exploring Replacement Policy for Memcached,” in *2020 International SoC Design Conference (ISOCC)*, IEEE, Oct. 2020, pp. 296–297. doi: 10.1109/ISOCC50952.2020.9332942.

[24] W. Cheng, F. Ren, W. Jiang, and T. Zhang, “Optimizing the Response Time of Memcached Systems via Model and Quantitative Analysis,” *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1458–1471, Sep. 2021, doi: 10.1109/TC.2020.3011619.

[25] Eslam G. AbdAllah, Changcheng Huang, and RandyKuang, “Generating Just-in-Time Shared Keys (JIT-SK) for TLS 1.3 Zero RoundTrip Time (0-RTT),” *Int J Mach Learn Comput*, vol. 12, no. 3, May 2022, doi: 10.18178/ijmlc.2022.12.3.1086.

[26] wolfSSL, “TLS 1.3 protocol support: Wolfssl embedded SSL/TLS Library. wolfSSL.” wolfSSL, 2022. Accessed: Jan. 18, 2023. [Online]. Available: https://www.wolfssl.com/docs/tls13/