Article

Stateless One-time Authenticated Session Resumption in TLS Handshake Using Paired Token

Byoungcheon Lee [1,†,‡] 0000-0002-1741-4192

[1] Department of Information Security, Joongbu University, 305 Dongheon-ro, Goyang-si, 10279 Korea; sultan@joongbu.ac.kr

Abstract: Transport Layer Security (TLS) is a cryptographic protocol that provides communications security between two peers and it is widely used in many applications. To reduce the latency in TLS handshake session resumption using pre-shared key (PSK) had been used. But current methods in PSK mode handshake uses a fixed session key multiple times for the lifetime of session ticket. Reuse of fixed session key should be very careful in the point of communications security. It is vulnerable to replay attacks and there is a possibility of tracking users. Paired token (PT) is a new secondary credential scheme that provides pre-shared key in stateless way in client-server environment. Server issues paired token (public token and secret token) to authenticated client. Public token represents signed identity of client and secret token is a kind of shared secret between client and server. Once client is equipped with PT, it can be used for many symmetric key based cryptographic applications such as authentication, authorization, key establishment, etc. It was also shown that it can be used for one-time authenticated key establishment using the time-based one-time password (TOTP) approach. In this paper we apply the PT and TOTP approach to TLS to achieve stateless one-time authenticated session resumption. Server executes full handshake of TLS 1.3 and issues PT to authenticated client. Then client and server can execute one-time authenticated session resumption using PT in stateless way in server side. In every runs of session resumption distinct session keys are established that the same PT can be used safely for longer lifetime. If anonymous PT is used with renewal issuing, user privacy, untraceability and forward security can be achieved easily. It will provide a huge performance gain in large-scale distributed services.

Keywords: Transport Layer Security; Handshake; Session resumption; Paired token; Stateless; One-time authenticated session resumption; Privacy; Untraceability

1. Introduction

Transport layer security (TLS) [1] is a cryptographic protocol that provides end-to-end communications security and it is widely used in many applications in the real world. The main design goal of TLS is providing authentication, confidentiality, and integrity in end-to-end communications. It consists of two sub-protocols; handshake protocol and record protocol. In handshake protocol client and server authenticate each other and establish a secure session key, and then in record protocol all end-to-end communications are encrypted with the secure session key.

The full handshake protocol of TLS is computationally expensive due to certificate-based authentication and Diffie-Hellman key exchange. In recently released TLS 1.3 (RFC8446) [1] reducing latency in handshake protocol was a hot issue. Main approaches for reducing latency in handshake were reducing round trip time (RTT) and using session resumption with pre-shared key (PSK). If PSK mode is enabled in TLS 1.3, server issues NewSessionTicket to authenticated client at the end of a successful full handshake and client stores session key and NewSessionTicket. In subsequent connection requests client sends NewSessionTicket in PSK extension, then server can recover the previous session key and can skip the heavy full handshake.

NewSessionTicket contains PSK identity, either session identifier in session cache or self-encrypted PSK in session ticket. Session cache approach is not practical in large-scale distributed service, since it requires the management of session cache and the session key retrieval is a stateful operation. Self-encrypted PSK in session ticket [2] approach has the advantage that it does not require server-side state, but it reuses a fixed session key multiple times during the lifetime of the session ticket. Reuse of fixed session key should be very careful in the point of communications security. It is also vulnerable to replay attacks [3] and denial of service (DOS) attack. There is a possibility of tracking users [13].

Quick UDP Internet Connections (QUIC) is Google's transport layer security protocol that provides secure connections over UDP [8]. To enhance the performance of handshake protocol it provides quick session resumption using public key cryptography. Recently there is an approach to combine TLS 1.3 and QUIC together, i.e., replace the handshake of QUIC with the TLS handshake and PSK-based session resumption [9–12].

Paired token (PT) is a new secondary credential scheme that provides stateless PSK in client-server environment [15,16,19]. Assume that there is an independent authentication system using some primary credential. Server authenticates the client with the primary credential and then issues paired token (public token and secret token) to authenticated client as a secondary credential. Public token has the role of signed identity that represents the authenticated state of client. Secret token is a kind of shared secret between client and server with a special property that server can compute secret token anytime from a given public token, thus server does not need to store issued client tokens. This feature provides the stateless management of client credential in server side. PT can be applied to many symmetric key based cryptographic applications such as authentication, authorization, secure communications, etc. Specially it was shown that it can be used for one-time authenticated key establishment using the time-based one-time password (TOTP) approach. PT can provide identification of client (with public token), time-based one-time authentication of client and key establishment (with secret token) in a single logical step. This feature was applied to achieve stateless re-association in WPA3 [17,18].

In this paper we apply the PT-based one-time authenticated key establishment to TLS 1.3 to achieve stateless one-time authenticated session resumption. In our approach server issues PT to authenticated client after the full handshake is finished successfully, and then in subsequent connection requests client uses PT for session resumption. The resulting session resumption protocol establishes distinct session keys for every connections that the same PT can be used multiple times for longer period of lifetime. We show that the proposed session resumption protocol can improve the performance of TLS a lot. The proposed scheme has the following distinguished features.

1. It provides time-based one-time authenticated key establishment in session resumption in stateless way.
2. The same PT can be used multiple times for session resumption for longer period of lifetime.
3. It satisfies essential security features such as replay resistance, denial of service (DOS) resistance.
4. With the use of anonymous PT and secure renewal of PT, session resumption protocol can provide privacy, untraceability, and forward security.
5. In large-scale distributed environment it provides huge performance gain and scalability with the help of stateless service property.

This paper is organized as follows. Section 2 reviews TLS 1.3 and paired token. Section 3 presents the proposed stateless session resumption protocol in TLS. Section 4 provide security and performance analysis. Finally section 5 concludes the paper.

2. Related Works

2.1. TLS 1.3

Secure channel establishment protocols such as Transport Layer Security (TLS) are one of the most important cryptographic protocols that enables the security of Internet traffic. TLS provides authentication, confidentiality, and integrity in end-to-end communications. It consists of two sub-protocols; handshake protocol and record protocol. In handshake protocol client and server authenticate each other and establish a secure session key, and then in record protocol all end-to-end communications are encrypted with the secure session key. There are two handshake protocols; full handshake protocol for the first time connection and session resumption protocol for the efficient handshake with revisiting clients.

The full handshake protocol of TLS is computationally expensive due to certificate-based authentication and Diffie-Hellman key exchange. In recently released TLS 1.3 (RFC8446) [1] reducing latency in handshake protocol was a hot issue. Main approaches for reducing latency in handshake were reducing round trip time (RTT) and using session resumption with pre-shared key (PSK) [2].

Figure 1 shows the full handshake protocol in TLS 1.3. Full handshake consists of 3 parts; key exchange, server parameter transport, and authentication. Key exchange is the first part of the protocol which enables to generate fresh session key using Diffie-Hellman key agreement. Server parameter transport is used to send necessary server parameters to client. Authentication part is used to provide mandatory server authentication and optional client authentication by exchanging certificates and digital signatures. It is a heavy handshake due to public key computations in certificate verification, signature verification, and DH key agreement, and required round trip time is 2. Heavy full handshake is inevitable in initial handshake, but repeating it for every connections is not a good idea.

```
       Client                                               Server

Initial Handshake:
       ClientHello
       + key_share               -------->
                                                       ServerHello
                                                       + key_share
                                             {EncryptedExtensions}
                                             {CertificateRequest*}
                                                    {Certificate*}
                                              {CertificateVerify*}
                                                        {Finished}
                                 <--------     [Application Data*]
       {Certificate*}
       {CertificateVerify*}
       {Finished}                -------->
                                 <--------      [NewSessionTicket]
       [Application Data]        <------->      [Application Data]
```

Figure 1. Full handshake and issuing NewSessionTicket in TLS 1.3 [1].

```
Subsequent Handshake:
       ClientHello
       + key_share*
       + pre_shared_key          -------->
                                                       ServerHello
                                                  + pre_shared_key
                                                      + key_share*
                                             {EncryptedExtensions}
                                                        {Finished}
                                 <--------     [Application Data*]
       {Finished}                -------->
       [Application Data]        <------->      [Application Data]
```

Figure 2. Session resumption using PSK in TLS 1.3 [1].

2.2. *Session Resumption*

If PSK mode is enabled in TLS 1.3, efficient session resumption can be used. Figure 2 shows the session resumption protocol using PSK. In figure 1 note that server issues NewSessionTicket to client at the end of a successful full handshake and client stores session key and NewSessionTicket. In subsequent connection requests client sends NewSessionTicket in PSK extension, then server can recover the previous session key and can skip the heavy full handshake. NewSessionTicket contains PSK identity, either session identifier in session cache or self-encrypted PSK in session ticket.

In session cache approach server stores resumption secrets of recent sessions and issues their unique lookup keys to clients. It requires stateful management of PSKs in the form of database or cache. Session cache approach is not practical in large-scale distributed service, since it requires the management of session cache and the session key retrieval is a stateful operation.

In session ticket approach server has a long-term symmetric encryption key called the session ticket encryption key (STEK). Instead of storing client's resumption secret in a local database, the server encrypts it with the STEK to create a session ticket and gives it to client [2]. If client requests session resumption with the session ticket, server can decrypt it to retrieve the resumption secret, thus local management of PSK is not necessary. Session ticket approach has the advantage that it does not require server-side state, but it reuses a fixed session key multiple times during the lifetime of the session ticket. Reuse of fixed session key should be very careful in the point of communications security. It is also vulnerable to replay attacks [3] and denial of service (DOS) attack. There is a possibility of tracking users [13].

2.3. *Stateless One-time Authenticated Key Establishment Using Paired Token*

In OAuth 2.0 bearer token [4,5] and JSON web token (JWT) [6] server issues static bearer token to authenticated client. If a client presents a valid bearer token to the server in subsequent requests, server verifies the token, accepts the authenticated state of client and provides proper services. In this case the same static token is sent to the server multiple times during the lifetime of token and it is considered as a kind of credential. So it is subject to eavesdropping and replay attack that the whole web service should be protected with secure communication channel such as https [7].

Paired token (PT) [15,16,19] was originally proposed to solve the static nature of bearer token authentication and secure channel requirement in web environment. PT is a new secondary credential scheme that provides stateless pre-shared key (PSK) more efficiently in a client-server environment. It can be used for time-based one-time authenticated key establishment multiple times without requiring secure communication channel. Assume that there is an independent authentication system between client and server using some primary credentials. The server authenticates the client using a primary credential and then issues a paired token (public token and secret token) to the authenticated client as a secondary credential. The public token has the role of signed identity of the client that represents the authenticated state of the client. A secret token is a kind of shared secret between the client and server with a special property such that the server can compute secret token any time from a given public token; thus, the server does not need to save issued client tokens. Here, we describe the scheme in the following two stages.

2.3.1. Initial Authentication and Issuing Paired Token

Let's consider a simplified authentication model between client and server. Client is registered to the server and has some primary credential for initial authentication. Assume that server has a master secret key K which is used for issuing tokens. It is used only inside the server and never exposed outside.

In initial authentication client logs into the server using primary credential, for example, using ID and password. If initial authentication is successful, server computes two tokens as follows.

1. Public token $T_p = G_{JWT}(K, Info)$: a normal JSON web token (JWT) on user's authorization information $Info$.
2. Secret token $T_s = G_{JWT}(K, T_p)$: a recursive JWT on the above public token T_p.

Here $G_{JWT}(K, Info)$ is an abstract notation of issuing process of a JWT [4–6,19]. It represents that server prepares user-specific authorization information $Info$ and puts it in the Payload, prepares proper Header, and generates a Signature, a HMAC value of the Header and Payload using the server's secret K,

$$Signature = HMAC(K, Header||Payload).$$

Then $Token = [Header.Payload.Signature]$ is a valid JWT issued to the user by the server. $Info$ is a JSON object prepared by the server that server can decide which information is included in $Info$ according to its policy. To issue JWT with limited lifetime, $Info$ can have information on issuing time and expiration time. If T_p is used after its lifetime has passed, it will be invalidated. T_s is computed from T_p and it will be computed frequently in the server in later authentication stages. Therefore no time information is included in the computation of T_s to make these repeated computations be easy with no lifetime check. $< T_p, T_s >$ is a paired token that T_s is valid only if T_p is valid.

Server sends $< T_p, T_s >$ to client through a secure communication channel. In the issuing stage of PT, secure communication channel is required to send PT to client securely. Note that initial authentication requires secure communication channel to send the password securely and issuing paired token can use the same secure communication channel. As a secure communication channel we can use https, or other custom secure channel. Client stores paired token securely in application or key storage. In web security environment paired token can be stored in browser storage such as local storage.

Public token T_p represents a signed identity of the user and will be sent to the server to provide identification of client. Note that its validity can be verified only by the server who has issued it, since the master secret key K is needed in verification. Secret token T_s is a kind of shared secret between client and server, and it will never be sent to server directly. Server does not need to save $< T_p, T_s >$ in DB, since T_p will be presented by the client and T_s can be computed anytime from T_p. Therefore T_s is an inherently shared secret with the server in a stateless way. Maybe server can decide to store T_p for logging purpose, but it will not be used in later authentication and key establishment stage.

2.3.2. One-Time Authenticated Key Establishment Using Paired Token

If client is equipped with PT as shown above, single message quick one-time authenticated key establishment is possible using PT. Now client equipped with $< T_p, T_s >$ wants to establish a fresh session key with the server.

Client gets current time t, computes a time-based one-time authentication value $auth$, computes one-time authenticated key k as follows.

$$auth = HMAC(T_s, t||T_p), \tag{1}$$

$$k = HMAC(T_s, t||T_p||\text{“}key\text{”}). \tag{2}$$

Here "key" is a pre-agreed label for key generation. Client sends $< T_p, t, auth >$ to the server.

Upon receiving $< T_p, t, auth >$, server first verifies the validity of $auth$ as follows.

1. Verifies the validity of T_p and identifies who is requesting authentication.
2. Gets its own current time and checks that client's request time t is within allowed limit (checking liveness of request to defend against replay attack).

3. Computes the secret token $T_s = G_{JWT}(K, T_p)$ from T_p and then verifies the validity

$$auth \overset{?}{=} HMAC(T_s, t||T_p). \tag{3}$$

If it is valid, server computes the same one-time authenticated key k in (2) using T_s. Here $auth$ is a time-based one-time authentication of client and proves the possession of T_s. It is an application of time-based one-time password (TOTP) scheme [14] to paired token scenario to prove the possession of T_s without exposing it. Thus the same PT can be used multiple times for one-time authenticated key establishment.

PT is a fully hash-based secondary credential scheme that its use in authentication protocol is very efficient. It is specially designed credential that can be used in 1-to-1 communication in client-server environment. It cannot be used in other communication channels with other servers.

3. Stateless One-Time Authenticated Session Resumption Using Paired Token

Since PT is a secondary credential scheme that provide stateless PSK in client-server environment and it can be used for one-time authenticated key establishment, it is a perfect solution for efficient session resumption in TLS. We apply the stateless PSK feature of PT to TLS handshake protocol to achieve efficient session resumption. In TLS 1.3 there are two handshake protocols; full handshake and session resumption. We will modify TLS 1.3 protocols in the following ways.

1. In the full handshake protocol PT is issued to authenticated client in NewSessionTicket.
2. In the session resumption protocol session key is established using PT.

3.1. *Full Handshake and Issuing PT*

Basically this stage is the same as the full handshake in TLS 1.3 except that PT is issued in NewSessionTicket message. Full handshake contains mandatory server authentication using server certificate and optional client authentication using client certificate. Key exchange in the first two moves are used to compute fresh shared session key between client and server, and it is used to send NewSessionTicket (PT) securely to client. Server computes public token and secret token as follows.

1. Public token $T_p = G_{JWT}(K, Info)$.
2. Secret token $T_s = G_{JWT}(K, T_p)$.

Server encrypts $< T_p, T_s >$ using the shared session key and sends it to client. Then client decrypts it to recover $< T_p, T_s >$ and saves it in client system.

In public token server can include any client-specific information such as IP address, OS information, browser information, issuing time, lifetime of the token, etc, according to its policy. It is signed by the server with the master secret key K and its validity is verifiable only by the server who knows K. Secret token is generated by recursively signing the public token with no lifetime information. It can be generated only by the server.

Since the same PT will be used multiple times for session resumption during the lifetime of PT, this process of issuing PT is executed very infrequently only in full handshake. If client is equipped with a valid PT, session resumption using PT will be used more dominantly.

3.2. *Stateless One-Time Authenticated Key Establishment Using Paired Token*

If a client is equipped with PT, secure session key can be established very quickly in stateless way using PT. This process is executed in every resumption requests and produces different session keys though the same PT is used repeatedly. Here we mainly focus on how PSK is computed from the protocol in client and server. Then PSK-based real session key establishment and extra services can rely on the underlying TLS 1.3 PSK functions. According to the requirement of forward security we consider the following 2 protocols.

3.2.1. Model 1: PSK from Authenticated Key Transport

Client and server already share the same secret token in stateless way, thus they can establish PSK in any pre-agreed manner. For example, client prepares current time t and computes

$$auth = HMAC(T_s, t||T_p), \tag{4}$$

$$PSK = HMAC(T_s, t||T_p||\text{"}key\text{"}). \tag{5}$$

Client sends $< T_p, t, auth >$ to server.

Upon receiving $< T_p, t, auth >$, server verifies the validity of $auth$ in the following steps.

1. Verifies the validity of T_p using K and identifies who is requesting session resumption.
2. Gets his own current time and checks that the time difference from client's request time t is within certain allowed limit (checking liveness to defend against replay attack).
3. Computes the secret token $T_s = G_{JWT}(K, T_p)$ from T_p and then verifies the validity

$$auth \overset{?}{=} HMAC(T_s, t||T_p). \tag{6}$$

If all the above verifications are successful, server computes the PSK using the same equation 5.

Here $auth$ is a time-based one-time authentication (proof of knowledge of T_s) of client. It can be generated only by the client who knows T_s and its validity can be verified only by the server who knows K. Any eavesdropping and replay of the protocol at another time will be determined to be invalid. Eavesdropping attacker cannot compute PSK, since it does not have T_s.

Now client and server share the same PSK and it can be used to compute secure session key for record protocol using the underlying PSK-mode functions of TLS 1.3. Note that real session keys will be distinct depending on client's request time t. This is a real 0-RTT handshake, since client can send encrypted application level data in the first request message.

This is a single message, one-way, deterministic key establishment. It will be very useful for lightweight client and intermittent communications such as IoT applications.

3.2.2. Model 2: PSK from Authenticated Key Establishment and DH

The above key establishment protocol does not provide forward security. If an attacker gets access to T_s in any way, he can compute all the previous PSKs during the lifetime of PT. Here we incorporate ephemeral Diffie-Hellman key exchange to achieve forward security.

Client prepares current time t and ephemeral DH key share g^x and computes

$$auth1 = HMAC(T_s, t||T_p||g^x). \tag{7}$$

Client sends $< T_p, t, g^x, auth1 >$ to server.

Upon receiving $< T_p, t, g^x, auth1 >$, server verifies the validity of $auth1$ in the following steps.

1. Verifies the validity of T_p and identifies who is requesting authentication.
2. Gets his own current time and checks that the time difference from client's request time t is within certain allowed limit (checking liveness to defend against replay attack).

3. Computes the secret token $T_s = G_{JWT}(K, T_p)$ from T_p and then verifies the validity

$$auth1 \overset{?}{=} HMAC(T_s, t||T_p||g^x). \tag{8}$$

If all the above verifications are successful, server prepare its ephemeral DH key share g^y and computes

$$auth2 = HMAC(T_s, t||T_p||g^{xy}), \tag{9}$$

$$PSK = HMAC(T_s, t||T_p||g^{xy}||\text{“}key\text{”}). \tag{10}$$

Server sends $< T_p, t, g^x, g^y, auth2 >$ to client.

Then client can compute g^{xy} and verify the validity of $auth2$. If it is valid, client computes the same PSK using the same equation (10). Now client and server share the same PSK and compute secure session key for record protocol using the underlying PSK-mode functions of TLS 1.3. This key establishment protocol provides forward security with one round of extra communication.

If client wants to send encrypted application level data in the first request message, it can do it by using a temporal PSK'

$$PSK' = HMAC(T_s, t||T_p||g^x||\text{“}key\text{”}). \tag{11}$$

Server can compute the same PSK' and decrypt it. Thus, it can provide 0-RTT handshake, though the first message does not provide forward security.

3.3. *Model 3: Privacy and Untraceability using One-Time Anonymous PT*

In the proposed scheme public token is sent to the server in plain communication channel as an identification of client, therefore eavesdropping network attacker can identify the client from the communication traffic. If privacy of client is a prime concern, server can issue anonymous PT with no client-specific information in public token. If server needs to identify the client from the anonymous public token, server can keep the record of the relation between client and issued anonymous public token inside the server. It will depend on server's policy.

If a fixed anonymous PT is used for long period of time, network attacker can try to trace the activity of the same client. To provide untraceability, server can renew anonymous PT in every connections; i.e., server issues new anonymous PT to the same client and it is used only once. Issuing renewed PT to already authenticated client through an already established secure channel is not heavy in performance and it can be managed in automatic way. Server can trace the identity of client if it keeps the record of renewed PTs, but network attacker cannot trace the renewed PTs. Note that anonymous PTs issued by the server have no inter-relation. Therefore if one-time anonymous PTs are used and previous PTs are discarded, forward security can be achieved very easily.

3.4. *Discussion on Further Extensions*

We consider further extension scenarios according to service requirements.

Renewal of session key. To improve the security of symmetric key cryptography, session key needs to be renewed periodically. In the proposed scheme renewal of session key is very easy. Server can request renewal of session key to client, and then client can start new key establishment using new current time. It can be executed automatically between client and server. Renewal of session key will not be exposed to network attackers, since it is executed inside the previously established secure communication channel.

Per-request secure communications. If service is provided in intermittent manner, managing a secure session like https can be a burden. For example, normal UDP based services such as DNS has intermittent nature. Transactions between IoT devices and IoT server are quite intermittent. Current DTLS is a UDP security protocol, but it still uses session-based TLS handshake, so it is not best suited in this scenario. If we apply the Model 1 key establishment protocol, it is very efficient that we don't need to keep the session information. It is a real per-request secure communication ready for use in intermittent connection.

Delayed full authentication. Assume that client and server quickly resumed a secure communication channel using PT, but want to check the authenticity of peers again at later time. Then they can execute full authentication again inside the already established secure channel. If a party cannot be successful in this delayed full authentication, the secure connection is stopped and new full handshake will be requested. Full handshake is computationally heavy since it requires secure session establishment, but full authentication inside a secure communication channel is not heavy.

Rotation of server's master secret key *K*. To improve the security of service, server can renew master secret key K periodically. If K was renewed and client has PT issued by using old K, then client's PT is invalid and client will be requested to execute full handshake again. If K is renewed while client was communicating with the server in a previously established secure session, then server can temporarily use two master secret keys, issue new PT using new master secret key, and then guide the client to restart the session. All this renewal process can be done automatically inside previously established secure communication channel.

Scalability in distributed multiple server environment. Distributed multiple server environment is common in large scale services with huge number of concurrent clients. In this case it is a hot issue how to provide TLS secure communication service in scalable way. In the proposed scheme scalable TLS service is possible if the master secret key K is shared among multiple servers. Since multiple servers are normally managed by the same entity, sharing K securely in the pool of TLS servers is a reasonable assumption.

Backward compatibility of TLS. TLS 1.3 provides session ticket-based session resumption and the proposed scheme provides PT-based session resumption. These two handshakes can be implemented independently and one of them can be selected according to choice. So, backward compatibility can be achieved easily.

Separation of key establishment from full handshake. If PT-based key establishment is acceptable due to its efficiency, separation of key establishment function from full handshake can be a better choice, though it is a significant change in protocol compared with TLS 1.3. In this case the role of full handshake is limited to authenticating peers and issuing PT safely to authenticated client. Key establishment function is executed in separate protocol using PT. In this scenario full handshake will be executed very infrequently. In most lifetime of TLS usage, PT-based stateless key establishment will be used dominantly. More in-depth discussion will be required on this matter in the research community.

Integration with application level authentication. TLS is a transport layer security protocol and it normally provides communications security which is independent from application layer security. In TLS server authentication is mandatory, but client authentication is optional. Client authentication using certificate is hard to manage in the real world.

PT is a secondary credential that is issued to client by the server after a primary authentication is successful. This authentication scenario is very similar to application level authentication. If any proper API can be provided that can connect TLS handshake with application level authentication such as ID and password, then TLS handshake can be integrated with more explicit application level authentication and client authentication can be used more easily.

4. Analysis

4.1. Comparison of Features

We compare the features of the proposed PT-based session resumption with the session ticket-based session resumption.

In the session ticket-based session resumption, the session key itself is a credential and the same session key is used again in the resumed session. Thus reuse of the fixed session key multiple times in different sessions should be very careful. When client requests session resumption by sending session ticket, there is no explicit authentication mechanism that server cannot distinguish replay attacks or forged requests. Therefore it is subject to replay attack and denial of service (DOS) attack. If the same session key is reused multiple times, attackers can trace the activities of clients and forward security cannot be achieved [13].

On the other hand, the proposed PT-based session resumption provides explicit identification of client (by verifying T_p), explicit one-time authentication *auth* (proof of knowledge of T_s), and establishment of one-time session key in a single logical step. In this case credential is the secret token and session keys are computed from the secret token. Every resumed sessions will have distinct session keys because of the time information. Thus the same PT can be used for session resumption multiple times for longer period of lifetime. Since every session resumption requests contain explicit one-time authentications of client, it cannot be reused at later time. Any trial of DOS attack with forged request will be detected and stopped in earliest time. If renewal issuing of anonymous PT is used like in Model 3, privacy and untraceability of client is guaranteed. If anonymous PT is used only once and previous PT is discarded, then forward security is achieved easily.

Both approaches provide 0-RTT secure communication feature, but the proposed PT-based session resumption provides 0-RTT with distinct session keys in every connected sessions.

Table 1: Comparison of features; session ticket-based vs. PT-based session resumptions.

Features	Session ticket	PT-based (Model 3)
Credential	session key	secret token
Session key	fixed	one-time
Multiple usage	Yes	Yes
Authentication	No auth	one-time auth
Anti-replay	No	Yes
Anti-DOS	No	Yes
Anti-tracing	No	Yes
Forward security	No	Yes
0-RTT	Yes	Yes

Table 2: Comparison of features; Model 1 - 3.

Features	Model 1	Model 2	Model 3
Forward security	No	Yes	Yes
Anti-tracing	No	No	Yes
Performance	High	Low	Low

Now we compare the features of the proposed 3 models of session resumption.

Model 1 is the simplest and most efficient key establishment scheme. It cannot provide forward security and anti-tracing, but it is a single message one-way key establishment started by client. If the server can accept the trustworthiness of client, or if the server normally checks the authenticity of client in other ways, then this kind of one-way key establishment is very useful. If the communication model is intermittent rather than requiring continuous connection, this is best suited secure communication model. For example, in IoT applications communications between IoT end devides and

IoT server is quite intermittent that keeping session is heavy. In this case per-request secure communication is possible using Model 1 session resumption.

Model 2 provides forward security by using extra exchange of DH key shares. It requires computation of modular exponentiations and 1 additional round of communication. If the communication model requires stable session connection for longer time and forward security is a prime concern, this is a reasonable session resumption model. Since the same PT is used multiple times, it cannot provide anti-tracing.

Model 3 provides both forward security and anti-tracing by using one-time anonymous PT, i.e. anonymous PT is used only once and renewed in every resumed sessions. PT is issued and managed in server and client automatically by the software and this renewal process is executed safely inside the already established secure session. Thus this kind of renewal of PT is not heavy in performance. If the server needs to trace the client from the anonymous PTs, it has to keep the records of renewal history, which will require stateful operation in server side.

4.2. *Security Analysis*

Unforgeability. Public token is a publishable information and it is sent to server over plain communication channel, while secret token is assumed to be kept secret in client. Server can compute secret token anytime from given public token using the master secret key K. Network attackers can try to collect client's public token and protocol messages, and then try to compute the secret token or even server's secret key K. Attackers can try to forge another session resumption request without having secret token. The security of this kind of attacks will depend on the security of the underlying HMAC function. Note that JWT contains a HMAC value signed by server. A successful forgery of JWT will be reduced to a successful forgery of the underlying HMAC without having master secret key.

Resistance to replay attack. Any kind of eavesdropping and replaying attack will be difficult since time-based one-time authentication $auth$ is sent to the server in the first move of request, and the server will check its validity. If $auth$ is not valid, server will stop the session resumption protocol and will require full handshake. Simple replay attack will not work at another time. Network attackers can try to concurrently replay other client's session resumption request, but they do not have any advantage since they cannot compute the real session key without the secret token.

Resistance to DOS attack. Attackers can try to attack the availability of service by sending incorrect requests to the server. But the server can detect this kind of attacks in the earliest time. Client's request message in the first move message contains time-based one-time authentication $auth$ and the verification process is very efficient with just a few hash computations. Server can detect and stop invalid session resumption requests from attackers very early and the attackers will be requested to start from the full handshake again.

Resistance to MITM attack. Man-in-the-middle (MITM) attack is an issue related with the full authentication. Client has to be able to verify the mandatory server authentication and the server has to issue PT only to authenticated client. Once client is equipped with PT correctly issued by the server, client and server have a special 1-to-1 secure communication channel. If server issued PTs correctly with different $Info$ at different time, every clients will have different PTs. Any attacker in the middle cannot intrude into the secure communication channel established using PT between client and server.

Resistance to session hijacking attack. Any trial of simple session hijacking will not be successful, since attacker cannot continue the protocol without knowing the secret token and established session key is never exposed over the communication network.

Secrecy of messages. In PT-based session resumption protocol, client sends $< T_p, t, auth >$ to server in plaintext to start the session resumption. This is the only message exposed to network attackers. All other messages can be sent in encrypted form using the 0-RTT feature of the proposed protocol.

Forward security. Since PT is a secondary credential that is intended to be used multiple times during its lifetime, providing forward security is important. We have shown that Model 2 and Model 3 provide forward security with different approaches.
Privacy and untraceability. We have shown that Model 3 provides privacy and untraceability by using one-time anonymous PT and renewal of anonymous PT.
Systems security. As described above network attackers who do not have the secret token cannot do many things. Considering the fact that the same PT is used multiple times for longer period of lifetime, attackers will be more interested in system attacks that can get PT itself; such as OS hacking, malicious software, hacking browser, hacking application, hacking storage systems for tokens, etc.

If attacker is successful in hacking the system and get the PT itself, then every attacks are possible, such as sniffing or spoofing the attacked clients. Since secret token is a secondary credential that has to be stored and used in the client system, its security will highly depend on the system security, key storage security, and application security. Therefore, client system has to be kept secure using the best practice in the point of system security. This is a common system security argument in which credential is stored and used in the system itself.

4.3. *Performance Analysis*

Session ticket-based session resumption and Model 1-3 session resumption schemes have different features as shown in previous section that direct comparison of performance is difficult. Session ticket-based session resumption has limitations since the same session key is reused multiple times in different sessions. It can be used together with DH key exchange, which will result different session keys in different sessions.

The proposed Model 1-3 schemes provide session resumption service in stateless way and the same PT can be used multiple times for session resumption during the lifetime of PT. Because of the time-based one-time authenticated key establishment each session will have different session keys. Model 1-3 have different features and different performances that proper choice is necessary.

In large scale distributed server environment, scalability is a prime issue. In the proposed PT-based session resumption protocol scalability can be achieved very easily if the master secret key K is shared among the multiple servers.

5. Conclusion

Paired token is a useful secondary credential scheme that can provide stateless PSK between client and server. It looks like a useful cryptographic ticket that is issued by a server to an authenticated VIP client. In this paper we modified TLS 1.3 protocol such that server issues PT to authenticated client in full handshake protocol, and then stateless time-based one-time authenticated session resumption using PT is used dominantly. It can replace the traditional session ticket-based session resumption. It can provide forward security and anti-tracing with enhanced performance and scalability. It is conceptually simple since the same PT can be used multiple times in safe way to establish secure sessions that are distinct depending on time.

We think that the proposed session resumption schemes can be used in TLS 1.3 to improve security and performance. More in-depth discussions on various practical matters are required in the TLS research community.

Funding: This research received no external funding.

Acknowledgments: This work was conducted while the author visited the University of Alabama at Birmingham as a sabbatical research fellow with the support of Joongbu University in 2020. Special thanks to professor Yulian Zheng for the favor of invitation and many helpful discussions on this work.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Eric Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3, RFC8446, 2018. Available online https://tools.ietf.org/html/rfc8446
2. Joseph Salowey, Hao Zhou, Pasi Eronen, Hannes Tschofenig, Transport Layer Security (TLS) Session Resumption without Server-Side State, RFC5077, 2008. Available online https://tools.ietf.org/html/rfc5077
3. Nimrod Aviram, Kai Gellert, and Tibor Jager, Session Resumption Protocols and Efficient Forward Security for TLS 1.3 0-RTT, Eurocrypt 2019, Pages 117-150, Cryptology ePrint Archive: Report 2019/228, 2019. https://eprint.iacr.org/2019/228
4. Dick Hardt, The OAuth 2.0 authorization framework, RFC6749, 2012. Available online https://tools.ietf.org/html/rfc6749
5. Michael B. Jones and Dick Hardt, The OAuth 2.0 authorization framework: bearer token usage, RFC6750, 2012. Available online https://tools.ietf.org/html/rfc6750
6. Michael B. Jones, John Bradley, and Nat Sakimura, JSON web token (JWT), RFC7519, 2015. Available online https://tools.ietf.org/html/rfc75
7. Eric Rescorla, HTTP over TLS, RFC2818, 2000. Available online https://tools.ietf.org/html/rfc2818
8. A. Langley, W.T. Chang, QUIC crypto, 2014. Available online https://docs.google.com/
9. J. Iyengar and M. Thomson, QUIC: A UDP-based multiplexed and secure transport, June 2020. Available online https://tools.ietf.org/html/draft-ietf-quic-transport-29.
10. M. Thomson, S. Turner, Using TLS to Secure QUIC draft-ietf-quic-tls-34, Available online https://tools.ietf.org/html/draft-ietf-quic-tls-34.
11. Robert Lychev, Samuel Jero, Alexandra Boldyreva, Cristina Nita-Rotaru, How Secure and Quick is QUIC? Provable Security and Performance Analyses, Cryptology ePrint Archive: Report 2015/582, 2015. Available online https://eprint.iacr.org/2015/582
12. Shan Chen, Samuel Jero, Matthew Jagielski, Alexandra Boldyreva, and Cristina Nita-Rotaru, Secure Communication Channel Establishment: TLS 1.3 (over TCP Fast Open) vs. QUIC, ESORICS 2019, Cryptology ePrint Archive: Report 2019/433, 2019. https://eprint.iacr.org/2019/433
13. E. Sy, C. Burkert, H. Federrath, and M. Fischer, Tracking Users across the Web via TLS Session Resumption, ACSAC '18, December 3-7, 2018, San Juan, PR, USA.
14. D. M'Raihi, S. Machani, M. Pei, J. Rydell, TOTP: Time-Based One-Time Password Algorithm, RFC6238, 2011. Available online https://tools.ietf.org/html/rfc6238
15. Byoungcheon Lee, Strengthening of token authentication using time-based randomization, Journal of Security Engineering, vol. 14, no. 2, 2017, pp. 103-114.
16. Byoungcheon Lee, Stateless Randomized Token Authentication for Performance Improvement of OAuth 2.0 MAC Token Authentication, Journal of The Korea Institute of Information Security & Cryptology, VOL.28, NO.6, 2018, pp. 1343-1454.
17. Byoungcheon Lee, Efficient Wi-Fi Security Protocol Using Dual Tokens, Journal of The Korea Institute of Information Security & Cryptology, Vol. 29, No. 2, 2019, pp. 417-429.
18. Lee, B. Stateless Re-Association in WPA3 Using Paired Token. Electronics 2021, 10, 215. https://doi.org/10.3390/electronics10020215
19. Lee, B. Paired Token: A New Secondary Credential Providing Stateless Pre-Shared Key. Int. J. Inf. Secur. 2020. submitted.