ISSN 1330-3651 (Print), ISSN 1848-6339 (Online) https://doi.org/10.17559/TV-20240528001688
Preliminary communication

Enhancing TLS Handshake Security: A Novel Mutual Cryptographic Scheme

Yuting FENG

Abstract: The Transport Layer Security (TLS) Handshake Protocol serves as a critical mechanism in the security framework of internet communications, establishing a secure conduit between clients and servers. This protocol, not only ensures the confidentiality and integrity of data transferred over the internet but also facilitates the authentication of communicating parties. In this paper we investigate the transmission process and the message structure of the TLS Handshake protocol, construct a formal representation for the core transmission process. After that, we find the attack trace between client and server, which is caused by the secrecy of transported message. We introduce our enhanced cryptographic scheme, detailing the theoretical foundations, cryptographic mechanisms employed, and the integration process with the TLS protocol. Our approach significantly improves the resilience of client-server communications against potential attacks while maintaining efficiency. Formal verification confirms the enhanced security of our proposed scheme, demonstrating its potential for improving TLS implementations.

Keywords: mutual cryptographic scheme; security analysis; security protocol animator; transport layer security (TLS)

1 INTRODUCTION

TLS, the successor to Secure Sockets Layer (SSL), is among the most widely implemented cryptographic protocols. Since it was developed, it has been used to provide end-to-end privacy, authentication, and integrity. TLS consists of two primary components: the handshake protocol and the record protocol. The handshake protocol allows two entities to authenticate each other and establish a key for subsequent sessions. The Record protocol is responsible for the sending of data, which divides the data into several pieces for compression and encryption [1, 2].

Among the handshake protocol and the record protocol, the handshake protocol gets more attention. The handshake protocol is a crucial part of TLS, which is used to secure communication over computer networks. The TLS Handshake Protocol ensures privacy and data integrity between two communicating applications (e.g., a web browser and a server). The protection that the TLS Handshake provides relies heavily on adherence to best practices such as using strong cipher suites, regularly updating cryptographic libraries to patch known vulnerabilities, and using protection mechanisms like HSTS (HTTP Strict Transport Security) to prevent protocol downgrade attacks.

For the earliest version, TLS 1.0, Dean et al. describe a client-puzzle resolution against the DDos attack in SSL handshake [3]. Jonsson et al. show that the security of the TLS Handshake Protocol is related to the hardness of inverting RSA [4]. After the presentation of TLS 1.2, Voelker and Schoeller analyze DOS attacks on TLS and SSL and issue an efficient solution for the attack [5]. The above studies have revealed that there are difficult problems in the certification process of TLS Handshake Protocol. The researchers then began to propose multiple solutions to solve safety issues. Morrissey et al. study the security of the widely deployed Secure Session Layer/TLS key agreement protocol. They give a structure of security model for keys, which provides an idea of strengthening the key security [6]. In addition to enhancing security on keys, many researchers have also achieved results in handshake protocol and record protocol. Gajek focuses on the security of the handshake protocol, and evaluates the composition of key exchange functionalities realized by the TLS Handshake with the message transmission of the TLS based on the adaptation of the secure channel model [7]. Canetti et al. present the most complete analysis to date of the TLS Handshake Protocol and its application to data encryption (in the Record Protocol) [8]. These TLS Handshake Protocols adopt a certificate-based mechanism, which leads to complex certificate management overheads and long handshake latency. To overcome these disadvantages, Peng et al. present a series of handshake protocols that apply identity-based encryption, signature, and authenticated key agreement schemes, respectively [9]. Li et al. analyze the variants of TLS that make use of pre-shared keys (TLS-PSK). Li introduced a new and strong definition of ACCE security that covers protocols with pre-shared keys. Although it increases some bandwidth and memory overhead for the client, client-aided RSA provides the best performance among the algorithms in the SSL/TLS Handshake Protocol by transferring some cryptographic computation to the client [10]. On the performance front, Stebilaand Sullivan find that latency is slightly worse compared to the insecure approach, but still significantly better than the domain owner serving the content directly [11]. Fouladgar et al. show that the combination of the TLS-DHE Handshake protocol and the TLS Record Layer encryption is secure in this model [12]. Although these security schemes add security to TLS, they still cannot resist all kinds of attacks. Garman and Paterson provide new attacks against RC4 in TLS and report on a "proof of concept" implementation of the attacks for a specific application layer protocol [13]. Bhargavan et al. identify a new class of transcript collision attacks on TLS 1.2 client authentication [14]. Through the analysis of the above research results, it is known that the reliability of the safety scheme and the verification method are closely related. When analyzing protocol security using informal methods, unexpected attack scenarios often arise.

In this paper, we present a formal analysis approach to the TLS Handshake Protocol and find vulnerabilities in the authentication process. The TLS Handshake Protocol relies on certificates to establish trust during the handshake. If certificate validation is not done correctly or if a certificate authority (CA) is compromised, this could lead to man-in-the-middle attacks. It is critical that certificate trust chains are properly validated and that known weak certificate

authorities are not trusted. The session resumption attack is another noticeable vulnerability in the TLS Handshake Protocol. TLS provides mechanisms for session resumption to save time and resources for connections where the client and server have previously communicated. If not properly implemented, attackers can exploit weaknesses in session resumption to hijack sessions or reduce the security of the session keys. For these two attacks, we made some improvements to the handshake protocol. We encapsulated the packets transmitted in the protocol process with the encryption algorithm and adopted digital signatures for sensitive data. To verify the safety of the new scheme, we check it with the verification method SPAN. The result shows that the new scheme enhances the authentication, efficiency, integrity, and reliability of the TLS Handshake Protocol.

In this paper, we make the following three contributions:

- We give a secure analysis using the Security Protocol Animator SPAN for the TLS Handshake Protocol.
- We found the vulnerability of the TLS Handshake Protocol in the mutual authentication process.
- We proposed an optimized mutual cryptographic scheme for the TLS Handshake Protocol and verified it as safe.

The remainder of the paper is organized as follows: Section II describes the preliminaries of the TLS Handshake Protocol, such as its architecture and procedure; Section III introduces the secure analysis method for the TLS Handshake Protocol, as well as its verification results. In Section IV, we propose an improved cryptographic scheme for the TLS Handshake Protocol to avoid the security vulnerability. Finally, Section V contains the conclusions of the paper.

2 PRELIMINARIES

In this section, we introduce the background of our research. Above all, it is necessary to know the fundamental architecture of the TLS protocol and the procedure of the TLS Handshake Protocol.

2.1 TLS Architecture

TLS protocol is a cryptographic protocol designed to provide secure communication over a computer network [15-17]. TLS operates between the transport layer, which handles communication between hosts, and the application layer, where actual data exchange happens (e.g., HTTP for web pages). The primary purpose of TLS is to prevent eaves dropping, tampering, and message forgery by malicious actors. The architecture of the TLS protocol follows a client-server model, where the TLS client (usually a web browser) initiates a secure connection with the TLS server (usually a web server) [18].

The TLS client is responsible for initiating the TLS handshake process with the server. It typically runs on a client-side application, such as a web browser. The TLS client performs tasks such as validating the server's digital certificate, negotiating the cryptographic algorithms to use, generating a shared secret key, and encrypting the data to be transmitted. The TLS server, usually a web server, responds to the client's request to establish a secure connection. It accepts the TLS handshake initiated by the client and performs tasks such as presenting its digital certificate for authentication, negotiating cryptographic algorithms, generating a shared secret key, and encrypting the data to be transmitted [19].

On this basis, the TLS protocol uses a combination of cryptographic algorithms and digital certificates to provide secure communication over the internet. Digital certificates are used to authenticate the identities of servers and clients, while cryptographic algorithms are used to encrypt and decrypt data, ensuring that communication is confidential, secure, and authenticated. The server presents its digital certificate to the client during the TLS handshake. This certificate is issued by a trusted third-party CA and contains the server's public key. The client verifies the authenticity and validity of the certificate, ensuring it trusts the CA that issued it. This process helps establish a secure connection between the client and server. Cryptographic algorithms used in TLS include symmetric algorithms (like AES) for encryption, asymmetric algorithms (like RSA or ECC) for key exchange and authentication, and hash functions (like SHA) for integrity checks [20].

To achieve safety performance, the handshake protocol and the record protocol are both crucial for the TLS. The TLS Handshake Protocol facilitates the exchange of cryptographic parameters between the client and server. This includes negotiating the TLS version, selecting the cryptographic algorithms and key exchange methods, verifying the server's digital certificate, and generating the shared secret key used for encryption and decryption. The TLS record protocol provides a secure channel for transmitting data. It takes the data from higher-layer protocols (such as HTTP) and breaks it into manageable chunks called TLS records. These records are then encrypted, digitally signed, and authenticated before being transmitted over the network.

It would not be accurate to say which protocol is more important for TLS. But for our research work, the handshake protocol is crucial for initiating the secure session. Without the Handshake protocol, TLS would not be able to securely authenticate the parties involved, negotiate secure parameters, and generate the necessary encryption keys.

2.2 TLS Handshake Protocol Procedure

The handshake protocol can be viewed as the foundation layer of a secure TLS session, setting up all the parameters and keys that will be used to ensure security. It is used to establish a secure, encrypted connection between a client and a server. The procedure of the TLS Handshake Protocol involves four main phases: the Hello Phase, Server Authentication and Key Exchange Phase, Client Authentication and Key Exchange Phase, and Finish Phase [21].

The client starts the handshake by sending a ClientHello message to the server. This includes the client's TLS version number, a list of supported cipher suites, a list of supported compression methods, a random number (Client Random), and possibly a list of supported extensions. In the Hello Phase, the server responds with a ServerHello message. This contains the server's chosen protocol version (within the client's supported range), the

cipher suite selected from the client's list, a random number (Server Random), and a session ID for session resumption if applicable.

In the Server Authentication and Key Exchange Phase, the server sends its certificate to the client for authentication. The certificate contains the server's public key and is signed by a trusted CA. If the chosen cipher suite requires it (e.g., if the cipher suite uses ephemeral Diffie-Hellman (DH) key exchange), the server sends the key exchange parameters to the client along with a digital signature over these parameters. If client authentication is required, the server will request a certificate from the client. Then the server sends this message to indicate that it has finished sending messages to support the key exchange, and the remaining messages will depend on the client's response [22].

In the Client Authentication and Key Exchange Phase, if requested, the client sends its certificate to the server. If necessary, the client sends key exchange information to the server, typically including the pre-master secret, and then optionally sends a verification message for the certificate to complete the authentication phase.

In the Finish Phase, both the client and server indicate that subsequent messages will be encrypted using the newly negotiated parameters and exchange messages that verify the integrity of all handshake messages and confirm that the key exchange and authentication processes were successful.

After these four phases are successfully completed, the TLS handshake establishes a secure channel. The client and server can begin to communicate using symmetric encryption with the established session keys.

Fig. 1 shows the complete message flow of the Handshake protocol. In the message process, not all messages are mandatory. When the server's certificate contains an RSA public key that can be used for both encryption and key agreement, and if the cipher suite is using plain RSA for key exchange, then the server might not send a ServerKeyExchange message. Instead, the client uses the public key from the server's certificate to encrypt a pre-master secret and sends it to the server, where it is decrypted with the server's private key. Likewise, if the server is authenticated, the server must send the certificate message to the client. As the client is required for authentication, it must transmit its certificate to the server. Consequently, the CertificateVerify message is used to provide explicit verification of a client certificate. For the aforementioned reasons, the messages with a star in the top right corner mean optional or situation-dependent messages. Besides this, the ChangeCipherSpec message is used to change the encryption being used for both the client and server. It signifies that subsequent messages will be protected under the newly negotiated encryption parameters and keys. It functions as a trigger that tells the receiving party that the sender will start encrypting messages using the agreed-upon security settings. By being sent as a single-byte record of its own content type and not as a handshake message, ChangeCipherSpec ensures a clear demarcation between the unprotected handshake messages and the encrypted messages that follow [23].

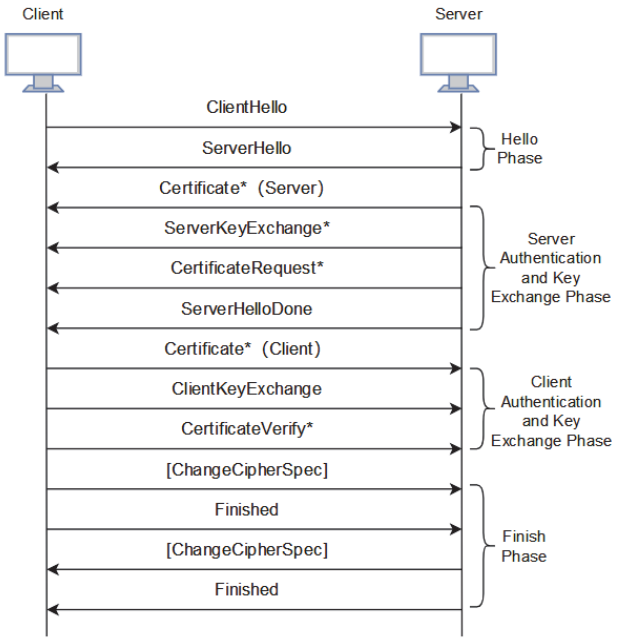

Figure 1 Complete message flow of the handshake protocol

2.3 TLS Handshake Protocol message structure

Once the Handshake Protocol has been completed, the TLS record layer receives handshake messages and encloses them in one or more TLS plaintext structures. These structures are then handled and sent according to the specifications of the currently active session state. TLS handshake messages are encapsulated within the TLS record layer, which provides the structure for sending data securely. The record layer is responsible for encapsulating different types of protocol data, including handshake messages, alert messages, change cipher specification messages, and application data.

The handshake message structure is designed to securely transfer data between the client and server. The record layer adds a layer of framing to efficiently handle different message types and allow for different TLS protocols to coexist and upgrade if necessary. Each record can encapsulate only one or a portion of a handshake message if it is too large to fit into one record, in which case it can be split across multiple records.

Based on the Handshake protocol message flow, there are ten kinds of handshake message types in the Handshake protocol: ClientHello, ServerHello, ServerCertificate, ServerKeyExchange, CertificateRequest, ServerHelloDone, ClientCertificate, ClientKeyExchange, CertificateVerify, and Finished. To analyze the security of a protocol, you not only need to understand the operation process of the protocol but also analyze the structure of the protocol packet.

(1) ClientHello

The ClientHello message is vital to begin the handshake process and allows the client and the server to agree on how to set up a secure communication channel. The structure of the ClientHello message is defined as follows:

Structure {ProtocolVersionclient_version;
Random random;
SessionIDsession_id;
CipherSuitecipher_suites;

CompressionMethodcompression_methods;
Extension extensions;
} ClientHello;

Protocol Version: The version of the TLS protocol by which the client wishes to communicate during this session.

Random: A client-generated random structure, composed of the current UNIX timestamp and 28 bytes of random or pseudo-random data.

Session ID: A variable-length session identifier. If the ID is non-empty, it indicates a session resumption request.

Cipher Suites: A list of supported cipher suites, ordered by client preference.

Compression Methods: A list of supported compression methods.

Extensions: A list of extensions that may include enhanced capabilities such as SNI (Server Name Indication), supported groups (formerly known as elliptic curve information), maximum fragment length, signature algorithms, and more.

(2) ServerHello

The ServerHello message is the server's response to the client's ClientHello message during the TLS Handshake Protocol. The structure of the ServerHello message is defined as follows:

Structure {ProtocolVersionserver_version;
Random random;
SessionIDsession_id;
CipherSuitecipher_suite;
CompressionMethodcompression_method;
Extension extensions
} ServerHello;

Protocol Version: The version of the TLS protocol that the server has decided to use.

Random: A random byte sequence generated by the server, which is used in subsequent cryptographic computations.

Session ID: The identifier for a session that can be reused. If the server agrees to reuse the session, the ID will match that provided by the client in its ClientHello. If not, this will be a new session ID.

Cipher Suite: The single cipher suite selected by the server from the list provided by the client in its ClientHello message.

Compression Method: The compression method selected by the server from the list provided by the client.

Extensions: The server can also include a list of extensions in the ServerHello message. These extensions can communicate additional parameters and settings that are not covered by the previous fields.

(3) ServerCertificate

The primary purpose of the ServerCertificate message is to provide a means for the server to authenticate itself with the client. By presenting its certificate, the server asserts its identity and assures the client that it is communicating with the legitimate owner of the domain or service the client intends to interact with. The server's certificate contains its public key, which is crucial for setting up a secure encrypted session. After the client receives and verifies the certificate, it can use the server's public key to encrypt data that only the server can decrypt with its corresponding private key. Depending on the certificate, specific extensions or key usage fields might activate additional security features for the session, such as certain encryption algorithms or key exchange protocols. The structure of the ServerCertificate message is defined as follows:

Structure {ServerCertificatecertificate_list;
} Certificate;

ServerCertificate: This field contains one or more certificates that make up the server's certificate chain. Each certificate contains both its length and the actual certificate data. The certificates that are included are typically in X.509 certificate format, which is a standard format for public key certificates. The client can use these certificates to verify the server's identity, check the validity period, and validate the certification path.

(4) ServerKeyExchange

When the cipher suite selected for the session uses an ephemeral DH or ephemeral Elliptic Curve DH key agreement protocol, the server uses the ServerKeyExchange message to send the parameters necessary for the client to engage in the key exchange. The structure of the ServerKeyExchange message is defined as follows:

Structure {select (KeyExchangeAlgorithm)
ServerDHParams params;
} ServerKeyExchange;

Depending on the key exchange algorithm being used, there are different ServerDHParams. For the RSA Key Exchange algorithm, the server includes its RSA public key in the ServerKeyExchange message. In the DH Key Exchange algorithm, the server includes the parameters used for the DH key exchange, such as the prime modulus, and the generator and its ECDH public key encoded in a specific format.

(5) CertificateRequest

This process is known as client certificate authentication, or mutual TLS. The CertificateRequest message is sent by the server to the client after the server's Certificate message but before the ServerHelloDone message. The structure of the ServerKeyExchange message is defined as follows:

Structure {ClientCertificateTypecertificate_types;
SignatureAndHashAlgorithm
supported_signature_algorithms;
DistinguishedNamecertificate_authorities;
} CertificateRequest;

ClientCertificateType: The server provides a list of the types of certificate types which the client might offer.

SignatureAndHashAlgorithm: types of algorithms that could be recognized and verified by the server.

DistinguishedName: This field contains a list of distinguished names of acceptable certificate authorities. This list guides the client in choosing which certificate to present; it would usually select a client certificate that is signed by one of the authorities in the list.

(6) ServerHelloDone

This message is sent by the server to signify the completion of the ServerHello and accompanying messages. It indicates to the client that the server has finished sending its part of the handshake and that it is the client's turn to respond with its own set of handshake messages. The structure of the ServerHelloDone message is defined as follows:

Structure { }ServerHelloDone;

(7) ClientCertificate

This message typically contains the client's public key along with identification information, and it is signed by a trusted CA. The structure of the ClientCertificate message is defined as the same as the ServerCertificate message. When the client does not have any certificates to send, the message will be an empty one. At this moment, the server could decide either to ignore the certificate or respond with an alert message.

(8) ClientKeyExchange

The core purpose of the ClientKeyExchange message is to provide the key exchange data from the client to the server for shared secret generation. When using DH or Elliptic Curve Diffie-Hellman (ECDH), the client sends its part of the DH parameters (public values), which will be combined with the server's public values to compute the shared secret. The structure of the ClientKeyExchange message is defined as follows:

Structure {select (KeyExchangeAlgorithm)
EncryptedPreMasterSecret;
} ClientKeyExchange;

The provided key exchange algorithms are listed in RFC 5246. According to different key exchange algorithms, different public keys are used in the ClientKeyExchange message.

(9) CertificateVerify

The CertificateVerify message provides a way for the client to prove to the server that it possesses the private key associated with the certificate it presented to the server earlier in the ClientCertificate message. The structure of the CertificateVerify message is defined as follows:

Structure {handshake_messages;
} CertificateVerify;

The client verifies the signature by reconstructing the digest of the handshake messages up to that point, then using the public key from the server's certificate to verify that the received signature was indeed made with the corresponding private key. If the signature verifies correctly, the client can trust that the server is in possession of the private key, and hence the identity asserted by the certificate.

(10)Finished

The Finished message is used to indicate the completion of the handshake phase and to provide a verification check of the integrity and authenticity of the proceeding handshake messages. Both client and server send a Finished message, and each must verify the Finished message received from the other side to ensure the security of the handshake. The structure of Finished message is defined as follows:

Structure {verify_data;
} Finished;
verify_data = PRF (master_secret,
finished_label,
Hash (handshake_messages));

The creation and verification of a Finished message rely on various keying materials that are derived during the handshake process, which are defined as verify_data. The Verify Data within the Finished message is calculated using specific keys derived from the main secret and involves applying a pseudorandom function (PRF) or a hash function over the handshake messages.

3 SECURITY ANALYSIS OF TLS HANDSHAKE PROTOCOL

Formal methods are crucial and widely used in cryptographic protocol analysis. Tools such as model checkers and theorem provers can be applied to formal representations to analyze properties like confidentiality, integrity, and authentication mechanisms embedded within protocols like TLS, IPSec, etc. Formal analysis of security protocols can reveal subtle bugs or security weaknesses that even extensive testing might fail to uncover [23-25].

The Security Protocol Animator SPAN is a tool related to formal verification, specifically in the context of analyzing and animating security protocols. Among the many formal validation tools, SPAN has performed well in the validation of security protocols. SPAN's developer team has successfully verified or found vulnerabilities in multiple commonly used security protocols, such as H.530, IKE, EKE etc. SPAN is actually a plug-in for the widely known AVISPA Tool, designed to allow users to interact with the security protocol analysis process more intuitively. It is used to animate the execution of security protocols specified using the High-Level Protocol Specification Language (HLPSL), which is the specification language used by AVISPA. The animation provides a visual means to understand how security protocols operate and how their messages are exchanged over a network. Since SPAN is integrated with the AVISPA framework, it can take advantage of the powerful back-end analyzers provided by AVISPA, such as OFMC and CL-AtSe, to carry out the formal verification process behind the scenes [26-28]. If AVISPA's back-end analysis tools find an attack on the protocol, SPAN can be used to animate this attack. This allows users to visualize how an attacker could exploit vulnerabilities in the protocol, which is beneficial for both educational and development purposes [29, 30].

To use a security protocol animator such as SPAN, the first step involves formally representing the protocol. This is done by writing a specification of the TLS Handshake Protocol in a formal language that the tool can understand. For the AVISPA Tool, this language is HLPSL (High-Level Protocol Specification Language).

3.1 Formal Representation of TLS Handshake Protocol

To facilitate modelling, the handshake process and data structure are represented formally. According to the message flow introduced in the Fig. 1, we use client.hello to represent the ClientHello message, server.hello to represent the ServerHello message, other messages are also abbreviated like this. The thirteen communication processes are expressed as follows.

C -> S: client.hello
S -> C: server.hello
S -> C: server.certificate
S -> C: server.key.exchange
S -> C: certificate.request
S -> C: server.hello.done
C -> S: client.certificate
C -> S: client.key.exchange
C -> S: certificate.verify
C -> S: change.cipher.spec

C ->S: finished
S -> C: change.cipher.spec
S ->C: finished

As the certificate request is optional, it depends on the authentication request of the server. To express the most basic authentication process of the handshake protocol, we neglect the handshake process without server authentication flow and change the cipher suite flow. The reduced process is as follows:

C -> S: client.hello
S -> C: server.hello
S -> C: server.certificate
S -> C: server.key.exchange
S -> C: server.hello.done
C -> S: client.certificate
C -> S: client.key.exchange
C -> S: certificate.verify
C ->S: finished
S ->C: finished

Then the key step is to substitute the message structure of each message for the message packet in the message flow. The formal representation of the process above is:

C -> S: version.random.session_id.cipher_suites.methods
S -> C: version.random.session_id.cipher_suites.methods
S -> C: certificatelists
S -> C: params
S -> C: done
C -> S: certificatelists
C -> S: params
C -> S: handshake_messages
C ->S: master_secret.finished.Hash (handshake_messages)
S ->C: master_secret.finished.Hash (handshake_messages)

To distinguish the message content issued by the server and the client, and accurately express the message structure, the above representation should be revised once more. In addition, the messages sent in the same direction of transmission are merged into a total message body.

C -> S: c.random.session_id
S -> C: s.random.session_id.public_key{pa_server}.secret.server.done
C -> S: c.public_key.{pa_client}secret.client.handshake_messages.Hash (handshake_messages)
S ->C: Hash (handshake_messages)

As cipher_suites define the cipher specification supported for client and server and compression methods define a list of the compression methods supported, which could not represent the security attribute of a message, they are neglected in the final representation. Similarly, the version of the TLS protocol could also be ignored. As everyone knows, certificates contain public keys and authorities. The certificate of the client or server could be represented as public_key. It should be noted that for different key exchange methods, the messages in steps "Server Key Exchange Message" and "Client Key Exchange Message" have different structures. For the RSA key exchange method, parameters sent by the server contain client random, server random, and a pre-master key. For the DH key exchange method, a pair of secret keys and a public key are generated, and the public key "pa_server" is sent in a param message from server to client. Then, the same as on the client side, the generated public key "pa_client" is sent in a param message from the client to the server. These two messages are encrypted in a digital signature by each sender. The master secret and the finished division do not have much impact; that could be neglected.

3.2 HLPSL Rewritings of TLS Handshake Protocol

The next step is to rewrite the model into the HLPSL language that SPAN can recognize. The client side is rewritten as A, while the server side is rewritten as B. The algorithm rewrites random numbers for the client and server as Rand1 and Rand2, respectively. Has is the sign for the hash function. PKS and PKC represent the public key of the server and the public key of the client.

Transmission of Client Side
transition

1. *State = 0 /\ RCV(start) =|> State' := 2 /\ Rand1' := new() /\ SND(Rand1'.session1)*
2. *State = 2 /\ RCV(Rand2'.session2. {PKS.ParaS'}_PKC. {B.PKS}_inv(PKS)) =|> State' := 4 /\ParaC':= new() /\ SND(A.{ParaC'}_PKS. {A.PKC}_inv(PKC). Has(Rand1,Rand2',ParaS',ParaC')) /\ witness(A,B,na_nb2,Rand1.Rand2')*
3. *State = 4 /\ RCV(B.Has(Rand1,Rand2,ParaS,ParaC')) =|> State' := 6 /\ request(A,B,na_nb1,Rand1.Rand2) /\ secret(A.{ParaC'}_PKS. {A.PKC}_inv(PKC). Has(Rand1,Rand2,ParaS,ParaC'),sec_c,{A,B}) /\ secret(B.Has(Rand1,Rand2,ParaS,ParaC'), sec_s,{A,B})*

TLS Handshake protocol is used to create secure links, mainly responsible for negotiation and key exchange. Authentication is the security property defined in RFC 5246. This model requires that both the client side and the server side can provide digital certificates to complete two-way authentication. In addition, since the entire message is encrypted before transmission, this message has security requirements, so the secrecy of the encapsulated message is required as a security property. Based on the requirements above, the security goal of this protocol is defined as follows:

Specification of Security Properties
goal
secrecy_ofsec_c,sec_s
authentication_on na_nb1
authentication_on na_nb2
end goal

After running this model in SPAN, we found attacks on a handshake protocol under this model. The attack path is as follows.

ATTACK TRACE
i -> (a,6): start
(a,6) -> i: Rand1(1).session1
i -> (b,3): x246.session1

(b,3) -> i:
{Rand2(2).session2}_inv(pks).ParaS(2).done.Crequest(2)
i -> (a,6): {x259.session2.x260}_inv(pki).done.x261
(a,6) -> i: ParaC(3).{x}_inv(pkc).{Rand3(3)}_pki
i -> (b,3): x274.{x}_inv(pkc).{x275}_pks
(b,3) -> i: {dummy_nonce}_pkc

The result shows the attacker i could participate in the communication between entities, playing one party gaining the trust of the other, and thus getting the complete transmission data. That means, the security properties are not satisfied. There exits an attack trace, which starts from the attacker i. The messages transmitted between the client and the server are captured and copied, including the encrypted messages. Although the content of the message was not read, the attacker succeeded in imitating it, resulting in the breaking of the authentication mechanism between the two entities. Therefore, the agreement under this security goal is not safe.

4 THE PROPOSED CRYPTOGRAPHIC SCHEME AND VERIFICATION

Many researchers have proposed remedial solutions for the security vulnerabilities of the TLS Handshake Protocol. In our work, we encapsulate the captured message with public key encryption according to the security vulnerability found by SPAN, which increases the security of the protocol based on simplification cost.

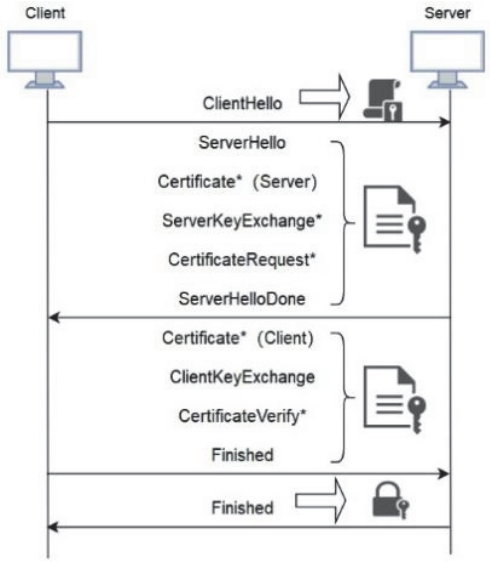

Figure 2 Optimized message flow of handshake protocol

According to the results of the formal analysis by animator SPAN, the messages are reinforced multiple times. First, use digital signatures for the messages sent by the client, and make the server confirm that the received message is not a fake message. Second, the certificate issued by the server to the client is encrypted with a hash function to ensure that the integrity of the message is not destroyed. Third, when the client verifies the message, all the messages are encrypted with the public key algorithm to improve the security of the message. The message flow of the optimized scheme could be indicated as Fig. 2.

Message flow is still divided into four parts: client sends "ClientHello" to server, server responses as "ServerHello", client then authenticates the server, at last server authenticates client.

To certify the identity of the client, the first "ClientHello" message is encrypted by the client's digital signature. The sign c.random indicates the random number generated by client. The sign session_id is the constant value of this session. The combination of the two values is signatured by client. This ensures that all random numbers issued by the client will not be changed. In formal representation, the message is rewritten as follows:

C->S: client_secretkey{c.random.session_id}

In the second "ServerHello" message, a random number generated by the server would also be encrypted by the server's digital signature. And to improve safety, the server's certificate is encrypted by the client's public key. In formal representation, the message is rewritten as follows:

S -> C:
client_publickey{server_secretkey{s.random.session_id}.
public_key{pa_server}.server_secretkey{secret.server.done}}

Besides that, the client certificate would also be hashed to avoid collision, which guarantees the authenticity and integrity of the message. Meanwhile, the hashed message would be encrypted by the server's public key to enhance security.

C->S:
server_publickey{Hash(c.public_key.{pa_client}secret.client.handshake_messages.Hash (handshake_messages))}

To achieve the same effect, the last message constructed by the server would also be done as in the message above. In formal representation, the message is rewritten as follows:

S->C: client_publickey{Hash(Hash (handshake_messages))}

In such a triple encryption system, the security property of secrecy would be accomplished. The HLPSL representation by SPAN of the optimized TLS Handshake Protocol process could be rewritten as follows:

Transmission of Client Side
transition

1. *State = 0 /\ RCV(start) =|>*
   State' := 2 /\ Rand1' := new() /
SND({Rand1'.session1}_inv(PKC))
2. *State = 2 /\ RCV({Rand1'.session1}_inv(PKC))*
=|>
State' := 4 /\ParaC':= new() /
({Has(A.{ParaC'}_PKS.{A.PKC}_inv(PKC).Has(Rand1,Rand2',ParaS',ParaC'))}_(PKS)) /
witness(A,B,na_nb2,Rand1.Rand2')
3. *State = 4 /*
RCV({Has(B.Has(Rand1,Rand2,ParaS,ParaC'))}_(PKS))
=|> State' := 6 /
request(A,B,na_nb1,Rand1.Rand2)
/
secret(Has(A.{ParaC'}_PKS.{A.PKC}_inv(PKC).Has(Rand1,Rand2,ParaS,ParaC')),sec_c,{A,B})
/
secret(Has(B.Has(Rand1,Rand2,ParaS,ParaC')),sec_s,{A,B})

When we put the optimized protocol model into SPAN to run, after the state access to 40 nodes, no attack trace was found. Use the same backend OFMC to verify the

security of the optimized TLS Handshake Protocol; the security property of secrecy is perfectly implemented.

5 CONCLUSION AND FUTURE WORKS

This study presents a significant enhancement to the TLS Handshake Protocol, addressing key vulnerabilities identified through formal analysis. Our proposed mutual cryptographic scheme, verified using SPAN/AVISPA, demonstrates improved security in the authentication process. By strategically combining asymmetric and symmetric encryption, we achieve a balance between security and efficiency. This work contributes to the ongoing efforts to strengthen internet security protocols, particularly in an era of increasing cyber threats. Future research should focus on real-world implementation and performance analysis of this enhanced protocol across various network environments.

The research is supported by the Anhui Province Quality Engineering Project, 2022kcsz238.

6 REFERENCES

[1] Ferreira, A., Giustolisi, R., Huynen, J. L., Koenig, V., & Lenzini, G. (2013). Studies in socio-technical security analysis: authentication of identities with TLS certificates. *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 1553-1558. https://doi.org/10.1109/TrustCom.2013.190

[2] Ohwo, O. B., Ayankoya, F. Y., Ajayi, O. F., & Alao, D. O. (2023). Advancing DNS performance through an Adaptive Transport Layer Security Model (ad-TLSM). *Ingénierie des Systèmesd' Information*, *28*(3), 777-790. https://doi.org/10.18280/isi.280329

[3] Dean, D. & Stubblefield, A. (2001). Using Client Puzzles to Protect {TLS}. *10th USENIX Security Symposium (USENIX Security 01)*.

[4] Jonsson, J. & Kaliski, B. S. (2002). On the security of RSA encryption in TLS. *Advances in Cryptology - CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara*, 127-142. https://doi.org/10.1007/3-540-45708-9_9

[5] Völker, L. & Schöller, M. (2007). Secure TLS: preventing DoS attacks with lower layer authentication. *Kommunikation in VerteiltenSystemen (KiVS) 15. Fachtagung Kommunikation in VerteiltenSystemen (KiVS 2007) Bern*, 237-248. https://doi.org/10.1007/978-3-540-69962-0_20

[6] Morrissey, P., Smart, N. P., & Warinschi, B. (2008). A modular security analysis of the TLS handshake protocol. *Advances in Cryptology-ASIACRYPT 2008: 14th International Conference on the Theory and Application of Cryptology and Information Security*, 55-73.

[7] Gajek, S., Manulis, M., Pereira, O., Sadeghi, A. R., & Schwenk, J. (2008). Universally Composable Security Analysis of TLS---Secure Sessions with Handshake and Record Layer Protocols. *Cryptology ePrint Archive*. https://doi.org/10.1007/978-3-540-88733-1_22

[8] Canetti, R., Krawczyk, H. (2002). Universally composable notions of key exchange and secure channels. *Advances in Cryptology - EUROCRYPT 2002: International Conference on the Theory and Applications of Cryptographic Techniques Amsterdam, the Netherlands*, 337-351. https://doi.org/10.1007/3-540-46035-7_22

[9] Peng, C., Zhang, Q., & Tang, C. (2009). Improved tls handshake protocols using identity-based cryptography. *2009 International Symposium on Information Engineering and Electronic Commerce*, 135-139. https://doi.org/10.1109/IEEC.2009.33

[10] Li, Y., Schäge, S., Yang, Z., Kohlar, F., & Schwenk, J. (2014). On the security of the pre-shared key cipher suites of TLS. *Public-Key Cryptography-PKC 2014: 17th International Conference on Practice and Theory in Public-Key Cryptography*, 669-684. https://doi.org/10.1007/978-3-642-54631-0_38

[11] Stebila, D. & Sullivan, N. (2015). An analysis of TLS handshake proxying. *2015 IEEE Trustcom / BigDataSE /ISPA*, *1*, 279-286. https://doi.org/10.1109/Trustcom.2015.385

[12] Fouladgar, S., Mainaud, B., Masmoudi, K., & Afifi, H. (2006). Tiny 3-TLS: A trust delegation protocol for wireless sensor networks. *Security and Privacy in Ad-Hoc and Sensor Networks: Third European Workshop, ESAS 2006, Hamburg*, 32-42. https://doi.org/10.1007/11964254_5

[13] Garman, C., Paterson, K.G., & Van der Merwe, T. (2015). Attacks Only Get Better: Password Recovery Attacks Against RC4 in TLS. *24th USENIX Security Symposium (USENIX Security 15)*, 113-128.

[14] Bhargavan, K. & Leurent, G. (2016). Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH. *Network and Distributed System Security Symposium--NDSS 2016*. https://doi.org/10.14722/ndss.2016.23418

[15] de Carné de Carnavalet, X. & van Oorschot, P. C. (2023). A Survey and Analysis of TLS Interception Mechanisms and Motivations: Exploring how end-to-end TLS is made "end-to-me" for web traffic. *ACM Computing Surveys*, *55*(13s), 1-40. https://doi.org/10.1145/3580522

[16] Bikku, T., Biyyapu, N. S., Sekhar, J. C., Kumar, M. K., Nokerov, S. M., & Pratap, V. K. (2024). The social network dilemma: Safeguarding privacy and security in an online community. *International Journal of Safety and Security Engineering*, *14*(1), 125-133. https://doi.org/10.18280/ijsse.140112

[17] Riadi, I., Sunardi, D., & Aprilliansyah, D. (2023). Analysis of Anubis Trojan attack on Android Banking application using mobile security labware. *International Journal of Safety and Security Engineering*, *13*(1), 31-38. https://doi.org/10.18280/ijsse.130104

[18] De la Hoz, E., Cochrane, G., Moreira-Lemus, J. M., Paez-Reyes, R., Marsa-Maestre, I., & Alarcos, B. (2014, June). Detecting and defeating advanced man-in-the-middle attacks against TLS. *2014 6th International Conference on Cyber Conflict (CyCon 2014)*, 209-221. https://doi.org/10.1109/CYCON.2014.6916404

[19] Sarvabhatla, M. & Vorugunti, C. S. (2014). A secure biometric-based user authentication scheme for heterogeneous WSN. *2014 Fourth international conference of emerging applications of information technology*, 367-372. https://doi.org/10.1109/EAIT.2014.23

[20] Manulis, M., Stebila, D., Kiefer, F., & Denham, N. (2016). Secure modular password authentication for the web using channel bindings. *International Journal of Information Security*, *15*, 597-620.

[21] Cai, J., Huang, X., Zhang, J., Zhao, J., Lei, Y., Liu, D., & Ma, X. (2018). A handshake protocol with unbalanced cost for wireless updating. *IEEE Access*, *6*, 18570-18581. https://doi.org/10.1109/ACCESS.2018.2820086

[22] Guo, H., Gao, Y., Xu, T., Zhang, X., & Ye, J. (2019). A secure and efficient three-factor multi-gateway authentication protocol for wireless sensor networks. *Ad Hoc Networks*, *95*, 101965. https://doi.org/10.1016/j.adhoc.2019.101965

[23] Hernandez-Ardieta, J. L., Gonzalez-Tablas, A. I., & Ramos, B. (2009). Formal Validation of OFEPSP+ with AVISPA. *Foundations and Applications of Security Analysis: Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security, ARSPA-WITS 2009*, 124-137. https://doi.org/10.1007/978-3-642-03459-6_9

[24] Shaikh, R. & Devane, S. (2010). Formal verification of payment protocol using AVISPA. *International Journal for*

Infonomics, *3*(3), 326-337.
[25] Rekik, M., Meddeb-Makhlouf, A., Zarai, F., & Obaidat, M. S. (2015). A MSCTP-Based Authentication Protocol: MSCTPAP. 2015 IEEE International Conference on Data Science and Data Intensive Systems, 617-623. https://doi.org/10.1109/DSDIS.2015.78
[26] Islam, S. (2014). Security analysis of LMAP using AVISPA. *International Journal of Security and Networks*, *9*(1), 30-39. https://doi.org/10.1504/IJSN.2014.059325
[27] Islam, S. H. (2015). Design and analysis of a three party password-based authenticated key exchange protocol using extended chaotic maps. *Information Sciences*, *312*, 104-130. https://doi.org/10.1016/j.ins.2015.03.050
[28] Raniyal, M. S., Woungang, I., & Dhurandher, S. K. (2017). An inter-device authentication scheme for smart homes using one-time-password over infrared channel. *Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments: First International Conference, ISDDC 2017, Proceedings 1*, 95-117. https://doi.org/10.1007/978-3-319-69155-87
[29] Raniyal, M. S., Woungang, I., Dhurandher, S. K., & Ahmed, S. S. (2018). Passphrase protected device-to-device mutual authentication schemes for smart homes. *Security and Privacy*, *1*(3), e42. https://doi.org/10.1002/spy2.42
[30] Cao, J., Ma, M., & Li, H. (2019). LPPA: Lightweight privacy-preservation access authentication scheme for massive devices in fifth Generation (5G) cellular networks. *International Journal of Communication Systems*, *32*(3), e3860. https://doi.org/10.1002/dac.3860

Contact information:

Yuting FENG, Associate Professor
Computer and Artificial Intelligence School,
Hefei Normal University,
Hefei 230601, China
E-mail: Fengyt@hfnu.edu.cn