Hindawi
Security and Communication Networks
Volume 2021, Article ID 1630223, 12 pages
https://doi.org/10.1155/2021/1630223

Research Article

A Systematic Approach to Formal Analysis of QUIC Handshake Protocol Using Symbolic Model Checking

Jingjing Zhang,[1] Xianming Gao,[1] Lin Yang,[1] Tao Feng,[1] Dongyang Li,[2] and Qiang Wang[1]

[1]*National Key Laboratory of Science and Technology on Information System Security, Institute of System Engineering, Chinese Academy of Military Science, Beijing 100039, China*
[2]*College of Command and Control Engineering, Army Engineering University of China, Nanjing 210007, China*

Correspondence should be addressed to Qiang Wang; 18513688908@163.com

Received 11 May 2021; Revised 14 July 2021; Accepted 6 August 2021; Published 20 August 2021

Academic Editor: Ruhul Amin

As a newly proposed secure transport protocol, QUIC aims to improve the transport performance of HTTPS traffic and enable rapid deployment and evolution of transport mechanisms. QUIC is currently in the IETF standardization process and will potentially carry a significant portion of Internet traffic in the emerging future. An important safety goal of QUIC protocol is to provide effective data service for users. To aim this safety requirement, we propose a formal analysis method to analyze the safety of QUIC handshake protocol by using model checker SPIN and cryptographic protocol verifier ProVerif. Our analysis shows the counterexamples to safety properties, which reveal a design flaw in the current protocol specification. To this end, we also propose and verify a possible fix that is able to mitigate these flaws.

1. Introduction

As a newly proposed secure transport protocol, QUIC aims to improve the transport performance of HTTPS traffic and enable rapid deployment and evolution of transport mechanisms. In the OSI reference architecture, QUIC is above the network layer and spans the transport layer, session layer, presentation layer, and application layer. It uses UDP instead of TCP in the transport layer. In the session layer and presentation layer, QUIC abandons the TLS1.2 protocol and self-encapsulates the TLS stack for protocol encryption. In the application layer, HTTP/2 is only responsible for HTTP protocol parsing, and QUIC can fulfill the functions of HTTP/2 multiplexing and link management. The position of QUIC in the HTTPS protocol stack is shown in Figure 1. Different from the traditional HTTP/2 + TLS + TCP scheme, QUIC can run completely in the user space rather than the system kernel based on UDP protocol. Therefore, QUIC can be rapidly deployed like an application program and continuously updated iteratively according to the usage requirements.

Being a key part of QUIC protocol, the QUIC handshake protocol is responsible for authenticating the identities of participants and establishing secure connection for subsequent communications. QUIC handshake protocol performs encryption in the transport layer, reducing the number of round trips required for setting up a secure connection. QUIC initial connections are common 1-RTT, meaning that all initial connection data can be sent immediately without waiting for a reply from the server, which is more efficient compared to the 3 round trips required for TCP/TLS before application data can be sent. QUIC handshake protocol is functionally equivalent to TCP/TLS/HTTP2, but implemented on top of UDP.

QUIC is still under review for standardization, which usually takes the format of an RFC: a natural language (normally English) document that offers implementation advice to protocol engineers. However, a natural language document is nonetheless ambiguous and open to various interpretations, some of which are even contradicting. As for the current QUIC handshake protocol, it is still unclear whether or not it conforms to the properties claimed in the IETF standardization documents. Safety and security are two types of properties that shall be accommodated in the design of cryptographic protocols. Safety refers to the capacity of

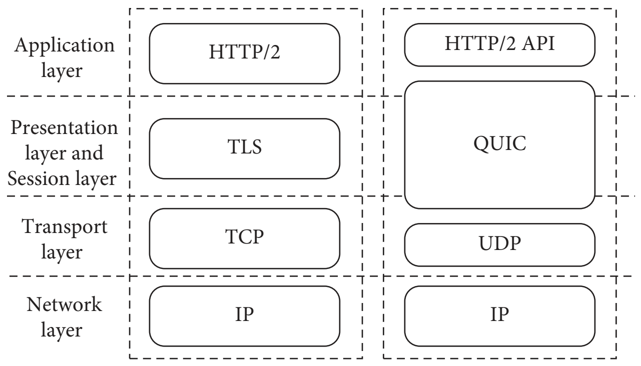

Figure 1: QUIC in the traditional HTTPS stack.

accommodating expected functions for cryptographic protocols, while security refers to the capacity of avoiding hostile attack in the public environment for cryptographic protocols. In order to ensure correct implementation, the cryptographic protocols shall have both safety and security before deployment.

As reported in [1], a possible way to resolve the ambiguities and rigorously validate the protocol design is through formal verification, where a formal model of the protocol is first constructed and then analyzed with respect to the specified properties. Symbolic model checking [2] has been a popular method for the formal verification of cryptographic protocols. Since the pioneering work in [3] that discovered the Needham–Schroeder protocol's design flaws, symbolic model checking has been widely and actively used to formally analyze cryptographic protocols [4–11].

However, most of the current works focus on security verification, which still lack consideration for safety verification of cryptographic protocols. Now, safety is mainly based on typical model checking for modeling and verification, while security mainly uses specific modeling and analysis technology such as process calculation for verification. In our work, we propose a new safety modeling and verification method for cryptographic protocols which lazily combines a typical model checker and a cryptographic protocol verifier. We also perform the safety verification of QUIC handshake protocol based on this method. The principle employed by our method is to be tool-agnostic, that is, it can be instantiated through any generic typical model checker and cryptographic protocol verifier. To achieve our goal, however, we need to address the following challenges. (C1) Modeling and verification cannot be implemented directly for safety property of cryptographic protocols with specific modeling and analysis technology such as process calculus. (C2) Safety verification of cryptographic protocols results from typical model checking may be not true or feasible for practical cryptographic protocols, that is to say, verification results may be fake counterexamples. (C3) The QUIC protocol lacks a formal specification and hence is prone to ambiguity and underspecification. To this end, we have made the following contributions in the work:

(1) We propose a method to solve the problem of fake counterexamples in safety verification.

(2) Based on the IETF standardized documents, we construct the formal model of the QUIC handshake protocol in applied pi calculus and give the transition system model of the QUIC handshake protocol. Through the transition system model, we present a enhance attacker model and construct the Promela [12] model of QUIC handshake protocol, which is used to verify the safety property of QUIC handshake protocol.

(3) We use our method to verify the QUIC handshake protocol and successfully find the real design flaw of QUIC handshake protocol.

(4) According to our verification results, we discuss the possible causes of the defects and suggest potential fixes of QUIC handshake protocol.

The organization of this paper is as follows. Section 2 gives the related works on formal analysis of cryptography protocols. Section 3 presents a thorough description and the safety requirement of QUIC handshake protocol. Section 4 presents the formal verification of QUIC handshake protocol. Section 5 reports the verification results. Finally, Section 6 concludes this paper.

2. Related Works

The properties considered in model checking of cryptographic protocols can be divided into 2 kinds: based on temporal properties and based on cryptographic properties. Previously, a lot of work focused on analyzing these properties of cryptographic protocols. In this section, we review the most relevant works.

Babenko et al. [13] used the formal verifier SPIN to analyze the temporal properties of the cryptographic protocol for e-voting. They found that the e-voting protocol correctly handles the case of an active attack on the parties' authentication.

Ninet et al. [14] performed a formal analysis of the IKEv2 specification using the SPIN. Their analysis showed that the reflection attack is not possible, due to IKEv2's Initiator and Response flags. And, they confirmed that IKEv2-Sig does not satisfy weak agreement.

In [15, 16], the authors used Tamarin [17] to model and analyze the cryptographic properties of the 5G AKA protocol. They found that the 5G AKA protocol lacks integrity protection for the identity of the server network.

Zhang et al. [18] used ProVerif to model and analyze the cryptographic properties of the 5G EAP-TLS protocol. Their analysis revealed several design flaws of authentication properties of 5G EAP-TLS protocol. They also proposed several strategies to repair these vulnerabilities.

Hussain et al. [19] proposed a systematic model-based adversarial testing approach LTEInspector that leverages the combined power of a symbolic model checker and a protocol verifier for analyzing the critical procedures of the 4G LTE network. They exposed ten new attacks.

Hussain et al. [20] proposed a property-guided formal verification of 5G control-plane protocols. Their evaluation of the 5G protocol model against 187 properties revealed 11 new exploitable protocol design weaknesses. They also discovered 5 prior attacks which 5G inherits from 4G LTE.

The security analysis for cryptographic protocols also includes some complexity-based formal methods, such as [21–23]. In this paper, our main consideration is whether there are defects in the design of cryptographic protocol. We conduct formal verification of the cryptographic protocol based on the symbolic model checking. This method relies on the symbolic model of cryptography and the Dolev–Yao attacker model [24]. The protocol messages are abstracted by terms, and the cryptographic primitives are abstracted by function symbols and assumed to be perfect (i.e., unbreakable). The algebraic properties of cryptographic primitives are described by equational relations over function symbols. Compared with complexity-based formal methods, the symbol model checking has the characteristics of automation.

In our previous work [25, 26], we verified the security of the QUIC handshake protocols with the cryptographic protocol verifiers ProVerif and Verifpal. However, in addition to the authentication and confidentiality of the QUIC handshake protocol, researchers may pay more attention to whether a certain function of the protocol can be realized. For example, in the scene of streaming media content transmission, the purpose of QUIC protocol is to handle more connections under the premise of ensuring security, so as to meet the function of providing content transmission quickly. Therefore, it becomes an important functional safety requirement of QUIC handshake protocol to ensure that the client can normally obtain the content transmitted by the server. Therefore, how to analyze and verify the temporal property of cryptographic protocols has become a significant research work.

3. The QUIC Handshake Protocol

In this section, we present a detailed review of the QUIC handshake protocol described in the IETF document [27].

The client of QUIC handshake protocol represents a subscriber's device (e.g., mobile phone or computer) that intends to start a secure connection to the network. The server of QUIC handshake protocol is where the client may connect to obtain a service. We assume that the public channel through which the client communicates with the server is under the control of malicious attackers.

(1) The flow of the QUIC handshake process as defined in the IETF document [27] is shown in Figure 2.

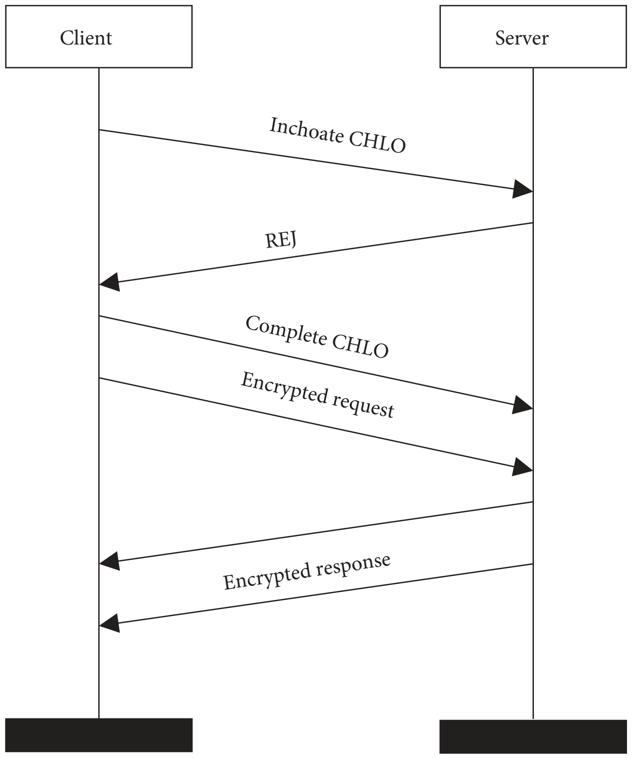

Figure 2: Initial handshake process of QUIC.

(2) Since the client does not cache the server configuration information in the beginning, the client needs to send a hello message (CHLO) to the server to get the reject message (REJ) from the server.

(3) When the server receives a CHLO message, it sends the REJ message to the client. The REJ message contains the following parts: (1) the config information including the server's long-term Diffie–Hellman public value, (2) a certificate chain authenticating the server, (3) a signature of the server config using the private key from the leaf certificate of the chain, and (4) a source address token (as an authenticated encryption block).

(4) If the handshake is successful, the server calculates the initial keys using the client's ephemeral Diffie–Hellman public value and its long-term Diffie–Hellman private value. Moreover, it calculates its own ephemeral Diffie–Hellman public value using its ephemeral Diffie–Hellman private value and calculates its final keys using the client's ephemeral Diffie–Hellman public value and its ephemeral Diffie–Hellman private value. It sends its ephemeral Diffie–Hellman public value encrypted with the initial secret key as a server hello message (SHLO) to the client. Finally, the server encrypts subsequent communication data using its forward-secure key.

(5) When the client receives the SHLO message, it sends the packet encrypted with its forward-secure key.

The detailed steps of the QUIC handshake process are depicted in Figure 3 and Table 1.

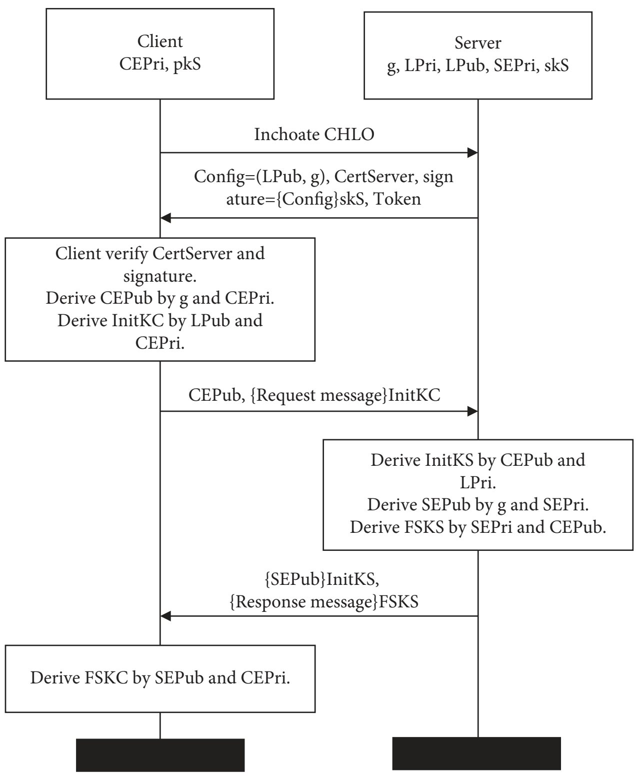
Client
CEPri, pkS
Server
g, LPri, LPub, SEPri, skS
Inchoate CHLO
Config=(LPub, g), CertServer, sign ature={Config}skS, Token
Client verify CertServer and signature.
Derive CEPub by g and CEPri.
Derive InitKC by LPub and CEPri.
CEPub, {Request message}InitKC
Derive InitKS by CEPub and LPri.
Derive SEPub by g and SEPri.
Derive FSKS by SEPri and CEPub.
{SEPub}InitKS,
{Response message}FSKS
Derive FSKC by SEPub and CEPri.

Figure 3: The detailed steps of the QUIC handshake process.

Table 1: Protocol messages and their annotations.

Message	Annotation
CEPri	Client's ephemeral Diffie–Hellman private value
LPri	Server's long-term DH private value
SEPri	Server's ephemeral Diffie–Hellman private value
LPub	Server's long-term DH public value
g	Primitive root
CEPub	Client's ephemeral Diffie–Hellman public value
SEPub	Server's ephemeral Diffie–Hellman public value
InitKC	Initial key of client
InitKS	Initial key of server
FSKC	Forward-secure key of client
FSKS	Forward-secure key of server
pkS	Public signature key of the server
skS	Private signature key of the server
{} skS	{} is signed using the private signature key of the server
{} InitKC	{} is encrypted using the initial key of client
{} InitKS	{} is encrypted using the initial key of server
{} FSKS	{} is encrypted using the forward-secure key of server

As an important function of the QUIC protocol, we consider whether the client of the protocol will eventually be able to obtain the data provided by the server. Based on the protocol state machine model, the functional requirements of the QUIC handshake protocol can be expressed by temporal property ϕ: the client will eventually reach the received data state when it reached the waiting data state. The linear temporal logic LTL formula of property ϕ is expressed as

$$(\text{Client Init} (\text{Client Finish}). \qquad (1)$$

4. Formal Verification of QUIC Handshake Protocol

4.1. Overview of Our Approach. The design framework for safety verification of QUIC handshake protocol is realized based on typical model checker SPIN and cryptographic protocol verifier ProVerif. Promela, as input to SPIN, is the formal model describing the state transfer of QUIC handshake protocol. Temporal properties φ are used to describe the safety requirements of QUIC handshake protocol. We give two strategies (Ref and Tran) to process ϕ to get the security properties $\varphi\prime$. We use ProVerif to verify the applied PI calculus [6] model and properties $\varphi\prime$ of QUIC handshake protocol so as to judge the counterexamples π of SPIN. The design framework of our method is shown in Figure 4.

4.2. Formal Models of QUIC Handshake Protocol. A. State transfer models of the QUIC handshake protocol.

Firstly, according to the process of QUIC handshake protocol, we present the state transfer models of the QUIC handshake protocol, as shown in Figures 5 and 6.

Initially, the client is at the initial state, and when it receives Restart, it sends an initial Hello message CHLO to the server. The client then enters the waiting configuration message (REJ) state. In this state, when the client receives the Restart or ICHLO_rej message, the initial Hello message CHLO is resent, and there is no state transition for the client. The ICHLO_rej message indicates that the server rejected the client's handshake request. When the client receives the configuration message REJ sent by the server, there are two possibilities. One is that the client receives the configuration message REJ from the server, and the server certificate T_CertS or the signature T_Sign Config of the configuration information is incorrect; then, the client will resend the initial Hello message CHLO, and the client state will not change. The other is that the client receives the configuration message REJ of the server, and the server certificate T_CertS and the signature T_Sign Config of the configuration information are correct; then, the client will send the completed Hello message CCHLO and the ciphertext Enc_ReqM of the request data to the server, and the client transfers to the waiting data state. At this point, if the client receives the CCHLO_REJ message sent by the server, it means that the server rejected the handshake message of Hello completed by the client. Then, the client will resend the initial Hello message CHLO to the server and transfer to the waiting REJ state. When the client receives the server's Hello message SHLO and the ciphertext Enc_ResM of the server's reply data, there are also two situations. One is that the client cannot decrypt the ciphertext sent by the server correctly; then, the client will send the initial Hello message CHLO to the server again. In the other case, the client can correctly decrypt the ciphertext sent by the server, indicating that the client has correctly obtained the data provided by the server. Then, the client sends the confirmation message Finish to the server, and the client transfers to the state of receiving the data. The annotations of state transfer messages are shown in Table 2.

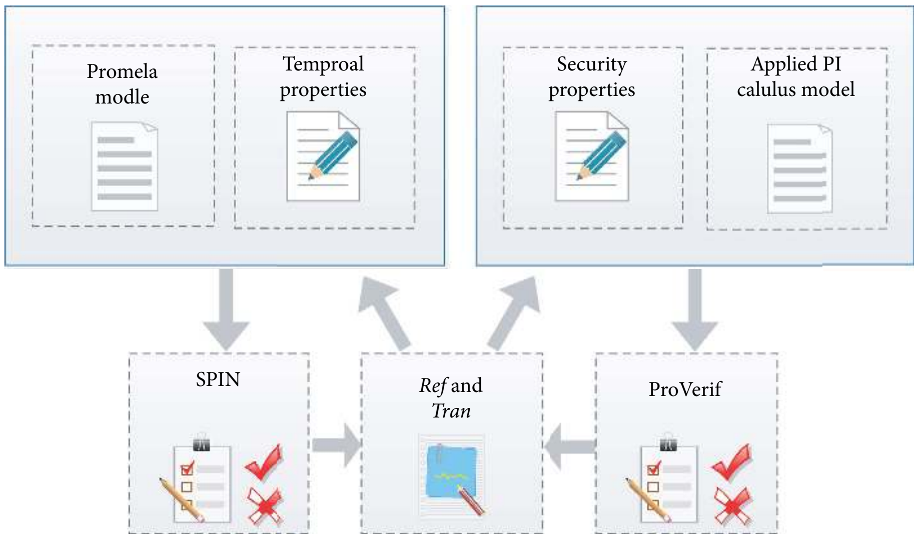

Figure 4: The design framework of the proposed method.

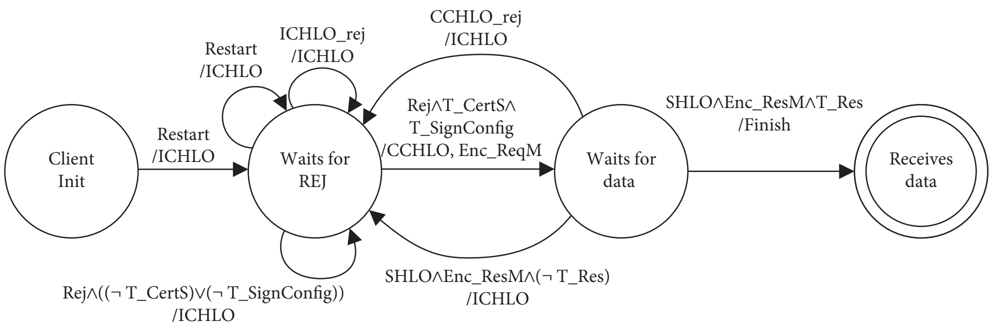
Client Init
Restart /ICHLO
Waits for REJ
Restart /ICHLO
ICHLO_rej /ICHLO
CCHLO_rej /ICHLO
Rej∧T_CertS∧ T_SignConfig /CCHLO, Enc_ReqM
Waits for data
SHLO∧Enc_ResM∧T_Res /Finish
Receives data
Rej∧((¬ T_CertS)∨(¬ T_SignConfig)) /ICHLO
SHLO∧Enc_ResM∧(¬ T_Res) /ICHLO

Figure 5: Client state transfer model of QUIC handshake protocol.

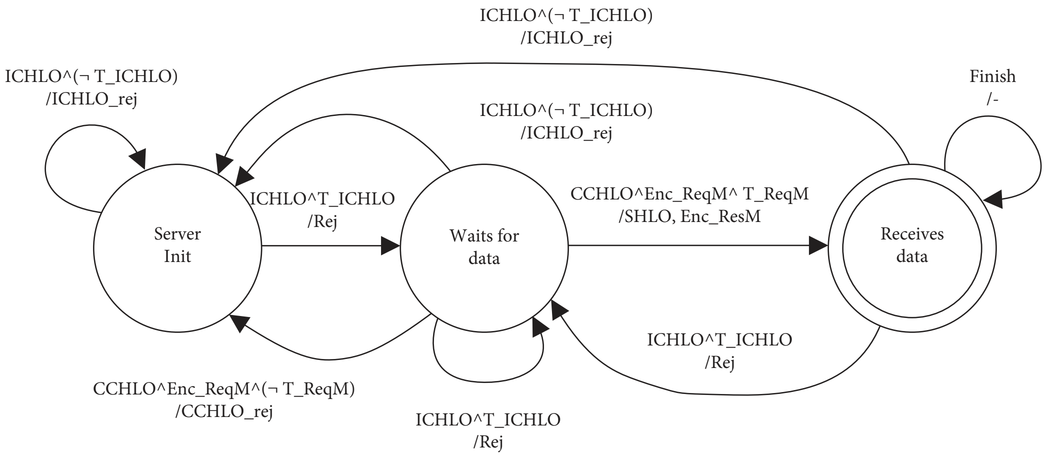
ICHLO^(¬ T_ICHLO) /ICHLO_rej
ICHLO^(¬ T_ICHLO) /ICHLO_rej
Finish /-
ICHLO^(¬ T_ICHLO) /ICHLO_rej
Server Init
ICHLO^T_ICHLO /Rej
Waits for data
CCHLO^Enc_ReqM^ T_ReqM /SHLO, Enc_ResM
Receives data
ICHLO^T_ICHLO /Rej
CCHLO^Enc_ReqM^(¬ T_ReqM) /CCHLO_rej
ICHLO^T_ICHLO /Rej

Figure 6: Server state transfer model of QUIC handshake protocol.

4.3. QUIC Handshake Protocol in Promela. Based on the state machines of the client and server, we use Promela to model the state machines of the QUIC handshake protocol. Firstly, we define and assign the type of data involved in the protocol. The data types are defined as byte and bool, and the named set is represented as follows:

$$\begin{aligned}&\text{byte} = \{\text{Restart, CHLO, Rej, CCHLO, SHLO}\}),\\&\text{byte} = \{\text{CHLO\_rej, CCHLO\_rej, Finish}\}),\\&\text{bool} = \{\text{ClientInit, Server Init}\},\\&\text{bool} = \{\text{Client Finish, Server Finish}\}.\end{aligned} \tag{2}$$

Table 2: State transfer messages and their annotations.

Message	Annotation
CHLO	Client's initial Hello message
ICHLO	rej It is the server's rejection message to CHLO
CCHLO	rej It is the server's rejection message to CCHLO
T_CertS	It indicates that the server's certificate is true
T_SignConfig	It indicates that the server's configuration message is true
SHLO	Server's Hello message
Enc_ResM	Ciphertext of the server's data
Enc_ResM	Ciphertext of the client's data
T_Res	It indicates that the server's response message is true
T_Req	It indicates that the client's request message is true

We preassign a fixed value to each byte type message, which is used by the client, server, and attacker to identify the message. We initially assign false to all bool variables. Client Finish and Server Finish assign true when the client and server successfully reach the final state, respectively. The assignment of the message is defined as follows:

$$\begin{aligned} \text{byte Restart} &= 0, \\ \text{CHLO} &= 1, \\ \text{Rej} &= 2, \\ \text{CCHLO} &= 3, \\ \text{SHLO} &= 4, \\ \text{byte CHLO\_rej} &= 5, \\ \text{CCHLO\_rej} &= 6, \\ \text{Finish} &= 7, \\ \text{bool Client Init} &= \text{false}, \\ \text{Server Init} &= \text{false}, \\ \text{bool Client Finish} &= \text{false}, \\ \text{Server Finish} &= \text{false}. \end{aligned} \tag{3}$$

For the communication between the client and server, we define two channels to carry the message transmission between the client and server. In the model, messages are sent and received synchronously. The message types in the channel are byte and bit. Variables of type byte, which have values ranging from 0 to 255, represent the data for protocol interactions. Variables of type bit have values ranging from 0 to 1. This type represents the correct value of protocol interaction data, where 1 means correct and 0 means incorrect:

$$\begin{aligned} \text{chan } C \text{ to } S &= [0] \text{of } \{\text{byte}, \text{bit}\}, \\ \text{chan } S \text{ to } C &= [0] \text{ of } \{\text{byte}, \text{bit}\}. \end{aligned} \tag{4}$$

According to the client state machine model shown in Figure 5, we used Promela to model the state transition of client, as shown in Figure 7.

In the client model, we define two temporary variables, Cmg and Ccond, to receive messages from the channel for the client. The value of Cmg represents the message sent by the server, and the value of Ccond indicates whether the message of Cmg is correct. For example, if the client expects to receive the configuration message REJ sent by the server, the client will determine whether the currently received Cmg value is 2 and whether the Ccond value is 1. The client chooses the next state transition based on the result of the judgment.

```
proctype Client(chan in, out)
{
byte Cmg;
bit Ccond;
S0: in?Cmg, Ccond;
        if
                :: (Cmg == 0 && Ccond == 1) -> out!ICHLO, 1; goto S1
        fi;
S1: in?Cmg, Ccond;
        ClientInit = true;
        if
                :: (Cmg == 0 && Ccond == 0) -> out!ICHLO, 1; goto S1
                :: (Cmg == 2 && Ccond == 1) -> out!CCHLO, 1; goto S2
                :: (Cmg == 2 && Ccond == 0) -> out!ICHLO, 1; goto S1
                :: (Cmg == 5 && Ccond == 0) -> out!ICHLO, 1; goto S1
                :: (Cmg == 6 && Ccond == 0) -> out!ICHLO, 1; goto S1
        fi;
S2: in?Cmg, Ccond;
        if
                :: (Cmg == 4 && Ccond == 1) -> out!Finish, 1; goto S3
                :: (Cmg == 6 && Ccond == 0) -> out!ICHLO, 1; goto S1
                :: (Cmg == 4 && Ccond == 0) -> out!ICHLO, 1; goto S1
        fi;
S3: ClientFinish = true;

}
```

Figure 7: Promela model of the client.

According to the server state machine model shown in Figure 6, we used Promela to model the state transition of the server, as shown in Figure 8.

In the server model, we also defined two temporary variables, Smg and Scond, to receive messages from the channel. The value of Smg represents the message sent by the client, and the value of Scond indicates whether the message Smg is correct. For example, when the server receives a message in its initial state, it determines the value of the message Smg. If the message Smg value is 1, the server thinks it received a CHLO message. At the same temporal, if the value of Scond is 0, then the server thinks the initial Hello message sent by the client is incorrect and sends the rejection message CHLO_rej and 0 to the channel. If the value of Scond is 1, then the server thinks it received a correct initial Hello message from the client. The server then sends the configuration information Rej and 1 into the channel and changes its state to the waiting data state. In addition, we output a message (0, 1) at the beginning of the server model to simulate the environment sending the Restart message to the client.

```
proctype Server(chan in, out)
{
byte Smg;
bit Scond;
out!0, 1;
S0: in?Smg, Scond;
        if
                ::(Smg == 1 && Scond == 1) -> out!Rej, 1; ServerInit = true;
goto S1
                ::(Smg == 1 && Scond == 0) -> out!ICHLO_rej, 0; goto S0
        fi;
S1: in?Smg, Scond;
        if
                ::(Smg == 3 && Scond == 1) -> out!SHLO, 1; goto S2
                ::(Smg == 3 && Scond == 0) -> out!CCHLO_rej, 0; goto S0
                ::(Smg == 1 && Scond == 1) -> out!Rej, 1; goto S1
                ::(Smg == 1 && Scond == 0) -> out!CCHLO_rej, 0; goto S0
        fi;
S2: in?Smg, Scond;
        if
                ::(Smg == 1 && Scond == 1) -> out!Rej, 1; goto S1
                ::(Smg == 1 && Scond == 0) -> out!ICHLO_rej, 0; goto S0
                ::(Smg == 7 && Scond == 1) -> ServerFinish = true; goto S2
        fi;
}
```

Figure 8: Promela model of the server.

The attacker's behavior based on the Dolev–Yao model is modeled, as shown in Figure 9. The attacker model receives messages from the channel through variables Img and Icond. When the attacker receives the messages, it determines the value of Img. If it is a rejected message, the attacker will resend the message to the channel. If the value of Img is a data message, the attacker will change the value of the message Icond from 1 to 0 and then sends Img and Icond to the channel. For example, when the value of Img is 3 and the value of Icond is 1, it means that the attacker intercepted the correct CCHLO message. Then, the attacker will send the messages Img and 0 to the channel. If the server receives these messages from the attacker, it means that the server cannot decrypt the ciphertext of CCHLO. Therefore, the server will send a rejection message CHLO_rej to the client.

4.4. QUIC Handshake Protocol in Applied PI Calculus. Client process is shown in Figure 10. First, the client sends a CHLO message on the channel c. We remark that an attacker can access this public channel. Then, it waits for the message including five variables bounded to variables $x1$, $x2$, $x3$, $x4$, and pkX, respectively. After that, the client checks if the variables $x3$ and pkX are, respectively, certificate and public key pkS of the server. Then, the client checks the signature of $x2$ using variable pkX, which is the public key belonging to the server. If the result of the signature check is equivalent to $x1$, then the client calculates its initial key InitKC using $x1$ and its ephemeral private value CEPri. Subsequently, it sends the ephemeral public value CEPub $(\exp(g, \text{CEPri}))$ and the ciphertext of the request message $(\text{enc}(\text{ReqM}, \text{InitKC}))$ on the channel. Then, it waits for a message of form $(x5, x6)$. When the client captures the message, it decrypts the variable $x5$ using its initial key InitKC and then bounds the return to variable $x7$. Normally, the variable $x7$ should be the server's ephemeral public value. Finally, the client calculates its forward-secure key FSKC using variable $x7$ and its ephemeral private value CEPri and decrypts the ciphertext $x6$ using the forward-secure key FSKC.

The server process is shown in Figure 11. First, the server bounds the message of its input to variable $x1$. It checks whether $x1$ is the legal inchoate client hello CHLO. Then, the server sends its certificate Cert Server, the long-term public value LPub $(\exp(g, \text{LPri}))$, the signature of LPub and Token, and its public key Token on the channel c. Next, the server waits for the message including two parts which are, respectively, bounded to $x2$ and $x3$. It obtains its initial key InitKS using $x2$ and the long-term private value LPri. The server checks if the ciphertext $x3$ is the request message using InitKS. Then, the server calculates SEPub with SEPri and obtains its forward-secure key FSKS with $x2$ and SEPri. Finally, it sends the ciphertext of SEPub and response message ResM on the channel c.

4.5. Our Verification Approach. In this section, we propose a verification approach to verify the safety property of QUIC handshake protocol. In this approach, we consider the existence of an attacker who has complete control over the message on the public channel. In other words, the attacker can intercept, tamper, and replay the information on the public channel. However, the attacker cannot decrypt the ciphertext without knowing the correct secret key. This attacker model is called the Dolev–Yao model and denoted by I. In addition, we define a stronger attacker model called I^+, which does not consider cryptographic operations.

The algorithm of the QUIC handshake protocol safety verification method is shown in Table 3.

According to Table 2, the input M and M' are, respectively, the Promela model and the applied PI calculus model of the QUIC handshake protocol. π is the counterexamples of the verification of M and temporal properties φ by the model checker SPIN. In the process of verification for the Promela model, attacker's capacity follows attacker model I^+. The attacker can capture and change message in the public channel or privately falsify the encrypted message without the secret key and without influence from the encryption mechanism. Therefore, verification counterexample π created by the attacker cannot be realized in the process of implementing specific protocol. For example, the attacker falsifies an encrypted message which is sent by an honest participant so that the receiver cannot decode encrypted message or obtain correct instruction for decryption, leading to a counterexample from model verification. However, the attacker cannot falsify encrypted message in practice due to lack of the secret key.

For the above uncertainty in verification counterexample of the Promela model, this paper classifies counterexamples based on involvement of attacker behavior in cryptogrammic operation. The counterexample π from temporal

```
proctype I(chan inC, outC, inS, outS)
{
byte Img;
bit Icond;
    do
        ::inC?Img, Icond ->
            atomic{
                if
                    :: (Img == 1 && Icond == 1) -> outS!ICHLO, 0
                    :: (Img == 1 && Icond == 1) -> outC!Restart, 0
                    :: (Img == 3 && Icond == 1) -> outS!CCHLO, 0
                    :: (Img == 7 && Icond == 1) -> outS!Finish, 1

                    :: (Img == 0 && Icond == 0) -> outC!Restart, 0
                    :: (Img == 1 && Icond == 0) -> outS!ICHLO, 0
                    :: (Img == 3 && Icond == 0) -> outS!CCHLO, 0
                    :: (Img == 7 && Icond == 1) -> outS!Finish, 1
                fi;
                }
        ::inS?Img, Icond ->
            atomic{
                if

                    :: (Img == 2 && Icond == 1) -> outC!Rej, 0
                    :: (Img == 4 && Icond == 1) -> outC!SHLO, 0

                    :: (Img == 5 && Icond == 0) -> outC!ICHLO_rej, 0
                    :: (Img == 6 && Icond == 0) -> outC!CCHLO_rej, 0

                    :: (Img == 0 && Icond == 0) -> outC!Restart, 0

                    :: (Img == 2 && Icond == 0) -> outC!Rej, 0
                    :: (Img == 4 && Icond == 0) -> outC!SHLO, 0
                fi;
                }
    od
}
```

Figure 9: Promela model of the attacker.

property verification based on the Promela model can be classified into

(1) Counterexample π where attacker creation cannot be determined
(2) True counterexample π where creation can be made by the attacker
(3) False counterexample π where creation cannot be made by the attacker

First, for counterexample classification, this paper proposes a refining strategy Ref with the temporal property based on the counterexample:

Ref 1: if counterexample π with temporal property φ is classified into Class 1, then it is impossible to create counterexample π for attacker behavior which shall be determined in formal verification for cryptographic protocols accommodating attacker model I. Therefore, temporal property φ is simplified into security constraint ω to be

```
let Client (sspuk:pkey) =
        out (c, CHLO);
        in(c, (x1:G, x2:bitstring, x3:bitstring, x4:bitstring));
        let (=x1) = checksign (x2, sspuk) in
        new CEPri: exponent;
        let InitCK = exp (x1, CEPri) in
        out (c, (exp (g, CEPri), enc (ReqM, InitCK)));
        in (c, (x5:G, x6:bitstring));
        let x7 = decP (x5, InitCK) in
        let FSKC = exp (x7, CEPri) in
        let ResMx = dec (x6, FSKC) in.
```

Figure 10: The client process in applied PI calculus.

```
let Server(sspuk:pkey, ssprk:skey) =
        in(c, x1:bitstring);
        if x1 = CHLO then
        new LPri: exponent;
        new Token: bitstring;
        out(c, (exp(g, LPri), sign(exp(g, LPri), ssprk), CertServer, Token));
        in(c, (x2:G, x3:bitstring));
        let InitSK = exp(x2, LPri) in
        let ReqMx = dec(x3, InitSK) in
        new SEPri: exponent;
        let SEPub = exp(g, SEPri) in
        let FSKS = exp(x2, SEPri) in
        out(c, (encP(SEPub, InitSK), enc(ResM, FSKS))).
```

Figure 11: The server process in applied PI calculus.

Table 3: Safety verification algorithm for QUIC handshake protocol.

```
Algorithm: safety verification method
Input: $M$, $M'$, $\varphi$
Output: $\pi$
(1) result = 0
(2) Input $M$ and $\varphi$ into SPIN to verify and obtain $\pi$
(3) if $\pi$! = null
(4) Refines $\varphi$ to get $\omega$ by refining strategy Ref 1.
(5) Transforms $\omega$ to the properties $\varphi\prime$ by the transformation strategy Tran.
(6) if verification result of $\varphi\prime$ is false
(7) result + +
(8) Refines $\varphi$ to get new $\varphi$ by refining strategy Ref 2.
(9) goto line 3
(10) else
(11) Refines $\varphi$ to get new $\varphi$ by refining strategy Ref 2.
(12) goto line 3
(13) if result = 0
(14) Model $M$ satisfies the property $\varphi$.
(15) else
(16) Model $M$ dose not satisfy the property $\varphi$.
```

verified under which the attacker cannot create the existing counterexample.

Ref 2: if counterexample π with temporal property φ is classified into Class 2 or Class 3, then the counterexample π is a true or false counterexample which is created by the attacker and not consistent with the temporal property of cryptographic protocols. In order to find other true counterexamples in protocol, it is necessary to simplify current temporal property φ and eliminate attacker's behavior of creating the current counterexample based on existing temporal property φ to obtain new temporal property φ_1.

When analyzing functional safety of cryptographic protocols, this section considers whether the attacker can falsify message sent by the honest participant in protocol. If the attacker can falsify such message, then the attacker also can send false message to protocol as an honest participant; as a result, cryptographic protocols cannot be executed normally in accordance with standards. Therefore, security constraint ω in refining strategy Ref 1 can be described as message m that cannot be falsified and captured by the attacker. Based on standard Dolev–Yao model I, security constraint ω can be classified into 1. message falsification not related to cryptogrammic operation; 2. message falsification related to cryptogrammic operation. According to classification for ω, the security constraint can be transformed into security strategy Tran:

Tran 1: Class 1 security constraint ω is transformed into confidentiality φ' based on ProVerif:

$$\text{query attacker}\,(m), \tag{5}$$

where m is the plaintext message sent by the honest participant. If an attacker can obtain message m, then this attacker can falsify the plaintext message m. Therefore, it is necessary to determine whether the attacker can obtain message m, i.e., whether message m is confidential.

Tran 2: Class 2 security constraint ω is transformed into confidentiality and consistency φ' based on ProVerif:

$$\begin{aligned}&\text{query attacker}\,(\text{key}),\\&\text{inj}-\text{event}\,(e\,(\text{key}))==\;>\text{inj}-\text{event}\,(e\,(\text{key})),\end{aligned} \tag{6}$$

where key is the secret key for creating message m. When capturing key, the attacker can falsify message m so that it is necessary to determine whether key is confidential. In addition, in key exchange protocol, the attacker may exchange key with other honest participants and then send falsified and false encrypted message to other honest participants as an honest participant so that cryptographic protocols cannot be implemented normally. Therefore, it is necessary to verify consistency of key for the Class 2 security constraint.

Based on the above property refining strategy and constraint transformation strategy, this paper proposes a temporal property verification method based on counterexamples:

(1) If φ is a temporal property of protocol to be verified, then Promela model and property φ are verified by SPIN to obtain counterexample π_1, and property φ is transformed into security constraint ω for analyzing attacker behavior with property refining strategy Ref 1.
(2) The type of security constraint ω is determined based on constraint transformation strategy Tran to give security property φ_1'.

(3) The applied PI calculus model M' is added with declaration of confidentiality and event of the consistency relationship based on security property φ_1'. Then, property φ_1' is verified with the cryptographic protocol verifier ProVerif based on model M'.

(4) If verification results of ProVerif are false, then the attacker can create counterexample π_1, and π_1 is classified into Class 2, i.e., π_1 is a true counterexample. If verification results are true, then the attacker cannot create counterexample π_1, and π_1 is classified into Class 3, i.e., π_1 is a false counterexample. Then, the Promela model is modified with property refining strategy Ref 2, and attacker behavior of creating existing counterexample is eliminated to obtain simplified temporal property φ_1. Besides, Promela model and property φ_1 are verified again with SPIN. If new counterexample π_2 exists in verification results of property φ_1, then property φ_1 is simplified with property refining strategy Ref 1, and Step 2 is jumped to. If there is no new counterexample in verification results, then verification is ended.

5. Verification Results and Analysis

We use the model checker SPIN to verify the property φ, and the verification results are shown in Table 4.

We have listed counterexamples of violations of property φ in Table 3. Next, we need to determine whether the attacker is able to construct these counterexamples. The security constraints that we will need to further verify are denoted by $\omega_1 \sim \omega_4$ in Table 2.

ω_1 is the Class 1 security constraint, so confidentiality verification is required for the CHLO messages sent by the client. Through the security constraint transformation strategy Tran 1, ω_1 transforms to the confidentiality φ_1' based on Proverif:

$$\text{qurry attacker}\,(\text{CHLO}). \tag{7}$$

ω_2 is the Class 2 security constraint, so confidentiality and consistency verification is required for the server's private key skS. Through the security constraint transformation strategy Tran 1 and Tran 2, ω_2 transforms to the confidentiality and consistency φ_2' based on Proverif:

$$\begin{aligned}&\text{qurry attacker}\,(\text{skS}),\\&\text{inj}-\text{event}\,(\text{accptskS}\,(\text{skS})) == \;>\text{inj}-\text{event}\,(\text{sendskS}\,(\text{skS})).\end{aligned} \tag{8}$$

ω_3 is the Class 1 and Class 2 security constraint, so confidentiality and consistency verification is required for the client's initial session key InitKC, and confidentiality is required for the client's temporary Diffie–Hellman public key CEPub. Through the security constraint transformation strategy Tran 1 and Tran 2, ω_3 transforms to the confidentiality and consistency φ_3' based on Proverif:

$$\begin{aligned}&\text{qurry attacker}\,(\text{CEPub}),\\&\text{qurry attacker}\,(\text{InitKC}),\\&\text{inj}-\text{event}\,(\text{accpt InitKC}\,(\text{InitKC}))\\&\;== \;>\text{inj}-\text{event}\,(\text{send InitKC}\,(\text{InitKC})).\end{aligned} \tag{9}$$

ω_4 is the Class 2 security constraint, so confidentiality and consistency verification is required for the server's initial session key InitKC and final session key FSKS. Through the security constraint transformation strategyTran 2, ω_4 transforms to the confidentiality and consistency φ_4' based on Proverif:

$$\begin{aligned}&\text{qurry attacker}\,(\text{FSKS}),\\&\text{qurry attacker}\,(\text{InitKS}),\\&\text{inj}-\text{event}\,(\text{accptFSKS}\,(\text{FSKS}))\\&\;== \;>\text{inj}-\text{event}\,(\text{send FSKS}\,(\text{FSKS})),\\&\text{inj}-\text{event}\,(\text{accpt InitKS}\,(\text{InitKS}))\\&\;== \;>\text{inj}-\text{event}\,(\text{send InitKS}\,(\text{InitKS})).\end{aligned} \tag{10}$$

We use ProVerif to verify the above properties based on the applied PI calculus model of QUIC handshake protocol. The results for these properties are shown in Table 5.

The results show that the properties φ_1', φ_3', and φ_4' are not valid and the attacker can forge the messages CHLO and CEPub because these two messages are transmitted in plaintext, so φ_1' and φ_3' are not satisfied. The attacker can impersonate the client to send the forged CEPub to the server and can complete the establishment of the initial session key and the final session key with the server. Therefore, the server's initial session key InitKS and the final session key FSKS do not satisfy the confidentiality and consistency, so φ_4' is not satisfied.

According to the above results, it can be found that the attacker has the conditions to attack the temporal property φ. That is to say, when the client is in a malicious environment, there is a possibility that the client cannot finally obtain the data provided by the server.

As a possible fix, we propose a revised QUIC handshake protocol and the newly added elements are marked in red. We use the client's private key to sign CEPub and CHLO, and it can be guaranteed that the attacker cannot forge the client's CEPub and CHLO message. We have verified the newly revised QUIC handshake protocol shown in Figure 12. Our analysis shows that this revised protocol satisfies the liveness property we have considered.

Table 4: Verification results of SPIN.

Model checker properties		Counterexamples		Security constraints	
φ	After the client reaches the waiting data state, it will finally reach the receiving data state	π	Received the message Restart,0; the client stays in waiting configuration state		
φ_1	The client does not receive the message Restart,0 when it reached the waiting data state	π_1	Received the message CHLO_rej,0; the client stays in waiting configuration state	ω_1	The attacker cannot forge the CHLO message sent by the client
φ_2	The client did not receive the message CHL0_rej,0 when it reached the waiting data state	π_2	Received the message Rej,0; the client stays in waiting configuration state	ω_2	The attacker cannot forge the certification of server and cannot forge the signature of message REJ
φ_3	The client did not receive the message Rej,0 when it reached the waiting data state	π_3	Received the message CCHLO_rej,0; the client transfers to waiting configuration state from waiting data state	ω_3	The attacker cannot forge CEPub and ciphertext EncReqM sent by the client
φ_4	The client did not receive the message CCHLO_rej,0 when it reached the waiting data state	π_4	Received the message SHLO,0; the client transfers to waiting configuration state from waiting data state	ω_4	The attacker cannot forge the ciphertext EncResM and EncSEPub sent by the server

Table 5: Verification results of ProVerif.

	Security properties	Results	
φ_1'	qurry attacker (CHLO)	False	False
φ_2'	qurry attacker (skS) inj − event (accptskS (skS)) == > inj − event (sendskS (skS))	True	True
φ_3'	qurry attacker (CEPub) qurry attacker (InitKC) inj − event (accpt InitKC (InitKC)) == > inj − event (send InitKC (InitKC))	False True	False
φ_4'	qurry attacker (FSKS) qurry attacker (InitKS) inj − event (accptFSKS (FSKS)) == > inj − event (sendFSKS (FSKS)) inj − event (accpt InitKS (InitKS)) == > inj − event (send InitKS (InitKS))	False False	False

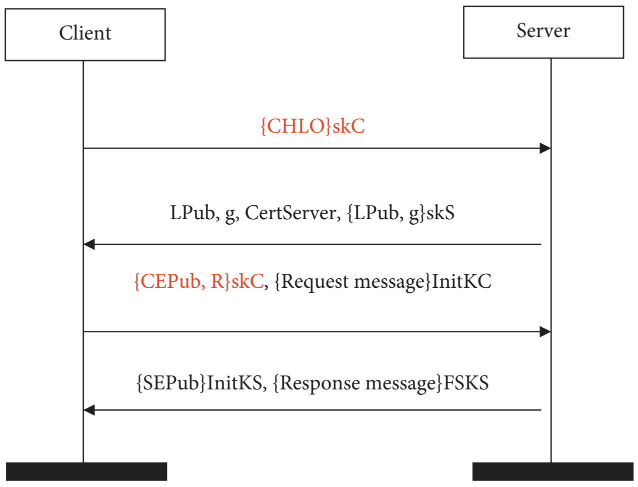
Client
Server
{CHLO}skC
LPub, g, CertServer, {LPub, g}skS
{CEPub, R}skC, {Request message}InitKC
{SEPub}InitKS, {Response message}FSKS

Figure 12: A possible fix for QUIC handshake protocol.

6. Conclusions

This paper proposes a new modeling and verification method for functional safety of cryptographic protocol to solve the problem of false counterexamples existing in verification for functional safety with typical model checking technology and that formal verification tool cannot directly conduct modeling for time property describing functional safety. This paper verifies functional safety of cryptographic protocol with the collaborative modeling method which realizes modeling for cryptographic protocol with Promela and PI calculation. In addition, this paper provides a time property refining strategy and a constraint transformation strategy based on counterexamples, as well as a functional safety verification method for cryptographic protocol based on typical model checker SPIN and formal verification tool ProVerif. With these methods, this paper conducts formal modeling and verification for safety, finds a defect of violating functional safety in design, and provides improvement actions for the design defect.

Regarding the disadvantages of the current work, we would like to remark that the analysis results by typical model checker and cryptographic protocol verifier are based on the symbolic protocol model, where we assume that the cryptography is perfect, and we do not take into account the computational strengths of the primitives. Though, this assumption is of theoretical research interest, it is too strong to be practical. Thus, if the underlying cryptographic primitives are broken, the protocol would also be faulty, even though it is proven correct and secure on the symbolic model.

In the future, we would like to go one step further to investigate the correctness of the protocol implementations, with respect to the specification. The idea is that ensuring the security of the protocol design is not enough, and we also need to ensure that the implementation of the protocol state machine is secure. We will also extend the current work to the computational cryptography model, where the probability of breaking cryptographic primitives is taken into account.

Data Availability

The data used to support the findings of this study have been deposited in the GitHub repository: https://github.com/bxk2008/Data-for-Hindawi.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] L. Hirschi, R. Sasse, and J. Dreier, *Security issues in the 5G standard and how formal methods come to the rescue*, ERCIM News, vol. 2019, no. 117, 2019.

[2] D. Basin, C. Cremers, and C. Meadows, *Model Checking Security Protocols*, Springer, New York, NY, USA, 2018.

[3] G. Lowe, "An attack on the Needham-Schroeder public-key authentication protocol," *Information Processing Letters*, vol. 56, no. 3, pp. 131–133, 1995.

[4] A. Armando and L. Compagna, "SATMC: a sat-based model checker for security protocols," in *Proceedings of the European Workshop on Logics in Artificial Intelligence*, pp. 730–733, Lisbon, Portugal, September 2004.

[5] D. Basin, S. Mödersheim, and L. Viganò, "OFMC: a symbolic model checker for security protocols," *International Journal of Information Security*, vol. 4, no. 3, pp. 181–208, 2005.

[6] B. Blanchet, "Modeling and verifying security protocols with the applied pi calculus and ProVerif," *Foundations and Trends in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, 2016.

[7] E. M. Clarke, S. Jha, and W. Marrero, "Verifying security protocols with brutus," *ACM Transactions on Software Engineering and Methodology*, vol. 9, no. 4, pp. 443–487, 2000.

[8] V. Cortier, S. Delaune, and P. Lafourcade, "A survey of algebraic properties used in cryptographic protocols," *Journal of Computer Security*, vol. 14, no. 1, pp. 1–43, 2006.

[9] J. C. Mitchell, M. Mitchell, and U. Stern, "Automated analysis of cryptographic protocols using mur/spl phi," in *Proceedings of the 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097)*, pp. 141–151, Oakland, CA, USA, May 1997.

[10] D. X. Song, "Athena: a new efficient automatic checker for security protocol analysis," in *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pp. 192–202, Mordano, Italy, June 1999.

[11] J. Zhang, L. Yang, W. Cao, and Q. Wang, "Formal analysis of 5G EAP-TLS authentication protocol using proverif," *IEEE Access*, vol. 8, pp. 23674–23688, 2020.

[12] S. Tripakis and C. Courcoubetis, "Extending promela and spin for real time," in *Proceedings of the Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop (TACAS'96)*, pp. 329–348, Passau, Germany, March 1996.

[13] L. Babenko and I. Pisarev, "Security analysis of the electronic voting protocol based on blind intermediaries using the spin verifier," in *Proceedings of the 2018 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pp. 43–435, Zhengzhou, China, October 2018.

[14] T. Ninet, A. Legay, R. Maillard, L. Traonouez, and O. Zendra, "Model checking the ikev2 protocol using spin," in *Proceedings of the 2019 17th International Conference on Privacy, Security and Trust (PST)*, pp. 1–7, Fredericton, Canada, August 2019.

[15] D. Basin, J. Dreier, L. Hirschi, S. Radomirović, R. Sasse, and V. Stettler, "A formal analysis of 5G authentication," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1383–1396, Toronto, Canada, October 2018.

[16] C. Cremers and M. Dehnel-Wild, "Component-based formal analysis of 5G-AKA: channel assumptions and session confusion," in *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, San Diego, CA, USA, February 2019.

[17] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols," in *Computer Aided Verification*, N. Sharygina and H. Veith, Eds., Springer, Berlin, Germany, 2013.

[18] J. Zhang, Q. Wang, L. Yang, and F. Tao, "Formal verification of 5G-EAP-TLS authentication protocol," in *Proceedings of the Fourth IEEE International Conference on Data Science in Cyberspace (DSC 2019)*, Hangzhou, China, June 2019.

[19] S. R. Hussain, O. Chowdhury, S. Mehnaz, and E. Bertino, "LTEinspector: a systematic approach for adversarial testing of 4G LTE," in *Proceedings of the 25th Annual Network and Distributed System Security Symposium*, San Diego, CA, USA, February 2018.

[20] S. R. Hussain, M. Echeverria, I. Karim, O. Chowdhury, and E. B. Greasoner, "A property-directed security and privacy analysis framework for 5G cellular network protocol," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 669–684, London, UK, November 2019.

[21] S. Qiu, D. Wang, G. Xu, and S. Kumari, "Practical and provably secure three-factor Authentication protocol based on extended chaotic-maps for mobile lightweight devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, p. 1, 2020.

[22] D. Wang and P. Wang, "Two birds with one stone: two-factor Authentication with security beyond conventional bound," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 4, p. 1, 2018.

[23] S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena, "Two-factor Authentication with end-to-end password security," in *Public-Key Cryptography–PKC 2018*, M. Abdalla and R. Dahab, Eds., Springer, New York, NY, USA, 2018.

[24] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[25] J. Zhang, L. Yang, X. Gao, G. Tang, J. Zhang, and Q. Wang, "Formal analysis of QUIC handshake protocol using symbolic model checking," *IEEE Access*, vol. 9, pp. 14836–14848, 2021.

[26] J. Zhang, L. Yang, X. Gao, and Q. Wang, "Formal analysis of QUIC handshake protocol using proverif," in *Proceedings of the 7th IEEE International Conference on Cyber Security and Cloud Computing, CSCloud 2020/6th IEEE International Conference on Edge Computing and Scalable Cloud, EdgeCom 2020*, pp. 132–138, New York, NY, USA, August 2020.

[27] I. Jana and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," in *Proceedings of the Internet-Draft QUIC Transport Protocol*, February 2020.