Formal Analysis of SDN Authentication Protocol with Mechanized Protocol Verifier in the Symbolic Model

Lili Yao, Jiabing Liu, Dejun Wang, Jing Li and Bo Meng
(Corresponding author: Bo Meng)

School of Computer Science, South-Central University for Nationalities
708 Minzu Ave, Hongshan Qu, Wuhan, Hubei Sheng 430074, China
(Email: mengscuec@gmail.com)
(Received Aug. 21, 2017; revised and accepted Nov. 1 & Oct. 21, 2017)

Abstract

With the wide development and applications of SDN, its security has attracted the attention of the people. In this study, in the beginning we apply applied PI calculus in symbolic model to formalize Mynah authentication protocol and mechanized analyze it with ProVerif. We find that there are two security vulnerabilities. And then, we propose an improved Mynah authentication protocol to address the vulnerabilities found by us. At the same time, the improved Mynah protocol is modeled by applied PI calculus and analyzed with ProVerif. Finally, we develop and deploy the improved Mynah authentication protocol to open source controller ONOS and switch Open vSwitch to validate its securities.

Keywords: Authentication; Formal Method; ProVerif; Security Protocol

1 Introduction

The purpose of introducing SDN is to establish a flexible data access and forwarding method in network and then to deal with the barriers for deploying the new technologies of the network protocols and to decrease cost and to overcome the difficulty of network management, especially provide a good environment for large-scale implementation of cloud computing and virtualization. With the wide development and applications of SDN [15, 16, 18], people have paid a special attention to its security [2, 25]. Owning to that the design and development of most SDN controllers that are the key component in SDN network at first focuses on the schedule and control of network resources and ignoring the security considerations of the controller itself [27]. SDN network is facing enormous security challenges, for example, the lack of trust mechanisms [25, 28]. Kloti and Kotronis [13] use STRIDE tool and Attack Tree to provide a security analysis of OpenFlow-based SDN and find that there are security risks such as information disclosure, denial of service and intervention vulnerabilities in a controller or a channel between the controller and the switch. In order to address the authentication [29], Shin *et al.* [26] present the FRESCO security application development framework in OpenFlow-based SDN. Based on Nox [9] and FRESCO, Porras *et al.* [23] introduce the FortNOX, a software add-on that applies a digital signature to implement the authentication of users. Mattos and Duarte [17] present AuthFlow which is a mechanism for authentication and access control based on host credentials. Dangovas and Kuliesius [7] introduce a SDN-based authentication and access control system to provide strong AAA (authentication, authorization and accounting) schemes.

Recently in order to address the vulnerability of DatapathID (DPID) duplication and provide the authentication between the controller and the switch, Kang *et al.* [12] propose Mynah authentication protocol and claim that it can address the two vulnerabilities. However, security analysis of Mynah protocol is not only not clear, but also DPID duplication problem has not fully been solved found by us. Hence in the study, we present formal analysis and an improved Mynah authentication protocol.

The main contributions of this study are summarized as follows:

1) The state-of-art of security research of SDN network is introduced in detail.

2) Apply applied PI calculus in the symbolic model to formalize Mynah authentication protocol and mechanized analyze it with mechanized tool ProVerif. The result shows that it cannot provide mutual authentication between the controller and the switch and is unable to deal with DPID duplication.

3) Propose an improved Mynah authentication protocol to address the security vulnerabilities found by

us. At the same time, the improved Mynah authentication protocol is modeled by applied PI calculus and mechanized analyzed with ProVerif. The result shows that the improved Mynah authentication protocol resolves DPID duplication and provides confidentiality of data and authentication between the switch and the controller.

4) Develop and deploy the improved Mynah authentication protocol to open source controller ONOS and switch Open vSwitch to validate authentication and confidentiality. The experiment result shows that it can provide confidentiality of data and authentication between the switch and the controller.

2 Related Works

In the beginning of design and development, most SDN controllers [11, 21, 24] focus on the scheduling and control of network resources, ignoring the security considerations of the controller itself [27]. So SDN is facing security challenges, for example, the lack of trust mechanisms [28].

With SDN development and applications, attacks of the controller and the switch significantly increased. Kloti and Kotronis [13] use STRIDE tool and Attact Tree to provide a comprehensive analysis of the security of OpenFlow protocol, whether it is a controller or a channel between the controller and the switch. There are security risks such as information disclosure, denial of service and intervention vulnerabilities. The existing OpenFlow protocol [8, 10, 22, 30] often uses SSL/TLS protocol and does not guarantee the authentication between the controller and the switch [3]. SSL/TLS protocol itself is overloaded and is not good choice to large-scale deployment. In addition, the controller is facing a man-in-the-middle attack, denial of service, bypassing the firewall and other traditional networks already exist security risks [14].

About the authentication [29]. Shin *et al.* [26] present the FRESCO security application development framework in OpenFlow-based SDN. The main function of FRESCO is to provide modular interface and deployment platform to build security services, through the corresponding FRESCO scripting language to define and implement security services, simplifying the development of security applications and complexity of debugging. It facilitates the controller to update and extend security services in the operating mode. Based on Nox [9] and FRESCO [26], Porras *et al.* [23] introduce the FortNOX, a software add-on to improve the flaws in the OpenFlow control plane and to deploy new security modules while using existing security services, each of the flow rules is signed using the role-based data source authentication method and the identity of the user is verified by verifying the signature data to ensure the security of the session. Mattos and Duarte [17] present AuthFlow which is an authentication and access control mechanism based on host credentials. The mechanism uses IEEE 802.1X standard and RADIUS authentication server to authenticate the host above the MAC layer, but AuthFlow has not used the signed certificate as access credentials. Dangovas and Kuliesius [7] introduce an authentication and access control system that binds the name (users, addresses) and user machines to the unambiguously defined network appliances and its ports and register the switch and host information to the controller and authenticate, satisfied strong AAA (authentication, authorization and accounting) schemes.

Recently in order to address the vulnerability of DPID duplication and provide authentication between the controller and the switch, Kang *et al.* [12] propose Mynah authentication protocol and claim that it can address the two vulnerabilities. However, security analysis of Mynah protocol is not only not clear, but also DPID duplication problem has not fully been solved found by us.

3 The Applied PI Calculus & ProVerif

The applied PI calculus [1] is proposed by Abadi *et al.* in 2001, which is a formal language [6, 19] used to formalize the modeling of concurrent processes. Applied PI calculus builds on pure PI calculus [20]. From pure PI calculus, we inherit constructs for communication and concurrency, also add functions and equations. In applied PI calculus, messages may then consist of atomic names or consist of values constructed from names and functions. The advantage of this is that we can easily treat standard data types, reducing the limitation of data representation. The applied PI calculus using functions to represent generic cryptographic primitives, such as encryption, decryption, digital signatures, *etc.* It does not need to construct a new cryptographic primitive for each cryptographic operation with good versatility. We can also describe attacks against protocols that rely on (equational) properties of some of those primitives. Therefore, it can express and analyze fairly sophisticated security protocols.

The grammar for processes of applied PI calculus language is similar to the PI calculus. Process P, Q as the basic unit, the input process is 0 that empty process; $Q|P$ is the parallel composition of P and Q; the replication $!P$ behaves as innumerable copies of P running in parallel. The process $vn.P$ that defines the variable name and $in(M, x : t)$ indicates that it is input in the channel in the process and $out(M, x : t)$ indicates that the process is output in the channel, $if M = N then P else Q$ indicates that the execution process is selected according to the judgment condition, let $x = M$ in P else Q presents the event evaluation, $R(M1, \cdots, Mk)$ represents the macro definition.

ProVerif [5] is a mechanized tool based on the Dolev-Yao model for automated verification of security protocol properties and is developed by Blanchet in 2001. It can be used to analyze and validate security protocols that use Horn clauses or applied PI calculus to model various

cryptographic primitives. Includes shared key cryptography, public key cryptography, digital signature, hash function and Diffie-Hellman key exchange. At the same time, it avoids the problem of the state space explosion. It can analyze and verify the strong confidentiality, authenticity, more general consistency and process of observation equivalent. ProVerif has successfully analyzed a large number of complex security protocols.

4 Mynah Authentication Protocol

4.1 Review

Mynah authentication protocol [12] based on OpenFlow protocol is designed to address the vulnerability of DPID duplication and to provide the authentication security service based on DPID. OpenFlow protocol uses DPID as the identifier of the data plane, but does not provide any means to authenticate DPID of the switch. The messages in Mynah authentication protocol are shown in Figure 1.

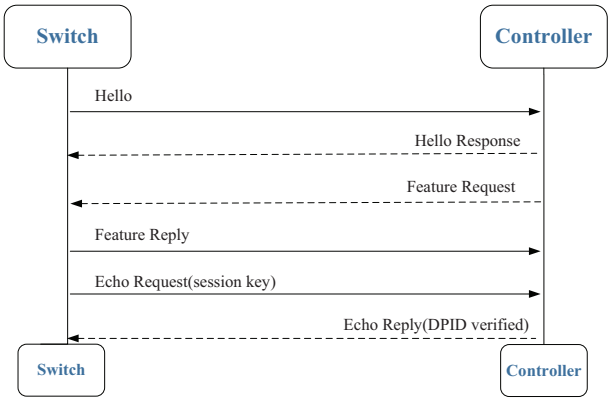

Figure 1: The messages of Mynah authentication protocol

Hello and Hello Response: After the switch and the controller establish a TCP connection, the switch sends a Hello message to the controller and the controller produces the Hello Reponse message to the switch to determine the negotiation on version of OpenFlow protocol used by the communicating parties. Hello message contains the highest version of OpenFlow protocol that the sender can support. The switch and the controller each received a Hello message from each other, compare the highest version supported by other parties with the highest version supported locally and finally the version with the lower version as the final version. If the negotiation process fails, a HELLO FAILED error message is returned to the peer and the connection is terminated.

Feature Request and Feature Reply: After the version of OpenFlow protocol is determined between the communicating parties, the controller sends a Feature Request message to the switch requesting configuration parameters and other related information for the switch. In the SDN network architecture, a controller manages the flow table updates of multiple switches at the same time. Therefore, it is necessary to save the independent information of the switch as an identification flag during the connection establishment process, so as to avoid interfering with the instructions sent. After receiving the Feature Request, the switch sends a Feature Reply message to the controller. Feature Reply message contains actions, DPID-based authentication and *etc.*

Echo Request and Echo Reply: The switch sends the Feature Reply message and indicates that the switch-controller can perform DPID-based authentication. The switch can send its session key in the Echo Request message. The session key depends on DPID, timestamp and transaction serial number. The session key should be encrypted using either asymmetric key algorithms or symmetric key algorithms. The switch encapsulates SessionKey encrypted with a public key into the Echo Request message and sends it to the controller. After the controller receives Echo Request message, it first checks DPID to verify the identity of the switch and then decrypts SessionKey using the corresponding private key. And then the controller checks whether the DPID, timestamp and transaction ID is valid or not. If any of the three parameters is invalid, the controller rejects the connection from the switch. If all information is valid but has a connection with the same DPID, the controller still rejects the connection. Finally, the controller generates a DPID verification message and encapsulates it into the Echo Reply message, which is encrypted with SessionKey and sends it to the switch.

4.2 Formalize Mynah Authentication Protocol Using the Applied PI Calculus

4.2.1 Function and Equational Theory

The functions and equations used in the modeling process are described in this section. This study uses applied PI calculus to formalize Mynah authentication protocol. Figure 2 depicts the Mynah authentication protocol function and equation theory.

The message x is encrypted by function $senc(x, PU)$ with public key PU and message x is decrypted by function $sdec(x, PU)$ with public key PU. The message x is encrypted by function $aenc(x, PU)$ with public key PU and the message x is decrypted with function $adec(x, PR)$ with private key PR. The private value is received by function $PR(y)$ as an input and a private key is generated as an output, at the same time the common value is received as an input through function $PU(y)$ and a public key is generated as an output.

4.2.2 Processes

The whole Mynah authentication protocol process consists of two processes: switch process and controller process. They together constitute the main process, as shown in Figure 3.

$$\begin{array}{l} Funaenc(x, PU). \\ Funadec(x, PR). \\ Funsenc(x, PU). \\ Funsdec(x, PU). \\ FunPU(y). \\ FunPR(y). \\ \\ equationadec(aenc(x, PU(y)), PR(y)) = x. \\ equationsdec(senc(x, PU(y)), PU(y)) = x. \end{array}$$

Figure 2: Function and equational theory

$$\|mainprocess = (processSwitch|processController)\|$$

Figure 3: Main process

$$\begin{array}{l} letprocessSwitch \triangleq \\ newmsgVersionS; newmsgTypeHelloS; newxid1; \\ out(c, (msgVersionS, msgTypeHelloSxid1)); \\ in(c, (= msgVersionCon, = msgType1, = xidRly1)); \\ in(c, (= msgType2, = xidRly2)); \\ newmsgTypeFeaReply; \\ out(c, (msgTypeFeaReply, xidRly2, datapathID)) \\ newtimestamp; newxid3; newmsgTypeEchoReq; \\ letsessionkeyS = getSessionKey \\ (timestamp, xid3, datapathID)in \\ letsecretKey = aenc(sessionkeyS, PU(keyop1)) \\ inout(c, (msgTypeEchoReq, xid3, secretKey)); \\ in(c, = msgType3, = xidRly3, = secretMessage); \\ ifsdec(secretMessage, PR(sessionkeyS)) \\ = OPmessagethenout(c, finished) \end{array}$$

Figure 4: Mynah authentication protocol switch process

The switch process is shown in Figure 4. First, it sends the protocol version number $msgVersionS$ to controller process through public channel c and then receives the protocol version number $msgVersionS$ from controller process through the public channel c for message version negotiation. After the version is determined, switch process receives the configuration information request Feature Request through public channel c, generates the response Feature Reply and sends its own DPID to the controller process through public channel c. After controller process receives the DPID, switch process uses the DPID, timestamp and transaction sequence xid3 to generate the SessionKey and uses asymmetric encryption algorithm to encrypt secretKey and sends it to controller process through public channel c. And then from controller process through open channel c to receive controller encrypted message, the use of existing SessionKey and symmetric decryption algorithm decryption secretMessage, if the decryption is successful to verify the correctness of key, though open channel c output finished, to the end of this protocol communication.

The controller process is shown in Figure 5. It sends the protocol version number $msgVersionS$ to switch process through public channel c and then receives protocol version number $msgVersionS$ of the switch from switch process through public channel c and performs the message version negotiation. This process is similar to the switch process. Once the version is determined, controller process immediately sends Feature Request over public channel c and waits for the Feature Reply of the receiving process. In the received Feature Reply response, the controller process obtains DPID of sender's process and saves it. And then through open channel c to receive the session process SessionKey, using private key $PR(keyop1)$ decryption secertkey get the session key $SessionKeyC$, the session key DPID and previously saved DPID comparison verification. If authentication is successful, the parameter OPmessage is encrypted using $SessionKeyC$ and sent to switch process via open channel c.

$$\begin{array}{l} letprocessController \triangleq \\ newmsgVersionC; newmsgTypeHelloC; \\ newxid4; \\ out(c, (msgVersionC, msgTypeHelloC, xid4)); \\ in(c, (= msgVersionSw, = msgType4, = xidRly4)); \\ newmsgTypeFeaReq, newxid5; \\ out(c, (msgTypeFeaReq, xid5)); \\ in(c, (= msgType5, = xidRly5, = datapathID)); \\ in(c, (= msgType6, = xidRly6, = secretkey)); \\ letsessionkeyC = adec(secretkey, PR(keyop1))in \\ newmsgTypeEchoReply; newflag4; \\ letsecretMessage = senc(OPmessage, \\ PR(sessionkeyC))in \\ newmsgTypeEchoReply, newxidRly6, newflag4; \\ out(c, (msgTypeEchoReply, xidRly6, \\ flag4, secretMessage)) \end{array}$$

Figure 5: Mynah protocol controller process

4.3 Automatic Verification of Authentication and Confidentiality of Mynah Authentication Protocol with ProVerif

Here we use the statements query attack (OPmessage) to verify confidentiality of OPmessage message and then use non-injective agreements to model authentication, Mynah protocol authentication is shown in Table 1.

The ProVerif inputs in Figure 7 are entered into the ProVerif and the analyze outputs are shown in Figure 8 to Figure 10.

Table 1: Authentication

Non-injective agreement	Authentication
ev:endauthcon_sMynah(x)==>ev:beginaauthcon_sMynah(x).	Verify the authentication from the controller to the switch
ev:endauthswit_cMynah(x)==>ev:beginaauthswit_cMynah(x).	Verify the authentication from the switch to the controller

Figure 8 is the result of confidentiality of the message OPmessage. The result is true. According to the specification of Mynah authentication protocol, the switch sends the session key in the Echo Request message. The session key is encrypted using either asymmetric key algorithms. The switch encapsulates SessionKey encrypted with a public key into Echo Request message and sends it to the controller. After the controller receives Echo Request message, it first checks DPID to verify the identity of the switch and then decrypts SessionKey using the corresponding private key. The attacker cannot obtain the private key and hence cannot decrypt the message OPmessage.

$$
\begin{array}{l}
funaenc/2. funadec/2. \\
funsenc/2. funsdec/2. \\
funPU/1. funPR/1. \\
fungetSessionKey/3. \\
\\
equationadec(aenc(x, PU(y)), PR(y)) = x. \\
equationsdec(senc(x, PU(y)), PU(y)) = x.
\end{array}
$$

Figure 6: Functions and equations in ProVerif

$$
\begin{array}{l}
queryattacker : OPmessage. \\
queryev : endauthcon\_sMynah(x) \\
==> ev : beginaauthcon\_sMynah(x). \\
(**ControllerauthenticatesSwitch**) \\
queryev : endauthswit\_cMynah(x) \\
==> ev : beginaauthswit\_cMynah(x). \\
(**SwitchauthenticatesController**) \\
..... \\
eventbeginaauthswit\_cMynah(echoRequest); \\
out(c, echoRequest); ..... \\
in(c, echoReply); \\
eventendauthcon\_sMynah(echoReply); \\
..... \\
in(c, echoRequest); \\
eventendauthswit\_cMynah(echoRequest); \\
..... \\
eventbeginaauthcon\_sMynah(echoReply); \\
out(c, echoReply).
\end{array}
$$

Figure 7: Mynah authentication protocol in ProVerif

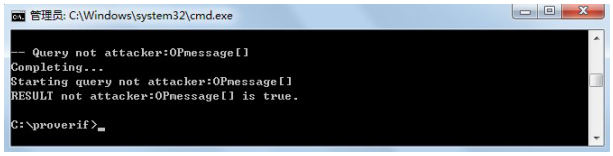

Figure 8: OPmessage confidentiality

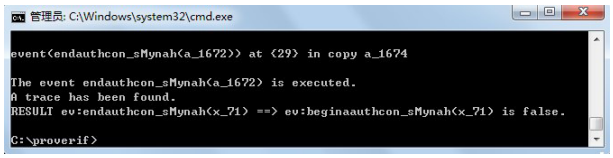

Figure 9: The analysis result of authentication from the controller to the switch

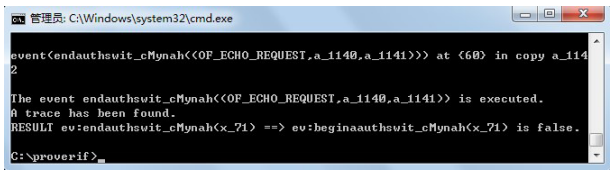

Figure 10: The analysis result of authentication from the switch to the controller

Figure 9 presents the result of *ev: endauthcon_sMynah(x) ==> ev: beginaauthcon_sMynah(x).* Figure 10 shows the result of *ev: endauthswit_cMynah(x) ==> ev: beginaauthswit_cMynah(x).* The results are false and show that the switch and the controller cannot authenticate each other. According to the specification of Mynah authentication protocol, there has no authentication mechanism between switch and controller.

About DPID duplication, owning to that the DPID is not classified, the attacker can get the DPID and then use the DPID to launch the communication early. According to the specification of Mynah authentication protocol, the attacker generates the Echo Request message and sends it to the controller. The controller checks whether the DPID, timestamp and transaction ID is valid or not. If any of the three parameters is invalid, the controller rejects the connection from the switch. If all information is valid but has a connection with the same DPID, the controller still rejects the connection. Because the DPID is

fresh and not is used, the controller generates a DPID verification message and encapsulates it into the Echo Reply message and sends it to the switch. So Mynah authentication protocol cannot prevent DPID duplication.

5 Improved Mynah Authentication Protocol

5.1 The Design of Improved Mynah Authentication Protocol

Improved Mynah automation protocol framework is shown in Figure 11. Firstly, there is a version negotiation between the switch and the controller. If it is successful, the controller gets switch configuration information, otherwise ends the conversation. Second, after the switch configuration information is obtained, the controller initiates the authentication request and then if the authentication request succeeds, the switch generates a session key. At the same time the digital signature mechanism is introduced to implement the authentication between the switch and the controller. If the request fails, terminate session. Finally, if authentication is successful, session key is used to encrypt the follow-up message. Otherwise, the session is terminated.

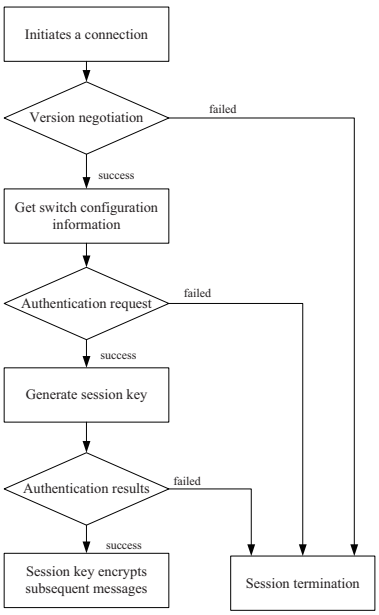

Figure 11: Improved Mynah authentication protocol framework

The improved Mynah authentication protocol introduces a digital signature to implement authentication between the switch and the controller, confidentiality of data and to prevent DPID duplication.

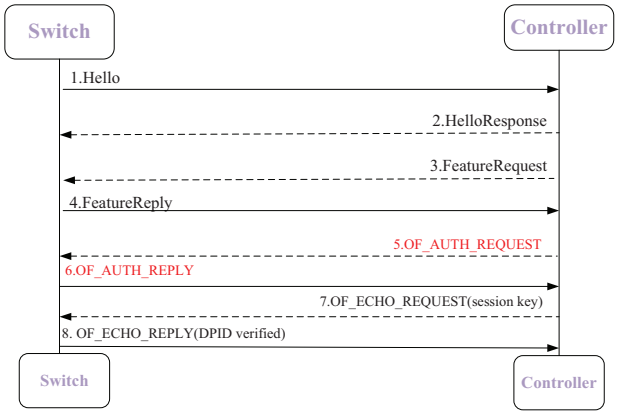

Figure 12: Improved Mynah authentication protocol messages

The improved Mynah message structures are shown in Figure 12. After the controller obtains DPID of the switch, it needs to verify the identity of the switch. The controller initiates an authentication request and uses a digital signature to generate a message digest for protocol type, event sequence number and DPID in the OF_AUTH_REQUEST message and encapsulates it into OF_AUTH_REQUEST message and then sends it to the switch. After receiving OF_AUTH_REQUEST message, the switch verifies the digital signature using public key of the controller. And then it generates a session key, encrypts the key using an asymmetric encryption algorithm, calculates the digital signature and encapsulates it in OF_AUTH_REPLY message to the controller. The controller verifies after receiving OF_AUTH_REPLY message. Finally, the controller encrypts subsequent communication message entries using the obtained session key.

1) The controller and the switch still need to complete version negotiation process. The controller obtains configuration information about the switch.

2) After obtaining DPID of the switch, the controller initiates the authentication request.

3) The controller uses its own private key to generate a digital signature for OF_AUTH_REQUEST message and appends the digital signature to message OF_AUTH_REQUEST as a whole message to switch. The switch verifies the digital signature through public key of the controller after receiving OF_AUTH_REQUEST message, verifies whether the initiator of request is indeed controller and then the switch generates session key and uses symmetric encryption algorithm to encrypt the session key, generates a digital signature of OF_AUTH_REPLY message using its own private key and sends it to the controller. After receiving OF_AUTH_REPLY message, the controller verifies the digital signature using the public key of the switch. If the verification of the digital signature is successful, the controller

Table 2: Improved Mynah authentication protocol message structures

Description of the improved Mynah authentication protocol field		
Message item	Field Name	Description
OF_AUTH_REQUEST	Header	Message head, Type=OFPT_AUTH_REQUEST
	DPID	Create a data plane identifier for the connected switch
	SignedMessage	Digital signature, authentication controller identity
OF_AUTH_REPLY	Header	Message head, Type=OFPT_AUTH_REPLY
	SecretKey	The encrypted session key
	SignedMessage	Digital signature, authentication switch identity
OF_ECHO_REQUEST	Header	Message head, Type=OFPT_ECHO_REQUEST
	SecretMessage	Encrypted message
OF_ECHO_REPLY	Header	Message head, Type=OFPT_ECHO_REPLY
	Flag	Verify that the authentication result is TRUE

obtains the session key sent by the switch using symmetric decryption algorithm. Otherwise the protocol is ended.

4) After the controller obtains the session key generated by the switch, the session key is used to encrypt data. And then the controller encapsulates it into OF_AUTH_REQUEST message and sends it to the switch. When the switch received OF_AUTH_REQUEST message, it uses the session key to decrypt it. Finally, the switch generates OF_ECHO_REPLY message and sends it to the controller.

OF_AUTH_REQUEST message is generated by the controller, which contains DPID of the switch and the digital signature generated by the controller using the private key. OF_AUTH_REPLY message is generated by the switch, which contains encrypted session key and the digital signature generated by the switch using the private key. The message fields and descriptions are shown in Table 2.

5.2 Formalize Improved Mynah Authentication Protocol Using the Applied PI Calculus

The function and the equation theory are shown in Figure 13 in formal model of the improved Mynah authentication protocol. The message x is digitally signed by function $sign(x, PR)$ with private key PR and the correctness of message x signature is verified by function $versign(x, PU)$ with public key PU. The message x is encrypted by function $senc(x, PU)$ with public key PU and message x is decrypted by function $sdec(x, PU)$ with public key PU. The message x is encrypted by function $aenc(x, PU)$ with public key PU and the message x is decrypted with function $adec(x, PR)$ with private key PR. The private value is received by function $PR(y)$ as an input and a private key is generated as an output and the common value is received as an input through function $PU(y)$ as an input and a public key is generated as an output.

$$
\begin{array}{l}
Funaenc(x, PU). \\
Funadec(x, PR). \\
Funsenc(x, PU).Funsdec(x, PU). \\
Funsign(x, PR).Funversign(x, PU). \\
FunPU(y).FunPR(y). \\
\\
equationadec(aenc(x, PU(y)), PR(y)) = x. \\
equationsdec(senc(x, PU(y)), PU(y)) = x. \\
equationversign(sign(x, PR(y)), PU(y)) = x.
\end{array}
$$

Figure 13: Improved Mynah authentication protocol function and equality theory

The switch process is shown in Figure 14. In the first part, the switch first completes the version negotiation and the FeatureReply response through public channel c, sends the relevant configuration information to the controller process and then switch process receives authentication request OF_AUTH_REQUEST through public channel c and then uses controller public key $PU(keyrp1)$ and function $versign(x, PU)$ to confirm the digital signature. If the verification result verifies the authenticity of signature, the session key is generated by DPID, timestamp and transaction sequence xid3 and the session key is encrypted using asymmetric encryption function $aenc(x, PU)$. Finally, switch private key $PR(keyrp2)$ and function $sign(x, PR)$ to sign above parameters and sends them to the controller process via open channel c. The second part, through public channel c from controller process to receive secretMessage, using the existing SessionKey and symmetric decryption algorithm $sdec(x, PU)$ to decrypt secretMessage. If the decryption is successful, the flag of OF_ECHO_REPLY message is set to true and then through public channel c sends the message to controller and the protocol communication ends.

The controller process is shown in Figure 15. First, it completes the version negotiation and message featureRequest to the switch process through open channel c and then obtains DPID of the switch. Second, use controller private key $PR(keyrp1)$ and function $sign(x, PR)$ to generate a digital signature and en-

capsulate it into the OF_AUTH_REQUEST message to send the authentication request through public channel c. Then the controller process receives the authentication response message OF_AUTH_REPLY through public channel c, it uses switch public key $PU(keyrp2)$ and function $versign(x, PU)$ to confirm the digital signature. If the verification result confirms the authenticity of signature, the function $adec(x, PU)$ decrypts the SessionKey and encrypts the OPmessage with session key, encapsulating it into the OF_ECHO_REQUEST message and sending it to the switch process through public channel c.

$$
\begin{array}{l}
\text{letprocessSwitch} \triangleq \\
newmsgVersionS; \\
newmsgTypeHelloS; newxid1; \\
out(c, (msgVersionS, msgTypeHelloS, xid1)); \\
in(c, (= msgType3, = xidRly3, \\
= datapathID, = SignedMessageC2S)); \\
ifversign(SignedMessageC2S, PU(keyrp1), \\
(msgType3, xidRly3, datapathID)) \\
= truethennewtimestamp; \\
newmsgTypeAuthReply; \\
letsessionkeyS = getSessionKey \\
(timestamp, xidRly3, datapathID) \\
inletsecretKey = aenc(sessionkeyS, \\
PU(keyop1))in \\
letSignedS2C = sign \\
((msgTypeAuthReply, xidRly3, secretKey), \\
PR(keyrp2))in \\
out(c, (msgTypeAuthReply, xidRly3, \\
secretKey, SignedS2C));
\end{array}
$$

Figure 14: The switch process

$$
\begin{array}{l}
letprocessController \triangleq \\
newmsgVersionS; \\
newmsgTypeHelloS; newxid1; \\
out(c, (msgVersionS, msgTypeHelloS, xid1)); \\
letSignedC2S = \\
sign((msgTypeAuthReq, xid7, datapathID), \\
PR(keyrp1))in \\
out(c, (msgTypeAuthReq, xid7, \\
datapathID2, SignedC2S, SignedC2S); \\
in(c, (= msgType7, = xidRly7, \\
= secretKey, = SignedMessage)); \\
ifversign(SignedMessageS2C, PU(keyrp2), \\
(msgType7, xidRly7, secretKey)) = true \\
thennewmsgTypeEchoReq; newxid8; \\
letsessionkeyC = \\
adec(secretKey1, PR(keyop1))in \\
letsecretMessage = \\
senc(OPmessage, PR(sessionkeyC))in \\
out(c, (msgTypeEchoReq, xid8, secretMessage));
\end{array}
$$

Figure 15: The controller process

$$
\begin{array}{l}
funaenc/2. \\
funadec/2. \\
funsdec/2. \\
funsenc/2. \\
funsign/2. \\
funversign/2. \\
funPU/1. \\
funPR/1. \\
\\
equationadec(aenc(x, PU(y)), PR(y)) = x. \\
equationsdec(senc(x, PU(y)), PU(y)) = x. \\
equationversign(sign(x, PR(y)), PU(y)) = x.
\end{array}
$$

Figure 16: Functions and equations in ProVerif

$$
\begin{array}{l}
queryattacker : OPmessage. \\
queryev : endauthcon\_s(x) \\
==> ev : beginaauthcon\_s(x). \\
queryev : endauthswit\_c(x) \\
==> ev : beginaauthswit\_c(x). \\
.....in(c, authRequest); \\
ifversign(SignedMessageC2S, PU(keyrp1)) = \\
(msgType3, xidRly3, datapathID1)then \\
eventendauthcon\_s(SignedMessageC2S); \\
.....eventbeginaauthswit\_c(SignedS2C); \\
out(c, authReply); \\
.....eventbeginaauthcon\_s(SignedS2C); \\
out(c, authRequest); \\
.....in(c, authReply); if \\
versign(SignedMessageS2C, PU(keyrp2)) = \\
(msgType7, xidRly7, secretKey1)then \\
eventendauthswit\_c(SignedMessageS2C);
\end{array}
$$

Figure 17: Improved Mynah authentication protocol in ProVerif

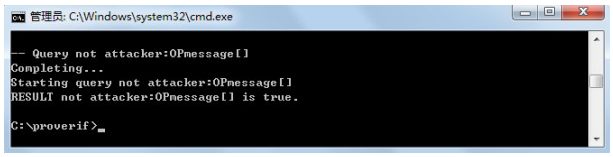

Figure 18: Confidentiality of OPmessage

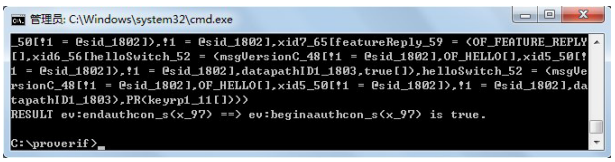

Figure 19: The analysis result of the authentication from the controller to the switch

Table 3: Authentication

Non-injective agreement	Authentication
ev:endauthcon_s(x)==>ev:beginaauthcon_s(x).	Verify the authentication from the controller to the switch
ev:endauthswit_c(x)==¿>ev:beginaauthswit_c(x).	Verify the authentication from the switch to the controller

Figure 20: The analysis result of the authentication from the switch to the controller

5.3 Automatic Verification of Authentication and Confidentiality of Improved Mynah Authentication Protocol with ProVerif

After the formal model of the improved Mynah authentication protocol was generated, ProVerif is used to perform the formal analysis. First the target that needs to be proved is defined and then the analysis with ProVerif is implemented. The authentication of the improved Mynah authentication protocol is shown in Table 3. This process is similar to the verification process of Mynah authentication protocol.

ProVerif scripts in Figure 17 are as input to ProVerif and the outputs of analysis are shown in Figure 18 to Figure 20.

Figure 18 shows the result of the formalize analysis of confidentiality of OPmessage and the result is true, it proved that the improved Mynah authentication protocol provides confidentiality of OPmessage.

Figure 19 shows the result of $ev : endauthcon\_s(x) ==> ev : beginaauthcon\_s(x)$. Figure 20 is the result of $ev : endauthswit\_c(x) ==> ev : beginaauthswit\_c(x)$.The two results are true and indicate that the switch and the controller can authenticate each other.

In the improved Mynah authentication protocol, because digital signature mechanism is adopted, the switch uses the private key to sign the message when sending the message OF_AUTH_REPLY. After receiving OF_AUTH_REPLY message, the controller needs to verify the digital signature using the public key of the switch, to ensure the authentication for the switch and integrity of the message OF_AUTH_REPLY. Similarly, the controller uses the digital signature for the message OF_ECHO_REQUEST, the switch receives the OF_ECHO_REQUEST and uses the controller's public key to verify the digital signature to ensure the authentication for the controller and the integrity of the message OF_ECHO_REQUEST.

About DPID duplication, according to the specification of the improved Mynah authentication protocol, the attacker cannot generate the digital signature for message OF_ECHO_REQUEST because the private key of controller is secret. At the same time the attacker also cannot produce the digital signature for OF_AUTH_REPLY because the private key of the switch is secret. So the improved Mynah authentication protocol can prevent DPID duplication.

6 Develop and Deploy the Improved Mynah Authentication Protocol

The improved Mynah authentication protocol was developed and deployed to open source controller ONOS [4] and switch Open vSwitch to validate its security. The improved Mynah authentication protocol program consists of the controller side developed with Java language and the switch side developed with C language. The improved Mynah authentication protocol program is deployed to the controller ONOS and the switch Open vSwitch and be recompiled. The improved Mynah authentication protocol development architecture is illustrated in Figure 21.

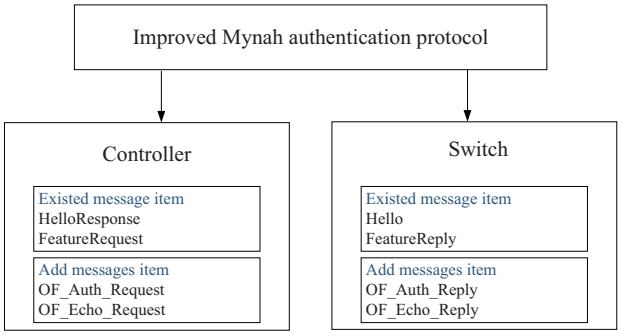

Figure 21: Improved Mynah authentication protocol development architecture

Running environment is composed of the hardware environment and the software environment. The hardware environment is for Intel Dore dual-core CPU, memory 2GB. The software environment is for the operating system for Ubuntu 14.0.1, the controller ONOS version 1.3, the switch Open vSwitch version 1.0, the virtual network simulation platform Mininet version 1.0, Apache Karaf version 3.0.2.

6.1 ONOS Controller Side Development

In the ONOS platform, the controller and the switch connection consist of three steps. The controller starts listening to port 6633, the switch launches the connection with controller, the controller and the switch make negotiation on version and the message transmission.

The main class in ONOS controller side is the OpenFlowControllerImpl class that implements the OpenFlowController interface. During the initialization of OpenFlowControllerImpl class, the controller object Controller class and OpenFlowSwitchAgent class are instantiated at first to monitor the state of the switch that has established the connection. And then it calls the Controller.start(OpenFlowAgent agent) method for parameter configuration and starts the server side, listens port, waits for the switch to establish a connection.

After the connection is established between the controller and the switch, the controller generates OFChannelHandler object and listens for the messages sent by the switch. After receiving the message sent by the switch, the controller first analyzes the type of message and performs different method calls according to the message type. When the message type is Feature_Reply, the controller should send the authentication request message OF_Auth_Request. When the message type is Auth_Reply, the controller should send the OF_Echo_Request message.

In ONOS design mode, the OFMessage interface is defined, which contains all the data items and operations in an OpenFlow message. Its subinterfaces are also defined for behavior and data items according to the explicitly defined message items in OpenFlow protocol specification. The implementation class developed by us process the data according to the method defined by the parent interface and the specific version information. The data that need to transmit between the controller and the switch is encapsulated into the instantiated object and communicates through the ChannelPipeline pipeline in ONOS platform. The protocol message structure is shown in Figure 22.

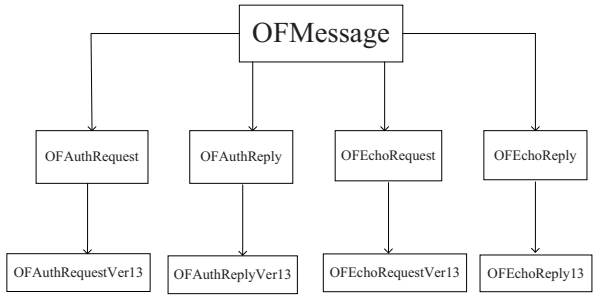

Figure 22: ONOS message structure

Because each message has a separate implementation class, the ONOS controller side need to check Type and Version of the message received and then find the corresponding implementation classes according to the Type and Version for message encapsulation, encryption, decryption and authentication *etc.* ONOS controller has implemented the implementation classes of HelloResponse message and FeatureRequest message, so we focus on the implementation classes and deployments of OF_Auth_Request message and OF_Echo_Request message.

When the controller sends an authentication request, it first needs to obtain parameters such as version number, Xid and DPID. Then the digital signature is generated for the DPID as part of the OF_Auth_Request message and sent to the switch. When the controller receives the OF_Auth_Reply message, it is also necessary to verify the digital signature using the public key. If the verification succeeds, the authentication from the controller to the switch is true. If the verification fails, the controller terminates the session and releases the connection. At this time the controller saves the DPID in its own database. If the attacker wants to use the same DPID to establish connection, then the controller needs to query database. The results show that there is a duplicate DPID, then the controller refused connection.

6.2 Open vSwitch Switch Side Development

When the switch receives the OpenFlow message, it also needs to perform the action according to the message type. When the message type is Auth_Request, the switch should send the OF_Auth_Reply message. When the message type is Echo_Request, the switch should send the OF_Echo_Reply message.

After receiving the OF_Auth_Request message, the switch first verifies the digital signature using the public key of the controller. If the verification succeeds, the authentication from the controller to switch is true. The switch then generates the session key and encrypts the session key and generates the digital signature as a field of the OF_Auth_Reply message sent to the controller. After receiving the OF_Auth_Reply message, the controller verifies the digital signature using the public key of the switch. If the verification is successful, the controller obtains the session key and can use it to encrypt the follow-up message to the switch. At the same time, the switch uses the session key to decrypt the message.

6.3 Results and Analysis

After the deployment of the improved Mynah authentication protocol program, we find that when DPID of the switch changes, the controller terminates and releases the connection, restarts the new session request by the switch and completes the authentication process again. Otherwise, the controller remains in listening state. When the digital signature verification fails, the controller or switch

also to terminate the session process. That is consistent with the results of the formal analysis with ProVerif.

7 Conclusion

With the rapid development and applications of SDN network, people have paid a special attention to its security. Recently Kang *et al.* propose an authentication protocol called Mynah authentication protocol and claim that it can address the vulnerability of DPID duplication and provide the authentication between the controller and the switch in OpenFlow-based SDN network. Owning to security analysis of Mynah authentication protocol is not clear, in this study we apply applied PI calculus in symbolic model to formalize Mynah authentication protocol and mechanized analyze it with mechanized tool ProVerif. We find that it does not provide mutual authentication between the controller and the switch. At the same time, Mynah authentication protocol can't prevent DPID duplication. Hence we propose an improved Mynah authentication protocol to address the vulnerabilities found by us. At the same time, the improved Mynah authentication protocol is modeled by applied PI calculus and mechanized analyzed with ProVerif. The results show that the improved Mynah authentication protocol can prevent DPID duplication and provide confidentiality of data and authentication between the switch and the controller. Finally, we develop and deploy the improved Mynah authentication protocol to open source controller ONOS and switch Open vSwitch to validate authentication and confidentiality. The experimental results show that it can provide confidentiality of data and authentication between the switch and the controller. In the near future, we will use the proof assistant Coq to prove the correctness of the improved Mynah authentication protocol implementation.

Acknowledgments

This study was supported in part by the Fundamental Research Funds for the Central Universities, South-Central University for Nationalities No.CZZ18003 and by the natural science foundation of Hubei Province under the grants No. 2014CFB249.

References

[1] M. Abadi and C. Fournet, "Mobile values, new names, and secure communication," in *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 104–115, Mar. 2001.

[2] M. Azrour, Y. Farhaoui, and M. Ouanan, "A new secure authentication and key exchange protocol for session initiation protocol using smart card," *International Journal of Network Security*, vol. 19, no. 6, pp. 870–879, 2017.

[3] K. Benton, L. J. Camp, and C. Small, "OpenFlow vulnerability assessment," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 151–152, Aug. 2013.

[4] P. Berde, M. Gerola, J. Hart, Y. Higuchi, and M. Kobayashi, "ONOS: Towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 1–6, Aug. 2014.

[5] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *Proceeding of the 14th IEEE Computer Security Foundations Workshop*, pp. 82–96, Cape Breton, June 2001.

[6] I. Bocic and T. Bultan, "Symbolic nodel extraction for web application verification," in *2017 IEEE/ACM 39th International Conference on Software Engineering*, pp. 724–734, Buenos Aires, Argentina, May 2017.

[7] V. Dangovas and F. Kuliesius, "SDN-driven authentication and access control system," in *The International Conference on Digital Information, Networking, and Wireless Communications*, pp. 20–23, Czech, June 2014.

[8] D. Erickson, "The beacon OpenFlow controller," in *Proceedings of the second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pp. 13–18, Hong Kong, China, Aug. 2013.

[9] N. Gude, T. Koponen, J. Pettit, B. Pfaff, and N. Mckeown, "NOX: Towards an operating system for networks," *ACM Sigcomm Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[10] F. Hu, Q. Hao, and K. Bao, "Survey on software-defined network and OpenFlow: From concept to implementation," *Communications Surveys & Tutorials IEEE*, vol. 16, no. 4, pp. 2181–2206, 2014.

[11] S. Jain, A. Kumar, S. Mandal, J. Ong, and L. Poutievski, "B4: Experience with a globally-deployed software defined WAN," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, pp. 3–14, Aug. 2013.

[12] J. W. Kang, S. H. Park, and J. You, "Mynah: Enabling lightweight data plane authentication for SDN controllers," in *International Conference on Computer Communication & Networks*, pp. 1–6, Las Vegas, USA, Aug. 2015.

[13] R. Kloti, V. Kotronis, and P. Smith, "OpenFlow: A security analysis," in *IEEE International Conference on Network Protocols*, pp. 1–6, Goettingen, German, Oct. 2014.

[14] D. Kreutz, F. M. V. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pp. 55–60, Aug. 2013.

[15] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, and S. Azodolmolky,

"Software-defined networking: A comprehensive survey," in *Proceedings of the IEEE*, vol. 103, no. 1, pp. 10–13, 2014.

[16] J. Liu, Y. X. Lai, Z. P. Diao, and Y. N. Chen, "A trusted access method in software-defined network," *Simulation Modelling Practice and Theory*, vol. 74, pp. 28–45, 2017.

[17] D. M. F. Mattos and O. C. M. B. Duarte, "AuthFlow: Authentication and access control mechanism for software defined networking," *Annals of Telecommunications*, vol. 71, no. 11-12, pp. 607–615, 2016.

[18] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a model-driven SDN controller architecture," *World of Wireless, Mobile & Multimedia Networks*, pp. 1–6, 2014.

[19] D. Mery and M. Poppleton, "Towards an integrated formal method for verification of liveness properties in distributed systems: With application to population protocols," *Software & Systems Modeling*, vol. 16, no. 4, pp. 1083–1115, 2017.

[20] R. Milner, "Communicating and mobile systems: The π-calculus," *Cambridge: Cambridge University Press*, pp. 1–5, 1999.

[21] A. A. Mohammed, M. Gharbaoui, B. Martini, F. Paganelli, and P. Castoldi, "SDN controller for network-aware adaptive orchestration in dynamic service chaining," in *IEEE NetSoft Conference and Workshops (NetSoft'16)*, July 2016.

[22] S. Natarajan, A. Ramaiah, and M. Mathen, "A software defined cloud-gateway automation system using OpenFlow," in *IEEE 2nd International Conference on Cloud Networking (CloudNet'13)*, pp. 219–226, San Francisco, Nov. 2013.

[23] P. Porras, S. Shin, V. Yegneswaran, M. Fong, and M. Tyson, "A security enforcement kernel for OpenFlow networks," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networks*, pp. 121–126, Helsinki, Finland, Aug. 2012.

[24] L. Schiff, S. Schmid, and P. Kuznetsov, "In-band synchronization for distributed SDN control planes," *ACM SIGCOMM Computer Communication*, vol. 46, no. 1, pp. 37–43, 2016.

[25] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2016.

[26] S. Shin, P. Porras, V. Yegneswaran, M. Fong, and G. Gu, "FRESCO: Modular composable security services for software defined networks," in *Proceedings of Network & Distributed Security Symposium*, 2013.

[27] R. Smeliansky, "SDN for network security," in *1st International Conference on Science and Technology*, Moscow, Russia, Oct. 2014.

[28] M. M. Wang, J. W. Liu, J. Chen andJ. Mao, and K. F. Mao, "Software defined networking: Security model, threats and mechanism(in chinese)," *Journal of Software*, vol. 27, no. 4, pp. 969–992, 2016.

[29] D. Yu, A. W. Moore, C. Hall, and R. Anderson, "Authentication for resilience: The case of SDN," *Lecture Notes in Computer Science*, vol. 8263, pp. 39–44, 2013.

[30] Q. Y. Zuo, M. Chen, G. S. Zhao, C. Y. Xin, and G. M. Zhuang, "Research on OpenFlow-based SDN technologies(in chinese)," *Journal of Software*, vol. 24, no. 5, pp. 1078–1097, 2013.

Biography

Lili Yao was born in 1993 in China. Now she is a postgraduate at School of Computer Science, South-Center for Nationalities, China. Her current research interests include protocol security and data storage security.

Jiabing Liu was born in 1993 and is now a postgraduate at the school of computer, South-Center University for Nationalities, China. His current research interest is the Formal analysis of security protocol.

Dejun Wang was born in 1974 and received his Ph.D. in information security at Wuhan University in China. Currently, he is an associate professor in the school of computer, South-Center University for Nationalities, China. He has authored/coauthored over 20 papers in international/national journals and conferences. His current research interests include security protocols and formal methods.

Jing Li was born in 1989 and graduate from the South-Center University for Nationalities, China. His main research direction includes the analysis of security protocols and formal methods.

Bo Meng was born in 1974 in China. He received his M.S. degree in computer science and technology in 2000 and his Ph.D. degree in traffic information engineering and control from Wuhan University of Technology at Wuhan, China in 2003. From 2004 to 2006, he worked at Wuhan University as a postdoctoral researcher in information security. From 2014 to 2015, he worked at University of South Carolina as a Visiting Scholar. Currently, he is a full Professor at the school of computer science, South-Center University for Nationalities, China. He has authored/coauthored over 50 papers in International/National journals and conferences. In addition, he has also published two books "Automatic generation and verification of security protocol implementations" and "secure remote voting protocol" in the science press in China. His current research interests include security protocols and formal methods.