Mathematical Biosciences and Engineering

MBE 16(5): 4359–4381.
DOI: 10.3934/mbe.2019217
Received: 28 February 2019
Accepted: 08 May 2019
Published: 17 May 2019

http://www.aimspress.com/journal/MBE

Research article

A novel software-defined network packet security tunnel forwarding mechanism

Zhibin Zuo, Rongyu He, Xianwei Zhu and Chaowen Chang*

Zhengzhou Institute of Information Science and Technology, Zhengzhou, 450001, China

*** Correspondence:** Email: changchaowen5@163.com.

Abstract: The OpenFlow protocol match field capacity is fixed and limited, and packet forwarding in software-defined network lacks valid authentication of data source, integrity verification, and confidentiality protection mechanism. OpenFlow only supports the MPLS label tunnel establishment, and therefore cannot establish a secure tunnel flexibly. In order to solve these problems, we propose P4Sec, a novel software-defined network packet security tunnel forwarding mechanism. As P4 allows the data plane to be reprogrammed to realize the characteristics of packet forwarding, we build a software-defined network security tunnel to prevent data malicious tampering, stealing, forgery and other malicious network behavior, implementing packet routing and forwarding based on gateway identity. Finally, we construct a P4Sec prototype system based on the software switch BMv2, verify the effectiveness of the mechanism through experimental analysis, and evaluate the overhead of the mechanism. The results demonstrate that P4Sec security mechanism ensure the authenticity, integrity, and confidentiality of forwarded data, and realize the secure forwarding requirements of data packets in software-defined network.

Keywords: software-defined network; packet forwarding; security tunnel; P4; identity-based signature

1. Introduction

Though traditional IP networks are widely used, their network structure is complex and difficult to manage. In a traditional IP network, the control plane determines how network traffic is handled, while the data plane forwards traffic based on decisions made by the control plane. Because these planes are bundled inside the network device, network flexibility is reduced, hindering the

innovation and evolution of network infrastructure. A software-defined network (SDN) is a new type of computer network architecture which separates the control and management logic of the network from the network equipment, as shown in Figure 1. This architecture allows us to manage the network service from a higher level of abstraction, providing great flexibility and scalability for the deployment and configuration of the network, and thus greatly improving the management efficiency of the network [1–4].

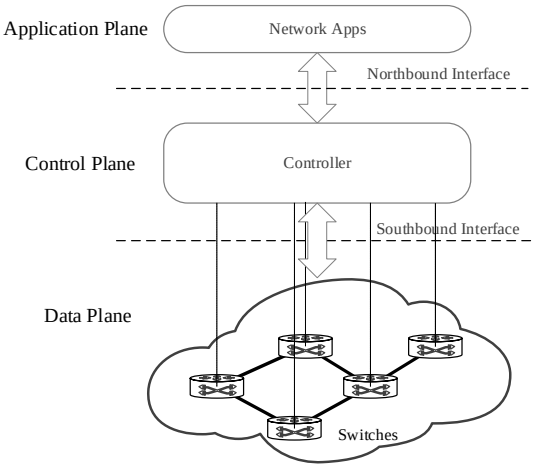

Figure 1. SDN Architecture.

Although SDN has many advantages not shared by traditional networks, such as the separation of control and forwarding, centralized network management, and network programmability, SDN still has many security risks and other problems [5–7]. SDN networks do not have an effective data source authentication mechanism for data packet forwarding, and attackers can therefore use spoofed source IP addresses to initiate denial of service attacks such as SYN Flood Attacks to controllers and forwarding devices, resulting in network paralysis. Furthermore, data packets in SDN networks lack data confidentiality and integrity protection, and are therefore susceptible to being stolen, deleted, forged, and so on [8,9]. The OpenFlow protocol [10], established by the ONF Standardization Organization, is the SDN southbound interface communication specification that specifies the basic components and functional requirements of an OpenFlow switch as an SDN data plane forwarding device, but its match fields are fixed, limited in number, and do not support dynamic increase match fields. Although the match field has been increased from 12 to 41 [11], it requires a large cost and a long period of time to support the new match fields. OpenFlow currently only supports the establishment of MPLS labels, and does not natively support the establishment of security tunnels. With the emergence of new protocols in the network, the OpenFlow will become increasingly bloated, and the addition of new match fields will become increasingly difficult due to the limited expansion capabilities of SDN networks based on OpenFlow technology.

P4 is a high-level language for programming protocol-independent packet processors [12]. Unlike OpenFlow, P4 supports a fully programmable parser, which allows us to define and parse new protocol headers. Because the P4 can programmatically define the switch processing logic, updating

the protocol version of the switch is performed by simply updating the switch processing logic through the controller programming. This enables the data forwarding plane to also have programmable capabilities, allowing the parsing and forwarding processes of packets to be programmatically controlled such that the software can truly define a network and its network devices. The P4 forwarding model is shown in Figure 2.

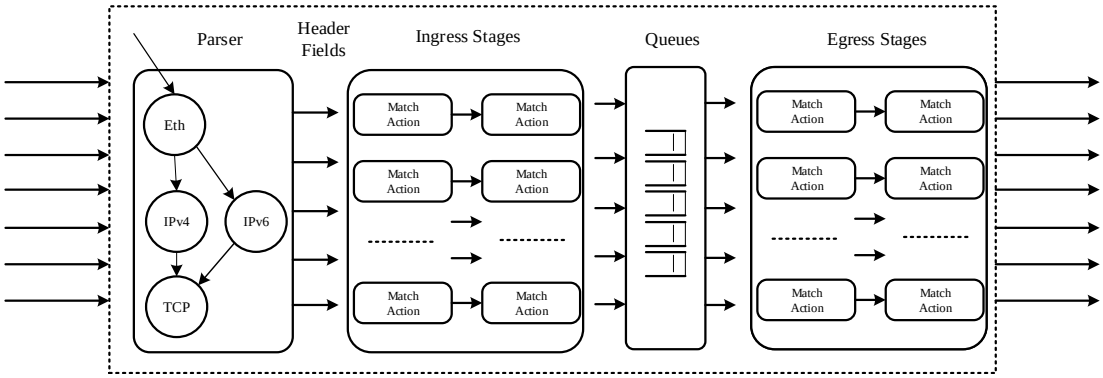

Figure 2. P4 forwarding model.

Combining the characteristics of SDN network with the advantages of P4, we propose P4Sec, a software-defined network packet security tunnel forwarding mechanism based on P4. Utilizing P4 to allow the data plane to be reprogrammed to realize the characteristics of packet forwarding, we define a new protocol header P4Sec, and build a P4-based SDN security tunnel to prevent malicious data tampering, stealing, forgery, and other malicious network behavior. Additionally, we implement packet routing and forwarding based on the gateway's identity. Using the routing and forwarding characteristics of P4Sec that are based on gateway identity, the device identity of the gateway is combined with the Diffie-Hellman (DH) key exchange algorithm to realize security negotiation for tunnel session key. Finally, we construct a P4Sec prototype system based on a BMv2 software switch, verify the effectiveness of the mechanism through experimental analysis, and evaluate the overhead of the mechanism.

The structure of this article is as follows. Firstly, some related research work regarding this topic is introduced in Section 2. The system design, working principle, and solution of the SDN packet security tunnel forwarding are then introduced in Section 3. In section 4, the specific implementation of P4Sec in SDN is described. Section 5 presents verification of the effectiveness of the mechanism through experimental analysis and evaluate the overhead of the mechanism. Section 6 then summarizes the work of this paper and states our future work.

2. Related work

The separation of the control plane from the data plane in SDN reduces the difficulty of attacking the packet forwarding of the data plane, which enables malicious activity such as copying packets, hijacking traffic, malicious tampering, and the insertion of false packets into the normal network flow. However, because the switch in the data layer only provides simplified data forwarding functions, the data forwarding verification scheme cannot simply be deployed in a traditional IP network.

SPHINX [13] provides data flow forwarding verification functionality to SDN, and obtains statistical information of the data layer through the control plane to detect the data packet loss and attacks such as traffic hijacking. However, due to the lack of an effective packet verification function, it cannot guarantee that the received statistical information is correct, nor can it guarantee consistency between the packets arriving at the destination and those sent by the source. Therefore, SPHINX cannot detect intelligent data plane attacks such as malicious discards and packet replay attacks. SDNsec [14] enables each switch to embed cryptographic markings in its forwarding packet header, verifying the normal packet forwarding by checking the cryptographic markings of the packet header. However, SDNsec requires the controller to generate a shared key with each forwarding device, and lacks confidentiality protection for data packets. Shin et al. [15] proposed a CloudWatcher method for SDN traffic monitoring in a cloud environment, automatically importing network traffic into a corresponding security device to implement the necessary network packet inspection. However, redirecting traffic to the security device for data source verification requires a complicated implementation, and the location of the security device must be comprehensively considered for the level of cooperation between different security devices and the granularity of SDN traffic control. Because this CloudWatcher method only supports match fields using common protocols in the first four layers of the network, its control granularity is limited [16].

3. System design

3.1. Basic principles

The SDN network packet security tunnel forwarding mechanism implements four basic functions to realize the secure tunnel forwarding of data packets. The four functions include key negotiation, authentication, encryption and decryption, and routing and forwarding based on gateway identity.

The key negotiation function refers to the generation of a session key by the gateway devices of the sender and receiver according to the improved DH key exchange algorithm during the tunnel negotiation initialization process. The improved key exchange algorithm combines the DH key exchange algorithm with the gateway device identity, and uses the identity-based signature algorithm to authenticate the negotiation data, realizing key agreement between gateway devices to prevent man-in-the-middle attacks. The generated session key is then used for secure data packet forwarding during the tunnel data interaction process.

The authentication function refers to applying an identity-based signature algorithm to the data packet during the tunnel data interaction process. This enables verification of the source and the integrity of the data packet as the gateway device forwards the data, which ensures the authenticity and integrity of the data.

The encryption and decryption function refers to encrypting and decrypting data packets during the tunnel data interaction process by using the session key generated during the tunnel negotiation initialization process, ensuring the confidentiality of data packets.

Routing and forwarding based on gateway identity refers to using the gateway identity as the match field of a data packet and forwarding the data packet based on the matched gateway identity, utilizing the combination of the gateway identity and the P4Sec header. This enables the network data packet routing and forwarding mechanism based on gateway device identity.

These four functions of P4Sec complement each other, and jointly construct the SDN data packet security tunnel forwarding mechanism.

3.2. Architecture

P4Sec security tunnel forwarding mainly consists of two processes and two main modules. The two processes include the tunnel negotiation initialization process and the tunnel data interaction process. The two modules are the matching and forwarding module and the security processing module. The P4Sec overall architecture is shown in Figure 3.

For the tunnel negotiation initialization process, the tunnel gateway devices generate a session key by using the improved DH key exchange algorithm through tunnel negotiation request/reply message. Using this session key shared between the sender gateway and the receiver gateway provides security support for the tunnel data interaction process.

For the tunnel data interaction process in the SDN network, the controller is the network manager and is responsible for the routing management of the entire network. The controller formulates the routing strategy according to the DGW_ID field of the gateway device identity in the P4Sec header, and sends the strategy to all forwarding devices in the network, instructing all forwarding devices to match and forward packets according to the DGW_ID field. The tunnel gateway devices then perform security processing on data packets for source authentication, integrity verification, and confidentiality protection when forwarding data.

The matching and forwarding module is located inside all network forwarding devices, including the tunnel gateways. The module encapsulates or decapsulates the P4Sec header for the data packet, initializes the P4Sec protocol header, and either forwards the packet to the internal security processing module for security processing or forwards the packet to the adjacent forwarding device according to the tunnel destination gateway identity for further processing. The module matches and forwards the data flow according to the received flow rule sent by the controller, thereby implementing control forwarding based on the tunnel gateway identity.

The security processing module is located inside tunnel gateway devices. It provides security services for data packets, encrypts and decrypts data packets to implement data confidentiality protection, verifies signatures for packet authenticity and integrity, and ensures data forwarding security.

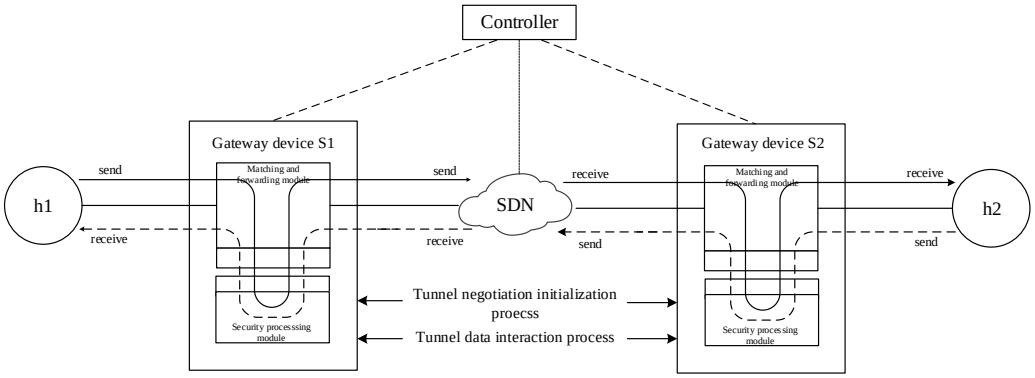

Figure 3. P4Sec overall architecture.

3.3. P4Sec header

The P4Sec protocol header consists of Proto_ID, Sessiontype, SeqNo, DGW_ID, Verification, and IV, as shown in Figure 4. These elements are defined as follows.

Proto_ID is a next header field that indicates the type of the next protocol header in the protocol header parsing process.

Sessiontype is a session type message field. P4Sec defines three kinds of message session types. Tunnel negotiation request messages have a REQ value, and are used to request tunnel establishment during the tunnel negotiation initialization process. Tunnel negotiation response messages have an ACK value, and are used for tunnel gateway response request messages during the tunnel negotiation initialization process. Tunnel communication messages have a NORMAL value, and are used for normal communication during the tunnel data interaction process.

SeqNo is a serial number field, and the sender gateway device maintains a separate SeqNo for each data packet. For each forwarded packet of a given tunnel, the tunnel gateway device increments SeqNo such that SeqNo is used as a unique identifier for each packet in the data flow. This can be used to prevent packet anti-replay attacks.

DGW_ID is the destination gateway device identity field, and stores the unique identifier of the destination gateway device identity. The switch matches this field according to the flow rule sent by the controller, using the gateway identity to forward data packets to implement route.

Verification is a field containing authentication information. This authentication information is generated by the gateway device using its own private key, and is used to sign the tunnel negotiation request packet, the tunnel negotiation response packet, and the tunnel communication message during the tunnel negotiation initialization process and the tunnel data interaction process. This signing enables verification of the authenticity and integrity of data packets.

IV is a field containing an initialization vector. When a data packet is encrypted, the IV is generated as an unpredictable piece of data that is 16 bytes in length, which is the block size in AES, and this data is unique to the message/key combination.

Temp is a data field used for tunnel negotiation. It is used to generate session key, and is used for the tunnel negotiation request message and the tunnel negotiation response message during tunnel negotiation initialization process.

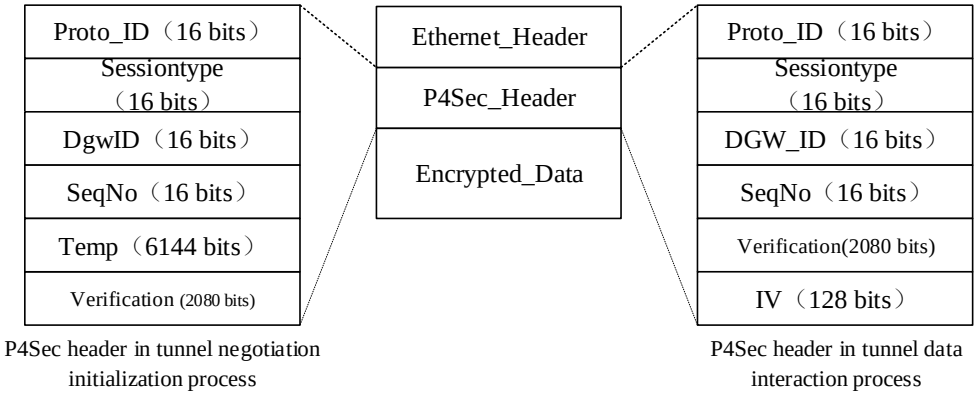

Figure 4. The structure of P4Sec header.

3.4. Tunnel data interaction process

The tunnel data interaction process is a process in which the terminal host relies on the SDN gateway device to conduct the matching and forwarding of packets and to secure processing. This process, shown in Figure 5, is outlined as a sequence of seven steps, as follows. In the process, the gateway devices S1 and S2 obtain the device identity through their own configuration files and generate the public key of device identity. They obtain the private key of device identity from the trusted Private Key Generator (PKG) through secure channels. P4Sec uses the Identity-Based Signature (IBS) algorithm [17] to generate corresponding public and private keys for the gateway device.

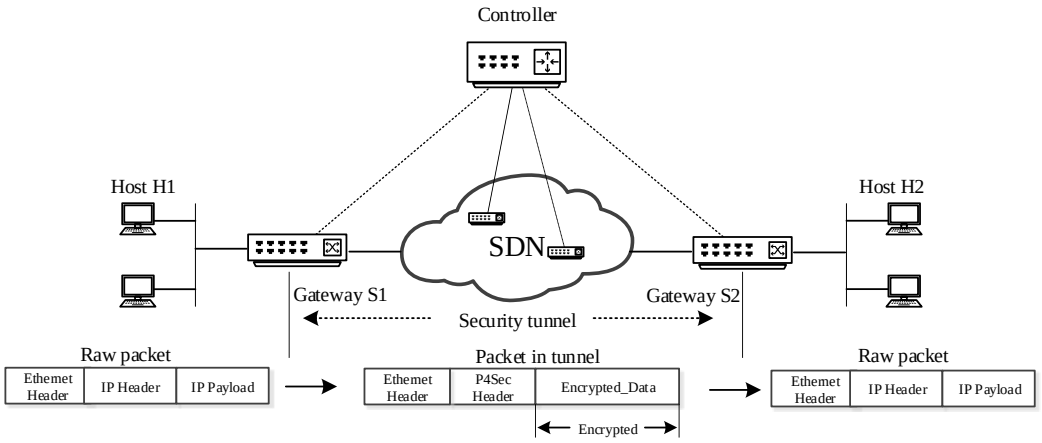

Figure 5. Tunnel data interaction process based on P4Sec.

(1) After Gateway S1 receives the original data packet sent by terminal host H1, the gateway uses session key K, generated during the tunnel negotiation initialization process, to symmetrically encrypt the IP datagram. The generated ciphertext is then stored in the Encrypted_Data field. The encryption algorithm is detailed in Section 4.3.2.

(2) Gateway S1 performs a hash operation on the ciphertext Encrypted_Data to generate a hash value, uses its own private key to sign the hash value, and the generated signature information is then stored in the Verification field. The signature algorithm is detailed in Section 4.3.1.

(3) Gateway S1 generates the P4Sec header and puts the header in the data packet. The values of each field in the P4Sec header are as follows: the Type field contains NORMAL, indicating that the P4Sec header is a tunnel communication message; the DGW_ID field contains the device identity of Gateway S2, indicating that Gateway S2 is to receive the message; the SeqNo field contains the serial number of the packet in the data flow, which is used to prevent replay attacks; the Verification field contains the signature information, which is used to verify the authenticity and integrity of the packet's Encrypted_Data field, preventing man-in-the-middle attacks.

(4) Gateway S1 uses the DGW_ID field in the P4Sec header as a match field, and forwards the data packet based on the matching rule sent by the controller to implement accurate matching and forwarding of the data flow based on the tunnel gateway identity.

(5) Other forwarding devices in network match and identify the DGW_ID field in the P4Sec header, and then forward the data packet to the destination Gateway S2 based on the routing and

matching rules sent by the controller.

(6) After Gateway S2 receives the data packet, the gateway resolves the P4Sec header and obtains the Verification field, which contains the signature information of the Encrypted_Data field. The security processing module of Gateway S2 then verifies the signature information to ensure the authenticity and integrity of the data packet. If the verification fails, the packet will be discarded. Meanwhile, the session key K generated during the tunnel negotiation initialization process is used to decrypt the Encrypted_Data field, the decrypted data is then stored in the original field, and the packet is discarded if the decryption fails.

(7) Gateway S2 deletes the P4Sec header in the data packet, submits the packet to the matching and forwarding module, and then matches and forwards the data packet to destination host H2.

4. Implementation

This chapter provides a detailed introduction for the implementation of the tunnel negotiation initialization process, the matching and forwarding module based on P4Sec, and the security processing module based on P4Sec.

4.1. Tunnel negotiation initialization

The tunnel negotiation initialization process is shown in Figure 6. In this implementation, the 6144-bit MODP Group of RFC 3526 recommendation [18] and SHA-256 are used to generate a 256-bit AES session key K, which is used in tunnel data interaction process. The 6144-bit MODP Group id is 17, the prime P is $2^{6144} - 2^{6080} - 1+2^{64}\{[2^{6014}\ \pi\ ] + 929484\}$, the generator a is 2, and [x] represents a largest integer less than or equal to x. The Hash function is SHA-256, which generates a hash value of 256 bits in length. The specific implementation of the signature algorithm is detailed in Section 4.3.1.

(1) Gateway S1 first queries the configuration file to get the device identity of Gateway S2, and then provides its own device identity and that of Gateway S2 to the controller.

(2) The controller uses the centralized control and the whole network view to determine the route and forwarding rule according to the identity of Gateway S2 at the receiving end, and then delivers to the forwarding devices of the entire network. The forwarding devices, including Gateway S1, match and forward the data to receiving Gateway S2 according to the flow rule sent by the controller.

(3) Gateway S1 selects a secret random number X_A (540 bit or more), calculates $a^{X_A}\ mod\ P = Y_A$, and then stores the calculation result of Y_A in the Temp field of the tunnel negotiation request message.

(4) After Gateway S1 calculates the hash value of Y_A, the gateway uses the private key of its own device identity to sign the hash value, generates signature information (R_A, S_A), and then stores that information in the Verification field (260 bytes).

(5) Gateway S1 sends a tunnel negotiation request message to Gateway S2. The values for the fields of P4Sec header in message are as follows: the Type field is REQ, indicating that the packet is a tunnel negotiation request packet; the DGW_ID field contains the device identifier of Gateway S2, indicating that Gateway S2 is to receive the request message; the SeqNo field contains the packet sequence number that is use to prevent replay attacks; the Temp field

contains the Y_A value sent by Gateway S1 to Gateway S2 to generate the session key; and the Verification field contains the signature information.

(6) After Gateway S2 receives the tunnel negotiation request message sent by Gateway S1, Gateway S2 queries the tunnel configuration file to obtain the established tunnel name, the gateway S1 device identity, and related parameters *a* and P.

(7) Gateway S2 obtains the Y_A value from the Temp field, obtains its signature information (R_A, S_A) from the Verification field, and uses the device identity public key of Gateway S1 to calculate the signature information of Y_A, providing authenticity and integrity verification to prevent man-in-the-middle attacks.

(8) Gateway S2 selects a secret random number X_B (540 bit or more), calculates $a^{X_B} \ mod \ P = Y_B$, and stores the calculation result for Y_B in the Temp field of the tunnel negotiation response message.

(9) After Gateway S2 calculates the hash value of Y_B, Gateway S2 uses the private key of its own device identity to sign the hash value, generates the signature information (R_B, S_B), and stores that information in the Verification field (260 bytes).

(10) Gateway S2 sends a tunnel negotiation response message to Gateway S1. The values for the fields of P4Sec header in message are as follows: the Type field is ACK, indicating that the packet is a tunnel negotiation response packet; the DGW_ID field contains the device identity of Gateway S1, indicating that Gateway S1 is to receive the response message; the SeqNo field contains the packet sequence number that is used to prevent replay attacks; the Temp field contains the Y_B value sent by Gateway S2 to Gateway S1 to generate the session key; and the Verification field contains the signature information.

(11) After Gateway S1 receives the tunnel negotiation response message sent by Gateway S2, Gateway S1 obtains the Y_B value from the Temp field, obtains the signature information (R_B, S_B) from the Verification field, and uses the public key of Gateway S2 to verify the Y_B value to achieve authenticity and integrity verification and prevent man-in-the-middle attacks.

(12) Gateway S1 calculates $Y_B^{X_A} \ mod \ P = K$, Gateway S2 calculates $Y_A^{X_B} \ mod \ P = K$, and Gateway S1 and Gateway S2 use their respective calculation results for K as the session key.

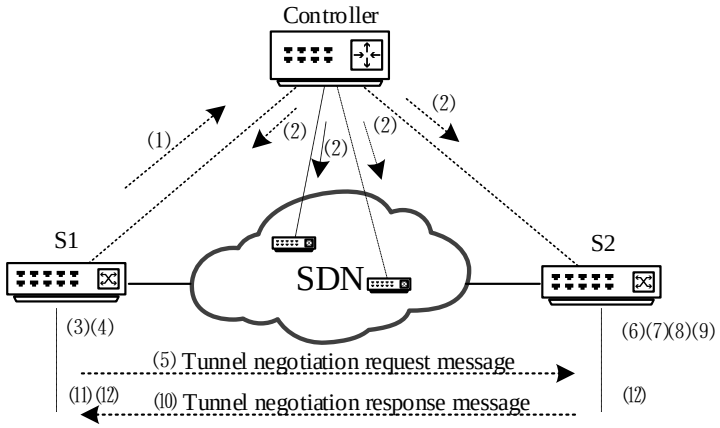

Figure 6. Tunnel negotiation initialization process.

4.2. Matching and forwarding module

Matching and forwarding module is implemented by developing a P4 application on forwarding devices. There are currently two parallel language specifications, including P4-14 and P4-16.The development of matching and forwarding module uses P4-14. Since the target for our P4 programs is BMv2, we uses the p4c-bmv2 as the compiler.

Packets arriving at the P4 forwarding device are first processed by the packet parser. The parser finds and extracts the previously defined and supported headers, including ethernet_header, p4sec_header, ipv4_header, and so on. The extracted headers are then passed to the match-action table for processing. The control program determines the jump relationship of the data packet in different match-action tables, which are detailed as follows.

(1) Header. The header is a list of ordered fields, each with a corresponding name and length. Each header has a corresponding instance of the header to store the data specific to each field. As shown in Table 1, the P4Sec security tunnel forwarding mechanism defines the corresponding protocol header.

Table 1. Structure of the P4Sec header.

Structure of the P4Sec header

header_type p4sec_t {
fields {
proto_id : 16;
sessiontype : 16;
dgw_id : 16;
seqno : 16;
verification : 2080;
iv : 128;
}
}

(2) Parser. After defining the headers, we also need to define the relationship between the headers and the corresponding relationships of packets parsing. The parser is used to parse the byte flow into corresponding protocol headers for subsequent flow table matching and action execution. The P4Sec header is added between the Ethernet header and the IPv4 header, and the P4Sec header type is 0x1212. When the Ethernet etherType is 0x1212, the parser should jump to the P4Sec header for subsequent parsing as shown in Figure 7. When the proto_id of P4Sec is 0x0800, the parser should jump to the IPv4 header for subsequent parsing.

(3) Table. The format of the table is Match+Action, meaning it contains match fields and their corresponding execution actions. When the match field defined in the table matches a field in the packet successfully, a corresponding action is performed. For the P4Sec security tunnel forwarding mechanism, the flow_forward table and the data_to_verify table are mainly involved. The flow_forward table and the data_to_verify table are designed and implemented, and described as follows.

flow_forward table: According to the flow rule delivered by the control plane, the flow_forward table uses the DGW_ID field in the P4Sec header as the match field to match and forward the

received data flow. In the Table 2, reads is the value of the match field, and the matching type is exact match; actions is the action performed after the matching succeeds, where in this case, the custom forwarding behavior my_forward sets the value of standard_metadata.egress_spec to determine the egress port of data flow. The custom forwarding behavior my_forward achieves the routing and forwarding of the data flow according to the gateway device identity. _drop is the default action of unsuccessful matching, and size is the maximum entry size of the matching table.

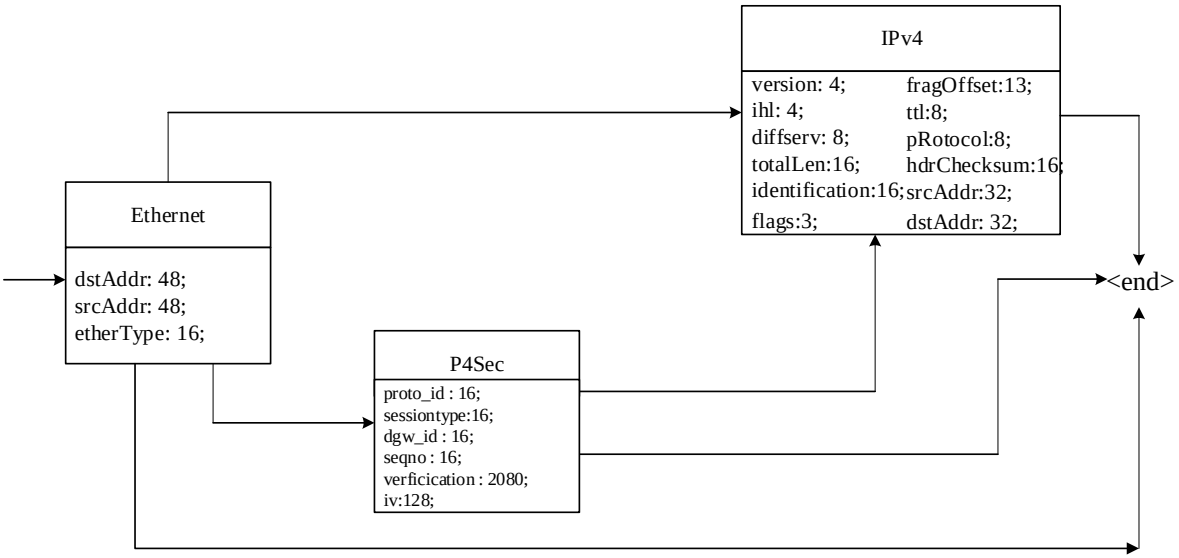

Figure 7. Progression of header parsing.

Table 2. Structure of the flow_forward table.

Structure of the flow_forward table
table flow_forward {
reads {
p4sec.dgw_id: exact;
}
actions = {
my_forward;
_drop;
}
size = 1024;
}

Data_to_verify table: This table preprocesses the data packets that need to be forwarded to the security processing module for processing, meaning the do_pkt_to_verify action primitive is used to send the pending data packets to the port connected to the security processing module. The security processing module performs encryption and decryption as well as authenticity and integrity verification of the forwarded data packets, and its specific implementation is detailed in Section 4.3.

Table 3. Structure of the data_to_verify table.

Structure of the data_to_verify table
table data_to_verify {
reads {
meta.if_index: exact;
}
actions = {
do_pkt_to_verify;
_drop;
}
}

(4) Control Program. The control program determines the order in which the data packets are processed, meaning the jump relationships of data packets in different matching tables. After match tables is defined and implemented, a control program is also needed to determine flow control between the different tables. This process of the control program is shown in Figure 8.

In the ingress process, data packets are checked for whether they have been processed by the security processing module (Whether there is a specific header that identifies that the packet has been processed by security module). If a data packet has been processed, the packet is sent to the flow_forward table, matching the DGW_ID field for packet forwarding according to the flow rule sent by the controller; if it has not been processed, the packet is sent to the data_to_verify table, as it is ready to be sent to the security processing module for security processing.

In the egress process, data packets are checked for packet source and the corresponding headers are added or deleted according to the data source. If the packet has been processed by the security processing module and is from network, and the P4Sec header and other redundant header are deleted and ready to be forwarded according to the output port. If the packet has been processed by the security processing module and is from terminal host, the P4Sec header will be valid. If the packet is not processed by the security processing module, a specific header is added to identify the packet which is forwarded to the secure processing module port.

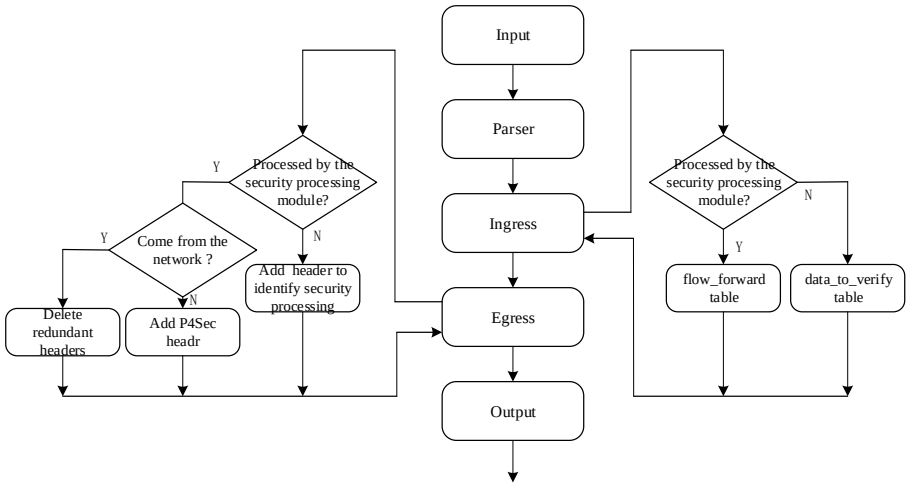

Figure 8. Process of the control program.

4.3. Security processing module

The security processing module includes authenticity and integrity verification as well as encryption and decryption processing.

4.3.1. Authenticity and integrity verification

In this solution, an IBS algorithm is used to provide data authenticity and integrity authentication for the tunnel negotiation initiation process and the tunnel data interaction process. IBS is a public key cryptography. It does not use a digital certificate, and instead directly uses the unique identity of a gateway device as the public key. The private key is generated and distributed to users by the trusted PKG. Because the public key is the public identity information of the gateway device itself, the trust authority does not need to create and distribute the public key certificate, which solves the problems of certificate generation, issuance, backup, and revocation, reducing the cost and complexity of system establishment and maintenance while greatly saving storage space and network bandwidth.

(1) Setup. Taking the security parameter *k* as input, the A-type curve parameter provided by the open source pairing-based cryptography library (pbc library) is used as the system parameter to generate the system master key *msk*. This process is completed by PKG, and secretly saves the system master key *msk*. H_2 uses SHA1, and outputs 160 bits.

Algorithm 1: Setup Algorithm

Input: a secret k

Output: msk

1: BEGIN

2: generate group G_1、G_2 of order q;

3: generate a bilinear map $e: G_1 \times G_1 \to G_2$;

4: choose a generator $P \in G_1$;

5: randomly choose a secret $msk = s \in Z_q^*$;

6: choose one-way function: $H_1: \{0,1\}^* \to G_1^*$;

7: choose one-way function: $H_2: G_2 \to \{0,1\}^n$;

8: compute $P_{pub} = sP$;

9: $params = (G_1, G_2, e, P, P_{pub}, H_1, H_2)$;

10: return msk;

11: END

(2) Extract. This algorithm is used to generate the public and private key of the gateway of the sender. According to the sender gateway identity GW_ID, we can get the public key Q_{ID} of the sender gateway. Enter the system master key *msk* and the sender gateway identity GW_ID, we can get the private key d_{ID}.

Algorithm 2: Extract Algorithm

Input: msk, GW_ID

Output: the private key d_{ID}、the public key of Q_{ID}

1: BEGIN

2: compute the public key : $Q_{ID} = H_1(GW\_ID)$;

3: compute the private key : $d_{ID} = sQ_{ID}$;

4: return Q_{ID}, d_{ID};

5: END

(3) Sign. Use the signature algorithm on the message *m* to generate the signature information (R,S), which are stored together in the Verification field of the P4Sec header. The signature information R and S are each a 130-bit string, occupying 260 bytes total (2080 bits).

Algorithm 3: Signature Algorithm

Input: the message m, the private key d_{ID}

Output: the signature (R, S)

1: BEGIN

2: randomly choose $k \in Z_q^*$;

3: compute $R = kP = (x_R, y_R)$;

4: compute $S = (H_2(m)P + x_R d_{ID})k^{-1}$;

5: return (R, S);

6: END

(4) Verify. The Verification field, the sender gateway identity GW_ID, and the received message *m* are used as input, and the verification algorithm is applied to verify the correctness of the Verification field. If Value1=Value2 in the verify algorithm, the verify passes. Otherwise, the verify fails and the data message is discarded.

Algorithm 4: Verification Algorithm

Input: the signature (R,S), GW_ID, the message m

Output: VResult

1: BEGIN

2: compute $H_2(m)$;

3: compute $Value1 = e(R,S)$;

4: compute $Value2 = e(P,P)^{H_2(m)} e(P_{pub}, Q_{ID})^{X_R}$;

5: if ($Value1 = Value2$) VResult=1 , else VResult=0;

6: return VResult;

7: END

4.3.2. Data encryption and decryption processing

In this scheme, the Output Feedback (OFB) mode of the AES encryption algorithm [19] is used to provide data confidentiality protection for the tunnel data interaction process. In the OFB mode [20], the key stream is first generated by a block cipher generator, and then the key stream is XOR with the plaintext stream to obtain a ciphertext stream. Decryption is then performed using a block encryption generator to generate the key stream first, and then the key stream is XOR with the ciphertext stream to get plaintext. Due to the symmetry of the XOR operation, the process of encryption and decryption is identical. The OFB mode uses a block algorithm to implement stream algorithm, and the plaintext does not need to be aligned according to the size of the block. The plaintext is the same length as the ciphertext, which is convenient for the encryption of streaming data, and can be timely encrypted and for the transfer of data smaller than the block size.

The encryption algorithm is shown in Algorithm 5. The message Data is the data to be encrypted, and the K is the session key generated during the tunnel negotiation initialization process described in Section 4.1. The encrypted Ciphertext and initialization vector IV are the output. The OFB mode of AES requires the use of an Initialization Vector (IV). An IV is unique to the combination of a message and key, and must be known by both the encrypter and the decrypter.

Algorithm 5: Encryption Algorithm (AES OFB mode)

Input: the message Data, the key K

Output: the encrypted Data Ciphertext, the initialization vector IV

1: BEGIN

2: Initialize Data, K to byte type;

3: compute $Ciphertext = AES.new(K, AES.MODE\_OFB).encrypt(Data)$;

4: compute $IV = AES.new(K, AES.MODE\_OFB).iv$;

5: return $Ciphertext$, IV;

6: END

The decryption algorithm is as shown in Algorithm 6. The input is the Ciphertext to be decrypted, the initialization vector IV, and the session key K. Ciphertext is the Encrypted_Data field in the received packet, IV is the IV field of the P4Sec header, and the K is the session key generated during the negotiation initialization process. The output is the decrypted plaintext.

Algorithm 6: Decryption Algorithm

Input: the encrypted data *Ciphertext*, the initialization vector *IV*, the key *K*

Output: the decrypted data *Plaintext*

1: BEGIN

2: Initialize *Ciphertext* , *IV*, *K* to byte type;

3: compute $Plaintext = AES.new(K, AES.MODE\_OFB, iv = IV).decrypt(Ciphertext)$;

4: return *Plaintext*;

5: END

5. Experiments and analysis

5.1. Experimental environment

We implemented and deployed a simple P4Sec prototype using a virtual BMv2 switch and Mininet. The controller uses runtime_CLI and the northbound interface is the P4 Runtime. We added a matching and forwarding module and a security processing module for the P4Sec tunnel gateway device, signed the data using an identity-based signature algorithm, and encrypted and decrypted the data using the AES symmetric encryption algorithm. Figure 9 shows the evaluation environment we created to evaluate system effectiveness, safety features, time consumption.

The experimental environment was built on a host with an Intel Core i3-4170 CPU 3.70GHz and 16 GB of memory, which hosted BMv2 software switch and Mininet. S1 and S2 are access forwarding devices (i.e., tunnel gateway devices), S3 and S4 are central forwarding devices, and H1 and H2 are terminal hosts. Scapy was used to generate data packets on sender H1, and the Wireshark was used to analyze the device port packets.

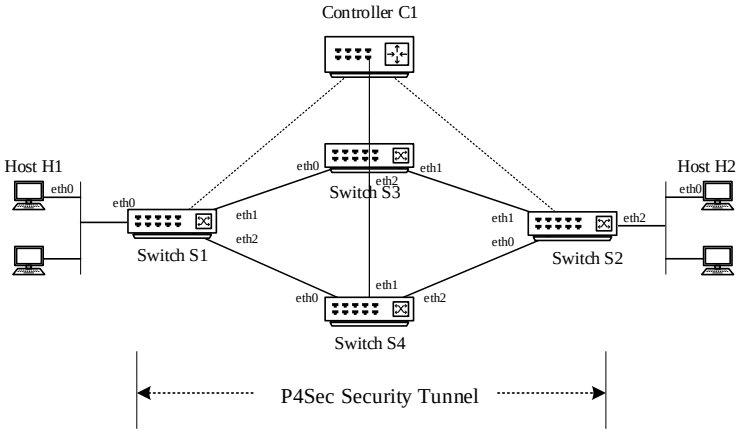

Figure 9. The experimental topology.

5.2. Matching and forwarding validity

The test showed that the forwarding device used the gateway identity as the match field, and matched the destination gateway device identity to control the routing and forwarding of data packets. The terminal host H1 sent packets to the H2 using Scapy, and the transmission rate was 20 packets

per second. The DGW_ID fields of S1, S2, S3, and S4 were set to 1001, 1002, 1003, and 1004, respectively. The controller sent the following flow rules: for S1, data packets with DGW_ID fields containing 1002 were forwarded to S3; for S3, data packets with DGW_ID fields containing 1002 were forwarded to S2. Monitor the port statuses of forwarding devices S1, S2, S3, and S4. The experiment was run for 10 s for each instance and was repeated 10 times, after which the 10 experimental data were averaged. The test results are shown in Figure 10.

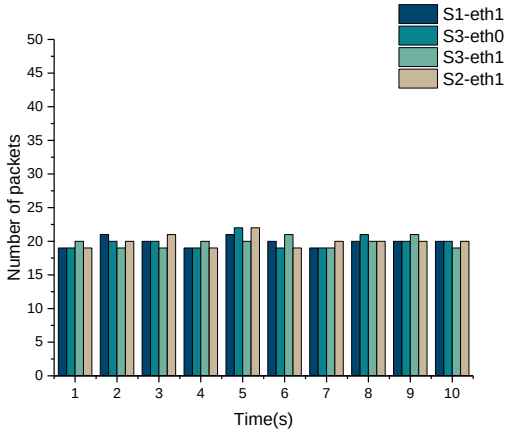

Figure 10. Packets status of S1-eth1, S2-eth1, S3-eth0, S3-eth1 ports.

As can be seen from Figure 10, the traffic values of the S1-eth1, S3-eth0, S3-eth1, and S2-eth1 ports are approximately the same, an average of 200 packets per port. After analysing packets on the S4, S3 and S2 ports, it was seen that the data packets received by S3 and S2 carried the P4Sec protocol header with DGW_ID fields containing 1002, and the traffic values of S4-eth0, S4-eh2, and S2-eth0 ports were zero. This resulted from the tunnel gateway S1 adds P4Sec headers with DGW_ID field of 1002 to the data packets, the forwarding device S1, S2 and S3 obtained the gateway identification as the match field and matched and forwarded the data flow accurately based on the tunnel gateway identification.

5.3. Security processing effectiveness

The terminal host H1 simultaneously sends data flow A, B, and C to H2, and each data flow is sent at a rate of 20 pps for a duration of 10 s. The tunnel gateway S1 adds valid P4Sec headers to the data flow A, B, and C, and the DGW_ID field of P4Sec header contains 1002. The forwarding policy was delivered to S1, S2 and S3: the data flow forwarding path for DGW_ID fields containing 1002 was S1->S3->S2. When the forwarding device S3 forwards the data flow B, the Verification field of the data flow B is tampered; when the forwarding device S3 forwards the data flow C, the Data field of the data flow C is tampered. Each experiment was repeated 10 times, after which the data of the 10 experiments were averaged. The test results are shown in Figure 11.

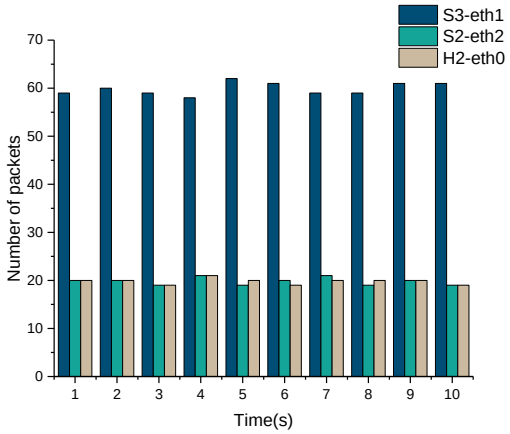

Figure 11. Packets status of S3-eth1, S2-eth2, H2-eth0 ports.

As can be seen from Figure 11, the number of S3-eth1 port packets was about three times as many as that of the S2-eth2 and H2-eth0 ports on average. After analysing packets on the S3, S2 and H2 ports through Wireshark, it was learned that packets sent by S3-eth1 port included data flow A, data flow B, and data flow C, for a total of 600 data packets; while packets sent through the S2-eth2 port were identical to those received by H1, an average of 200 packets, carrying data flow A and the DGW_ID field of 1002. This resulted from S2 refusing to forward error-signed packets and tampered packets after verifying the Verification fields in the P4Sec header in the process of forwarding data flow.

5.4. Delay analysis

5.4.1. Delay analysis in key exchange

The processing time of the improved DH key exchange algorithm was compared with the original DH key exchange algorithm used in the P4Sec tunnel negotiation initialization process. Figure 12 shows the time comparison for S1 to generate tunnel request messages. Figure 13 shows the time comparison for the generation of session keys by S2 after receiving a tunnel request message. Figure 14 shows the time comparison for the generation of session keys by S1 after receiving a tunnel answer message. In Figure 12, 13 and 14, DH represents the processing time of the original DH algorithm. P4Sec_DH represents the processing time of the improved DH algorithm.

In Figure 12, P4Sec_DH represents the processing time for generating tunnel negotiation request messages by S1, including the time for generating and signing the Temp field. Compared with the original DH key exchange algorithm, the processing of signing the Temp field was added. We generated 200 different tunnel negotiation request messages respectively, and obtained the processing time for the two DH algorithms to generate tunnel negotiation request messages. After calculation, the average processing time of the original DH algorithm was 65.8 ms, while the average time of the improved DH algorithm was 86 ms. The processing time was increased by 20.2 ms, which is 30.1% more than the original DH algorithm.

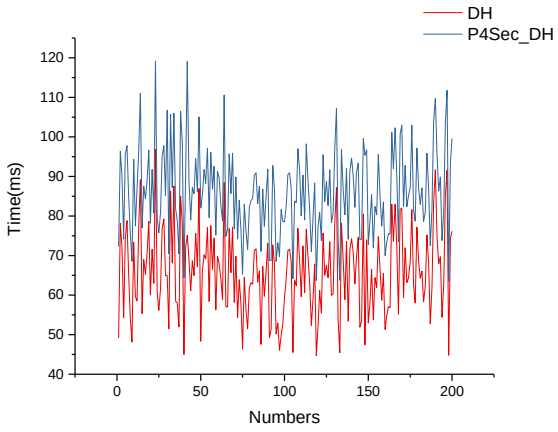

Figure 12. Tunnel request packet generation time for S1.

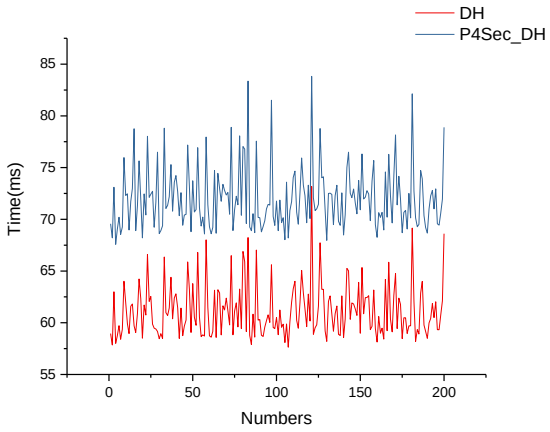

Figure 13. Session key generation time for S2.

In Figure 13, P4Sec_DH represents the processing time for generating session keys after receiving a tunnel request message by S2, including the time for verifying the Verification field and generating the session key. Compared with the original DH key exchange algorithm, the processing of verifying the Verification field in the tunnel request message was added. We received 200 different tunnel negotiation request messages by S2 respectively, and obtained the processing time for the two DH algorithms to generate session keys. After calculation, the average processing time of the original DH algorithm was 61.1 ms, while the average time of the improved DH algorithm was 72.1 ms. The processing time was increased by 11 ms, which is 18% more than the original DH algorithm.

In Figure 14, P4Sec_DH represents the processing time for generating session keys after receiving a tunnel response message by S1, including the time for verifying the Verification field and generating the session key. Compared with the original DH key exchange algorithm, the processing of verifying the Verification field of the tunnel response message was added. After calculation, the average processing time of the original DH algorithm was 61.9 ms, while the average time of the improved DH algorithm was 72.8 ms. The processing time was increased by 10.9 ms, which is 17.6%

more than the original DH algorithm.

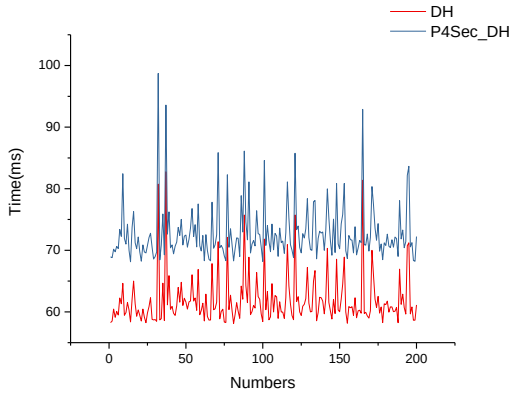

Figure 14. Session key generation time for S1.

Experiments show that, compared with the original DH protocol, the improved DH protocol has some difference in processing time, but the impact on the P4Sec tunnel negotiation initialization process is within an acceptable range.

5.4.2. Delay analysis in security processing

To analyze the impact of tunnel gateway security processing on data forwarding, we verified 300 data packets and obtained the total processing time, data signature and verification time, and encryption and decryption time during the tunnel gateway forwarding. The total processing time of the tunnel gateway S1 at the sending end includes matching and forwarding time, signature time, and data encryption time. The total processing time of the tunnel gateway S2 at the receiving end includes matching and forwarding time, verification time, and data decryption time. The comparison of the security processing time of the sender S1 is shown in Figure 15. The comparison of the security processing time of the receiver S2 is shown in Figure 16.

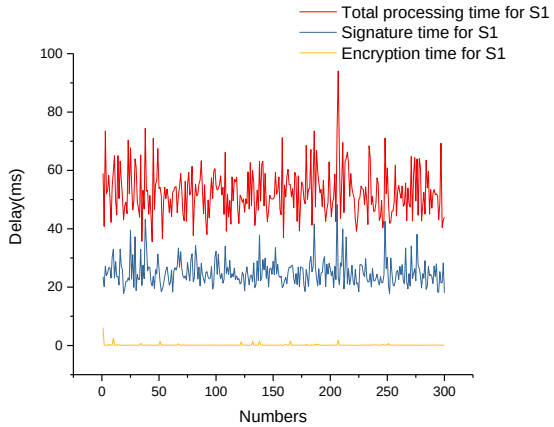

Figure 15. The comparison of the processing time for S1.

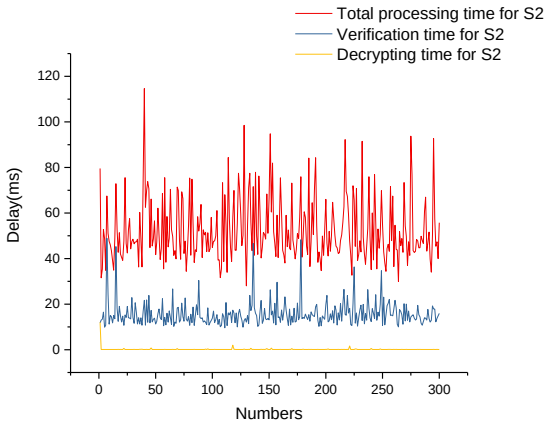

Figure 16. The comparison of the processing time for S2.

As can be seen from Figure 15, the average total time for processing of the tunnel gateway S1 is 52.6 ms, in which the data signature time is 25 ms, which accounts for 47.5% of the total processing time; the data encryption time is 0.19 ms, which accounts for 0.3% of the total processing time. As can be seen from Figure 16, the average total time for processing of the tunnel gateway S2 is 52.5 ms, in which the data verification time is 15.7 ms, which accounts for 29.9% of the total processing time; the data decryption time is 0.18 ms, which accounts for 0.3% of the total processing time. According to the analysis, the signature algorithm accounts for the largest proportion in security processing, and the complexity of signature algorithm in security processing directly determines the total forwarding delay of tunnel gateway.

6. Conclusion

In this paper, we propose P4Sec, a novel software-defined network packet security tunnel forwarding mechanism. For this mechanism, in light of the lack of effective data confidentiality protection, authenticity and integrity verification, as well as other problems relevant to SDN, we apply P4 technology in the SDN, design the P4Sec header, and utilize of its four functions to realize security tunnel forwarding based on P4Sec. Based on the open source software switch BMv2, we initially implemented security tunnel forwarding based on P4Sec, and then valuated the effectiveness, safety features, time consumption of the mechanism. The experimental evaluation shows that tunnel forwarding based on P4Sec effectively implements encryption, decryption, signatures, and verification of data packets in SDN, ensuring the confidentiality, authenticity, and integrity of the data packets transmitted in the tunnel. In the process of data packet forwarding, routing and forwarding are based on gateway identity such that network forwarding behavior is defined according to gateway identity, creating a novel method of routing and forwarding for SDN. For future work, we will also study the SDN data plane forwarding security problem, and further improve data processing capability.

Acknowledgments

This work was supported by the National Science Foundation of China (No. 61572517).

Conflict of interest

All authors declare no conflicts of interest in this paper.

References

1. N. McKeown, How SDN will shape networking, Open Networking Summit, (2011).
2. H. Kim and N. Feamster, Improving network management with software defined networking, *IEEE Commun. Mag.*, **51** (2013), 114–119.
3. J. A. Wickboldt, W. P. De Jesus, P. H. Isolani, et al., Software-defined networking: management requirements and challenges. *IEEE Commun. Mag.*, **53** (2015), 278–285.
4. D. Kreutz, F. M. Ramos, P. Verissimo, et al., Software-defined networking: A comprehensive survey, *P. IEEE*, **103** (2015), 14–76.
5. I. Ahmad, S. Namal, M. Ylianttila, et al., Security in software defined networks: A survey, *IEEE Commun. Surv. Tut.*, **17** (2015), 2317–2346.
6. Z. Shu, J. Wan, D. Li, et al., Security in software-defined networking: Threats and countermeasures, *Mobile Netw. Appl.*, **21** (2016), 764–776.
7. Z. Cai, C. Hu, K. Zheng, et al., Network security and management in SDN, *Secur. Commun. Netw.*, (2018).
8. A. Shaghaghi, M. A. Kaafar, R. Buyya, et al., Software-defined network (sdn) data plane security: Issues, solutions and future directions, (2018), arXiv preprint arXiv:180400262.
9. S. Gao, Z. Li, B. Xiao, et al., Security threats in the data plane of software-defined networks, *IEEE network*, **32** (2018), 108–113.
10. N. Mckeown, T. Anderson, H. Balakrishnan, et al., OpenFlow: Enabling innovation in campus networks, *ACM SIGCOMM Comp. Com.*, **38** (2008), 69–74.
11. Open Networking Foundation, OpenFlow Switch Specification Version 1.4.0., 2013. Available from: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.
12. P. Bosshart, D. Daly, G. Gibb, et al., P4: Programming protocol-independent packet processors, *ACM SIGCOMM Comp. Com.*, **44** (2014), 87–95.
13. M. Dhawan, R. Poddar, K. Mahajan, et al., SPHINX: Detecting security attacks in software-defined networks, *NDSS*, (2015), 8–11.
14. T. Sasaki, C. Pappas, T. Lee, et al., SDNsec: Forwarding accountability for the SDN data plane, *IEEE*, (2016), 1–10.
15. S. W. Shin and G. Gu, Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks, *IEEE*, (2012), 1–6.
16. P. Bosshart, G. Gibb, H. S. Kim, et al., Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN, *ACM SIGCOMM Comp. Com.*, **43** (2013), 99–110.
17. A. Shamir, Identity-based cryptosystems and signature schemes, *Springer*, (1984), 47–53.
18. T. Kivinen and M. Kojo, RFC 3526: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE), 2003. Available from: http://tools.ietf.org/html/rfc3526.

19. N. F. Pub, Advanced encryption standard (AES), Federal information processing standards publication, **197** (2001), 0311.
20. M. Dworkin, Recommendation for block cipher modes of operation. *NIST,* (2001).