An SDN-enabled Elliptic-curve Diffie-Hellman Key Exchange towards Secure P2P Networking

Wenjun Fan, Siyuan Wu and Hao Chen
School of Advanced Technology, Xi'an Jiaotong-Liverpool University
Suzhou, Jiangsu, 215123, P.R.China
Wenjun.Fan@xjtlu.edu.cn; {Siyuan.Wu16, Hao.Chen22}@student.xjtlu.edu.cn

***Abstract*—The network connections based on the Transmission Control Protocol (TCP) are reliable but lack security protection. The conventional Transport Layer Security (TLS) protocol adds an extra layer over the transport layer which enables payload encryption of the TCP segment. However, TLS relies on certificate authority (CA) to distribute the public key (for preventing Man-in-the-Middle attack), which involves nonnegligible overhead and constrains its use, e.g., TLS is not appropriate to peer-to-peer (P2P) networks due to the huge communication overhead. Therefore, this paper proposes a novel key distribution mechanism towards securing the TCP connection on P2P network. The mechanism applies an SDN-enabled approach to facilitate the Elliptic-curve Diffie-Hellman key exchange. With this mechanism, the key exchange can prevent Man-in-the-Middle attack with minimal communication overhead. The experimental results built on the prototype show that this approach is efficient.**

***Index Terms*—Software-defined Networking, Elliptic-curve Diffie–Hellman, TCP Connection, P2P Network**

I. Introduction

The Transmission Control Protocol (TCP) [1] was proposed in 1974. During the past decades (almost half a century), the protocol itself as well as its various implementations by different platforms and operating systems (OS) have evolved many times for patching the security vulnerabilities [2]–[4] and enhancing the performance. Owing to the security issues of the plain-text TCP connection, Transport Layer Security (TLS) [5] was proposed in 1999, which was built on the now-deprecated Secure Sockets Layer (SSL). TLS is a cryptographic protocol designed over TCP to provide communications security. Although TLS provides protection for the TCP segment's payload, the root of trust relies on the security of the certificate authority (CA) adopted. The CA is applied to distribute and verify the public keys, otherwise, TLS still suffers Man-in-the-Middle (MITM) attack. However, CA is not always secure [6]. In July 2011, e.g., the DigiNotar security breach resulted in an attacker using the company's CA infrastructure to issue hundreds of rogue digital certificates for high-profile domains, including one for google.com that was later used in a mass surveillance attack against Internet users [7]. In fact, taking CAs for granted when we blindly trust and use them is the opposite of Zero Trust [8], [9], which instructs us never to trust and always to verify the authenticity.

Moreover, the public key distribution via CA incurs non-negligible communication overhead, which deters TLS from being adopted by peer-to-peer (P2P) networks. For instance, the permissionless cryptocurrency P2P networks (e.g., the Bitcoin P2P network) don't use CA-based TLS because of the huge communication overhead, whereas the Bitcoin P2P network encounters critical security issues due to the plain-text TCP connection [10].

In order to solve the single point of failure stemmed from centralized CA and apply a less costly cryptographic protection to the TCP connection on P2P network, in this paper, we are motivated to propose a novel SDN-enabled approach for facilitating the Elliptic-curve Diffie-Hellman key exchange (ECDH). As we know, ECDH is often used to provide forward secrecy, while itself is a non-authenticated key agreement protocol, and thus, it is still vulnerable to MITM. In other words, the public key exchange in the ECDH protocol needs protection. In theory, our approach is appropriate to any cryptographic technique, however, we focus on ECDH since it offers better security with a smaller key size.

Therefore, the objective of this proposal is to provide a secure way to distribute public keys of ECDH for the two end-points in the context of Software-defined Networking (SDN), which has been regarded as the next generation of network architecture. The most recently emerged software-defined wide area network (SD-WAN) technology [11] has attracted even more extensive attention since it aims at long-distance data transmission across multiple domains. It is estimated that the SD-WAN market size will grow at a CAGR of 35.94% from 2023 to 2030 [12]. Thus, the SDN-enabled approach has a promising prospect for the future network development.

The contributions of this paper are summarized as follows:

- This paper proposes a novel SDN-enabled public key distribution mechanism for the ECDH protocol.
- A proof-of-concept of the approach is implemented for facilitating the secure and efficient TCP connection for the P2P networking.
- This paper conducts a number of experiments built on the prototype in order to evaluate the efficiency of this proposed approach.

The rest of this paper is organised as follows: Section II outlines the threat model; Section III proposes the design of the approach and performs a security analysis; Section IV demonstrates the implementation; Section V presents the experimental results built on the prototype; Section VI reviews

979-8-3503-7099-7/24/$31.00 ©2024 IEEE

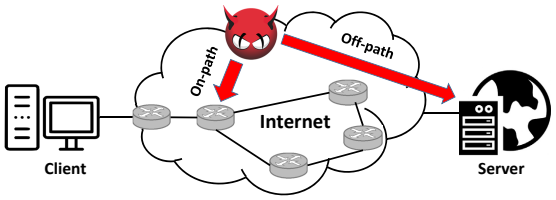

Fig. 1. Threat model including both on-path and off-path attacks.

the related work; Section VII concludes the paper and proposes avenues for future research.

II. Threat Model

This section presents the threat model that this approach builds on. The threat model is in compliance with normal Internet scenario, in which a client connects a server through a number of router hops. It is worth indicating that this threat model is not limited to the client-server (C/S) network architecture, while it is also applicable to the P2P network architecture, since with the latter, every peer acts as both server and client simultaneously.

Figure 1 graphically shows a general scenario of the threat model used in this paper. An adversary (with evil appearance in the figure) on the Internet can launch either off-path attacks or on-path attacks (such as MITM). In regard to executing off-path attack, the adversary is assumed to be able to blindly guess the TCP connection state or use some side-channel approaches [13]–[18] to obtain the expected TCP sequence number in order to hijack the connection. By contrast, should the adversary be powerful enough, a privilege position between the client and server is obtained, i.e., at least one router/switch on the route path is compromised or controlled by the adversary, then the MITM state is built. With this, the MITM attacker can intercept the TCP connection while the client and server are even not aware of its existence. In particular, the MITM attacker can observe, discard, and modify all the traffic between the client and server. However, the packets discarding is out of the scope of the threat model, since such denial-of-service (DoS) attack is always free as if the attacker already gets the privilege position. Besides, the client and the server over here are secure and they can prevent themselves from being compromised.

III. Approach

The section mainly proposes the design of the SDN-enabled ECDH mechanism and the security analysis elucidating how the mechanism would resist the corresponding attacks.

A. A Brief of ECDH

Foremost, an elliptic curve, E, is a plane curve over a finite field which consists of the points satisfying the following Equation (1):

$$E : y^2 = x^3 + ax + b \tag{1}$$

Thus, given an E, the point multiplication is defined as the repeated addition of a point along that curve, which denotes as $nG = \sum_{i=1}^{n} G$ for some scalar (integer) n and a point $G = (x, y)$ that lies on the curve. This type of curve is known as a Weierstrass curve. The security of modern Elliptic Curve Cryptography (ECC) depends on the intractability of determining n from $Q = nG$ given known values of Q and G if n is large (known as the elliptic curve discrete logarithm problem by analogy to other cryptographic systems).

TABLE I
The Definition of Notation

Notation	Definition
p	The prime used to set the size of the finite field
$\mathbb{F}_p$	The finite field given by the integers modulo p
a, b	The two constants less than p for defining the elliptic curve
G	The base point or generator
n	The order of G
h	The cofactor
$\mathcal{O}$	The identity element
$E(\mathbb{F}_p)$	The elliptic curve satisfying $y^2 \equiv x^3 + ax + b \mod p$, where (x, y) are pairs of non-negative integers less than p
d_C	The client's private key
Q_C	The client's public key
d_S	The server's private key
Q_S	The server's public key
x_k	The shared secret key

To use ECC, two parties must agree on all the elements defining the elliptic curve, that is, the domain parameters of the scheme. The size of the field used is typically a prime[1] denoted as p. The elliptic curve E is defined by the constants a and b used in the defining Equation (1). The cyclic subgroup is defined by its generator (a.k.a. base point) G. The order of G, which is the smallest positive number n such that $nG = \mathcal{O}$ (the point at infinity of the curve, and the identity element), is normally prime. Since n is the size of a subgroup of $E(\mathbb{F}_p)$, it follows from Lagrange's theorem that the number $h = \frac{1}{n}|E(\mathbb{F}_p)|$ is an integer. In cryptographic applications, h is called the cofactor and must be small (i.e., $h \leq 4$) and, preferably, $h = 1$. To summarize, in the prime case, the domain parameters are (p, a, b, G, n, h), which should be generated by a 3rd trusted party like a CA server for the C/S networking or a bootstrap server for the P2P networking. All the above notations are defined in Table I.

Given the domain parameters, the ECDH protocol works as follows. First, each party must have a key pair suitable for elliptic curve cryptography, consisting of a private key d (a randomly selected integer in the interval $[1, n-1]$) and a public key represented by a point Q (where $Q = d \cdot G$, that is, the result of adding G to itself d times). Let client's key pair be (d_C, Q_C) and server's key pair be (d_S, Q_S). Each party must know the other party's public key prior to execution of the protocol. Thereafter, the client computes point $(x_k, y_k) = d_C \cdot Q_S$, while the server computes point $(x_k, y_k) = d_S \cdot Q_C$. The shared secret is x_k (the x coordinate of the point). The shared secret calculated by both parties is equal, because $d_C \cdot Q_S = d_C \cdot d_S \cdot G = d_S \cdot d_C \cdot G = d_S \cdot Q_C$.

[1]Or, it is a power of two (2^m), which is called the binary case. In this brief, we only describe the prime case.

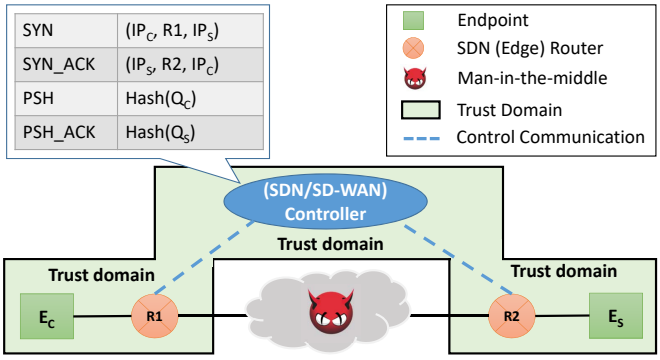

Fig. 2. General model of the proposed SDN-enabled ECDH mechanism towards secure exchange of the public keys.

As we can see, with ECDH, the only information about the shared secret key is the public keys of both sides and the domain parameters. No party except the client and server can determine their private keys, and therefore, no party other than them can computer the shared secret, unless that party can solve the elliptic curve discrete logarithm problem.

Nevertheless, ECDH is still not enough to resist the MITM attack, since in the above process, Both parties must be able to confirm that the public key received is indeed the other party's. We prove this below.

Theorem 1. *In the context of the ECDH protocol, if two parties want to communicate, if one party can verify that the received public key Q' is equal to the public key Q transmitted by the other party, then the ECDH protocol can resist MITM.*

Proof. If both parties can verify the transmitted public keys Q_C and Q_S, even if an MITM attacker can obtain these public keys, it cannot calculate the shared key because it does not know the corresponding private key. As a result, the MITM attacker can no longer break the encrypted channel. □

B. Mechanism Design

The proposed SDN-enabled ECDH mechanism is mainly designed for resisting the MITM attack in compliance with the aforementioned threat model. Figure 2 presents the design of the proposed mechanism. As we can see, this mechanism is based on the SDN network, which includes control plane and data plane. The control plane includes SDN/SD-WAN controller depending on the network type, i.e., local area network (LAN) or wide area network (WAN). Our approach should be no limitation to either of them. Also, the data plane includes the SDN switch or SDN edge router (a.k.a. middlebox) depending on the network type. In the figure, the router $R1$ directly links to the client E_C, and on the other side, the router $R2$ directly links to the server E_S. Moreover, the trust domain in our case is comprised of the controller, the client and server as well as their conjunct SDN routers/switches. That means all these entities as well as the control communication between controller and SDN routers won't be compromised by the attacker.

First of all, it is assumed that the public keys have been generated by the endpoints respectively. On the controller, an information table will be created, which will record the public key information (i.e., the hash value of Q) and the link information consisting of the source IP address IP_C and destination IP address IP_S (or the other way around according to the traffic direction) and the conjunct SDN switch.

With such setup, the SDN-enabled ECDH mechanism will go through the following steps.

1) First, the client E_C initiates the TCP connection by sending out the `SYN` packet. Once $R1$ receives this packet, since it doesn't have any flow entry to process this `SYN` packet, $R1$ will hand it in to the controller via sending a `Packet_In` packet through the OpenFlow protocol. When the controller gets the `Packet_In` packet, it can obtain and save the link information as a 3-tuple (IP_C, $R1$, IP_S). Afterwards, the controller sends a `Packet_Out` back to $R1$ and makes the `SYN` packet forward. Note that at this moment, still no flow entry is installed on $R1$. When the `SYN` packet reaches $R2$, another round of `Packet_In` and `Packet_Out` will proceed, whereas the controller will neither save any information nor install flow entry to $R2$. Later, $R2$ forwards the `SYN` packet to E_S.
2) Once E_S receives the `SYN` packet, it will respond a `SYN_ACK` packet to E_C. So, the `SYN_ACK` packet will first arrive at $R2$. Since having no flow entry (of course), $R2$ will hand it in to the controller. At this time being, the controller will check the saved IP_C and IP_S to make sure it is the required `SYN_ACK` packet and then store this link information to the information table as another 3-tuple (IP_S, $R2$, IP_C). Thereafter, the `SYN_ACK` packet will go through the same route back to E_C. Later, E_C will send out an `ACK` packet to complete the 3-way handshake and establish the TCP connection between the two endpoints.
3) Soon after the TCP connection builds, E_C will send out a `PSH` packet that includes Q_C it generated. The `PSH` packet will first go to the controller as the first step due to empty flow table of $R1$. Thus, the controller can calculate and save the hash value of Q_C, i.e., $Hash(Q_C)$. Later, the `PSH` packet containing Q_C will arrive at $R2$. Since $R2$ has no flow entry either, it will hand the packet to the controller. Now that the controller has $Hash(Q_C)$, it can compare the hash value of the one given by $R2$ with the saved hash value. If they are the same, forward the `PSH` packet to E_S. Therefore, E_S receives the desirable Q_C.
4) Eventually, E_S responds a `PSH_ACK` packet containing its Q_S back to E_C. The Q_S contained packet will go through the same process as the above step. Thereby, both the two endpoints finish the exchange of Q, and a secret key can be agreed upon.
5) After a configurable time interval, the middlebox can notify the controller to remove the public key and link information from the information table for preventing data leakage.

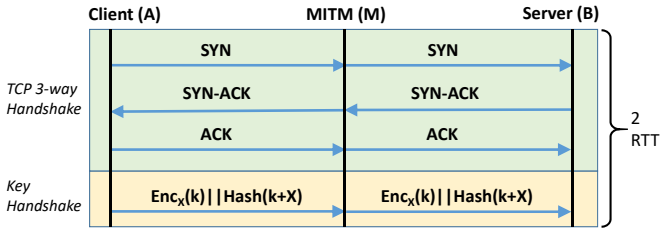

Fig. 3. The ephemeral key distribution for every new TCP connection.

Further, the firstly shared secret key can act as a master secret key (X) to distribute ephemeral secret key (k) for each new TCP connection. Since the client and server already have X, when a new TCP connection needs to be built between them, the client will generate a new k, and then send $Enc_X(k)||Hash(k+X)$ to the server, where $Enc_X(k)$ stands for encrypting k with X, and $Hash(k+X)$ is the message authentication code (MAC) which concatenates k with X to create $k+X$ and calculates the hash value. As the server knows X, it can decrypt the ciphertext by Equation (2).

$$k = Dec_X(Enc_X(k)) \tag{2}$$

Also, the server can verify the authentication and message integrity of k via comparing the received MAC code and the calculated MAC code using Equation (3).

$$Hash(k+X) = Hash(Dec_X(Enc_X(k)) + X) \tag{3}$$

Once k is agreed, an application can use the shared secret key for encrypting the messages and transmit the ciphertext on the public network. Figure 3 shows the ephemeral key k distribution approach for every new TCP connection. We can see that it (plus TCP 3-way handshake) takes 2 round-trip times (RTT), which is less than the 3 RTT spent by TLSv1.3.

C. Security Analysis

With our approach, the MITM attacker can exist in the untrusted domain. However, the MITM attacker cannot modify the data, in particular, it cannot modify the exchanged public keys. The security properties provided by our approach can be discussed as follows:

- Unlike a 3rd party CA, an **SDN controller**[2] is often managed by domain network administrator, and the communication between controller and packet switch often rides on proprietary channel, which makes such a trust domain more controllable than domains that rely on 3rd party CAs. Thus, the MITM attacker cannot modify the link information stored in the controller, since any middle link information sent to the controller will be regarded as created by an attacker and thus discarded immediately.
- The **link information** is used to ensure the identity. Assuming the MITM attacker would like to take part in the transmission, it has to add/modify the link information in the controller's information table. The MITM attacker is not able to add a link information before the conjunct SDN switch/router, because the controller only accepts the first `SYN` or `SYN_ACK` packets sent by the conjunct $R1$ or $R2$.
- The **public key information** is stored and matched by the controller. Since the controller has the global view of the whole SDN network, and especially, it has the control communication channel with the middlebox which is inside the trust domain, it plays a vitally trusted party for verifying the public key.
- The **information table** only stores the link and public key information provisionally, which means once the key exchange is done, the corresponding information should be removed. Although the controller is in the trust domain, we still consider preventing the potential threats like internal attackers, external infiltration, or attempts to exhaust the size of the information table.

[2]To avoid single point of failure, the SDN controller can adopt a decentralized solution using the emerging technology like Blockchain [19].

Overall, our approach is counterfeit-sensitive since it uses the controller to store not only the public key information but also the link information. The link information can ensure the identity of the client and server, while the hash value of the public key can achieve the tamper-resistance.

IV. Implementation

This section presents the implementation for evaluating the proposed mechanism including the software specification and the testbed setting up.

A. Software Specification

Regarding the SDN controller, the Ryu SDN framework (supporting OpenFlow 1.5) [20] is employed, which is open source, python programming based, and well-documented. In our case, the endpoint link information and public key information recording application is developed upon the northbound API of the Ryu SDN framework.

For SDN switch, we use Open vSwitch (OVS) [21], a widely adopted open-source software switch in SDN. It consists of several modules working in both kernel space and user space. In its architecture, `ovsdb-server` is the database server that uses the Open vSwitch Database Management Protocol (OVSDB) to manage OVS. The kernel datapath performs packet forwarding, which caches flow rules that are used to match and execute actions on packets received directly in the kernel space, enabling high performance of packet forwarding. In case of a cache miss, the kernel datapath delivers the packet to the userspace daemon termed `ovs-vswitchd`, where the packet is handled, and flow rules are given back to the kernel datapath for handling subsequent packets that they match against. This process is known as an upcall. The communication between `ovs-vswitchd` and the kernel module is done via a netlink socket. For performance consideration, most actions in OVS are implemented in the kernel datapath to avoid making copies and context switches between kernel and userspace.

We use the recommended elliptic curve domain parameters `secp256r1` provided by the Standards for Efficient Cryptography Group (SECG) in [22], where the domain parameters are

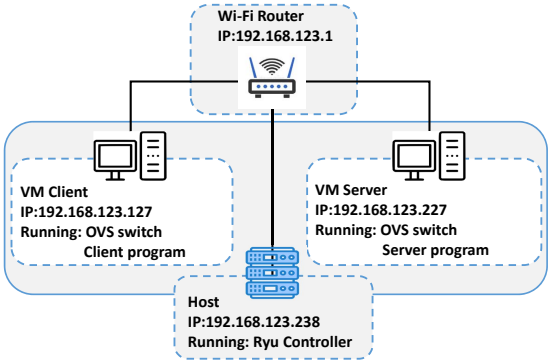

Fig. 4. Testbed setting up for evaluating the prototype.

chosen verifiably at random as specified in ANSI X9.62 [23]. The `secp256r1` curve, also known as the NIST P-256 curve, is widely adopted due to its strong security properties and efficiency. In addition, for implementing a proof of concept with the ECDH protocol in our case, we use SHA256 as the hash function, and AES-256 with mode ECB as the symmetric encryption algorithm.

B. Testbed Setting Up

The specification of the testbed is detailed as follows. The host is an x64-based PC with CPU 12th Gen Intel(R) Core(TM) i9-12900H 2.50 GHz and RAM 32 GB. The host installs two virtual machines (VM) using VMware workstation as the VM hypervisor. The two VMs both have the same specification: 4-core@5.1GHz Intel Core i7, 16GB memory, and Ubuntu 20.04. Further, both VMs bridge to the host so as to get real IP addresses from the router. Our router's architecture specification is MediaTek MT7621 ver:1 eco:3, and it has the wireless communication speed 866/866 Mbps. One VM working as the client deploys an OVS switch and the client side program, while the other VM working as the server deploys an OVS as well as the server side program. The host machine runs the Ryu (SDN) controller.

With the above mentioned specification, the testbed can be set up as Figure 4 shows. The router manages a private network with IP prefix 192.168.123.1/24. The two VMs (i.e., VM Client and VM Server) use bridge network linking to the host. Thus, the router assigns IP addresses of 192.168.123.238, 192.168.123.127, and 192.168.123.227 to the host, server, and client, respectively. Since all the nodes (the VMs and host) locate on the same network, the SDN controller can connect to the OVS switches.

V. Experimental Results

This section shows the experimental results built on the prototype implementation.

A. Processing Time with Different Public Key Lengths

We first test the processing time in terms of various public key lengths using our approach. The processing time indicates the period that the client sends `SYN` to the server till it receives Q_S from the server. Hence, this time calculation includes the TCP 3-way handshake, the ECDH public key exchange, the process for `Packet_In` and `Packet_Out` as well as the flow entries installation as the entire overhead. Figure 5 (Box & Whisker plot) shows the processing time in terms of different public key lengths.

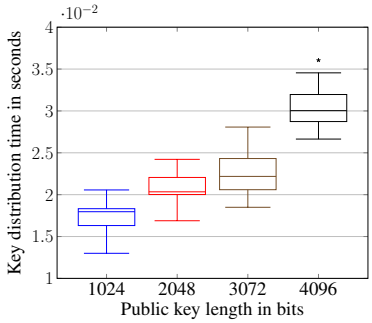

Fig. 5. Processing latency with different lengths of public key.

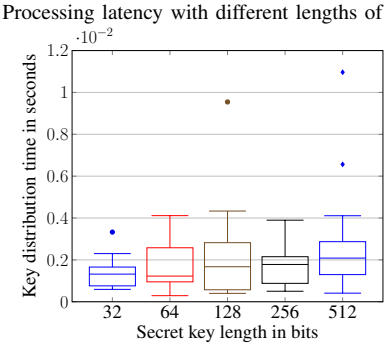

Fig. 6. Processing latency with different lengths of secrete key.

As the figure presents, with the increasing length of the public key when the private key length is set to 256 bits, the processing time for distributing the public key grows slightly. In our case, it is worth noting that the payload generation time for the elliptic curve private key is 0.000039 seconds. According to our astute analysis, the most time-consuming stems from the controller processing the public keys using `Packet_In`, which takes 0.016 seconds in mean for two rounds because the controller does not install flow entries after processing the TCP 3-way handshake. Whereas even using a key length value of 4,096 bits, the average processing latency is around 0.03 seconds and the outlier value is 0.036 seconds, which only brings negligible overhead for the entire data transmission.

B. Processing Time with Different Secret Key Lengths

Also, we test the processing time with various ephemeral secret key lengths when we fix the master secret key length to 256 bits. It is worthy of denoting that the AES key generation time is very short, around 0.0000012 seconds on average in our case. Figure 6 shows the testing result, whereby we can see that the different ephemeral secret key lengths do not impact the processing time significantly. The average processing time only increases a little.

In particular, the ephemeral key distribution time is significantly shorter than the public key distribution time, because the controller does not need to process the key contained `PSH` packet and compare the `Packet_In` packet with the value in the information table. The SDN controller can install the flow

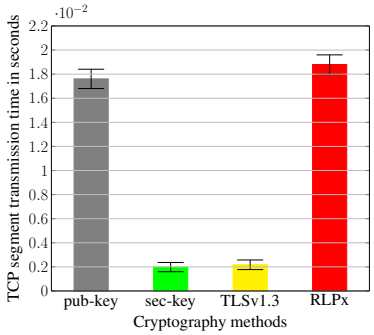

Fig. 7. Comparison with TLS1.3 and RLPx about the TCP segment transmission latency (the former two are our approach for public key exchange and ephemeral secret key distribution).

entries instantly when dealing with the TCP 3-way handshake segments. Only the server side's verification against the newly created secret key takes a little time latency. When we set the secret key length as 512 bits, though some outliers go beyond 0.01 seconds, the average processing time is 0.00208 seconds, which is still in the order of a thousandth of a second. With this result, in real practice, we could choose a relatively large secret key length to enhance the security degree.

C. Comparison with TLSv1.3 and RLPx

In addition, we conduct a contrast test amongst TLSv1.3 using TLS_AES_256_GCM_SHA384 as the cipher, RLPx [24] using elliptic curve integrated encryption scheme (ECIES), and our approach for public key exchange and ephemeral secret key distribution (with the key length at 512 bits).

Figure 7 shows the contrast results. We can see that our approach uses an average time latency of 0.0176 seconds and 0.0019 seconds for public key exchange and ephemeral secret key distribution, separately. By contrast, TLSv1.3 spends 0.0021 seconds in mean for public key exchange, and RLPx takes 0.0188 seconds in mean which is even longer. That is because RLPx never uses CA and needs to exchange the public key in order to derive and create the shared secret key for every connection. It is worth signifying that in our approach the public key exchange will only take place in one time[3], and once the master secret key is shared, the main latency is originated from the ephemeral secret key distribution.

Thus, we can conclude that our approach is slightly more efficient than TLSv1.3, while significantly more efficient than RLPx for P2P networking. That is because our approach mainly uses symmetric encryption for ephemeral secret key distribution. Nevertheless, TLSv1.3 still relies on a 3rd party CA which is out of the control under the domain network administrator, while our approach never suffers that. Though RLPx also does not rely on CA since every node's ID indicates the node's public key (which makes the spoofing attack difficult), it brings nonnegligible networking overhead.

[3]TLS also needs the parties to download the certificate and CA's public key in the first place, which we don't compare in this paper.

VI. Related Work

This section reviews the related work with respect to secure networking approaches and key distribution methods.

A. Secure Networking Approaches

Cryptography is widely used for protecting the network communications between endpoints. TLS [5] adds an additional layer between the application layer and the transport layer for protecting the segment payload, while it has nothing to do with the segment header fields. Thus, it is unable to resist the TCP reset attacks [25] and off-path hijacking attacks [13]–[18]. Further, IPsec [26] is a secure network protocol suite that authenticates and encrypts packets of data to provide secure encrypted communication between two computers over an Internet Protocol network. It is used in virtual private networks (VPNs). However, IPSec is far too complex, and the complexity has led to a large number of ambiguities, contradictions, inefficiencies, and weaknesses [27]. In addition, Quick UDP Internet Connection (QUIC) [28] was proposed to be nearly equivalent to a TCP connection but with much-reduced latency for even using cryptographic protection. QUIC makes the exchange of setup keys and supported protocols part of the initial handshake process. Thereby, QUIC-Crypto just needs 0-RTT[4] for encrypting application data. However, QUIC-Crypto still relies on public key certificate like TLS, which has its inherent security problem.

B. Key Distribution Methods

Symmetric cryptography is often used to encrypt the data transmission on the Internet as it is efficient. However, the symmetric cryptography suffers from the issue of secret key distribution between trusted endpoints. In other words, the endpoints must first have a secure and authenticated way for exchanging the secret key, which however is hard to achieve in the first place. Public key cryptography stems from the secret key distribution problem, however, it still needs to prevent the public key exchange from MITM attack. A widely used approach for distributing the public key to resist against the MITM attack is using certificate authority (CA) [29], [30]. However, CA itself as a centralized public key infrastructure (PKI) has security problems [31]. Though several decentralized PKI solutions were proposed to solve the single point of failure issue, such as the blockchain based methods [32], the overhead is still a nonnegligible problem. In order to reduce the complexity of public key management and distribution, identity-based cryptography (IBC) [33] was proposed, in which a user's identity and a trusted third party issued master public key are incorporated for generating the user's public key. Whereas, with IBC, a new issue namely key escrow arises, whereby the user's private key is created by the trusted third party, which needs the user to have an unconditional trust on the third party. Even worse, it is impossible to revoke the user's credentials and issue new

[4]Conceptually, in terms of the QUIC-Crypto document, all handshakes in QUIC are 0-RTT (round-trip time), it's just that some of them fail and need to be retried.

credentials without either changing the user's identity or changing the master public key and re-issuing private keys. To address the problem of key escrow, certificateless public key cryptography (CL-PKC) [34] was proposed, which is a joint approach taking advantage of both certificate-based and identity-based public key cryptographies. In CL-PKC, there also exists a trusted third party termed key generation center (KGC), which only provides the partial public and private keys of all users. Nevertheless, KGC also faces the single point of failure issue. If KGC is compromised, though it is less devastating than that in IBC, the adversary can issue new "fake" keys on behalf of legitimate users and pretend that they are authentic keys.

VII. Conclusion

This paper proposed a novel SDN-enabled Elliptic-curve Diffie-Hellman mechanism to protect the TCP connections in the context of permissionless P2P networking. Taking advantage of the SDN technology, this mechanism never relies on any third party CAs, however, it can still resist against MITM attacks when it distributes the public keys. Further, the experimental results presented that this approach can yield a higher performance with negligible overhead than other cryptographic techniques due to an appropriate implementation. For future work, we plan to apply this mechanism to real-world permissionless cryptocurrency P2P networks to achieve literal utility. In particular, we will stress on evaluating the communication performance in terms of scalability.

Acknowledgment

This work was supported in part by XJTLU Research Development Funding RDF-21-02-012 and XJTLU Teaching Development Funding TDF21/22-R24-177. This work was also partially supported by the XJTLU AI University Research Centre, Jiangsu Province Engineering Research Centre of Data Science and Cognitive Computation at XJTLU and SIP AI innovation platform (YZCXPT2022103).

References

[1] V. Cerf and R. Kahn, "A protocol for packet network intercommunication," *IEEE Transactions on communications*, vol. 22, no. 5, pp. 637–648, 1974.

[2] G. C. Kessler, "An overview of tcp/ip protocols and the internet," *InterNIC Document, Dec*, vol. 29, p. 42, 2004.

[3] M. De Vivo, G. O. de Vivo, R. Koeneke, and G. Isern, "Internet vulnerabilities related to tcp/ip and t/tcp," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 1, pp. 81–85, 1999.

[4] A. Medina, M. Allman, and S. Floyd, "Measuring the evolution of transport protocols in the internet," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 2, pp. 37–52, 2005.

[5] T. Dierks and C. Allen, "The tls protocol version 1.0," Tech. Rep., 1999.

[6] S. B. Roosa and S. Schultze, "Trust darknet: Control and compromise in the internet's certificate authority model," *IEEE Internet Computing*, vol. 17, no. 3, pp. 18–25, 2013.

[7] J. R. Prins and B. U. Cybercrime, "Diginotar certificate authority breach "operation black tulip"," *Fox-IT, November*, vol. 18, 2011.

[8] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero trust architecture," National Institute of Standards and Technology, Tech. Rep., 2020.

[9] C. Buck, C. Olenberger, A. Schweizer, F. Völter, and T. Eymann, "Never trust, always verify: A multivocal literature review on current knowledge and research gaps of zero-trust," *Computers & Security*, vol. 110, p. 102436, 2021.

[10] W. Fan, S.-Y. Chang, X. Zhou, and S. Xu, "Conman: A connection manipulation-based attack against bitcoin networking," in *2021 IEEE Conference on Communications and Network Security (CNS)*, 2021, pp. 101–109.

[11] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu, "Software-defined wide area network (sd-wan): Architecture, advances and opportunities," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, 2019, pp. 1–9.

[12] vMR Team, "Sd wan market size and forecast," *Verified Market Research*, April 2023.

[13] Z. Qian, Z. M. Mao, and Y. Xie, "Collaborative tcp sequence number inference attack: how to crack sequence number under a second," pp. 593–604, 2012.

[14] Z. Qian and Z. M. Mao, "Off-path tcp sequence number inference attack-how firewall middleboxes reduce security," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 347–361.

[15] Y. Gilad and A. Herzberg, "Off-path tcp injection attacks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 16, no. 4, pp. 1–32, 2014.

[16] Y. Cao, Z. Qian, Z. Wang, T. Dao, S. V. Krishnamurthy, and L. M. Marvel, "Off-path tcp exploits: Global rate limit considered dangerous." in *USENIX Security Symposium*, 2016, pp. 209–225.

[17] Y. Cao, Z. Qian, Z. Wang, T. Dao, S. V. Krishnamurthy, and L. M. Marvel, "Off-path tcp exploits of the challenge ack global rate limit," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 765–778, 2018.

[18] X. Feng, C. Fu, Q. Li, K. Sun, and K. Xu, "Off-path tcp exploits of the mixed ipid assignment," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1323–1335.

[19] W. Fan, S.-Y. Chang, S. Kumar, X. Zhou, and Y. Park, "Blockchain-based secure coordination for distributed sdn control plane," in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*. IEEE, 2021, pp. 253–257.

[20] "Ryu sdn framework," https://ryu-sdn.org/.

[21] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15, USA, 2015, p. 117–130.

[22] Certicom Research, "Sec 2: Recommended elliptic curve domain parameters," https://www.secg.org/sec2-v2.pdf, 2010.

[23] American National Standards Institute (ANSI), "Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ecdsa)," American National Standard X9.62-2005, 2005.

[24] "The rlpx transport protocol," accessed on 19th May 2023 https://github.com/ethereum/devp2p/blob/master/rlpx.md.

[25] P. Watson, "Slipping in the window: Tcp reset attacks," *Presentation at*, 2004.

[26] N. Doraswamy and D. Harkins, *IPSec: the new security standard for the Internet, intranets, and virtual private networks*. Prentice Hall Professional, 2003.

[27] N. Ferguson and B. Schneier, "A cryptographic evaluation of ipsec," 1999.

[28] J. Roskind, "Quic: Multiplexed stream transport over udp," *Google working design document*, 2013.

[29] E. Gerck *et al.*, "Overview of certification systems: x. 509, ca, pgp and skip," *The Black Hat Briefings*, vol. 99, 1997.

[30] J. Weise, "Public key infrastructure overview," *Sun BluePrints OnLine, August*, pp. 1–27, 2001.

[31] J. A. Berkowsky and T. Hayajneh, "Security issues with certificate authorities," in *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference*, 2017, pp. 449–455.

[32] H. Bhanushali, A. Arthena, S. Bhadra, and J. Talukdar, "Digital certificates using blockchain: an overview," in *2nd International Conference on Advances in Science & Technology*, 2019.

[33] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Advances in Cryptology: Proceedings of CRYPTO*, 1985, pp. 47–53.

[34] S. S. Al-Riyami, K. G. Paterson *et al.*, "Certificateless public key cryptography," in *Asiacrypt*, vol. 2894. Springer, 2003, pp. 452–473.