Building Generic Scalable Middlebox Services over Encrypted Protocols

Cong Liu*, Yong Cui*, Kun Tan†, Quan Fan*, Kui Ren‡, and Jianping Wu*

*Tsinghua University
†Huawei Technologies
‡State University of New York at Buffalo

***Abstract*—The trends of the increasing middleboxes make the middle network more and more complex. Today, many middleboxes work on application layer and offer significant network services by the plain-text traffic, such as firewalling, intrusion detecting and application layer gateways. At the same time, more and more network applications are encrypting their data transmission to protect security and privacy. It is becoming a critical task and hot topic to continue providing application-layer middlebox services in the encrypted Internet, however, the state of the art is far from being able to be deployed in the real network. In this paper, we propose a practical architecture, named *PlainBox*, to enable session key sharing between the communication client and the middleboxes in the network path. It employs Attribute-Based Encryption (ABE) in the key sharing protocol to support multiple chaining middleboxes efficiently and securely. We develop a prototype system and apply it to popular security protocols such as TLS and SSH. We have tested our prototype system in a lab testbed as well as real-world websites. Our result shows PlainBox introduces very little overhead and the performance is practically deployable.**

I. Introduction

Although originally the Internet was designed to have a simple core and push complex functions into edges, the existing network is no longer simple. There are an increasing number of network appliances, or middleboxes, inside the network and perform critical network functions, such as firewalling, load balancing, intrusion detecting, and application layer gateways [1]. Many of these middleboxes need to access packet data, and some of them may even need to modify the packets on-the-fly [2]. For example, application proxies may be deployed by an enterprise IT department to reduce the network traffic and provide better response time for users [3]. As another example, an Internet Service Provider (ISP) may want to deploy an intrusion detection and prevention system (IDPS) that may scan the packet data for intrusion signatures. Moreover, there are an increasing number of middleboxes chaining on the same path with the rapid deployment of NFV [4] and Service Function Chaining [5].

However, the recent trend of applying encryption in communications imposes new challenges to these middlebox services. The rapid adoption of secure communication protocols is mainly driven by the requirement to protect user security and privacy. These security protocols employ cryptography and may secure communication at different layers. For example, TLS [6] secures TCP payloads, SSH [7] provides application layer security and IPsec [8] encrypts data at network layer. Since the keys are only negotiated at the communication ends, the middleboxes cannot decode the encrypted packets and as a consequence, many functions have to be disabled if they need to look into the plain-text of the user data, such as in-network caching and intrusion detection.

To address this challenge, previous work has either proposed new methods to work without decrypting the packets (e.g., Blindbox [1]), or modified certain security protocols (e.g., mcTLS [2]) to allow sharing session keys between communication end points and the middlebox services. However, previous approaches are far from ideal. Blindbox only supports very limited middlebox functions on limited data types (e.g., keywords matching), and even for these functions the performance (e.g., accuracy) may also degrade largely [1]. When a middlebox targets at a flow, it still requires the communication endpoints to provide their session key and decrypt user traffic to get the plain-text data.

On the other hand, though mcTLS provides a mechanism for TLS client and server to share their session keys with a middlebox, it is not practical for two reasons. First, it requires the TLS server to be involved into the key exchange process. In today's Internet, TLS has already been largely deployed in websites and to update all websites is very expensive. Even a large ratio of websites are still using quite old version of SSL, which has been deprecated for security reason. It does not even support the key sharing between a user device and a middlebox owned by the user side without updating the TLS server. Second, mcTLS explicitly defines new messages in the TLS handshake protocol to exchange middlebox certificates and session keys. The new messages must be standardized (e.g., in IETF) before it can be actually deployed, which may be very difficult and time consuming due to security reasons [9]. Defining new message also require a lot of changes to application source code, which is not practical especially on user-side devices that all application code should be modified and recompiled. Due to the difficulty in changing user-side software, the mcTLS work even didn't port a full-blown web browser in its implementation [2].

To solve the problem, we want to design a more practical session key exchange mechanism by using an out-of-band channel for its additional information exchange. A direct solution is to create a separate keep-alive TCP connection between the user device and the middlebox. However, since

Yong Cui is the corresponding author.

This work is supported by National Key R&D Plan of China (No. 2017YFB1010002) and NSFC Project (No. 61422206).

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

Authorized licensed use limited to: Tsinghua University. Downloaded on June 15,2022 at 04:21:40 UTC from IEEE Xplore. Restrictions apply.

the middlebox must provide its certificate for authentication and it is always the middlebox to first talk to the user, due to the largely deployed Network Address Translation (NAT), it is very difficult for a middlebox to initiate a TCP connection with a user. In addition, a keep-alive connection doesn't work because a user cannot know whether a known middlebox will locate on the path of the future flows to be created. Meanwhile, many networks perform ECMP load balancing on 5-tuples (source and destination IP addresses, source and destination ports, and transport protocol) so the TLS flow and the side channel may go through different paths.

In this paper, we propose a practical architecture, named *PlainBox*, to enable session key sharing among the communication endpoints and the middleboxes in the network path. Instead of modifying existing security protocols, PlainBox provides a secure out-of-band control plane to authenticate the middlebox services and it allows users to specify their sharing policies. The message exchange is embedded into the original 5-tuple data flow by a system-level agent to pass through NAT and ensure forwarding path over ECMP load balancing. PlainBox provides a simple user device API, through which applications may specify the policies and input session keys. Based on these policies, PlainBox will share the session keys to only trusted middleboxes, which may later use the keys to decipher encrypted packets. PlainBox employs Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [10] in the key sharing protocol design. It uses only a single message exchange to share keys to multiple middleboxes in a chain. Since the original protocol handshake is not changed in PlainBox, the server-side application service does not need to be modified, and the network function of the client-side application does not need to be changed neither. As a result, it would be much easier to extend PlainBox support to new applications.

We apply PlainBox to a set of protocols such as TLS, IPsec, SSH, and their combinations (e.g., TLS over IPsec). We have implemented PlainBox in Linux for user devices and extended the middleboxes to support PlainBox for accessing encrypted traffic. We have also built supports on different applications to use PlainBox API to share session keys with trustworthy middlebox services. Supporting PlainBox only require very few changes to user-side applications, and we can even support web browsers (e.g., Chrome and Firefox) without any changes to their source code or recompilation. We have tested our prototype system for performance on both handshake and data processing. Our results show PlainBox introduces very little overhead for key sharing and achieves good performance, so it is practically deployable in real network.

The rest of the paper is organized as follows. We introduce the background information for our system in §II. §III describes the PlainBox system design. We then describe the specific design of the key sharing protocol in §IV. In §V-VI we describe the implementation of PlainBox and gives the evaluation result. Then we discuss related work in §VII. Finally, we conclude our work in §VIII.

II. Background

A. Middleboxes

Conventionally, packets on the Internet are transmitted just using clear text, from IP to application-layer protocols (e.g., HTTP). Nowadays people have paid more and more attention to security and privacy in network transportation because any device on the path can look into the flow data and even modify the content. It brings two types of middle-network watchers. The attackers collect user privacy for malicious usage and modify the packet to attack the hosts or servers. There are also friendly watchers which improve the network performance, functionality, and even security by accessing to plain-text data of network traffic.

Today middleboxes have already been widely deployed by ISPs and ICPs. For example, an ISP may deploy an intrusion detection and prevention system (IDPS) to improve network security or provide in-network web caching to enhance web services. A middlebox service may also be owned by the network user to provide better network function while locates at the cloud across the middlebox network [3], [11].

Different middlebox services may require different access to the plain-text user data. For example, a silent firewall reads plain-text data and just block the whole session once it matches the rule. A load balancer reads data for better selection without modifying the packet content. When the firewall is able to write into the plain-text data flow, it can send a warning to the user which results in a better service. An in-network caching system may need to be able to send redirect message to a user and it also requires writing to the plain-text data. An application-layer gateway requires writing to the plain-text data to modify protocol data online [12].

B. Security Protocols

To improve transportation security and privacy for end users, the best approach is to do encryption at data transmission. It simply encrypts all the payload and prevents middle attackers from understanding the original plain-text content. The encryption may be performed at several protocol layers, including network layer and transport layer.

Today many protocols that with data encryption have been widely used on the Internet. SSL/TLS [6] is a popular transport-layer encryption protocol which is widely used by HTTP as HTTPS. SSH [7] is also used for carrying other protocols or directly providing applications such as sftp. The mature development on cryptography protocols attracts applications to directly use them as underlying protocols instead of re-designing a new protocol (e.g., using TLS for DNS [13], XMPP [14], and Email [15]).

III. PlainBox Design Overview

In this section, we introduce the PlainBox system design. We summarize the design goals and then describe the system functions. Given the middlebox's identity, we allow the user to decide different permissions to the middlebox. We then discuss the security issues in our system. The specific description of the protocol design is at §IV.

A. Design Goals

Keep Security Protocol Unmodified In order to do session key sharing, we need to import additional message exchange between the communication endpoint and the middlebox.

Authorized licensed use limited to: Tsinghua University. Downloaded on June 15,2022 at 04:21:40 UTC from IEEE Xplore. Restrictions apply.

However, it is very expensive to insert new content into the original handshake protocols due to the difficulty in standardization. Without the standardization, it is very difficult to upgrade all client and server applications to support the new protocol messages in order to deploy the new function. Therefore, our design tries to keep the security protocols unmodified and keep client-side applications (e.g., web browser) unchanged by using out-of-band message exchange.

Achieve Server-side Transparency Though both the user device and the server share the same symmetric keys in security protocols, a system that requires updates on the server side is very difficult to be deployed. Since we could not update all the servers on the Internet at a stroke, if some of the servers do not support the new protocol requirements, the key-sharing function has to be disabled when a user device connects to the servers. In the other hand, for a client-side mechanism, an updated user device can make full use of the new feature regardless of whether other user devices are updated. To be more practical, we only involve the user device into the key sharing system and keep the server transparent. It allows us to easily deploy PlainBox with only a few software updates on a user device.

Efficient Middlebox Authentication In the real world, a user device typically generates multiple data flows from one or more applications when accessing the Internet. A middlebox may also serve a user device for multiple flows on multiple different protocols. Consider that PlainBox should be generic to support multiple security protocols and require minimal effort to extend the support for new protocols, we separate the middlebox authentication process from the key-sharing handshake.

B. Threat Model

In PlainBox, we require the user device to perform authentication of middlebox device, so we can assume that the user is aware of all the middleboxes in the protocol. In order to make sure the middlebox does not perform malicious operations over user traffic, we require there to be a centralized certificate authority to validate and distribute certificates with valid attributes to middleboxes. Based on the PKI model, we allow the clients to share their session keys with the trusted middleboxes. However, it is still possible that a middlebox may be compromised due to software attack or even malicious OS and operator. In this case, hardware-based trusted execution environment (e.g., Intel SGX [16]) can be used on the middlebox to prevent the attacks from OS and privileged users.

On the user device, we assume all the software environments are trusted, including the applications and OS. Since we import an additional user agent to manage application keys and permissions, in case the user device software environments are untrusted, SGX should be deployed to protect the user agent. The server is transparent to PlainBox protocol, so we do not introduce any requirement to the server.

C. Design Overview

The main task of the PlainBox is to safely share the user's session keys with the middleboxes and to enable the middleboxes access to the plain-text of the network traffic. In order to protect user security, we require that the middlebox should be non-transparent to a user and allow the user to authenticate a middlebox. When a user decides to trust a middlebox, it can then execute the key sharing process when it creates an encrypted connection. The user device and the middleboxes use a protocol based on ABE for security and scalable key exchange.

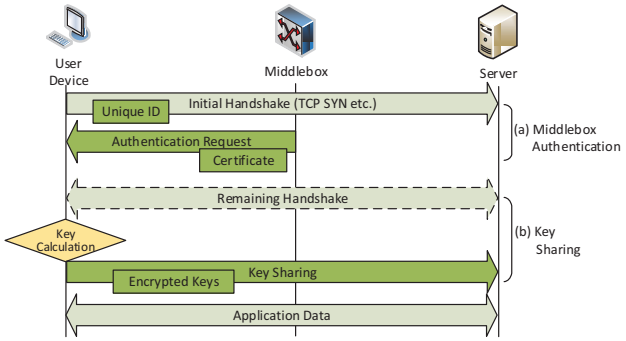


Fig. 1: PlainBox handshake flow for middlebox authentication and key sharing.

The middlebox classifies all input traffic for interested flows. Once an interested flow enters, the middlebox identifies the user identity of the flow. If the flow is sent from a new user device, the middlebox initiates an authentication request to the user device by sending its certificate and target information (e.g., target protocol and target destination domain). The authentication process follows the public key infrastructure (PKI) and the certificate should be signed by trusted certificate authority (CA). Once the authentication successes, the middlebox is able to receive the user keys of the requested protocols on each encrypted flow generated by the user. Then the middlebox can decrypt the encrypted session data and then use the plain-text data for further processing.

On the user side, we design a unified user device agent to manage permissions and policies for key sharing. The user agent is responsible for authentication requests from middleboxes and verifies the identity of the middlebox. The agent interacts with the local applications running encrypted protocols by providing an API for key registration. On the generation of a session, the corresponding application calls the API to register its key to the user agent.

Since PlainBox requires exchanging extra information between the user and the middleboxes for middlebox authentication and key exchange, it introduces two extensions to the protocol handshake as shown in Figure 1. The additional protocol exchange generally happens at the beginning of an encrypted flow during the handshake phase of the original encryption protocol. In addition to the original handshake, our information exchange is out of band between user device agent and the middleboxes, and doesn't involve the server or client-side application at the original end points for generic purpose.

IV. PlainBox Protocol Design

In this section, we describe the design of the PlainBox protocol on key sharing and middlebox authentication. We start from the basic out-of-band handshake protocol, then we introduce an optional mechanism to share keys with multiple middleboxes efficiently.

Authorized licensed use limited to: Tsinghua University. Downloaded on June 15,2022 at 04:21:40 UTC from IEEE Xplore. Restrictions apply.

A. Basic Handshake Protocol

The basic handshake flow of PlainBox is shown in Figure 1, including per-user middlebox authentication and per-session key sharing. In addition to the original protocol handshake, we need to exchange extra information between the user device and the middlebox, including the middlebox's certificate and the user's session keys for sharing. In order to exchange information between users and middleboxes, either changing existing protocol handshake process or issuing out-of-band message exchange is necessary. We find that directly modifying protocols requires defining new message and states for the specific protocol which lacks extensibility to other protocols. Therefore, in the PlainBox design we choose out-of-band message exchange for both middlebox authentication and key sharing.

The authentication requires the middlebox to send its certificate and related information to the user. In the traditional IPv4 Internet, the user is likely to behind NAT and is using a private IPv4 address. In this case, it is difficult for a middlebox to directly send messages to the user in a separate flow. Moreover, when there are multiple middleboxes requiring the keys from the same flow, we want to do key sharing only once to reduce the overhead on traffic and computation. However, the load balancer in real world network performs load balancing on 5-tuples. A message sent from the user device to the server may go through a different path with the original data flow so that the middleboxes may be unable to receive the message.

To solve the issue, we encapsulate our message into the packets with the same characteristic as the original flow through our additional message exchange is out-of-band. For example, if the original protocol is based on TCP, we also send our messages through the same TCP flow. In order not to affect original protocol, the original server and client application are required to ignore the transmission messages we choose in each protocol.

To reuse existing protocols, we need to use options in protocol headers to carry additional information and signal the packet content. The server can directly drop such messages and does not need to understand the meaning. However, using protocol options may cause the packet dropped by routers in middle network. According to previous measurements on IP options [17], [18], IPv6 extension headers [19] and TCP options [18], [20], [21], we know that the TCP options and IPv6 extension headers have been widely supported by middle network and can be used safely, however using IPv4 options may face a high risk to be dropped in the middle network. Therefore in PlainBox protocol design, we prefer using TCP options when the original handshake protocol is using TCP, and also use IP options/extension headers for non-TCP protocols.

B. 1:N Key Sharing between User and Middleboxes

In order to safely share the user keys with the middleboxes, the keys to be sent must be encrypted using the public keys from the middleboxes. In mcTLS [2] regular asymmetric encryption is used. It only supports the scenarios that only a small number of middleboxes present on the network path requiring the plain-text of the flow. For each middlebox on the path, the user devices have to encrypt the keys and send the encrypted result out once. The overhead on computing and transmission increases with the number of middleboxes so it does not scale in practice.

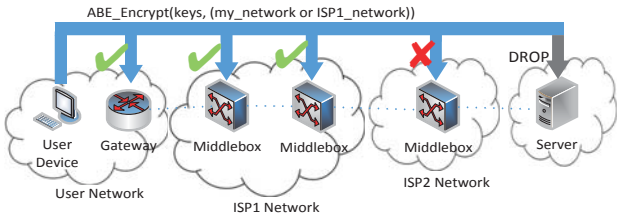


Fig. 2: Key sharing with multiple chaining middleboxes. The keys are encrypted using ABE by the user and sent to the server. Only middlebox from the local network and the ISP1 network is able to decrypt the data and get the keys. The server ignores the key-sharing message.

In the PlainBox, we design a more efficient key-sharing mechanism for the multiple chaining middleboxes on the same path. By merging the key-sharing message, we want to achieve sending only one copy of encrypted keys to all trusted middleboxes while still keeping the security on the sharing process. The key sharing process is shown in Figure 2. To achieve this, we make use of the Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [10], [22] mechanism. In the traditional public-key cryptography, the data is encrypted using a public key and only the one who owns the private key is able to decrypt the plain-text data. In CP-ABE, each ciphertext is specified with a policy which is a boolean expression of several attributes. Each key is associated with a set of attributes. A user can decrypt a ciphertext only if user key's attributes match the ciphertext's attributes. It is therefore possible to import CP-ABE into the user-middlebox scenario that each middlebox is assigned with several attributes while an encrypted protocol session can be specified with an access policy.

The user directly sends the encrypted session keys to the original destination when doing key sharing. The keys are encrypted by CP-ABE. In each encryption, the user specifies a policy to indicate who can decrypt this message. The policy on each flow is decided according to the user's permission management. Each middlebox is assigned with a set of attributes, including country, institution, application type, and any other necessary information. Thus a middlebox can decrypt the message to get the session keys if and only if it is trusted by the user and selected in the ABE policy.

In order for all the trusted middleboxes to receive the key-sharing message, the additional message must follow the same network path of the original data flow. In practice, network packets are usually load-balanced and distributed into different paths, however, packets in the same flow identified by 5-tuple are generally ensured to go through the same path [23]. Therefore, we put the key sharing message into the same type of packets as the original flow, using the TCP option or IP option/extension headers to indicate the specific payload type. Besides re-using the 5-tuple flows, the ISP that runs the middlebox can also use the service function chaining (sfc) [5] techniques to ensure the original flow and the extra key-sharing

Authorized licensed use limited to: Tsinghua University. Downloaded on June 15,2022 at 04:21:40 UTC from IEEE Xplore. Restrictions apply.

message to be forwarded on the same path.

C. Middlebox Authentication

In the design of middlebox authentication, we make use of a per-user scale authentication instead of per-flow authentication in mcTLS [2]. The reason is that in the access to the Internet, a user device typically generates a large number of different sessions. Nielsen reported that each U.S. user generated 51 sessions in December 2009 [24]. Even accessing the same site in the same application such as in web browsers, the user device may also generate several concurrent connections to improve performance [25]. Meanwhile, a middlebox typically serves a user for many connections. If a user device authenticates the same middlebox in each session, it may generate many repetitive transmissions and increase the traffic overhead in the protocol handshake.

We next consider how the middlebox identifies a user and then initiate authentication request. In order to identify whether a packet comes from a user that has already recognized the middlebox, it has to use IP address in the packet as an identifier. In the case of IPv4 NAT that an IPv4 address may be shared among multiple different user devices, we have to put additional fields into the packet to help the middlebox. Consider that the IPv6 address is enough for unique user identification, the additional field is only required by IPv4 flows. Moreover, consider that the key sharing may only happen at limited applications on limited flows, there is no need to put the additional identifier on packets not affected by PlainBox and the overhead of the additional field can be reduced. The user device only puts an additional identifier on the outgoing flows' handshake packets of which the session keys may potentially be shared. Since the identifier should be globally unique, we choose the hash of the DHCP Unique Identifier (DUID) from DHCPv6 [26], which is widely used and easy to calculate, as the user's unique ID. We further perform a hash to protect the privacy [27].

D. Permission Management

The user agent manages middlebox permissions similar to the permission management in Android. The permissions are identified by requested middlebox, target protocol, target destination, and requested operation. The permission decides whether the requested middlebox can read or rewrite the data from this host on the target protocol to the target destination. The user agent can pre-configure a policy on whether to grant the permission, e.g., allow all middleboxes from the same enterprise network and deny all other middleboxes. It can also interact with the user by popping up a message and let the user to decide whether to trust.

When the middlebox targets a flow and it has not been authenticated by the sender, i.e. there is no trusted identifier in the handshake packets sending from the user side, it initiates an authentication request to the user carrying its certificate signed by a trusted certificate authority (CA) and other request information. It contains the middlebox's public key and attributes, including country, institution, application type, and any other necessary information.

The operation types on permission include read/only (r/o) and read/write (r/w). In some middle services, the middlebox only logs the information from the plain-text flows or directly blocks the flow, so it only requires r/o permission of the flow. In other middle services, it also needs to modify the flow content, such as the content rewrite operations in ALG services, or sending HTTP redirect messages to users in middle network storage and firewall services. In this case the r/w permission is required.

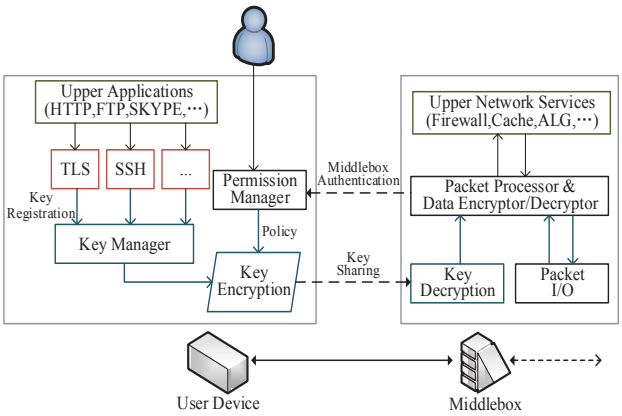


Fig. 3: System architecture in PlainBox user device and middlebox.

In security protocols, the Message Authentication Code (MAC) is used for data integrity authentication. With session encryption key, the middlebox is able to get the original plain-text however it is unable to modify the content or add insert new messages since the content is protected by the MAC. To be able to write to the flow, it has to get the MAC key to change the validation fields. Therefore, we enable r/o permissions by only sharing encryption key and r/w permissions by sharing both encryption key and MAC key.

V. System Implementation

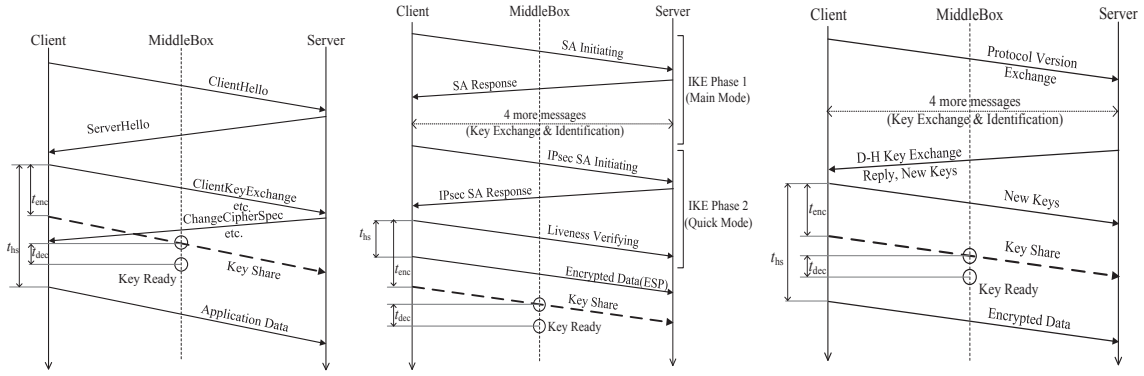
We have implemented a prototype system that supports middlebox authentication and key sharing of TLS 1.2 and SSH. Figure 3 shows the system architecture and the modules in PlainBox. Our middlebox successfully gets and decrypts user data from applications including Firefox(HTTPS) and OpenSSH(SSH) by displaying the correct plain-text of the flows.

PlainBox also supports protocol combination such as the case of HTTPS over IPsec. In this case, our user agent is able to manage session keys in all encryption protocols. Once the user decides to trust a middlebox on security protocols, we are able to share the keys from these applications on their generation. The middlebox is then able to decrypt the data from encapsulated packets.

A. User Device Agent

We implement a user agent program on user operating systems in C++. The task for the user agent is to communicate with middleboxes, manage middlebox permissions, manage local session keys, interact with local applications, and interact with the user. The general logic of our user device agent is cross-platform while the specific user interface and network socket programming are OS-specific.

Authorized licensed use limited to: Tsinghua University. Downloaded on June 15,2022 at 04:21:40 UTC from IEEE Xplore. Restrictions apply.



(a) TLS (b) IPsec (c) SSH

Fig. 4: PlainBox key sharing process in protocol handshakes.

The user agent tracks all local network packets. It is able to insert content into outgoing packets based on Netfilter and directly send packets using raw socket in order to send key-sharing messages to middleboxes. The user OS ignores all additional fields added by middleboxes by default. When the user agent receives an authentication request, it first checks the certificate from the middlebox. If the certificate is valid, it decides whether to trust the middlebox by the pre-configured policy or pop up a notification to the user to let the user make the decision. Once the middlebox is trusted, it is remembered until the timeout.

The user agent waits for new encryption handshake by listening on both outgoing packets and application API calls depending on the specific application implementation. Once a new handshake is caught, the user agent requests the session key from the application and shares the key with all trusted middleboxes. In most encryption protocols, the session keys are generated after the initial steps of the handshake process and there are still some remaining steps mainly for authentication before the transmission of encrypted application data. After the session keys are generated, the user agent encrypts them and sends to the trusted middleboxes through the same flow as the original handshake. Figure 4 demonstrates when the session key is shared in each protocol's handshake process. The encrypted keys are transmitted in an extra packet by marking a specific TCP option or IP option to signal this is a PlainBox key-sharing message to the middleboxes. It is important to ensure that the middlebox could obtain the plaintext of key materials before it receives encrypted application data so that the user agent must handle the calculation quickly enough. Since the encryption on keys takes some time, when the encrypted application data is generated faster than key encryption, our user device agent will buffer the first data packet until the keys are encrypted and shared to the middleboxes. Our evaluation results (§VI-A) show that the PlainBox generates very small delay on the protocol handshake even using CP-ABE for key sharing.

In order for the user agent to get the keys from different applications, the applications need to provide their keys to the agent. We design a unified API for applications to register their session keys. In popular web browsers, there are already built-in APIs that can be used for key registration. In our prototype, we use the SSLKEYLOGFILE API to export the master secrets. It is supported by both Firefox and Chrome. For applications that don't have built-in key sharing functions, we modify their source code to call our API for key registration. We add 47 lines of C code to OpenSSH for the API calls. As running on the user device, the user agent is considered trustable for getting all session keys on different applications. We put the centralized permission management on the user devices to improve the security of user keys. The session keys are still safe when exported to the user agent and will not be obtained by any untrusted middleboxes.

B. Middlebox

The middlebox is also implemented as software in C++. Though traditional network routers and middleboxes were often implemented as hardware, the trends on NFV [4] points out that the middlebox on the next generation network will be software running on virtualized servers. Our software middlebox implementation could be easily deployed on a new network and be easily extended to support new applications, protocols, and functionalities.

The control plane of the middlebox is responsible for initiating authentication requests with users and obtaining session keys. It listens to all passed traffic for the initial packets of its interested protocols. When the first packet of the interested protocol arrives and the user is considered not trusting this, the middlebox sends an authentication request to the user device with the next packet coming back from the server to the client. In TCP-based protocols, the middlebox decides whether the flow comes from a trusting user by the TCP SYN packet and initiates the authentication request along with the TCP SYN-ACK packet. By this way, the user can authenticate the middlebox as soon as possible.

The middlebox waits for the session keys from the user agent on each target flow on handshake state. For target protocols, it maintains per-flow state on session state and keys for further data plane processing. Once the session key of a flow is present, the flow is then handled by the data plane decryptor. We use openssl for data encryption and decryption.

Authorized licensed use limited to: Tsinghua University. Downloaded on June 15,2022 at 04:21:40 UTC from IEEE Xplore. Restrictions apply.

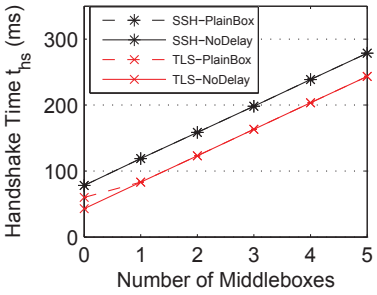


Fig. 5: Extra handshake time on each protocol with/without key sharing

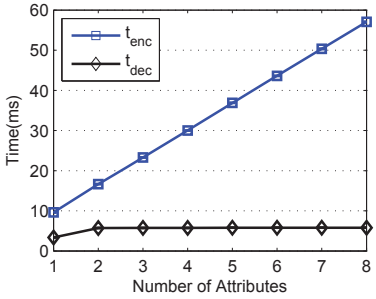


Fig. 6: The measurement of t_{enc} and t_{dec} on different key length and policy attributes.

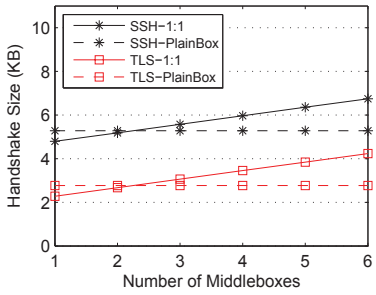


Fig. 7: Handshake size v.s. #middleboxes for 1:1 and Plainbox 1:N key sharing.

In some protocols, the session key may change during the transmission of the flow. The control plane is also responsible for the key changing events and obtaining new keys from the user agents. For the protocols that split their transport content into small pieces of chunks before doing encryption, the middlebox can support handling active flows from the middlebox of the transmission. In this case, the middlebox will send authentication requests and wait for session keys from the user side. Once the key is ready, it begins decrypting the flow from the first integrated encrypted chunk it receives.

The data plane encryption/decryption engine in middlebox handles input packets and decrypts the data using the session keys. When the middlebox services require to rewrite the content or insert extra messages into the flow, it uses the MAC keys shared by the users to construct new encrypted packets. It also supports decrypting encapsulated protocols, e.g., HTTPS over IPsec. In this case, when it finds an IPsec flow, it decrypts the flow and redirects a copy of the plain-text output to the input of the control plane engine. The engine re-processes it using necessary network-layer information for the original packet. When a new inner TCP/TLS handshake is identified, the control plane waits for its session key and then redirects the TLS flow back to the data plane decryptor.

VI. Evaluation

In this section, we evaluate the performance and overhead of the PlainBox system. On the control plane, we focus on the overhead on time delay by the key sharing protocol, and the system overhead caused by CP-ABE. On the data plane, we focus on the overhead on data decrypting and shows the system performance.

Experimental Setup In the testbed, we run the middleboxes as well as the clients and the servers at Linux servers running Ubuntu 14.04. Each server has Intel E5500 2.8Ghz CPU with 4GB RAM and two 1Gb NICs. We run Firefox as the client browser and Nginx as the server for HTTPS, OpenSSH for SSH, and also develop a set of benchmark tools for performance test. The servers are connected in a chain by 1Gb links. We add 20ms delay to each link following mcTLS [2]. The error bars indicate standard deviation.

A. Handshake Time

We first measure the time overhead on the protocol handshake after applying PlainBox key-sharing logic. Since PlainBox don't change the original handshake protocol in each application, the client application begins data transmission once the session key is present and the handshake is finished. In all protocols, the client generates the complete key materials in the middle of the handshake process. After that, it still has to spend some time on the remaining handshake and waiting for upper applications.

We denote the time between the client computes the session key materials and it generates the first encrypted packet for data transmission as t_{hs}. The PlainBox user agent is able to obtain session keys once the client has generated them. Before the keys are sent to the middleboxes, the encryption time on the keys cannot be ignored. After receiving the encrypted keys, the middlebox also needs some time on the decryption before it is able to decrypt user data. We denote the time on session key encryption and decryption as t_{enc} and t_{dec}. If we could ensure $t_{enc} + t_{dec} < t_{hs}$, the middlebox can have the session key ready before it receives the first encrypted data packet thus the PlainBox generates no delay time on the original protocol. Otherwise, we have to let the user device to buffer the first data packet until the session keys are encrypted and ready for sharing, and possibly buffer data packets at the middlebox before it is ready to decrypt the data.

We measure the handshake time in TLS and SSH by capturing the handshake messages and count the time interval between the first handshake packet after client key calculation and the first encrypted data packet. We count the time interval for ten times and report the mean. The t_{hs} is counted from the ClientKeyExchange message to the first ApplicationData message in TLS (Figure 4(a)) and from the ClientNewKeys message to the first encrypted packet in SSH (Figure 4(c)). The t_{hs} in TLS and SSH includes one RTT between the client and the server. We use eight attributes for the ABE encryption which is enough for common key-sharing cases. The lines marked as NoDelay are the original handshake time not affected by PlainBox in each protocol, and the lines marked as PlainBox denote the handshake time after buffering at the user side.

The result in Figure 5 shows that in TLS and SSH with a

Authorized licensed use limited to: Tsinghua University. Downloaded on June 15,2022 at 04:21:40 UTC from IEEE Xplore. Restrictions apply.

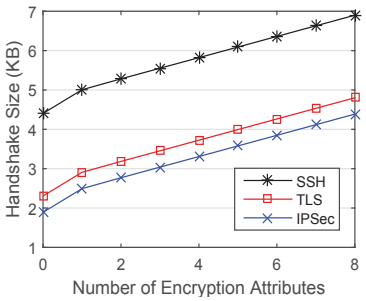


Fig. 8: The measurement of handshake size v.s. attributes.

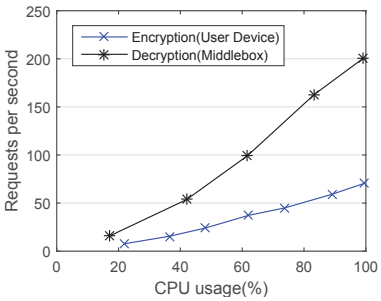


Fig. 9: Key-sharing requests per second on single core.

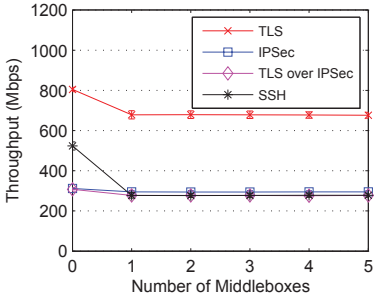


Fig. 10: PlainBox throughput v.s. number of middleboxes.

significant RTT, importing our key-sharing process doesn't increase the handshake time since there is enough time reserved. Thus we can conclude that the PlainBox key sharing generally provides an efficient handshake with a small time overhead.

B. Key Sharing Overhead

We next show the key preparing time t_{enc} and t_{dec} caused by the PlainBox. In the normal 1:1 client-middlebox key-sharing mode, the keys are encrypted and decrypted using normal public/private key cryptography. According to Crypto++ benchmarks [28], it takes several milliseconds for the RSA-based encryption and decryption which is lower than the time cost by ABE, therefore we focus on the cost of ABE encryption. Figure 6 shows the key preparing time t_{enc} and t_{dec} on TLS using CP-ABE. Each test case consists of 1,000 runs for average. Since our middlebox does not send out packets when a session key is ready, we measure the time directly in the program instead of capturing packets. Since the time complexity on ABE encryption/decryption is mainly decided by the number of attributes [10], we test the time on different numbers of attributes. From the result, we know that the encryption time is the major cost growing with the attribute numbers, while the decryption time is minor. The total key-preparing time on eight attributes is around 60ms. Comparing with the handshake time on encryption protocols, it generally causes very small delay on the whole handshake.

Figure 7 shows the total handshake size for different protocols with the increase of middleboxes in the path. It compares the handshake size between 1:1 user-middlebox key sharing and our PlainBox 1:N key sharing using ABE encryption on two attributes. In normal 1:1 key sharing, the total handshake size is a linear relation with the number of middleboxes, while in PlainBox key sharing it's constant. In addition, the overhead on total handshake packet size by ABE-based key sharing in each protocol is shown in Figure 8. Zero attributes stand for not sharing keys. Although the additional size on handshake size has a linear relation with the number of attributes in ABE encryption, it shows that in a limited number of attributes the size is still acceptable. Therefore, we can conclude that the ABE-based key-sharing in PlainBox provides high usability for user policy management while ensuring low overhead on multiple middleboxes.

C. System Performance

The computational overhead in PlainBox is mainly caused by ABE-based encryption and decryption. In each key-sharing request, the client computes encryption once and each middlebox computes decryption once. Since some protocol may support running time key re-generation, a data session may contain more than one key-sharing request. Figure 9 shows the CPU usage and the key-sharing requests per second on single CPU core. The encryption on the client is slower than decryption on middlebox. Consider that generating new sessions is not a critical task on user devices, the encryption performance is acceptable. According to NGINX's measurement on TLS server [29], a single-core server handles about 350 RSA-based requests per second. It shows that our middlebox performance has the same order and is also acceptable.

In the application data processing, our middlebox performs normal symmetric encryption/decryption which is the same as the client and server. Figure 10 shows the system throughput for different protocols on different middlebox numbers. We transmit 100MB file in each protocol test case, using HTTPS for TLS and SFTP for SSH. Each test consists of ten concurrent connections and repeats for twenty times. The result shows that although importing data decryption on the middlebox causes a small slow down on the throughput, adding more middleboxes doesn't impact the throughput, therefore our system is scalable.

VII. Related Work

Decrypting network traffic at middleboxes People have develop solutions for middlebox decryption on specific protocols. There are several proposals in the IETF on designing an explicit proxy for TLS [30], [31]. It requires modifications on the original TLS handshake and cannot support the TLS-over-other case, therefore is not a suitable solution. Without modifying the TLS protocol, transparent proxies can be used to decrypt the TLS traffic by replacing fake certificates [32]. Although such traffic has already been taking significant place in the Internet today [33], it is equivalent to a man-in-the-middle attack and may cause serious security problem.

mcTLS [2] is the first integrated design on the secure key sharing with middlebox for the TLS protocol. It provides a certificate-based authentication mechanism for clients and servers to authenticate middlebox and designs a key-sharing

Authorized licensed use limited to: Tsinghua University. Downloaded on June 15,2022 at 04:21:40 UTC from IEEE Xplore. Restrictions apply.

protocol to share session keys with middleboxes. Han et al. [34] designed a framework to use Intel SGX technology to protect middleboxes on key sharing scheme for encrypted traffic.

Computing on ciphertext For some specific functionalities (e.g., keyword searching), there have been encryption mechanisms to directly compute on the ciphertext [35]. BlindBox [1] is a system for middlebox to perform DPI over encrypted HTTPS traffic, including keyword search without decrypting the traffic and a simple method for session key sharing. Embark [36] extends BlindBox to support a wide range of middleboxes. It is a system for outsourcing middlebox to process encrypted traffic without decryption based on PrefixMatch scheme. SPABox [37] proposed a middlebox based system that supports DPI over encrypted traffic without decrypting the traffic.

VIII. Conclusion

In this paper, we propose the PlainBox architecture to enable session key sharing between the communication client and the middleboxes in the network path. It allows the user to share his session keys with trusted middleboxes using a scalable key-sharing protool on multiple middleboxes with little overhead. We implement a prototype system on several security protocols such as TLS, IPsec, SSH, and their combinations (e.g., TLS over IPsec), and supporting different applications including web browsers and sftp. Our evaluation results show that PlainBox only introduces a minor overhead and provides a good processing performance. Therefore, it is a practical and efficient solution of building generic and scalable middlebox services over encrypted protocols.

References

[1] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "BlindBox: Deep Packet Inspection over Encrypted Traffic," in *SIGCOMM '15*, 2015.

[2] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. Lopez, K. Papagiannaki, P. R. Rodriguez, and P. Steenkiste, "Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS," in *SIGCOMM '15*. ACM, 2015.

[3] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service," in *SIGCOMM '12*. ACM, 2012.

[4] J. R. Santos, Y. Turner, G. J. Janakiraman, and I. Pratt, "Bridging the Gap between Software and Hardware Techniques for I/O Virtualization." in *USENIX Annual Technical Conference*, 2008, pp. 29–42.

[5] P. Quinn and T. Nadeau, "Problem Statement for Service Function Chaining," Internet Requests for Comments, RFC 7498, April 2015.

[6] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," Internet Requests for Comments, RFC 5246, August 2008.

[7] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," Internet Requests for Comments, RFC 4251, January 2006.

[8] S. Kent and K. Seo, "Security Architecture for the Internet Protocol," Internet Requests for Comments, RFC 4301, December 2005.

[9] J. Arkko, "Tackling Internet Challenges," *ICEST 2014*, June 2014.

[10] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-Policy Attribute-Based Encryption," in *IEEE Symposium on Security and Privacy*. IEEE, 2007, pp. 321–334.

[11] C. Wang, X. Yuan, Y. Cui, and K. Ren, "Towards Secure Outsourced Middlebox Services: Practices, Challenges, and Beyond OCA," in *Accepted by IEEE Network, 201706*.

[12] C. Liu, Y. Cui, C. Zhang, and J. Wu, "Generic Application Layer Protocol Translation for IPv4/IPv6 Transition," in *IEEE ICC 2017*.

[13] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)," Internet Requests for Comments, RFC 7858, May 2016.

[14] P. Saint-Andre and T. Alkemade, "Use of Transport Layer Security (TLS) in the Extensible Messaging and Presence Protocol (XMPP)," Internet Requests for Comments, RFC 7590, June 2015.

[15] A. Melnikov, "Updated Transport Layer Security (TLS) Server Identity Check Procedure for Email-Related Protocols," Internet Requests for Comments, RFC 7817, March 2016.

[16] Intel. Intel Software Guard Extensions. [Online]. Available: https://software.intel.com/en-us/isa-extensions/intel-sgx

[17] R. Fonseca, G. M. Porter, R. H. Katz, S. Shenker, and I. Stoica, "IP Options are not an option," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2005-24, Dec 2005.

[18] A. Medina, M. Allman, and S. Floyd, "Measuring Interactions Between Transport Protocols and Middleboxes," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*.

[19] F. Gont, J. Linkova, T. Chown, and W. Liu, "Observations on the Dropping of Packets with IPv6 Extension Headers in the Real World," Internet Requests for Comments, RFC 7872, June 2016.

[20] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it Still Possible to Extend TCP?" in *Proc. IMC '11*.

[21] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP," in *NSDI '12*.

[22] H. Tang, Y. Cui, C. Guan, J. Wu, J. Weng, and K. Ren, "Enabling Ciphertext Deduplication for Secure Cloud Storage and Access Control," in *AsiaCCS 2016*.

[23] D. Thaler and C. Hopps, "Multipath Issues in Unicast and Multicast Next-Hop Selection," Internet Requests for Comments, RFC 2991, November 2000.

[24] The Nielsen Company. (2010) TOP U.S. WEB BRANDS AND SITE USAGE DECEMBER 2009. [Online]. Available: http://www.nielsen.com/us/en/insights/news/2010/top-u-s-web-brands-and-site-usage-december-2009.html

[25] Microsoft. Connectivity Enhancements in Windows Internet Explorer 8. [Online]. Available: https://msdn.microsoft.com/en-us/library/cc304129(v=vs.85).aspx

[26] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)," Internet Requests for Comments, RFC 3315, July 2003.

[27] S. Krishnan, T. Mrugalski, and S. Jiang, "Privacy Considerations for DHCPv6," Internet Requests for Comments, RFC 7824, May 2016.

[28] W. Dai, "Crypto++ 5.6.0 benchmarks," *Website at http://www.cryptopp.com/benchmarks.html*, 2009.

[29] NGINX. (2014) NGINX SSL Performance. [Online]. Available: https://www.nginx.com/wp-content/uploads/2014/07/NGINX-SSL-Performance.pdf

[30] D. McGrew, D. Wing, Y. Nir, and P. Gladstone, "TLS Proxy Server Extension," IETF, Internet-Draft draft-mcgrew-tls-proxy-server-01, July 2012.

[31] Y. Nir, "A Method for Sharing Record Protocol Keys with a Middlebox in TLS," IETF, Internet-Draft draft-nir-tls-keyshare-02, March 2012.

[32] A. Kingsley-Hughes, "Gogo in-flight Wi-Fi serving spoofed SSL certificates," in *ZDNet*, 2015.

[33] M. O'Neill, S. Ruoti, K. Seamons, and D. Zappala, "TLS Proxies: Friend or Foe?" *IMC 2016*.

[34] J. Han, S. Kim, J. Ha, and D. Han, "SGX-Box: Enabling Visibility on Encrypted Traffic using a Secure Middlebox Module," in *Proceedings of the First Asia-Pacific Workshop on Networking*. ACM, 2017.

[35] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic Searchable Symmetric Encryption," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 965–976.

[36] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, "Embark: Securely Outsourcing Middleboxes to the Cloud," in *NSDI '16*.

[37] J. Fan, C. Guan, K. Ren, Y. Cui, and C. Qiao, "SPABox: Safeguarding Privacy During Deep Packet Inspection at a MiddleBox," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3753–3766, 2017.

Authorized licensed use limited to: Tsinghua University. Downloaded on June 15,2022 at 04:21:40 UTC from IEEE Xplore. Restrictions apply.