mdTLS: How to make middlebox-aware TLS more efficient?

Taehyun Ahn[0009−0007−2339−286X], Jiwon Kwak[0009−0008−1560−7442], and Seungjoo Kim*[0000−0002−2157−0403]

School of Cybersecurity, Korea University, Seoul 02841, South Korea
{thyun_ahn,jwkwak4031,skim71}@korea.ac.kr

Abstract. Recently, many organizations have been installing middleboxes in their networks in large numbers to provide various services to their customers. Although middleboxes have the advantage of not being dependent on specific hardware and being able to provide a variety of services, they can become a new attack target for hackers. Therefore, many researchers have proposed security-enchanced TLS protocols, but their results have some limitations. In this paper, we proposed a middlebox-delegated TLS (mdTLS) protocol that not only achieves the same security level but also requires relatively less computation compared to recent research results. mdTLS is a TLS protocol designed based on the proxy signature scheme, which requires about 39% less computation than middlebox-aware TLS (maTLS), which is the best in security and performance among existing research results. In order to substantiate the enhanced security of mdTLS, we conducted a formal verification using the Tamarin. Our verification demonstrates that mdTLS not only satisfies the security properties set forth by maTLS but also complies with the essential security properties required for proxy signature scheme.[1]

Keywords: maTLS · Middlebox · Proxy signature · Formal verification

1 Introduction

The advent of the COVID-19 pandemic has instigated substantial transformations in the business landscape. Notably, a significant proportion of enterprises have transitioned from conventional in-office working arrangements to facilitating remote work options for their workforce. Concurrently, the pandemic has spurred innovative shifts in operational methodologies, exemplified by the substitution of face-to-face business procedures, historically reliant on in-person meetings, with video conferencing solutions. As a result of these shifts, there has been a discernible escalation in network traffic, with notable statistics from the Telegraph indicating a remarkable 47% surge in internet traffic between 2019 and 2020 [28].

* Corresponding Author

[1] All of the formal models and lemmas are open to the public through the following url https://github.com/HackProof/mdTLS

Especially during the COVID-19 pandemic, the security of confidential information of various companies and individuals has been emphasized as most social activities, including business, are conducted remotely over the network. Among the most prominent and widely adopted technologies addressing network security concerns during this period is HTTPS (HyperText Transfer Protocol Secure) [36].

HTTPS represents a communication protocol that integrates the HTTP (HyperText Transfer Protocol) [13] to the TLS (Transport Layer Security) protocol [10], with the overarching objective of ensuring the confidentiality and integrity of data transmitted over networks. This protocol finds utility not only in desktops but extends its application domain to encompass a diverse array of embedded devices, including IoT (Internet of Things) devices. HTTPS offers several fundamental security attributes, including the following:

- *Encryption:* It serves as a pivotal mechanism within HTTPS, facilitating the obfuscation of sensitive information by encoding the data exchanged between communicating entities. Commonly employed encryption algorithms encompass symmetric key algorithms like Advanced Encryption Standard (AES) [19].
- *Authentication:* It constitutes an integral component of HTTPS, operating to ascertain the identity of entities by utilizing digital certificates.
- *Integrity:* It is another crucial facet of HTTPS, operating as a mechanism to detect unauthorized tampering or forgery of messages. Conventional algorithms used to maintain message integrity involve the implementation of Message Authentication Codes (MACs), such as the Secure Hash Algorithm (SHA) [9], to uphold the veracity and unaltered state of a network connection.

According to the Google transparency report, there has been a consistent increase in the loading speed of HTTPS pages in the chrome browser since 2014 [17]. Moreover, among the top 100 non-Google websites on the internet, which collectively constitute approximately 25% of global website traffic, 96 websites have embraced HTTPS, with 90 of them making HTTPS their default protocol. Additionally, according to Gartner's article [34], edge computing technology is anticipated to evolve into a core IT technology. This technology facilitates the secure communications of data collected through embedded systems deployed across various domains, relying on TLS protocols. Consequently, TLS communication is expected to assume an increasingly pivotal role. However, the robust encryption mechanisms employed by TLS to protect data can also be exploited by attackers to hide malware within network traffic, thereby evading detection by conventional security measures. In fact, according to research by Cisco and Sophos, TLS is vulnerable to detecting malicious traffic, and the number of such cases continues to increase [5, 14]. As a result, TLS cannot be considered a complete solution against cybersecurity threats.

For this reason, numerous organizations have deployed specialized middleboxes with distinct functionalities designed to enhance security for their clients, such as firewall and intrusion detection [39]. For instance, some companies have integrated Transport Layer Security Inspection (TLSI) [30] capabilities into middleboxes to identify and intercept malicious traffic attempting to infiltrate their internal networks. TLSI represents a technology devised to thwart unauthorized actions perpetrated by hackers on encrypted network traffic, and numerous entities, including industry giants such as Microsoft, are actively leveraging this technology [27].

However, according to a survey conducted in the United States, more than 70% of employees still believe that hackers can exploit middleboxes. Also, 50% of the respondents answered that their personal information could be infringed by exploiting vulnerabilities in the middleboxes [33]. Ironically, middleboxes, initially installed to fortify data security within TLS communications, have emerged as potential targets for cyberattacks. Consequently, safeguarding data transmitted over TLS communications necessitates a holistic approach considering network components, such as middleboxes, from the inception of communication channel construction. This approach goes beyond simply installing security-hardened components into an existing network.

As a consequence, numerous researchers have proposed a range of TLS extension protocols to enhance security during communication via the TLS protocol. However, prior research endeavors, driven primarily by a pursuit of security, have inadvertently encountered performance-related challenges. In this study, we will introduce the mdTLS protocol, which is meticulously designed based on the proxy signature scheme. The mdTLS is subject to comparative evaluation against maTLS [24], widely recognized as the most exemplary among prior researches in terms of both security and performance. First, we investigated the amount of arithmetic operations that must be performed for each designed protocol to compare the performance of the mdTLS and maTLS protocols. We then formally verified that the mdTLS satisfies not only the security properties verified in maTLS, but also three other security properties related to the proxy signature scheme. To ensure methodological consistency in our experimental setup, we employed the Tamarin [26, 37, 40], utilized in prior maTLS research, during the security analysis.

The remainder of the paper is organized as follows. First, we analyzed the strengths and weaknesses of related works (Section 2). Next, we introduced our mdTLS protocol (Section 3). After that, we compare the performance between maTLS and mdTLS (Section 4). In Section 5, we verified our protocol using Tamarin (Section 5). We showed that the performance can be further improved when the Schnorr digital signature is used in the protocol (Section 6). Finally, we present our concluding remarks (Section 7).

2 Related works

Many researches have been conducted to improve TLS protocol. They are categorized into two types. One is the TLS-encryption extension-based approach. Their research is to improve the mechanism itself inside the protocol. The other one is the Trusted Execution Environment (TEE) based approach. Their research is to improve the protocol by using specific hardware.

2.1 TEE based approaches

A typical example of the Trusted Execution Environment (TEE) based approach is SGX-Box [18]. It utilized the remote attestation of Intel SGX. The server performs remote attestation to verify the integrity of the SGX-Box module in middleboxes. If remote attestation succeeds, they create a secure channel to prevent sensitive information from leaking between them. However, it is limited in that it is too dependent on its specific hardware (Intel SGX). Besides SGX-Box, there are many researches such as STYX [42], EndBox [16], and ShieldBox [41]. However, they also had the same limitations mentioned above.

2.2 TLS-extension based approaches

A typical example of the TLS-extension approach is SplitTLS [20]. In SplitTLS, middleboxes act as servers and clients at the same time. This feature gives them too many privileges. It can cause some security incidents. For example, middleboxes such as CDN service providers could receive the private key to act as a server. It accidentally exposes the private key during the key-exchange phase. The worst thing is that when the middleboxes become compromised, malicious users (attackers) could abuse their privileges. Unlike SplitTLS, mcTLS [32] provides the least privilege to middleboxes. Middleboxes can read or write the TLS payload by obtaining MAC key pair from each endpoint. For example, they can only read the TLS packets when they get a unique key for reading. The advantage of mcTLS is that it does not force middleboxes to create or install further objects. Since the mcTLS uses only one key when creating a session, it is considered insecure. In the performance view, it has a limitation in that additional latency occurs when establishing the first connection. Furthermore, it does not follow TLS standards. David Naylor, who had proposed mcTLS, proposed an extended version of mcTLS called mbTLS [31]. mbTLS was created to improve compatibility with TLS standards. mbTLS establishes two types of sessions. One is the mbTLS session, and the other is the standard TLS session. If one of the endpoints does not use mbTLS, then traditional TLS sessions are activated. Overall, mbTLS offers improvements over mcTLS, which causes latency when adding a secondary session. maTLS [24] is another extended protocol to address security issues in SplitTLS. It treats middleboxes as equivalent entities to the server and includes them in the TLS session. As the server's certificate, middleboxes' certificates are issued by the Certificate Authority (CA), and by

introducing the Middlebox Transparency (MT) log server, the middleboxes certificate contains a Signed Certificate Timestamp (SCT) [2, 23]. This guarantees middleboxes' audition and improves the reliability of the middleboxes' certificates. Also, unlike SplitTLS, this procedure shows middleboxes can create their own official certificates without using custom root certificates or server certificates. However, these security elements entail performance issues. To make every session in each section, maTLS handshakes are essential between every entity. This is why maTLS's initial handshake takes more time than the original version of TLS.

3 mdTLS: middlebox-delegated TLS protocol with proxy signature scheme

In this section, we described the mdTLS protocol. At first, we defined the adversary model and security goals related to the mdTLS. After that, we described each phase in the protocol in detail.

3.1 Adversary model

We considered the attacker's capability under the Dolev-Yao model [11]. Attackers can obtain and analyze messages in the network. Furthermore, they can get public keys. They aim to obtain certificates, perform an impersonation attack via forged certificates, and reveal private keys.

3.2 Security goal

TLS currently provides the following properties in multi-party cases. Among them, we define "secure" for mdTLS by extending three security properties to cover the "delegation" concept.

Authentication: The notion of authentication was defined as that every entity must be able to verify whether they are talking to the "right person". This goal was divided into two sub-goals. First, each entity(client or server) can verify whether the other endpoint is operated by the expected middleboxes. It is called *entity authentication*. Second, If a session between two endpoints consists of an ordered set of middleboxes MB_1 ... MB_{n-1}, then any data received by MB_j must be a prefix of the data sent by MB_{j-1} or MB_{j+1}, where $1 < j < n-1$. It is called *data authentication*. We refined *entity authentication* into two security goals. First, the client ensures the delegated middleboxes by verifying the warrant in signature. It is called *verifiability*. Second, each middlebox can be identified as an appropriately delegated middlebox by checking its public key from the proxy signature. It is called *strong-identifiability*.

Secrecy: The notion of secrecy can be defined as that adversaries should learn nothing more from observing ciphertext in network connections. This goal is divided into two sub-goals. First, each mdTLS segment sent from entities should be encrypted with a strong ciphersuite. It is called *segment secrecy*. Second, each segment should have its own security parameters, such as a unique session key, to prevent the data from being reused. It is called *individual secrecy*.

Integrity: The notion of integrity means that only authorized or delegated entities can make or modify messages under their permissions. This goal is divided into two sub-goals. First, the entity can confirm which middleboxes have made each modification to the message. It is called *modification accountability*. Second, endpoints can determine the list and order of middleboxes that messages pass through. It is called *path integrity*. In mdTLS, we defined one security goal additionally. Delegated middleboxes can generate valid signatures. It means, in converse, undelegated entities cannot modify messages because they cannot generate and verify the signatures. Hence, it is called *strong-unforgeability*.

3.3 Overview of mdTLS protocol

The mdTLS applies a proxy signature scheme based on the *partial delegation with warrant* [6, 22, 25] to improve performance while having the same security level as maTLS.

Proxy signature scheme [25] is a technique in which a proxy signer electronically signs on behalf of the original signer. When the original signer is temporarily absent, a proxy signer receives signature authority from the original signer and performs the proxy signing. This signing authority delegation technique can be used in various distributed systems, such as edge computing. There are four types of delegation in the proxy signature scheme: *full delegation*, *partial delegation*, *delegation by warrant*, and *partial delegation with warrant* [22, 25].

- *Full delegation:* The proxy signer uses the original signer's private key to generate the proxy signature.
- *Partial delegation:* This method generates a proxy signing key using the private keys of both the original and the proxy signers. The advantage is that it can prevent the original signer from arbitrarily proxy signing, but there is no way to revoke or limit proxy signing authority.
- *Delegation by warrant:* This method uses a warrant that specifies the proxy delegation period and message space to limit proxy signing authority. It can compensate for the shortcomings of partial delegation, but performance in verification deteriorates because the verifier must additionally verify the warrant when verifying the proxy signature.
- *Partial delegation with warrant:* Kim et al. [22] first introduced this type of delegation. This method utilizes the advantages of both *partial delegation* and *delegation by warrant*. Proxy signing authority can be restricted or revoked through a warrant. Additionally, since this method only verifies the proxy signature, the verification efficiency can be improved.

The details of the mdTLS are shown in Figure 1, 2. For reader's convenience, notation definitions are listed in Table 1. mdTLS is divided into 3 phases.

- *Generating certificates phase:* Before negotiation, server certificates are generated.
- *Handshake phase:* Negotiation between two endpoints on a network – such as a client and a server – to establish the details of their connection. During handshake, ECDH and ECDSA [21, 29] are used in key exchange and digital signature, respectively.
- *Record phase:* Data communications are encrypted between the two entities.

The following statements below Table 1 are detailed sequences in which each entity establishes a secure communication channel based on the mdTLS.

Table 1: Notations in mdTLS

	Notation	Meaning
Entities	C	Client
	S	Server
	MB_i	i-th middlebox ($0 < i < n$)
	e_i	i-th entity (e_0: client, e_n: server)
ECDH	$(d_{e_i}^{ex}, Q_{e_i}^{ex})$	e_i's ECDH key pairs
ECDSA	p	A prime number
	E	An elliptic curve on $\mathbb{F}_p$
	q	A field size (prime number)
	G	A base point on E having prime order q
	d_{e_i}	A private key with $0 < de_i < q$
	Q_{e_i}	A public key with $d_{e_i} \cdot G$ on E
	H	Cryptographic hash function ($\{0,1\}^* \rightarrow \mathbb{F}_q$)
	$S^H(d_{e_i}, m)$	Sign message m with private key d_{e_i} using H
	$V^H(Q_{e_i}, m, \sigma)$	Verify signature σ generated by $S^H(d_{e_i}, m)$
Proxy-signature	$PS(skp, m)$	Proxy signing the message m with proxy signing key skp
	$PV(Q_{e_i}, m, \sigma_p)$	Proxy verification for proxy signature σ_p, with Q_{e_i}

Phase 0. Generating certificates

1. Server sends Certificate Signing Request (CSR) to Certificate Authority (CA).
2. CA verifies CSR, creates pre-certificates, and submits to the Certificate Transparency (CT) log server to get SCTs [2].
3. After the CT log server adds pre-certificates to the logs, it returns SCTs to CA. Due to the Certificate Transparency policy [2, 23], at least 2 SCTs from different CT log servers are required for certificates.
4. Using the X.509 v3 [7] extension, CA attaches SCTs to the certificate and issues the certificate to the server.

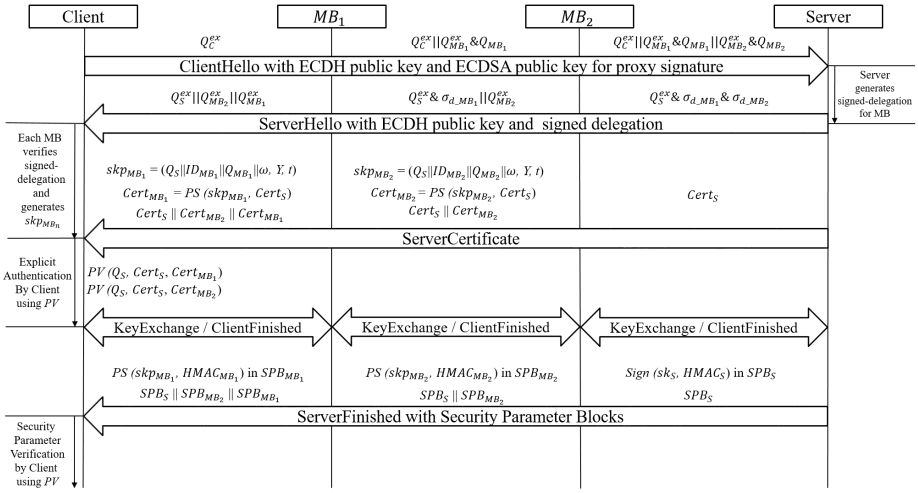


Fig. 1: Handshake phase of mdTLS

Phase 1. Handshake

1. Client generates ECDH key pair, and the public key Q_C^{ex} will be sent by ClientHello message.
2. Middleboxes attach their two types of keys to the ClientHello message. One is ECDH public key, $Q_{MB_i}^{ex}$, and the other is ECDSA public key, Q_{MB_i}, which will be used in the proxy signature scheme.
3. Server, the original signer, also creates its ECDH and ECDSA key pairs as middleboxes. When the server receives a ClientHello message, it operates the designation process to delegate middleboxes as proxy signers. Outputs of this process are called signed delegations $\sigma_{d\_MB_i}$. For delegation, the server has to sign the hash value of the delegation message. This message consists of Q_S, the identity of proxy signer ID_{MB_i}, Q_{MB_i}, and a warrant ω containing the message space and delegation period. In addition, 0 is prepended to represent that it is for the proxy signature scheme. $\sigma_{d\_MB_i}$ can be represented as (x_{Y_d}, s_d) according to ECDSA form. Signed delegations will be sent by ServerHello message with Q_S^{ex}.
   - $\sigma_{d\_MB_i} \leftarrow S^H(d_S, 0||Q_S||ID_{MB_i}||Q_{MB_i}||\omega)$
     - random value y_d ($0 < y_d < q$)
     - $Y_d \leftarrow y_d \cdot G$
     - $x_{Y_d} \leftarrow$ x-coordinate of Y_d
     - $c \leftarrow H(m_d)$ $(m_d = 0||Q_S||ID_{MB_i}||Q_{MB_i}||\omega)$
     - $s_d \leftarrow (c + d_S \cdot x_{Y_d}) \cdot y_d^{-1} \bmod q$
     - $\therefore \sigma_{d\_MB_i} = (x_{Y_d}, s_d) =$ signed delegation
4. Middleboxes attach their own ECDH public key $Q_{MB_i}^{ex}$ to the ServerHello message. Then, middleboxes check whether signed delegations from the server are valid. If validation succeeds, middleboxes generate their proxy signing key skp_{MB_i}.

- $skp_{MB_i} \leftarrow (Q_S||ID_{MB_i}||Q_{MB_i}||\omega, x_{Y_d}, t)$
  - $c \leftarrow H(m_d)$ $(m_d = 0||Q_S||ID_{MB_i}||Q_{MB_i}||\omega)$
  - $r \leftarrow H(Q_S||ID_{MB_i}||Q_{MB_i}||\omega||c)$
  - $t \leftarrow r + d_{MB_i} \cdot H(Y_d||\omega) \bmod q$
    - $Y_d \leftarrow y_d \cdot G = s_d^{-1} \cdot (c + d_S \cdot x_{Y_d}) \cdot G$

5. Due to the ServerCertificate message, the server sends its certificate $Cert_S$ to the client and middleboxes. Middleboxes generate their own certificates $Cert_{MB_i}$ by proxy signing the received server's certificate. Then, their certificates are sent to the client by appending to the ServerCertificate message.
   - $PS(skp_{MB_i}, Cert_S)$ returns $Cert_{MB_i}$, which can be shown as below:
     - $(ID_{MB_i}, Q_{MB_i}, \omega, (x_{Y_d}, s_d), S^H(t, 0||Cert_S||Q_S||ID_{MB_i}||Q_{MB_i}||\omega||x_{Y_d}||s_d||r))$
       - $(x_{Y_p}, s_p) \leftarrow S^H(t, 0||Cert_S||Q_S||ID_{MB_i}||Q_{MB_i}||\omega||x_{Y_d}||s_d||r)$
6. The client, a verifier, verifies certificates to authenticate entities in TLS session. Unlike $Cert_S$, the client has to use proxy verification, PV, to verify $Cert_{MB_i}$, which requires the client to generate proxy public keys PKP_{MB_i} corresponding to each middleboxes. With PKP_{MB_i}, the client verifies $Cert_{MB_i}$.
   - $PV(Q_S, Cert_S, Cert_{MB_i})$
     - $Cert_{MB_i} \leftarrow (ID_{MB_i}, Q_{MB_i}, \omega, (x_{Y_d}, s_d), (x_{Y_p}, s_p))$
     - If $Cert_S \notin \omega$ then *return false*;
     - Else $PKP_{MB_i} \leftarrow r \cdot G + H(s_d^{-1} \cdot (c \cdot G + x_{Y_d} \cdot Q_S)||\omega) \cdot Q_{MB_i}$;
       - $c \leftarrow H(0||Q_S||ID_{MB_i}||Q_{MB_i}||\omega), r \leftarrow H(Q_S||ID_{MB_i}||Q_{MB_i}||\omega||c)$
     - $V^H(PKP_{MB_i}, 0||Cert_S||Q_S||ID_{MB_i}||Q_{MB_i}||\omega||x_{Y_d}||s_d||r, (x_{Y_p}, s_p))$
7. Server sends ServerFinished message with security parameter block (SPB). These blocks consist of signatures of $HMAC$. This $HMAC$ generates authentication code from security parameters such as ciphersuite and handshake messages. For middleboxes, they have to proxy sign their blocks with their generated skp_{MB_i}. For a client, it must verify middleboxes' signed blocks with its generated proxy public keys PKP_{MB_i}.

Phase 2. Record

- Modification log is attached to the message and helps to check whether a message is modified. Besides, endpoints can also check whether unauthorized entities modify messages without permission.

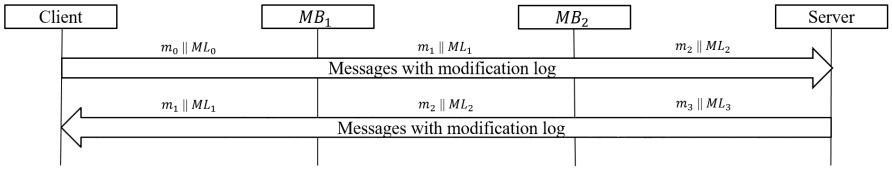


Fig. 2: Record phase of mdTLS

4 Performance analysis for mdTLS

In this section, we analyzed the performance of the mdTLS by conducting a comparative analysis with maTLS, which we consider to be among the best of the existing TLS-extension protocols. Our performance analysis is focused on the number of computations in protocols. Both mdTLS and maTLS rely on ECDSA for the generation of security parameters. ECDSA, being based on the Elliptic Curve Discrete Logarithm Problem (ECDLP), involves a substantial number of point multiplication operations. These operations can significantly influence the performance of both protocols. Therefore, we conducted a performance analysis employing algorithms capable of measuring the number of point multiplication operations. It is important to note that this analysis is based on server-only authenticated TLS version 1.2 and assumes that 3 SCTs are created for each certificate through the Certificate Transparency policy [1–3, 23].

4.1 Preliminaries for performance analysis

To facilitate performance comparisons between two protocols that offer the same 128-bit security strength, we have set the elements within the protocols, as shown below [12].

- Types of elliptic curve: Secp256r1
- Private key size: 256 bits
- Hash size: 256 bits

4.2 Analyzing the performance between maTLS and mdTLS

To measure the number of point multiplication operations, we employed the double-and-add algorithm, which averages 1 point doubling and 0.5 point additions per bit. Therefore, we considered an average of 1.5 point multiplication operations per bit. Following this, we divided the protocol into two segments and measured the number of point multiplication operations. The first segment corresponds to the generation and verification of certificates for utilization in the handshake phase. The number of computations for each protocol in this segment is detailed in Table 3 and 4 below. The second segment is where entities (server, client, middlebox) create and verify security parameters to be exchanged at the handshake phase. The number of computations for each protocol in this segment is detailed in Table 2 below.

Table 2: Computational analysis for security parameter blocks

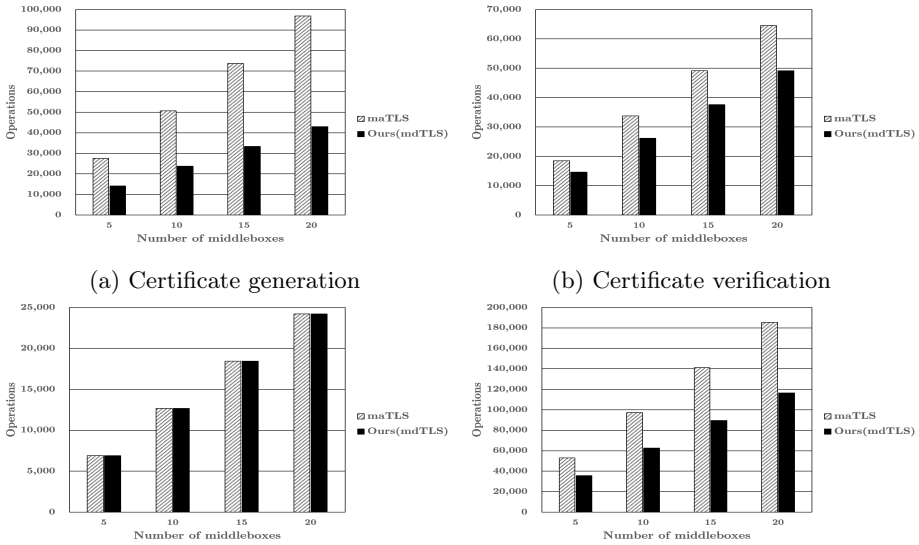
Descriptions	maTLS	mdTLS
Server generates security parameter blocks.	384	384
Middlebox generates security parameter blocks.	384N	384N
Client verifies blocks from the server.	768	768
Client verifies blocks from the middleboxes.	768N	768N

Table 3: Computational analysis for generating certificates

Descriptions	maTLS	mdTLS
- Server side		
Server generates keys and signature for CSR to CA.	768	768
CA verifies CSR signature.	768	768
CT log servers generate keys and signatures for 3 SCTs.	2,304	2,304
CA generates keys and signs for server's certificate.	768	768
- Middlebox side for maTLS		
Middleboxes generate keys and signature for CSR to CA.	768N	-
CA verifies CSR signature.	768N	-
MT log servers generate keys and signatures for 3 SCTs.	2,304N	-
CA generates keys and signs for middleboxes' certificate.	768N	-
- Middlebox side for mdTLS		
Each middlebox generates its keys.	-	384N
Server generates signed delegations to assign proxy signers.	-	384N
Middlebox verifies signed delegation and generate proxy signing key.	-	768N
Middleboxes generate certificates with proxy signing key.	-	384N

Table 4: Computational analysis for certificates verification

Descriptions	maTLS	mdTLS
Client verifies the signature and 3 SCTs in the server's certificate.	3,072	3,072
Client verifies the middleboxes' certificates.	3,072N	2,304N



(a) Certificate generation

(b) Certificate verification

(c) Security parameter blocks

(d) Overall performances

Fig. 3: Performance of protocols when using ECDSA

We have implemented certain components essential for the functionality of mdTLS. We mainly implemented internal functions for computing data required during the handshake phase, such as key or signature generation and verification. We implemented and analyzed its performance within a virtual environment, specifically using docker container. The rest of our testbed in docker image is as follows:

- Ubuntu 22.04.3 LTS
- Intel(R) Core(TM) i5-10400 CPU @ 2.90GHz
- 2GiB RAM

Table 5: Average execution time in implementation

Features	maTLS	mdTLS
ECDSA signing	1.4ms	1.4ms
ECDSA verification	2.5ms	2.5ms
Proxy signing	-	1.6ms
Proxy verification	-	8.9ms

Table 5 shows the time spent when signing and verifying the CSR files. Since the proxy signature scheme requires additional keys, the execution time of mdTLS is longer than maTLS. However, by reusing these keys when processing the security parameter block, the execution time of mdTLS can become similar to maTLS.

5 Security analysis for mdTLS

In this section, we conducted a security analysis of mdTLS using an approach similar to the one employed for maTLS [24], involving formal specification and verification through the Tamarin [40]. Tamarin is an automated formal verification tool based on multiset rewriting rules in the theory of equations. It has been continuously updated to maintain its effectiveness. Using this tool, maTLS formally verified six security lemmas: *server authentication*, *middlebox authentication*, *data authentication*, *path integrity*, *path secrecy*, and *modification accountability*. In the case of mdTLS, we successfully verified not only the same lemmas as previously done in maTLS but also three novel lemmas related to the proxy signature scheme following the same approach and tools: *verifiability*, *strong-unforgeability*, and *strong-identifiability*. However, in this paper, we only described three novel lemmas related to the proxy signature scheme, taking into consideration the maximum page limit imposed by the conference guidelines. The rest can be found on our GitHub [4].

5.1 Experimental setup

To analyze the security of the mdTLS, we established an experimental environment, as illustrated below. Our goal was to confirm that the formal model of mdTLS aligns with the security lemmas within our testing environment.

- Amazon Elastic Compute Cloud (Amazon EC2) c5a.24xlarge instance
- 96 vCPUs, 192 GiB RAM
- Ubuntu 22.04.2 LTS

5.2 Formal specification

We have formalized the mdTLS, specifying the detailed operations conducted by each entity during the handshake and record phases in the form of rules. For cryptographic primitives like hash, signature, and PRF (Pseudo-Random Function) [15], we used the built-in functions provided by Tamarin. Details of all rules can be found in the spthy file uploaded to our github [4]. The script below illustrates an example of the detailed operations concerning ServerHello messages. In the handshake phase, when the server receives a ClientHello message from the client, it responds by sending a ServerHello message to initiate mutual authentication. In this process, mdTLS sends a ServerKeyExchange message, a signed delegation, a Diffie-Hellman public key, and a ServerCertificate message. The delegation in this context consists of the server's public key, the middlebox's public key and identification information, and a warrant providing an explanation of the delegation.

```
rule Server_Hello:
  let
        server_hello_msg
            = < 'server_hello', ~ns, server_chosen_details >
                        ...
        server_key_exchange = s_dhe_pub
        server_key_exchange_signed
            = < server_key_exchange, sign(h(server_key_exchange)
                , ltk) >
        server_cert = < $S, pk(ltk) >
        warrant = ~warrant_fresh
        proxy_delegation = < pk(ltk), $M, mb_pubkey, warrant >
        proxy_delegation_signed = sign(h(proxy_delegation), ltk)

        Y_d = calcY_d(~y, 'G_skp')
        y_d_x = pointx(Y_d)
        c = h(proxy_delegation)
        s_d = multp( plus(multp(ltk, y_d_x), c), inv(~y) )
        proxy_delegation_signed_pair
         = < proxy_delegation, proxy_delegation_signed, <y_d_x, s_d> >
  in
    [   In( <mb_client_hello_msg, c_mb_extension> )
        , !PrivateKey('server', $S, ltk)    ]
  --[
        ServerSendDelegation(ltk_pub, mb_pubkey, warrant, proxy_delegation)
    ]->
    [   Out( <server_hello_msg, server_key_exchange_signed
              , proxy_delegation_signed_pair, s_extension
                , server_cert> ) ...    ]
```

5.3 Formal verification

A Tamarin-based formal model is a set of multiple rules, and these single rules are made up of three basic components. Facts represent detailed information about the current execution in the model. States are multisets of facts. During formal verification, user-defined functions called rules can add or remove facts from the state. This is often denoted as l →[a] →r, indicating that fact "l" is removed from the state and replaced by fact "r," with this process traced through the action denoted as "a." Tamarin, following these principles, can verify whether a lemma, which is desired to be satisfied throughout the protocol, holds even as the state changes in the operation. Tamarin's verification process is based on tracing the protocol's state through actions. To evaluate the security of our protocol, we defined nine security lemmas and one source lemma. Among them, security lemmas consist of six security lemmas of maTLS and three security lemmas related to the proxy signature scheme. As previously noted, we described three security lemmas associated with proxy signatures. Prior to describing them, we described an additional description of a source lemma designed to assist Tamarin in accurately verifying the formal specifications.

Source lemma. A source lemma is a concept used for formally verifying the security lemmas that a security protocol must adhere to during its execution. When conducting formal verification of an overall protocol, Tamarin adopts a strategy of deconstructing the protocol into smaller, more manageable components for analysis. The verification outcomes for these individual subsets are then used as supporting evidence to confirm that the entire protocol operates correctly and meets its prescribed security lemmas. However, during the verification process of these subsets, if Tamarin encounters difficulties in distinguishing between variables as nonce values or ciphertexts, it may face challenges in completing the verification. This is commonly referred to as a "partial deconstruction". To address such issues, it becomes necessary to establish a source lemma that precisely specifies the origin of these variables. From this source lemma, a refined source is generated, comprising a new set of sources. All security lemmas are subsequently verified using these refined sources, underscoring the importance of validating the source lemma to ensure the accurate computation of these refined sources [8, 40]. When we initially omitted the definition of source lemmas, the formally specified mdTLS model yielded 120 partial deconstructions. Consequently, we defined source lemmas to enable Tamarin to discern the origins of these problematic variables. Upon closer analysis, it was determined that the issue of partial deconstruction occurred in 4 distinct segments, one of which pertained to the scenario where a middlebox received an encrypted request message sent by the client. To resolve this particular issue, we formulated a source lemma indicating that the encrypted message `enc` received by the middlebox had been transmitted from the client through the `OutClientRequest()` action, as shown below. By employing this approach, we could generate refined sources in a state of "deconstructions completed". This strategic use of source lemmas proved in-

strumental in addressing the partial deconstruction challenge and facilitating the successful verification process within the mdTLS model.

```
All enc msg #i.
      InMbClientRequest( enc, msg ) @ i
      ==> (Ex #j. KU(msg) @ j & j < i)
           | (Ex #j. OutClientRequest( enc ) @ j & j<i)
```

Security lemma. After resolving the partial deconstruction issue, we verified that our protocol meets the nine security lemmas outlined in Section 3.2. In this section, we define three of the nine security goals related to proxy signature scheme. We also defined detailed information about the formulas that convert informal definitions into mathematical formulas called lemmas.

- *Verifiability:* The client must verify whether the middlebox's certificate, the proxy signature, was created with the consent of the server. To verify this lemma, we have to check whether the middlebox generated its certificate based on delegation and warrant sent by the server through the ServerHello message, as specified in *rule Server_Hello*.

```
All warrant mbLtk mbCert #tc.
    ClientReceivedProxySign(warrant, pk(mbLtk), mbCert) @tc
    ==> Ex delegation gy #tmb.
        MbGenerateProxySign(delegation, mbLtk, gy, warrant, mbCert)
        @tmb & KU(gy) @tmb & not(Ex #tmb. KU(mbLtk) @tmb)
        ==> Ex sPub #ts.
            ServerSendDelegation(sPub, pk(mbLtk), warrant, delegation)
            @ts & (#ts < #tmb) & KU(sPub) @ts
```

- *Strong-unforgeability:* The proxy signer's private key, which is used to generate the proxy signature, must not be revealed. Otherwise, the proxy signature can be forged by an adversary.

```
All warrant mbLtk mbCert #tc.
    ClientReceivedProxySign(warrant, pk(mbLtk), mbCert) @tc
    ==> All delegation gy sPub #tmb.
        (MbGenerateProxySign(delegation, mbLtk, gy, warrant, mbCert) @tmb
         & KU(gy)@tmb & not(Ex #tmb.KU(mbLtk) @tmb))
        & (MbReceiveProxyDelegation(sPub, pk(mbLtk), delegation) @tmb)
        ==> All #ts.
            ServerSendDelegation(sPub, pk(mbLtk), warrant, delegation)@ts & KU(sPub)@ts
            ==> Ex #tmbclient. MbSendPublicKey(pk(mbLtk)) @tmbclient
                & KU(pk(mbLtk)) @tmbclient
```

- *Strong-identifiability:* The identification of a proxy signer can be proved by its public key. The public key of the middlebox included in the proxy signature sent to the client must be the same as the public key of the middlebox sent to the server for proxy delegation.

```
All warrant mbPub mbCert #tc.
    ClientReceivedProxySign(warrant, mbPub, mbCert)@tc
    ==> All delegation mbLtk gy sPub #tmb.
        (MbGenerateProxySign(delegation, mbLtk, gy, warrant, mbCert)
          @tmb & KU(gy)@tmb & not(Ex #tmb. KU(mbLtk) @tmb))
        & (MbReceiveProxyDelegation(sPub, pk(mbLtk), delegation) @tmb)
        ==> All #ts.
            ServerSendDelegation(sPub, pk(mbLtk), warrant, delegation)
            @ts & KU(sPub)@ts
            ==> Ex #tmbclient. MbSendPublicKey(pk(mbLtk)) @tmbclient
                & KU(pk(mbLtk)) @tmbclient & (mbPub = pk(mbLtk))
```

Results of verification The overall result of formal verification is shown in Figure 4. Figure 4 illustrates that our mdTLS protocol not only satisfies the three security lemmas introduced above but also aligns with the lemmas validated for maTLS. Furthermore, Figure 5 shows mathematical proofs (verification process) demonstrating the consistent validity of the *verifiability* lemma within our mdTLS protocol among the security lemmas outlined in Figure 4.

```
/* All well-formedness checks were successful. */

end

==============================================================================
summary of summaries:

analyzed: mdTLS_ecdsa.spthy

  source_lemma (all-traces): verified (5660 steps)
  server_authentication (all-traces): verified (10 steps)
  middlebox_authentication (all-traces): verified (12 steps)
  middlebox_path_integrity (all-traces): verified (8 steps)
  path_secrecy (all-traces): verified (2 steps)
  modification_accountability (all-traces): verified (6 steps)
  data_authentication (all-traces): verified (2 steps)
  proxy_verifiability (all-traces): verified (10 steps)
  proxy_strong_unforgeability (all-traces): verified (12 steps)
  proxy_strong_identifiability (all-traces): verified (12 steps)

==============================================================================
```

Fig. 4: Overview of formal verification results

As mentioned earlier, Tamarin formally verifies whether the rules always satisfy the lemma, called validity. A typical approach to verifying validity is negating the formulas and checking for inconsistencies. Figure 5 shows the negated lemma for verifiability, followed by verifying whether this formulation leads to contradictions. Following this process, we have validated all six security lemmas mentioned earlier.

```
lemma proxy_verifiability:
  all-traces
  "(∀ warrant mbLtk mbCert #tc.
          (ClientReceivedProxySign( warrant, pk(mbLtk), mbCert ) @ #tc) ⇒
          (∀ delegation ydx sd #tmb.
            ((((MbGenerateProxySign( delegation, mbLtk, ydx, sd, warrant,
                                     mbCert
                ) @ #tmb) ∧
              (!KU( ydx ) @ #tmb)) ∧
             (!KU( sd ) @ #tmb)) ∧
            (¬(∃ #tmb.1. !KU( mbLtk ) @ #tmb.1))) ⇒
           (∃ sPub #ts.
             ((ServerSendDelegation( sPub, pk(mbLtk), warrant, delegation
               ) @ #ts) ∧
              (#ts < #tmb)) ∧
             (!KU( sPub ) @ #ts)))) ∧
        (∀ warrant mbLtk mbSign #tc.
          (ClientReceivedProxySignForSpb( warrant, pk(mbLtk), mbSign
           ) @ #tc) ⇒
          (∀ delegation ydx sd #tmb.
            ((((MbGenerateProxySignForSpb( delegation, mbLtk, ydx, sd, warrant,
                                           mbSign
                ) @ #tmb) ∧
              (!KU( ydx ) @ #tmb)) ∧
             (!KU( sd ) @ #tmb)) ∧
            (¬(∃ #tmb.1. !KU( mbLtk ) @ #tmb.1))) ⇒
           (∃ sPub #ts.
             ((ServerSendDelegation( sPub, pk(mbLtk), warrant, delegation
               ) @ #ts) ∧
              (#ts < #tmb)) ∧
             (!KU( sPub ) @ #ts))))"
```

Fig. 5: Proof of verifiability lemmas in Tamarin

6 Discussion

We proposed an ECDSA-based cryptographic protocol. However, during the research, we found new insights for improvement. The insight is to use the Schnorr algorithm instead of ECDSA for the algorithm that generates the digital signature. Boldyreva et al.'s research [6] used Schnorr signature, and they shows better outcomes in terms of both performance and security than ECDSA.

- Performance: Schnorr does not have modular inverse calculations that significantly affect performance.
- Security: Since Schnorr is strongly unforgeable under chosen message attack (SUF-CMA), Schnorr is provably secure in the random oracle model [35].

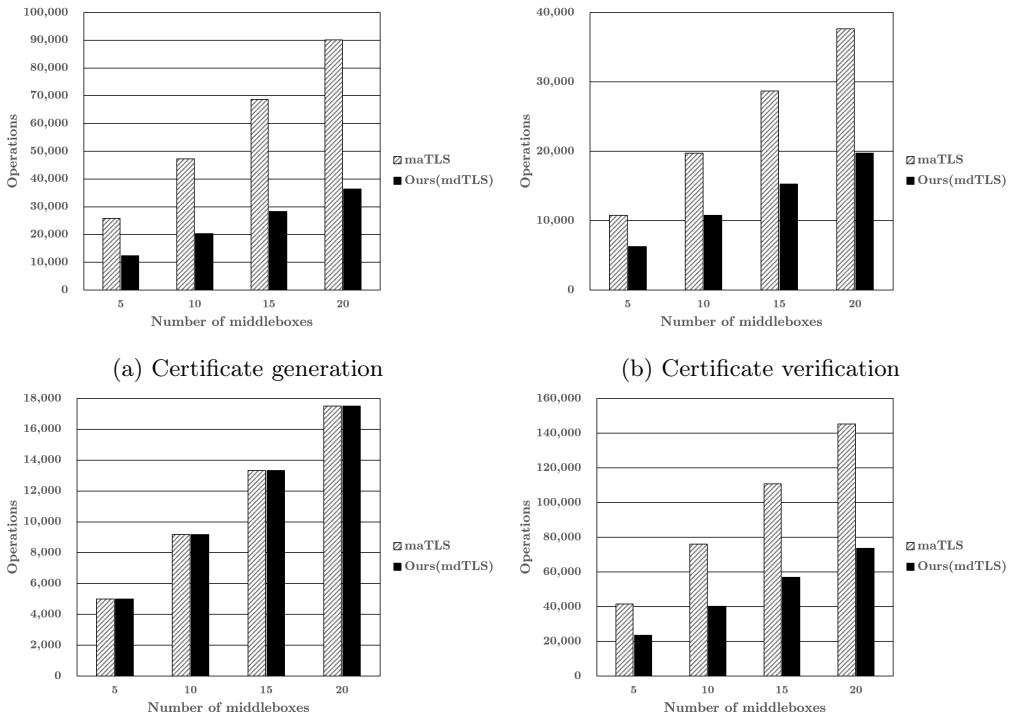
So we compared the performance of the maTLS and mdTLS protocols assumed that both protocols use the Schnorr signature. To measure the performance of Schnorr, the number of modular multiplication operations was calculated using the square-and-multiply algorithm. This algorithm requires 1.5 modular multiplications per bit on average. Besides, as mentioned in Schnorr's paper [38], we calculated the modular multiplications of the Schnorr verification equation by multiplying by 1.75 per bit. When the security level is set to 128-bit, the related parameters' sizes can be shown below [12].

- Public key size: 3,072 bits
- Private key size: 256 bits
- Hash size: 256 bits

Table 6 shows the number of modular multiplications at each stage. Here, N represents the number of middleboxes. The mdTLS reduces the number of modular multiplications by 51.8% compared to maTLS, demonstrating better performance when using Schnorr than when using ECDSA. Nevertheless, the TLS standard mandates the utilization of the ECDSA algorithm for digital signature creation, rendering the adoption of the Schnorr signature algorithm impractical now.

Table 6: Modular multiplications in maTLS and mdTLS

Stages	maTLS	mdTLS
Certificate generation	4,293N + 4,293	1,603N + 4,293
Certificate verification	1,792N + 1,792	897N + 1,792
Security parameter blocks	833N + 833	833N + 833
Overall	6,918N + 6,918	3,333N + 6,918



(a) Certificate generation

(b) Certificate verification

(c) Security parameter blocks

(d) Overall performances

Fig. 6: Performance of protocols when using Schnorr

7 Conclusion

In this paper, we proposed a middlebox-delegated TLS protocol in which only middleboxes that have been permitted can participate in the network. To demonstrate the excellence of our proposed protocol, we verified our protocol from two aspects of view: performance and security. In the performance view, we calculated the number of computations in the protocol. We found that the mdTLS reduces about 39% of the computations compared to maTLS. Also, we formally verified that our proposal achieved nine security lemmas: *server/middlebox/data authentication*, *path integrity*, *path secrecy*, *modification accountability*, *verifiability*, *strong-unforgeability*, and *strong-identifiability*. Especially among them, the latter three security lemmas are newly defined for our protocol by extending existing concepts. The primary contribution of this work is to show that using the proxy signature scheme can enhance performance efficiency and maintain its security level.

Acknowledgements This work was partly supported by Institute of Information communications Technology Planning Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2018-0-00532, Development of High-Assurance (EAL6) Secure Microkernel, 100) and supported by Korea University.

References

1. Apple's Certificate Transparency policy Homepage, https://support.apple.com/eng/HT205280, Last accessed 21 May 2023
2. Certificate Transparency Homepage, https://certificate.transparency.dev, Last accessed 21 May 2023
3. Chrome Certificate Transparency Policy Homepage, https://googlechrome.github.io/CertificateTransparency/ct_policy.html, Last accessed 21 May 2023
4. Hackproof github Homepage, https://github.com/HackProof/mdTLS, Last accessed 26 May 2023
5. Anderson, B.: Detecting Encrypted Malware Traffic (Without Decryption), https://blogs.cisco.com/security/detecting-encrypted-malware-traffic-without-decryption, Last accessed 26 September 2023
6. Boldyreva, A., Palacio, A., Warinschi, B.: Secure Proxy Signature Schemes for Delegation of Signing Rights. Journal of Cryptology **25**, 57–115 (2012)
7. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W.: Internet X. 509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) profile (2008), https://www.rfc-editor.org/rfc/rfc5280.txt, Last accessed 23 September 2023
8. Cortier, V., Delaune, S., Dreier, J.: Automatic generation of sources lemmas in TAMARIN: towards automatic proofs of security protocols. In: Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part II. pp. 3–22. Springer (2020)

9. Dang, Q.H.: Secure Hash Standard (2015), https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf, Last accessed 23 September 2023
10. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2 (2008), https://www.rfc-editor.org/rfc/rfc5246.txt, Last accessed 23 September 2023
11. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on information theory **29**(2), 198–208 (1983)
12. Elaine, B.: Recommendation for Key Management: Part 1 – General (2020), https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf, Last accessed 23 September 2023
13. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext Transfer Protocol–HTTP/1.1 (1999), https://www.rfc-editor.org/rfc/rfc2616.txt, Last accessed 23 September 2023
14. Gallagher, S.: Nearly half of malware now use TLS to conceal communications, https://news.sophos.com/en-us/2021/04/21/nearly-half-of-malware-now-use-tls-to-conceal-communications, Last accessed 26 September 2023
15. Goldreich, O., Goldwasser, S., Micali, S.: How to Construct Random Functions. Journal of the ACM (JACM) **33**(4), 792–807 (1986)
16. Goltzsche, D., Rüsch, S., Nieke, M., Vaucher, S., Weichbrodt, N., Schiavoni, V., Aublin, P.L., Cosa, P., Fetzer, C., Felber, P., et al.: EndBox: Scalable Middlebox Functions Using Client-Side Trusted Execution. In: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 386–397. IEEE (2018)
17. Google: Google Transparency Homepage, https://transparencyreport.google.com/https/overview?hl=en, Last accessed 9 May 2023
18. Han, J., Kim, S., Ha, J., Han, D.: Sgx-Box: Enabling Visibility on Encrypted Traffic using a Secure Middlebox Module. In: Proceedings of the First Asia-Pacific Workshop on Networking. pp. 99–105 (2017)
19. Heron, S.: Advanced Encryption Standard (AES). Network Security **2009**(12), 8–12 (2009)
20. Jarmoc, J., Unit, D.: SSL/TLS Interception Proxies and Transitive Trust. Black Hat Europe (2012)
21. Johnson, D., Menezes, A., Vanstone, S.: The Elliptic Curve Digital Signature Algorithm (ECDSA). International journal of information security **1**, 36–63 (2001)
22. Kim, S., Park, S., Won, D.: Proxy Signatures, Revisited. In: Information and Communications Security: First International Conference, ICIS'97 Beijing, China, November 11–14, 1997 Proceedings 1. pp. 223–232. Springer (1997)
23. Laurie, B., Langley, A., Kasper, E.: RFC 6962: Certificate Transparency (2013), https://www.rfc-editor.org/rfc/rfc6962.txt, Last accessed 23 September 2023
24. Lee, H., Smith, Z., Lim, J., Choi, G., Chun, S., Chung, T., Kwon, T.T.: maTLS: How to Make TLS middlebox-aware? In: NDSS (2019)
25. Mambo, M., Usuda, K., Okamoto, E.: Proxy Signatures: Delegation of the Power to Sign Messages. IEICE transactions on fundamentals of electronics, communications and computer sciences **79**(9), 1338–1354 (1996)
26. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In: Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25. pp. 696–701. Springer (2013)
27. Microsoft: Microsoft Azure firewall Homepage, https://learn.microsoft.com/ko-kr/azure/firewall/premium-features, Last accessed 9 May 2023

28. Miller, J.: Telegeography Homepage, https://blog.telegeography.com/2021-global-internet-map-tracks-global-capacity-traffic-and-cloud-infrastructure, Last accessed 9 May 2023
29. National Institute of Standards and Technology: Digital Signature Standard (DSS) (2023), https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf, Last accessed 23 September 2023
30. National Security Agency: Cybersecurity and Infrastructure Security Agency Homepage, https://www.us-cert.gov/ncas/current-activity/2019/11/19/nsa-releases-cyber-advisory-managing-risk-transport-layer-security, Last accessed 9 May 2023
31. Naylor, D., Li, R., Gkantsidis, C., Karagiannis, T., Steenkiste, P.: And Then There Were More: Secure Communication for More Than Two Parties. In: Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies. pp. 88–100 (2017)
32. Naylor, D., Schomp, K., Varvello, M., Leontiadis, I., Blackburn, J., López, D.R., Papagiannaki, K., Rodriguez Rodriguez, P., Steenkiste, P.: Multi-context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS. ACM SIGCOMM Computer Communication Review **45**(4), 199–212 (2015)
33. O'Neill, M., Ruoti, S., Seamons, K., Zappala, D.: TLS Inspection: How Often and Who Cares? IEEE Internet Computing **21**(3), 22–29 (2017)
34. Panetta, K.: Gartner Homepage, https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2019, Last accessed 9 May 2023
35. Pointcheval, D., Stern, J.: Security Arguments for Digital Signatures and Blind Signatures. Journal of cryptology **13**, 361–396 (2000)
36. Rescorla, E.: HTTP Over TLS (2000), https://www.rfc-editor.org/rfc/rfc2818.txt, Last accessed 23 September 2023
37. Schmidt, B., Meier, S., Cremers, C., Basin, D.: Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties. In: 2012 IEEE 25th Computer Security Foundations Symposium. pp. 78–94. IEEE (2012)
38. Schnorr, C.P.: EFFICIENT IDENTIFICATION AND SIGANTRUES FOR SMART CARDS. In: Advances in Cryptology—CRYPTO'89 Proceedings 9. pp. 239–252. Springer (1990)
39. Sherry, J., Hasan, S., Scott, C., Krishnamurthy, A., Ratnasamy, S., Sekar, V.: Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service. ACM SIGCOMM Computer Communication Review **42**(4), 13–24 (2012)
40. The-Tamarin-Team: Tamarin-Prover Manual, https://tamarin-prover.github.io/manual/master/tex/tamarin-manual.pdf, Last accessed 17 May 2023
41. Trach, B., Krohmer, A., Gregor, F., Arnautov, S., Bhatotia, P., Fetzer, C.: ShieldBox: Secure Middleboxes using Shielded Execution. In: Proceedings of the Symposium on SDN Research. pp. 1–14 (2018)
42. Wei, C., Li, J., Li, W., Yu, P., Guan, H.: STYX: A Trusted and Accelerated Hierarchical SSL Key Management and Distribution System for Cloud Based CDN Application. In: Proceedings of the 2017 Symposium on Cloud Computing. pp. 201–213 (2017)