Graphene: A Secure Cloud Communication Architecture

Abu Faisal and Mohammad Zulkernine

School of Computing, Queen's University, Kingston, ON, Canada
{faisal, mzulker}@cs.queensu.ca

Abstract. Due to ubiquitous-elastic computing mechanism, platform independence and sustainable architecture, cloud computing emerged as the most dominant technology. However, security threats become the most blazing issue in adopting such a diversified and innovative approach. To address some of the shortcomings of traditional security protocols (e.g., SSL/TLS), we propose a cloud communication architecture (**Graphene**) that can provide security for data-in-transit and authenticity of cloud users (CUs) and cloud service providers (CSPs). Graphene also protects the communication channel against some most common attacks such as man-in-the-middle (MITM) (including eavesdropping, sniffing, identity spoofing, data tampering), sensitive information disclosure, replay, compromised-key, repudiation and session hijacking attacks. This work also involves the designing of a novel high-performance cloud focused security protocol. This protocol efficiently utilizes the strength and speed of symmetric block encryption with Galois/Counter mode (GCM), cryptographic hash, public key cryptography and ephemeral key-exchange. It provides faster reconnection facility for supporting frequent connectivity and dealing with connection trade-offs. The security analysis of Graphene shows promising protection against the above discussed attacks. Graphene also outperforms TLSv1.3 (the latest stable version among the SSL successors) in performance and bandwidth consumption significantly and shows reasonable memory usage at the server-side.

Keywords: Cloud Computing · Security Protocol · Data-in-Transit · Authentication · Perfect Forward Secrecy.

1 Introduction

Security concerns such as data breaches and tampering, weak identities and access management, malicious insiders, system and application vulnerabilities and shared technology vulnerabilities have hazardous impact on the cloud as reported by the Cloud Security Alliance (CSA) [15]. To deal with these concerns, the majority cloud service providers (CSPs) implement a mixture of security and privacy controls to provide services to their customers. Cloud users (CUs) connect to the cloud services using internet connectivity. However, existing traditional security protocols (e.g., SSL/TLS) that protect this connectivity, should be more efficient to handle cloud communication related security issues. Every now and then, a new security threat is raised. In most cases, man-in-the-middle

(MITM) (including eavesdropping, sniffing, identity spoofing, data tampering), sensitive information disclosure, replay, compromised-key, repudiation and session hijacking attacks can happen in cloud communications [8, 15]. Traditional security protocols (e.g., SSL/TLS) are not always able to satisfy the growing demand of security in cloud communications for various reasons. These reasons are mainly related to maintaining middlebox compatibility, backward compatibility for older systems, downgrading due to unavailability of the selected protocol version or cipher suites and some recent attacks (e.g., BEAST, DROWN, CRIME, BREACH, WeakDH and Logjam, SSLv3 fallback, POODLE and ROBOT attacks) [3–5, 7, 10, 13, 17–19, 28].

The final draft of TLSv1.3 [30] is published recently. It claims to have some improvements over TLSv1.2 [31] in terms of security and performance. TLSv1.3 stops supporting all legacy symmetric encryption algorithms and static RSA and Diffie-Hellman cipher suites. It adds (EC)DHE to the base specification. Also, it uses only authenticated encryption with associated data (AEAD) algorithms. However, it still has some vulnerabilities. In TLSv1.3, the first two roundtrip handshake messages are merged into a single roundtrip message. This merged message includes the client key-exchange information, supported cipher suites information and "`ClientHello`" message together in *unencrypted* form. Most importantly, before the client receives "`ServerHello`" message, all communications are performed in unencrypted form. The client key-exchange information is one half of the key-exchange mechanism that is generated by random guessing of the server-side algorithm. Therefore, if the server does not agree or support that algorithm or the client sends no key-exchange information, the client needs to generate and send the key-exchange information again using the agreed algorithm which increases the roundtrip time. It still uses pre-shared key (PSK) cipher suites along with the above changes. Also, superfluous messages such as "`ChangeCipherSpec`" are eliminated while keeping a backdoor open for middlebox compatibility.

In this paper, we propose a comprehensive secure cloud communication architecture (**Graphene**). This architecture can effectively mitigate the existing threats of cloud communications between cloud entities. Graphene ensures security for data-in-transit and authenticity of cloud users (CUs) and cloud service providers (CSPs). It does not have any middlebox or backward compatibility. Either both parties communicate using the supported cipher suites recommended by the NIST or the secure channel cannot be established. We perform security analysis based on the man-in-the-middle (MITM) (including eavesdropping, sniffing, identity spoofing, data tampering), sensitive information disclosure, replay, compromised-key, repudiation and session hijacking attacks. Thus, we show that this architecture can efficiently mitigate these attacks. Graphene protects the cloud communication channels with significantly less negotiation and bandwidth overhead, reasonable memory usage and faster connectivity than the traditional security protocols (e.g., TLSv1.3).

Our main contribution in this paper is a comprehensive secure cloud communication architecture called Graphene. More specifically, the paper makes the

following contributions:
• Graphene provides a novel high-performance cloud focused security protocol. This protocol efficiently utilizes the strength and speed of symmetric block encryption, cryptographic hash, public key cryptography and ephemeral key-exchange mechanism.
• It utilizes new highly-compact message structures to support secure session establishment, reconnection and data transmission. These message structures help achieve minimal bandwidth consumption and reasonable memory usage than TLSv1.3 (the latest stable version among the SSL successors) and embed other communication protocols inside it.
• Graphene ensures security of the data-in-transit and all associated secret keys. It maintains perfect forward secrecy (PFS) by performing ephemeral key-exchange on each session and encrypting the session with a new secret key.
• It is applicable to both TCP and UDP-based communications. It has no dependency on the SSL/TLS/DTLS implementations at any part of the communication channel.

The rest of this paper is organized as follows. Section 2 presents the related work. Section 3 discusses the proposed secure cloud communication architecture. Section 4 provides the implementation and experimental environment of the architecture. Section 5 presents the results. Finally, Section 6 summarizes the paper.

2 Related Work

The main purpose of the existing cloud security research is to secure data during cloud communications (data-in-transit) as well as the cloud data storages (data-at-rest).

Google [20–22] uses multi-layer encryption model to secure the data-at-rest while relying on default TLS for protecting the data-in-transit. Amazon Web Services (AWS) [9] and Microsoft Azure [27] focus on protecting data integrity using keyed-HMAC. AWS uses temporary non-stored session keys in EC2 load balancers and Azure uses two-factor authentication to prevent unauthorized access of data. However, they still use secure sockets layer (SSL) to provide transmission protection to their customers. They also maintain MD5 compatibility for older systems. It is clearly visible that the CSPs are mostly concerned to secure the data stored in their data centers by using multi-layer encryption model, keyed-HMAC, two-factor authentication etc. However, they rely on default TLS and sometimes even SSL for protecting the data-in-transit which makes the existing implementations vulnerable to all recent SSL/TLS related attacks [3–5, 7, 10, 13, 17–19, 28]. On the contrary, Graphene ensures security of data-in-transit and authenticity of cloud entities. It does not support any security techniques which have any known vulnerabilities. Graphene has its own novel protocol, highly compact message structures and secure session management. It provides higher level of security with lower level of bandwidth consumption and reasonable memory usage.

AbdAllah *et al.* [6] propose a generic trust model (TRUST-CAP) for cloud-based applications by focusing on infrastructure-as-a-service (IaaS). However,

it does not provide any specific protocol for securing cloud communications. Conversely, Graphene provides a cloud-focused security protocol. It ensures adequate protection to the communication channel and its associated cloud entities against some most common cloud attacks. As Graphene follows the security objectives of the TRUST-CAP model, it can be used as the security protocol for cloud communications in TRUST-CAP as well.

Kaaniche *et al.* [23] propose a cloud data sharing framework (CloudaSec) that encrypts the data at the server-side using the hash of the data as the symmetric key. Then, it encrypts the symmetric key using recipient's public key and includes that in the response metadata. Basically, it uses a form of hybrid cryptography [2] where the symmetric key remains the same for unchanged data. On the other hand, Liang *et al.* [26] and Chandu *et al.* [14] also propose a similar approach that follows the generic hybrid cryptography. It uses AES to encrypt the data at the client-side. Then, it encrypts the AES key using owner's RSA public key. After that, it uploads both the encrypted data and the encrypted key to the cloud storage. Both approaches are vulnerable to compromised-key, permanent data tampering, identity spoofing, MITM and MATE attacks.

Khanezaei *et al.* [24] propose a secure cloud storage service using server's public-key to perform RSA encryption to protect the data during communication and storage. The service generates an AES secret key and stores it in the database along with the RSA encrypted data for future sharing. However, this solution imposes tremendous computing overhead on the cloud server for decrypting large amount of RSA encrypted data on every request which is a very cumbersome and slow process. The RSA encrypted data uses server's public key which can be decrypted easily using server's private key. Moreover, the AES secret key is stored along with the RSA encrypted data which makes this solution highly prone to compromised-key, permanent data tampering, identity spoofing, MITM and MATE attacks.

Kerberos [29] is a network authentication protocol that provides authentication in a non-secured environment. It uses a key distribution center (KDC) which receives request for tickets from the clients. Then, the KDC generates ticket-granting tickets (TGT) and encrypts it using client's password. After receiving the encrypted response, the client decrypts it using the password. This is different from the central key server (CKS) mechanism in Graphene. The CKS is a root public key management system which helps Graphene architecture to prevent MITM in all phases. It is designed to store, revoke and distribute root public keys securely.

All the above discussed related work are focused on a specific facet of security and operational behavior. They are mostly concerned about the data-at-rest and their confidentiality, integrity or access control. Also, some recent attacks such as BEAST, DROWN, CRIME, BREACH, WeakDH and Logjam, POODLE, ShellShock and ROBOT attacks [3–5, 7, 10, 13, 17–19, 28] have shown that the existing security protocols are not able to mitigate the increasing threats of cloud communications. It becomes a major hindrance while expanding towards IoT, fog or edge computing, connected vehicles etc. Therefore, a comprehensive

secure cloud communication architecture is mandatory to mitigate the rising threats against cloud communications.

3 Graphene Architecture

This section presents the proposed secure cloud communication architecture in detail. In the following section, we discuss about the design of this architecture and different communication phases of it. In Section 3.2, we explain the sequence of events executed at both user and server ends.

3.1 Design Specification

Graphene focuses on the security of data-in-transit in cloud computing. It guarantees the authenticity of cloud entities by using a new Central Key Server (CKS) mechanism. The CKS is designed to store, revoke and distribute root public keys securely. Graphene efficiently combines and utilizes the strength and speed of the symmetric block encryption, cryptographic hash, public key cryptography and ephemeral key-exchange mechanism. Symmetric encryption provides confidentiality, cryptographic hash enables integrity and public key cryptography ensures authenticity and non-repudiation. It embeds these four essential security elements into the communications in such a way so that a cloud user (CU) can have a secure communication channel with the Cloud Front End (CFE) server. It ensures security for both the data and the cryptographic keys.

The system does not use any long-term keys. Each session is encrypted with a new secret key thus ensuring perfect forward secrecy (PFS). Graphene is applicable for both TCP and UDP-based communications. It works in the application layer. Thus, it can be easily integrated with any protocols and server systems. Graphene utilizes seven highly compact new message structures: i) publish (PUB), ii) acknowledge (ACK), iii) reconnect (RECON), iv) request (REQ), v) response (RES), vi) expired (EXP), and vii) error (ERR). They make Graphene more efficient in terms of performance, bandwidth consumption, memory usage and integration with the existing protocols. These message structures facilitate the secure session establishment, reconnection, data transmission and error handling between cloud entities.

The architecture consists of six different communication phases such as registration, initialization, session establishment, data transmission, termination and reconnection. First, the cloud entities need to register their root public keys to the central key server (CKS) in the registration phase. After that, when any cloud user wants to communicate to the cloud server, temporary cryptographic keypairs and hash functions are initialized in the initialization phase to establish an encrypted session. Then, both the entities exchange their temporary public keys with each other, signed by their respective root private key. The key-exchange of temporary public keys is secured by hybrid-crypto mechanism [2, 16] using AES-GCM for data encryption and RSA/ECC for key encryption during the session establishment phase.

After that, both entities generate common symmetric encryption key using ephemeral key-exchange. Then, they start transmitting encrypted signed data to each other in the data transmission phase. After sending the response payload

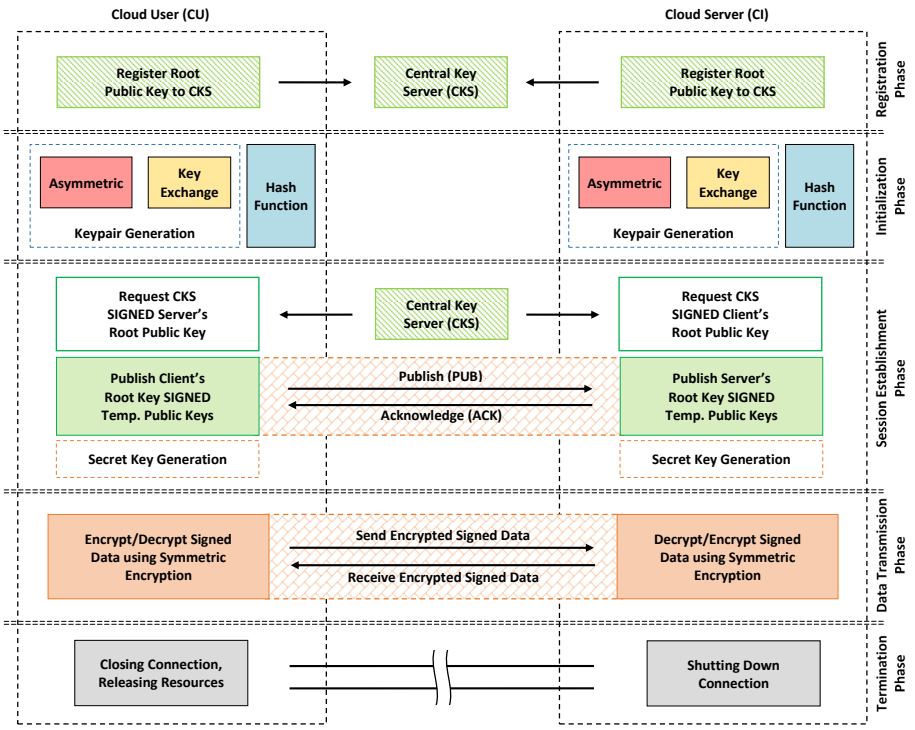

Fig. 1. Different communication phases between cloud entities in Graphene architecture

successfully to the cloud user, cloud server terminates the connection which is called the termination phase. At this point, the server keeps the encrypted session information till the session expires. Within that period, the cloud user can send a reconnection request and re-establish the encrypted session for further data transmission which is called the reconnection phase. Fig. 1 shows these phases of communications used by Graphene which we discuss in detail in the following paragraphs.

Registration Phase. All cloud entities must register their root public keys to the central key server (CKS) prior to any communication. The CKS public keys must also be installed in the cloud entity systems, to ensure integrity and authenticity of the data communicated between the CKS and the cloud entities. The CKS itself and all communications (key registration, revocation and distribution) with it are assumed to be secured at this point.

Initialization Phase. In case of cloud server instance (CI), this phase occurs at the very beginning when the CI is initiated. However, for the cloud user (CU), it occurs when a new cloud connection is created to commence communication with the cloud front end (CFE) server. During this phase, each cloud entity generates a pair of temporary public-private keypairs. One keypair (RSA/ECC) is for maintaining the authenticity and integrity of the payloads. The other

keypair (DHE/ECDHE) is for the ephemeral key-exchange. Each cloud entity also initializes cryptographic hash functions according to the design specification.

Session Establishment Phase. When a CU tries to connect to the CI for the first time, a temporary encrypted session is initialized between the CU and the CI. During this time, a pair of messages (PUB-ACK) are transmitted between them. Both parties store the other party's pair of public keys in that temporary session protected by a 64-byte hashed session key. Then, they generate a common secret key to proceed with the data transmission phase. The 64-byte hashed session key is updated after every successful transaction (request-response). The CU always receives the updated session key hidden inside the encrypted response. When this session expires, all the negotiated public keys and generated common secret key are destroyed automatically.

Data Transmission Phase. After establishing the secure session, both parties use the common secret key to perform symmetric block encryption for maintaining the confidentiality of the request and response payloads. The negotiated temporary keypair (RSA/ECC) is used to perform payload signing and verification that ensures authenticity and integrity of the payload throughout the session. Every signing operation performed in this architecture involves timestamp to protect against replay attacks. During this phase, a cloud focused cryptographic hash function (Blake2b [1]) is used to protect the data integrity.

Termination Phase. In this phase, when the CI sends encrypted response back to the CU successfully, the communication channel is terminated. The existing session remains valid for reconnection until it is expired.

Reconnection Phase. This phase is not explicitly shown in Fig. 1. It has implicit activity in this architecture. After the termination phase, if the CU again connects to the server and sends a valid reconnection (RECON) packet with the last received session key, the encrypted session is re-established between the CU and the CI. The CFE maintains a session key mapping of the CIs. Based on the session key, it reconnects the CU to the appropriate CI. Both parties use the previously negotiated pair of public keys and the stored common secret key. Therefore, re-keying the block cipher during the session is not needed.

3.2 Flow of Execution

This section explains how this architecture establishes a secure encrypted channel for communications and all the internal steps illustrated by the sequence diagram shown in Fig. 2.

Step-1. In this step, the cloud user (CU) initializes a cloud connection. A pair of temporary public-private keypair is generated and the cryptographic hash functions are initialized.
Step-2. After initializing the connection, the cloud user fetches the cloud server's root public key which is signed by the central key server (CKS) that ensures authenticity and non-repudiation for both parties.
Step-3. The cloud user (CU) connects to the cloud front end (CFE) server and a cloud instance (CI) is allocated for this connection.

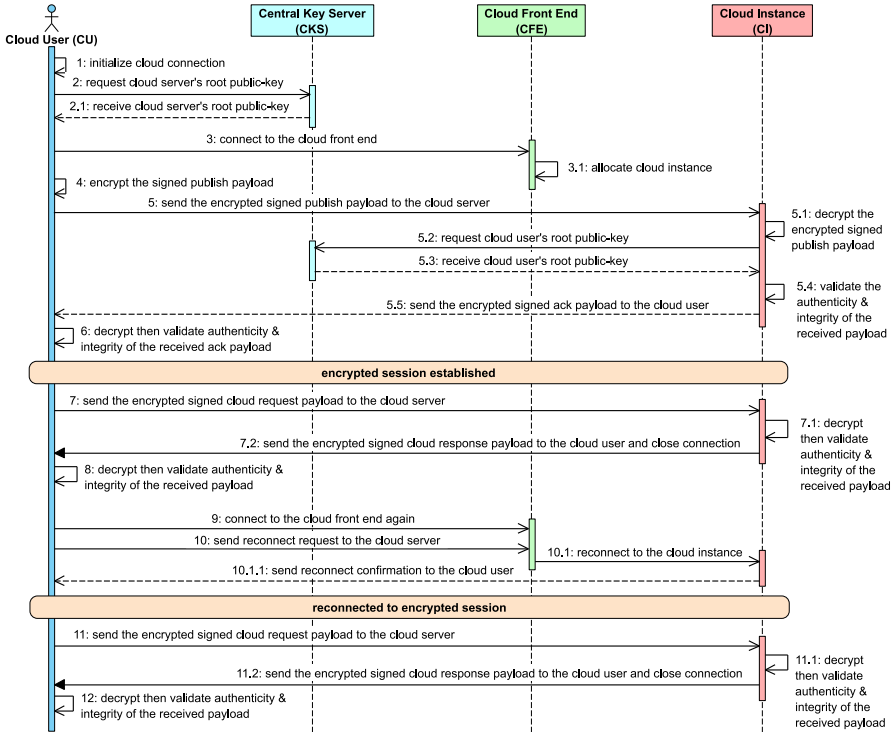

Fig. 2. Sequence diagram showing the flow of execution in Graphene architecture

Step-4. The CU signs its temporary public keys with own root private key to protect authenticity and integrity of the "publish" payload (PUB). After that, the signed "publish" payload is encrypted using symmetric block encryption to maintain the confidentiality of the payload in a hybrid-crypto mechanism [2, 16].
Step-5. The CU and CI utilize the "publish" and "acknowledge" packets (PUB-ACK) to share all temporarily generated public keys to each other. The CU sends the encrypted signed "publish" payload to the CI. After decrypting the received packet, the CI requests the cloud user's root public key from the central key server (CKS). Then, the CI validates the authenticity and integrity of the received "publish" payload. After validation, the CI sends the encrypted signed "acknowledge" payload (ACK) to the CU. This approach protects the session establishment phase from man-in-the-middle (MITM) attacks.
Step-6. When the CU receives the encrypted signed "acknowledge" packet (ACK), it also validates the authenticity and integrity of the received payload. The cloud user stores cloud server's temporary public keys in the session. After finishing session establishment phase, the common secret key is generated at both ends using the ephemeral key-exchange mechanism (DHE or ECDHE). A secure encrypted communication channel is established without using any pre-shared key or transmitting any part of the secret key. This generated secret key is used to perform symmetric block encryption on the signed cloud payload.

Step-7, 8. In this step, both parties perform data transmission (request-response) which is first signed and then encrypted to protect confidentiality, integrity and authenticity of the data. After sending the response, the cloud instance (CI) terminates the connection with the cloud user (CU).
Step-9, 10. When the CU again wants to accomplish any more data connectivity and it has the valid session information, it can send a reconnection packet (RECON) to the cloud front end (CFE) server. If any associated session is found, the secure channel is re-established between the CU and the CI. They do not need to perform the session establishment steps again. Otherwise, the CU must go through Step-4 to Step-6 again.
Step-11, 12. Once the secure session is re-established, both the CU and the CI can do data transmission again. After the response is sent back to the CU, the CI closes the connection.

4 Implementation and Experimental Environment

This section explains the implementation and experimental environment used to evaluate Graphene in terms of performance, bandwidth consumption and memory usage. In the following section, we briefly discuss the implementation details of the architecture. Then, in Section 4.2, we explain the experimental environment.

4.1 Implementation

Graphene is developed using Java and Java Cryptography Architecture (JCA). It has no dependency on any other platforms, tools and libraries. Therefore, our implementation can be deployed in any platform or environment where Java runtime environment (JRE) is available. To compare Graphene against TLSv1.3, we run all our experiments in Java11.0.1 (LTS) which includes an implementation of the TLSv1.3 specification [30]. A novel high-performance cloud focused security protocol is designed and implemented with seven highly compact message structures. Any types of payload data (e.g., HTTP, XML, JSON and Binary) can be sent and received using this protocol with minimal changes in the existing infrastructures and applications.

Graphene uses public-key cryptography for signing the payloads and ephemeral Diffie-Hellman (at least 2048-bit) using MODP groups [25] as the key-exchange mechanism. A latest cryptographic hashing algorithm Blake2b [1] is used for maintaining the integrity of the data-in-transit. It is faster than SHA-families and as secure as SHA-3 at minimum, which makes it a perfect candidate for cloud communications and large volume of data hashing. SHA-512 is used to generate temporary session keys from the connection properties and the client supplied information. AES-256 with Galois/Counter mode (GCM) is used as the symmetric block encryption for ensuring confidentiality throughout all the communication phases. The system operates over 256-bit encrypted channel which is the approved encryption standard for *top secret* information by both the National Institute of Standards and Technology (NIST) and the National Security Agency (NSA) of the USA.

This architecture is configurable to use any of the supported (RSA/ECC) public-key cryptographic algorithms for payload signing and verification. However, the minimum key size recommended by the NIST is 2048-bit for RSA and 224-bit for ECC [12]. Our implementation *strictly* follows these recommendations made by the NIST at all steps [11, 12]. AES (128/192/256-bit) encryption is used as the supported symmetric block encryption in Graphene. AES-256 is the highest level (military-grade) of symmetric encryption available at present. It is also the default choice for confidentiality in Graphene. However, Graphene can be configured to use any of the other key sizes or encryption algorithms if this level of security is not required.

4.2 Experimental Environment

As illustrated in Fig. 3, cloud instances (CIs) are configured according to the requirement. Each CI has 1 hyper-threaded vCPU core (4.0GHz frequency with turbo boost), 4GB of RAM, 20GB of local SSD storage. Each cloud instance runs CentOS 7 (*minimal version*) to have less interference from other processes. The cloud instances are setup and controlled by a cloud front end (CFE) server. The CFE server has a built-in basic load balancer which works in a simplified round-robin fashion. It is responsible for distributing all incoming traffics to these cloud instances equally by assigning the same weight to each instance (CI) unless the incoming traffic is a reconnection request with valid session information.

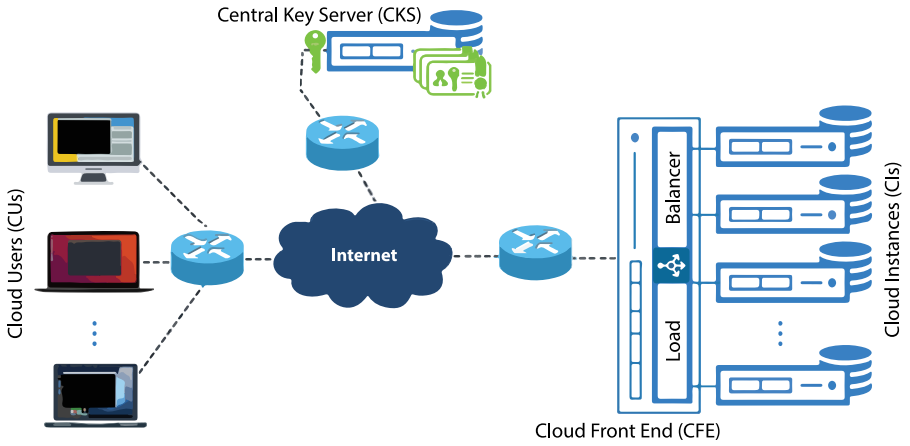

Fig. 3. Experimental environment of the Graphene architecture

The CI records execution time for session establishment (if any), request and response at the server-side for plaintext, TLSv1.3, TLSv1.2 and Graphene with and without session-reconnection mechanism. However, the cloud user (CU) monitors roundtrip time information at the client-side for further analysis. All CUs run in an iterative fashion and send request with a specific size (100B, 500B, 1KB, 500KB or 1MB) of data every time. A separate secure public key registration and distribution server runs as a central key server (CKS) for managing root public keys. In CKS, all cloud entities have their root public keys registered

against their unique identifier. In Graphene, the CFE server and the CUs have their public keys registered against their IP addresses and assigned random string tokens. All experiments are performed in an iterative fashion (1000 times). Each request belongs to a temporary encrypted session which has a hashed session key generated from the connection properties and the client supplied information.

The reason behind comparing with TLSv1.3 in our experiment is that it is the latest stable version among the SSL (Secure Sockets Layer) successors. TLSv1.3 is claimed to be more secure than TLSv1.2, where TLSv1.2 is proved to have a steady and secure implementation than SSL, TLSv1.0 and TLSv1.1. SSLv3 and TLSv1.0 are already declared obsolete and some vulnerabilities are reported for TLSv1.1. Due to the severe data breaches caused by recent attacks, TLSv1.3 is now recommended for secure communications over the internet. If TLSv1.3 is not available, at least TLSv1.2 should be used for secure communications.

5 Results and Analysis

This section presents the results and analyzes the solution. All prominent cryptographic technologies (public key cryptography, digital signature and verification, symmetric block encryption and cryptographic hash) are evaluated iteratively for different payload sizes (100B-20MB) to select the optimal choice for implementing a high-performance cloud focused security protocol (i.e. Graphene) that efficiently utilizes these technologies with respect to their strength and speed. The following section presents a thorough security analysis of Graphene against different types of attacks. After that, we evaluate the performance of Graphene in terms of execution time on server-side, roundtrip time on client-side, bandwidth overhead with respect to plaintext, memory usage at server-side and impact of different payload sizes in the above mentioned scenarios.

5.1 Security Analysis

To show the level of defense provided by Graphene with respect to MITM (including eavesdropping, sniffing, identity spoofing, data tampering), sensitive information disclosure, replay, forward secrecy (compromised-key), repudiation and session hijacking attacks, we provide a thorough analysis.

i) Man-in-the-Middle (MITM) Attack. This attack is basically a combination of different security attacks like eavesdropping, sniffing, identity spoofing and data tampering. In MITM attacks, an adversary can actively eavesdrop to a private communication between two legitimate users or even create separate connections to each of the users to appear as a legitimate entity to both parties (identity spoofing). Then, the attacker captures all the packets (sniffing) and forwards them to the other party in such a way so that the victims are forced to believe that they are communicating directly to each other over a private connection. In the later approach, the attacker has full control over the communication and can easily steal valuable information or even manipulate the packets (data tampering) sent to the victims. In order to analyze Graphene against these attacks, we investigate two types of connections made from any entity in Graphene. One is from cloud user (CU) or cloud instance (CI) to the

central key server (CKS) and the other is in between CU and CI as discussed below.

a) CU/CI to CKS. When any CU/CI requests any public key from the CKS, the CKS responds with the requested public key payload signed by its own root private key. The root public key of CKS is installed to all entity systems during setup time. Thus, the receiver can verify the authenticity and integrity of the received public key payload from the CKS which prevents identity spoofing and data tampering. Since the payload is a public key and it is meant to be shared publicly, confidentiality of this type of payload is not required at all. Therefore, even if any adversary is eavesdropping or sniffing to this connection, the adversary cannot tamper with the payload. Hence, MITM attacks are not possible for this type of connection.

b) Between CU and CI. All communications between the CU and the CI are securely protected (signed and encrypted). Each packet is signed by their root or temporary private key based on the communication phase. Thus, the other entity can always verify the authenticity of the sender by using sender's root or temporary public key. Signing each packet ensures the authenticity and integrity of the received payload in all phases which prevents the identity spoofing and data tampering attacks on DHE key-exchange and request-response payloads. Finally, due to AES-GCM encryption, the adversary can never see the payloads transmitted through this channel at any time which eliminates the scope of eavesdropping or sniffing. Thus, ensuring MITM attacks cannot be successful on this connection at all.

ii) Sensitive Information Disclosure. This attack often happens where the payload is transmitted in plaintext or the encryption technique used is prone to cryptanalysis attacks. In this scenario, the adversary can capture all the packets and steal transmitted sensitive information without the knowledge of the user. However, in Graphene, all communications between CU and CI are performed using AES-GCM encrypted channel (at least 128-bit) from the transmission of first packet. Thus, no sensitive information can be accessed without establishing a proper communication channel.

iii) Replay Attack. This is one of the most common attacks which helps the attacker to intercept valid payloads and retransmit those captured payloads repeatedly to perform some malicious or fraudulent activities. In Graphene, we designed the architecture in a manner so that this kind of attack cannot be successful. First, all our payload signing involve timestamp to create randomness in the output. Then, temporary session key is updated after every successful transaction (request-response) during the data transmission phase. This timestamp-based signing and temporary session key enable Graphene to prevent replay attacks. Thus, at no point, an adversary can gain any benefit from repeating any previously captured data.

iv) Forward Secrecy. In cryptography, forward secrecy is a feature that ensures compromising any secret key does not compromise the security of the past payloads communicated between the entities. In our approach, we maintain perfect forward secrecy (PFS) through ephemeral Diffie-Hellman key-exchange with

at least 2048-bit key size on each new session and by generating all associated cryptographic keys per session as well. Therefore, even if one session is compromised, other past and future sessions remain secure.

v) Repudiation. This means denying the responsibility of any actions performed. In Graphene, all entities must be registered to CKS prior to any communication. The session establishment phase is performed using their registered root public-private keypair and both entities (CU and CI) negotiate temporary keypairs for this session. Later on, all communications are authenticated using these temporary public-private keypairs. This ensures authenticity and non-repudiation of the entities throughout this communication. Thus, this attack is not feasible by any means over this communication channel.

vi) Session Hijacking. In session-based communications, attackers often try to capture session related information. More specifically, they try to lookup session keys or nonce information. In our approach, we use temporary hashed session keys generated based on connection properties and client supplied information. This session key enables cloud entities to re-establish their previous encrypted session if not expired already. Each session key is updated after every successful transaction (request-response) and most importantly, all transmitted packets in Graphene are AES-GCM encrypted.

vii) Some Recent Attacks. Some hazardous attacks such as DROWN, CRIME, BREACH, BEAST, WeakDH and Logjam, SSLv3 fallback, POODLE and ROBOT attacks [3–5, 7, 10, 13, 17–19, 28] happen on traditional security protocols (e.g., SSL/TLS) that highly threaten the existing cloud infrastructures and their expansion towards fog or edge computing, IoT, connected vehicles, smart city etc. Some of the attacks are performed by exploiting weaknesses in the security technologies whereas some are caused by misconfiguration of the system. Due to the advancement of computing resources, security measures which deemed secure in the past become vulnerable to brute force attacks, adaptive chosen plaintext attacks, compression ratio leak, discrete logarithm or other cryptanalysis attack techniques. Graphene strongly follows the NIST recommendations in choosing suitable cryptographic algorithms and their minimum supported key sizes. This enables Graphene to prevent such attacks. It uses Galois/Counter mode (GCM) as the mode of operation for AES with new initialization vector (IV) values for each request. Graphene does not deal with any compression techniques. It strictly follows the recommended key sizes by the NIST [11, 12] for the minimum level of security and also uses MODP [25] groups (group id 14 or above) to perform ephemeral key-exchange.

5.2 Performance Analysis

This section presents the performance evaluation of the implemented architecture in terms of average execution time on the server-side, roundtrip time on the client-side, bandwidth overhead with respect to plaintext, TLSv1.3 and TLSv1.2 communications and memory usage at the server-side. Table 1 represents the specification of the experimental environment used for evaluating performance, bandwidth overhead and memory usage.

Table 1. Cloud Instance Specification

Parameters	**Values**
Virtual CPU(s), Memory	vCPUs: 1 (HyperThreaded), RAM: 4GB
VM Class	Regular (Non-Preemptible)
Processing Unit	4.0GHz with Turbo Boost (8M Cache)
Cloud OS & Storage	CentOS 7 (Minimal) with 20GB SSD Storage
CFE Load Balancer	Round Robin
Sample Data	100B, 500B, 1KB, 500KB, 1MB
Number of Iteration	1000

Fig. 4(a) shows the average execution time for one of the investigated cloud instances in milliseconds. We investigate the average execution times in different cloud instances for plaintext (yellow curve), TLSv1.3 (purple curve), TLSv1.2 (orange curve), Graphene without session-reconnection (blue curve) and Graphene with session-reconnection (green curve) for different payload sizes (100B, 500B, 1KB, 500KB and 1MB).

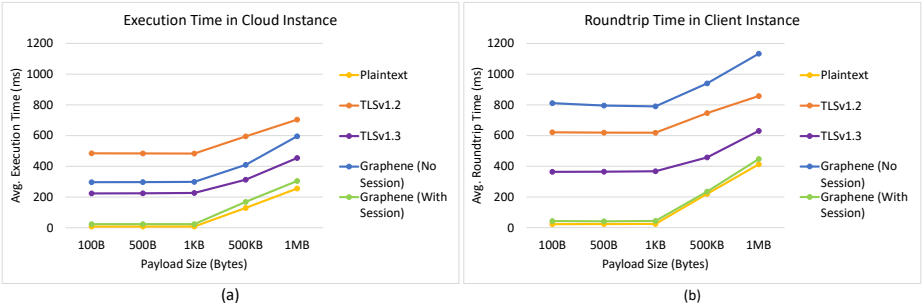

Fig. 4. Comparison of average (a) server-side execution time and (b) client-side roundtrip time in Graphene architecture (with/without session-reconnection) with respect to plaintext, TLSv1.3 and TLSv1.2 communications for different payload sizes

Graphene with session-reconnection mechanism (green curve) outperforms TLSv1.3 (purple curve) significantly for all payload sizes and lies very close to the plaintext (yellow) curve and behaves the same in all cloud server instances. Graphene with session-reconnection (green curve) performs around 90% faster than the TLSv1.3 communication. Our solution even shows better results with session-reconnection (green curve) and without session-reconnection (blue curve) mechanism with respect to TLSv1.2 (orange curve).

On the client-side, we have measured the average roundtrip time (in milliseconds) by taking the sum of observed durations for connection creation, session establishment (if present) and request-response time for different payload sizes. Fig. 4(b) presents the average roundtrip time for one of the investigated client instances under plaintext (yellow curve), TLSv1.3 (purple curve), TLSv1.2 (orange curve), Graphene without session-reconnection (blue curve) and Graphene with session-reconnection (green curve) for different payload sizes (100B, 500B, 1KB, 500KB, 1MB).

As observed from the performance curves of client-side average roundtrip time, Graphene with session-reconnection mechanism (green curve) performs

very close to that of the plaintext (yellow) curve and shows promising performance against TLSv1.3 (purple curve). The performance of Graphene without session-reconnection mechanism (blue curve) deteriorates in terms of average roundtrip time at the client-side. However, if it is used with session-reconnection mechanism, it is able to provide faster communication with higher level of security.

The bandwidth overhead graph shown in Fig. 5(a) is calculated with respect to the bandwidth consumption of the plaintext communication. It is readily noticed that the bandwidth overhead for 100 bytes of payload size is more than 280% for TLSv1.3 (purple column) and over 380% more for Graphene without session-reconnection mechanism (blue column). However, when Graphene is used with session-reconnection mechanism (green column), it shows only 80% overhead with respect to plaintext communication and provides 54% gain over TLSv1.3 communication.

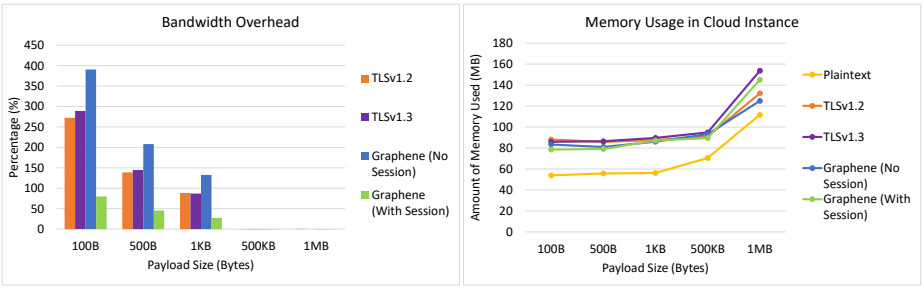

Fig. 5. Comparison of (a) bandwidth overhead and (b) average server-side memory usage in Graphene architecture (with/without session-reconnection) with respect to plaintext, TLSv1.3 and TLSv1.2 communications for different payload sizes

For 1KB of payload size, Graphene with session-reconnection mechanism provides 32% gain over the bandwidth consumption of TLSv1.3. The graph shows a decreasing trend with increasing payload sizes and for 500KB payload size the overhead becomes nearly 1% for all types of communications with respect to plaintext. Therefore, in case of large volume of data, it seems like the overhead is negligible. However, Graphene with session-reconnection performs noticeably well in smaller payload sizes as well as with the increasing payload sizes.

Fig. 5(b) shows the server-side memory usage (in MB) of Graphene in one of the investigated cloud instances with respect to plaintext, TLSv1.3 and TLSv1.2 communications. From the figure, it is readily noticed that Graphene with and without session-reconnection mechanism shows reasonable amount of memory usage for different payload sizes which lies very close to the memory usage of TLSv1.3 and TLSv1.2 communications. The usage pattern shows similar behavior in all the investigated cloud instances and the memory usage increases proportionally with the increase in payload size.

Overall, Graphene with session-reconnection mechanism performs significantly better than the TLSv1.3 in terms of server-side performance, client-side roundtrip time, bandwidth overhead and memory usage at server-side. Once

the session establishment phase is complete, it can efficiently establish 256-bit encrypted channel without causing any performance, bandwidth or memory overhead. However, Graphene without session-reconnection mechanism performs worse than TLSv1.3 because of the temporary keypair generations in each session at both ends (client and server). In every session, two temporary keypairs are generated at each side to establish the session. Communicating with the central key server (CKS) by the cloud user and the cloud instance does not have that much impact on the roundtrip time. Also, Graphene was not evaluated against TLSv1.3 0-RTT mode due to unavailability of the implementation of this mode in Java11.0.1 (LTS).

6 Conclusion

Most recent security attacks and vulnerabilities of the traditional security protocols (SSL/TLS), are the major road blocks in the expansion of cloud computing. In this paper, we propose a comprehensive secure cloud communication architecture (Graphene) that mitigates these attacks and vulnerabilities. In Graphene, security of data-in-transit and authenticity of cloud entities are ensured and firmly integrated into the communications to protect against wide range of cloud attacks. A novel high-performance cloud focused security protocol is designed and implemented. It has seven highly compact new message structures which establish a secure performance and bandwidth-efficient protocol with reasonable memory usage. This architecture can successfully prevent man-in-the-middle (MITM) (including eavesdropping, sniffing, identity spoofing, data tampering), sensitive information disclosure, replay, compromised-key, repudiation and session hijacking attacks. Graphene with session-reconnection mechanism shows 90% faster execution time than TLSv1.3 (the latest stable version among the SSL successors) on the server-side and exhibits similar performance at the client-side as well. In terms of bandwidth consumption, it shows 54% gain over TLSv1.3 and overall reasonable memory usage against different payload sizes. It enforces the NIST recommendation as the base level of security for data-in-transit in cloud computing. In the future, we will work on the applications of this architecture in different sectors.

Acknowledgment

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canada Research Chairs (CRC) program. We would also like to convey special thanks to Mohima Hossain from the TRL Lab at Queen's University for the fruitful discussion and her critics during this research work.

References

1. BLAKE2 — fast secure hashing. https://blake2.net/ (2017), accessed: 02 Sep. 2018
2. Hybrid CryptoSystem. https://en.wikipedia.org/wiki/Hybrid_cryptosystem (2017), accessed: 02 Sep. 2018

3. Weak Diffie-Hellman and the Logjam Attack. https://weakdh.org/ (2017), accessed: 02 Sep. 2018
4. CRIME. https://en.wikipedia.org/wiki/CRIME (2018), accessed: 02 Sep. 2018
5. Transport Layer Security: Attacks against TLS/SSL. https://en.wikipedia.org/wiki/Transport_Layer_Security#Attacks_against_-TLS/SSL (2018), accessed: 02 Sep. 2018
6. Abdallah, E.G., Zulkernine, M., Gu, Y.X., Liem, C.: Trust-cap: A trust model for cloud-based applications. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC). vol. 2, pp. 584–589 (July 2017). https://doi.org/10.1109/COMPSAC.2017.256
7. Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J.A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanella-Béguelin, S., Zimmermann, P.: Imperfect forward secrecy: How diffie-hellman fails in practice. In: Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security. pp. 5–17. CCS '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2810103.2813707, http://doi.acm.org/10.1145/2810103.2813707
8. Amara, N., Zhiqui, H., Ali, A.: Cloud computing security threats and attacks with their mitigation techniques. In: 2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC). pp. 244–251 (Oct 2017). https://doi.org/10.1109/CyberC.2017.37
9. Amazon Web Services: Amazon Web Services: Overview of Security Processes. https://d1.awsstatic.com/whitepapers/Security/AWS_Security_Whitepaper.pdf (May 2017), accessed: 02 Sep. 2018
10. Aviram, N., Schinzel, S., Somorovsky, J., Heninger, N., Dankel, M., Steube, J., Valenta, L., Adrian, D., Halderman, J.A., Dukhovni, V., Käsper, E., Cohney, S., Engels, S., Paar, C., Shavitt, Y.: Drown: Breaking tls using sslv2. In: USENIX Security Symposium. pp. 689–706 (2016)
11. Barker, E.B., Dang, Q.H.: SP 800-57 Pt3 R1. Recommendation for Key Management, Part 3: Application-Specific Key Management Guidance. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf (Jan 2015), accessed: 02 Sep. 2018
12. Barker, E.B., Roginsky, A.L.: SP 800-131A R1. Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar1.pdf (Nov 2015), accessed: 02 Sep. 2018
13. Böck, H., Somorovsky, J., Young, C.: Return of bleichenbacher's oracle threat (robot). In: Proceedings of the 27th USENIX Conference on Security Symposium. pp. 817–832. SEC'18, USENIX Association, Berkeley, CA, USA (2018), http://dl.acm.org/citation.cfm?id=3277203.3277265, accessed: 02 Sep. 2018
14. Chandu, Y., Kumar, K.S.R., Prabhukhanolkar, N.V., Anish, A.N., Rawal, S.: Design and implementation of hybrid encryption for security of iot data. In: 2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon). pp. 1228–1231 (Aug 2017). https://doi.org/10.1109/SmartTechCon.2017.8358562
15. Cloud Security Aliance: The Treacherous 12 - Top Threats to Cloud Computing + Industry Insights. https://cloudsecurityalliance.org/download/artifacts/top-threats-cloud-computing-plus-industry-insights/ (Oct 2017), accessed: 02 Sep. 2018
16. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM J. Comput. **33**(1), 167–226 (Jan 2004). https://doi.org/10.1137/S0097539702403773, http://dx.doi.org/10.1137/S0097539702403773

17. Duong, T., Rizzo, J.: Here come the xor ninjas. White paper, Netifera (2011)
18. Durumeric, Z., Li, F., Kasten, J., Amann, J., Beekman, J., Payer, M., Weaver, N., Adrian, D., Paxson, V., Bailey, M., Halderman, J.A.: The matter of heartbleed. In: Proceedings of the 2014 Conference on Internet Measurement Conference. pp. 475–488. IMC '14, ACM, New York, NY, USA (2014). https://doi.org/10.1145/2663716.2663755, http://doi.acm.org/10.1145/2663716.2663755
19. Fardan, N.J.A., Paterson, K.G.: Lucky thirteen: Breaking the tls and dtls record protocols. In: 2013 IEEE Symposium on Security and Privacy. pp. 526–540 (May 2013). https://doi.org/10.1109/SP.2013.42
20. Google: Encryption at Rest in Google Cloud Platform. https://cloud.google.com/security/encryption-at-rest/default-encryption/resources/encryption-whitepaper.pdf (Aug 2016), accessed: 02 Sep. 2018
21. Google: Encryption in Transit in Google Cloud. https://cloud.google.com/security/encryption-in-transit/resources/encryption-in-transit-whitepaper.pdf (Nov 2017), accessed: 02 Sep. 2018
22. Google: Google Infrastructure Security Design Overview. https://cloud.google.com/security/infrastructure/design/resources/google_-infrastructure_whitepaper_fa.pdf (Jan 2017), accessed: 02 Sep. 2018
23. Kaaniche, N., Laurent, M., Barbori, M.E.: Cloudasec: A novel public-key based framework to handle data sharing security in clouds. In: 2014 11th International Conference on Security and Cryptography (SECRYPT). pp. 1–14 (Aug 2014)
24. Khanezaei, N., Hanapi, Z.M.: A framework based on rsa and aes encryption algorithms for cloud computing services. In: 2014 IEEE Conference on Systems, Process and Control (ICSPC 2014). pp. 58–62 (Dec 2014). https://doi.org/10.1109/SPC.2014.7086230
25. Kivinen, T., Kojo, M.: More modular exponential (modp) diffie-hellman groups for internet key exchange (ike). https://tools.ietf.org/html/rfc3526 (2003), accessed: 02 Sep. 2018
26. Liang, C., Ye, N., Malekian, R., Wang, R.: The hybrid encryption algorithm of lightweight data in cloud storage. In: 2016 2nd International Symposium on Agent, Multi-Agent Systems and Robotics (ISAMSR). pp. 160–166 (Aug 2016). https://doi.org/10.1109/ISAMSR.2016.7810021
27. Microsoft: Trusted Cloud: Microsoft Azure Security, Privacy and Compliance. http://download.microsoft.com/download/1/6/0/160216AA-8445-480B-B60F-5C8EC8067FCA/WindowsAzure-SecurityPrivacyCompliance.pdf (Apr 2015), accessed: 02 Sep. 2018
28. Möller, B., Duong, T., Kotowicz, K.: This poodle bites: exploiting the ssl 3.0 fallback, 2014. Security Advisory (Sep 2014), accessed: 02 Sep. 2018
29. Neuman, D.C., Hartman, S., Raeburn, K., Yu, T.: The Kerberos Network Authentication Service (V5). RFC 4120 (Jul 2005). https://doi.org/10.17487/RFC4120, https://rfc-editor.org/rfc/rfc4120.txt
30. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Aug 2018). https://doi.org/10.17487/RFC8446, https://rfc-editor.org/rfc/rfc8446.txt
31. Rescorla, E., Dierks, T.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Aug 2008). https://doi.org/10.17487/RFC5246, https://rfc-editor.org/rfc/rfc5246.txt