STYX: A Trusted and Accelerated Hierarchical SSL Key Management and Distribution System for Cloud Based CDN Application

Changzheng Wei
Intel Asia-Pacific Research & Development Ltd. Shanghai, China.
changzheng.wei@intel.com

Jian Li*
School of Software,
Shanghai Jiao Tong University, China
li-jian@sjtu.edu.cn

Weigang Li
Intel Asia-Pacific Research & Development Ltd. Shanghai, China.
weigang.li@intel.com

Ping Yu
Intel Asia-Pacific Research & Development Ltd. Shanghai, China.
ping.yu@intel.com

Haibing Guan
Shanghai Key Laboratory of Scalable Computing and Systems,
Shanghai Jiao Tong University, China
hbguan@sjtu.edu.cn

ABSTRACT

Protecting the customer's SSL private key is the paramount issue to persuade the website owners to migrate their contents onto the cloud infrastructure, besides the advantages of cloud infrastructure in terms of flexibility, efficiency, scalability and elasticity. The emerging Keyless SSL solution retains on-premise custody of customers' SSL private keys on their own servers. However, it suffers from significant performance degradation and limited scalability, caused by the long distance connection to Key Server for each new coming end-user request. The performance improvements using persistent session and key caching onto cloud will degrade the key invulnerability and discourage the website owners because of the cloud's security bugs.

In this paper, the challenges of secured key protection and distribution are addressed in philosophy of "Storing the trusted DATA on untrusted platform and transmitting through untrusted channel". To this end, a three-phase hierarchical key management scheme, called STYX[1] is proposed to provide the secured key protection together with hardware assisted service acceleration for cloud-based content delivery network (CCDN) applications. The STYX is implemented based on Intel Software Guard Extensions (SGX), Intel QuickAssist Technology (QAT) and SIGMA (SIGn-and-MAc) protocol. STYX can provide the tight key security guarantee by SGX based key distribution with a light overhead, and it can further significantly enhance the system performance with QAT based acceleration. The comprehensive evaluations show that the STYX not only guarantees the absolute security but also outperforms the direct HTTPS server deployed CDN without QAT by up to 5x throughput with significant latency reduction at the same time.

CCS CONCEPTS

• **Networks → Cloud computing**; *Network privacy and anonymity*; *Location based services*; • **Computer systems organization → Cloud computing**; • **Hardware** → *Emerging architectures*;

KEYWORDS

Distributed applications, Security and Protection, QuickAssist Technology (QAT), Software Guard Extensions (SGX), Cloud Based Content Delivery Network

ACM Reference Format:
Changzheng Wei, Jian Li, Weigang Li, Ping Yu, and Haibing Guan. 2017. STYX: A Trusted and Accelerated Hierarchical SSL Key Management and Distribution System for Cloud Based CDN Application. In *Proceedings of SoCC '17, Santa Clara, CA, USA, September 24–27, 2017,* 13 pages. https://doi.org/10.1145/3127479.3127482

1 INTRODUCTION

Cloud-based content delivery network (CCDN) promotes the content delivery as a cloud service model [15][56][38] that leverages the cloud resources to provide the high scalability and low latency network on a large number of surrogate systems at globally distributed locations [56], such as Google Cloud CDN [26] and Amazon CloudFront [5].

The prerequisite of persuading the content owners to migrate their data onto cloud is the promise that the companies will obtain all benefits of the cloud infrastructure (DDoS attack mitigation, load balancing, WAN optimization), without any leakage risk of their private SSL keys [36][20][50][15]. An SSL key is the critical data that allows an organization to establish a secure connection with the authenticated customers. The leakage of an organization private SSL key will enable the malicious users to authenticate as the organization or spoof identity and intercept the traffic. The

*Corresponding Author
[1]STYX is the river in Greek mythology that formed the boundary between Earth and the Underworld. Oaths made by this river brings something 'worse than death' to the Oath Bearer if not fulfilled. If anyone bathes in the River STYX, that person becomes invulnerable to most physical attacks, like Achilles.

wide spread rumors about the security bugs and the curious administrator actions make the customers hesitate to share their site's SSL key with the cloud. The key security threat, such as the recently reported Heartbleed security bug in 2014 for example, incurred a buffer over-read vulnerability in Transport Layer Security (TLS) protocol on public server [23][41][61].

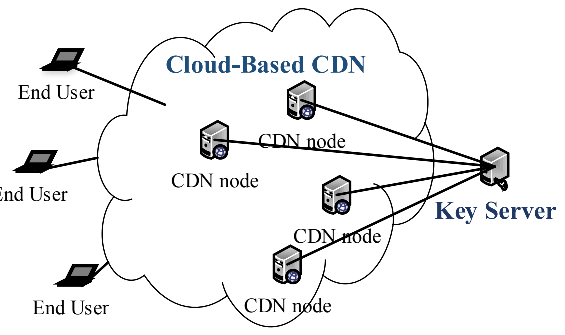


Figure 1: Keyless SSL model overview

In order to protect the key, the "Keyless SSL" solution was proposed by CloudFlare in 2015 [19]. The Keyless SSL solution allows the website owner to use the CloudFlare's CDN without requiring the customers to give up the custody of their private keys. Instead, when an end-user (client) tries to setup HTTPS connection with the website through the CDN node [16][38], the CDN node will transfer the HTTPS request back to the web server (key server) as it has never touched the website's private key.

As shown in Fig. 1, a web server generates a secret session key for the HTTPS request and sends it back to a CDN node. The CDN nodes are geographically distributed around the world that allow the end-users to communicate with a CDN node closest to them so as to achieve the low latency and high throughput. Keyless SSL can attain a significant performance enhancement from the cloud infrastructures since all messages except one are traveling over the short link between the CDN server and the end-users. The single long round trip to the key server is the authentication step that happens only once per session.

However, the HTTPS connection is required to be created by the web server, which makes the key server to be a bottleneck in serving the clients' HTTPS requests. Moreover, the short duration of the HTTPS sessions is actually the primary case with short session lifetime. Concretely, Google analytic reported that the average session duration is only 52 seconds with 1.45 page per session, 53% of the sessions last only for 0-10 seconds and the bounce rate is higher than 80% [25][8]. The short session will invoke the high frequent connections to the key server with the long round trip, which severely impacts the performance of Keyless SSL.

In this paper, a three-phases hierarchical key distribution system, called STYX, is proposed. The STYX system consists of one Key Distribution Center (KDC) and large-range distributed Key Sub-Centers (KSC), and provides the following advanced features:

(1) The KDC can deliver the SSL private keys to the KSCs through a secure channel constructed by remote attestation and provisioning protocol.
(2) The provisioned SSL private key will be protected by the KSC in a Trusted Execution Environment (TEE) in order to guarantee the security of the SSL private key from any unauthorized access or modification including the KSC node owner and administrator.
(3) The KDC can keep all KSCs in loyal with STYX's powerful revocation services.
(4) The KSCs located in distributed nodes can serve the nearby end-users with secured connection directly with provisioned SSL private key without invoking the KDC to achieve the efficient and effective HTTPS service.
(5) STYX introduces a secure channel mechanism between TEE on CPU and peripheral, which can offload CPU intensive operations onto hardware based accelerator. This will significantly improve the HTTPS performance without threat of the sensitive data leaks.

The implementation of STYX relies on Intel's trusted computing technologies in order to produce trusted and secured execution environment on untrusted platforms [39][59] [44]. Concretely, the STYX employs QAT (QuickAssist Technology)[1], SGX (Intel®Software Guard Extensions) [21][2][3] with remote attestation and provision services of Enhance Private ID (EPID) [11][12][13]. The Key Sub-Center accelerates the HTTPS communication performance by offloading the CPU-intensive calculations to QAT hardware device, including RSA decryption, RSA sign operation or other public key algorithms such as ECDH and ECDSA. This hardware assisted accelerated calculation can significantly increase the system efficiency and performance of HTTPS service in CDN node.

The rest of the paper is organized as follows: In Section 2, the challenges of SSL key leakage are introduced as the recent main obstacles for wide expansion of CCDN technology. Moreover, the current Keyless SSL solution is also described along with its shortcomings, which highlights the proposed design motivation. In Section 3, the design of STYX is proposed and the functionalities of its main modules and principle working procedures are described. Section 4 states the prototype implementation details of STYX, Section 5 evaluates the advantages of the proposed STYX by using experiments. Lastly, Section 6 summaries the paper and discusses the future work.

2 BACKGROUND AND MOTIVATION

2.1 Private Key Protection

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are the de-facto standards for securing Internet transactions [53][15] that can provide trusted identities via certificate chains and encrypted communication. Therefore, it must be guaranteed that the private keys used in SSL are not compromised by third parties.

If an organization's private SSL key is lost, the malicious user can authenticate as if he was it, spoof identity or intercept traffics. Therefore, it's a critical security event for an organization (a financial institute, for example) to lose its private key, since it will incur the widespread public embarrassment and loss of trust. Unfortunately, there are various threats of private key loss with serious event cases.

2.2 Heartbleed Challenge

The Heartbleed problem was publicly announced in April 2014 as one OpenSSL bug [23][6][61], which makes it possible for the attackers to inspect servers' memory contents, thereby potentially (and undetectably) revealing the servers' private keys [47].

For example, PKCS#8 [35][48][55] provides a scenario to protect the private key by encrypting the private key file on the disk with a user password. It can guarantee that the private key file is secured in the static storage, but at run time, the private key needs to be decrypted first and exported to the system memory. This introduces a risk that the private key can be attacked by memory snooping, or other memory attacks, so as to incur so-called "Heartbleed" vulnerability [61].

Recently, it has been reported that up to 17% of all HTTPS web servers are vulnerable to Hearbleed problem [41]. Even the fixed version of OpenSSL has been released with continuous deployment, the worries about other HeartBleed alike security bugs have become the most challenging problem [51][32][27] to persuade the customers (website owners) to migrate their contents onto the public cloud or third-party servers. Consequently, the sensitive data like private key should not be exposed as plaintext in memory. It is necessary to protect the private key in a "secure" memory place that is vulnerable to snooping from malicious hacker or curious infrastructure administrator.

2.3 Keyless SSL

The keyless SSL [19] provides an effective key protection solution by retaining on-premise custody of customers' private keys, which is compatible with the existing key management infrastructure constructed using the standard SSL algorithms.

CloudFlare's "Keyless SSL" solution was proposed in 2015. It allows websites to use the CDN cloud infrastructure without requiring the customers to give up the custody of their private keys. The request flow for Keyless SSL transactions with TLS protocol RSA handshake including five steps is shown in Fig. 2:

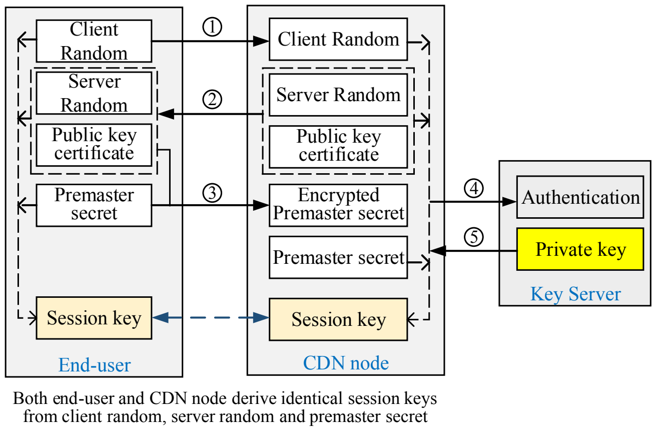


Figure 2: Keyless SSL transaction details

Step 1 & 2. The end-user connects to the closest CDN edge node and the CDN node responds with the certificate file as general TLS/SSL protocol.

Step 3. The end-user generates a secret key and sends the encrypted secret key to CDN node with SSL certificate.

Step 4. The CDN node connects with the key server and sends the encrypted secret key after authentication.

Step 5. The key server decrypts the secret key, and sends it back to the CDN node over a preset secure tunnel.

Thus, the end-user and the CDN node have established an HTTPS connection with a shared session key for website resource. Keyless SSL requires the CDN nodes to decrypt, inspect and re-encrypt the traffic for transmitting back to a customer's origin key server, while the key server keeps the exclusive access to the private key.

2.4 Performance Impact of Keyless SSL

Keyless SSL adds an additional connection between CDN nodes and the key server when an end-user connects to a Keyless SSL protected website. This incurs a significant overhead to handle the concurrent end-user requests and an increased latency due to the long round trip connection between the CDN node and the key server. The exact delay depends on the network latency between the user's CloudFlare edge server and the origin, and is generally in the tens or hundreds of milliseconds.

There are two types of technologies to mitigate these potential performance impacts using the Keyless SSL [18]:

(1) **Session cache and session tickets** allow the Keyless SSL to reuse the previously negotiated session symmetric keys, without requiring a new connection to the Keyless SSL client if a session has previously been negotiated with that user.
(2) **Persistent connections** allow the connection between the keyless client and the CDN node to remain open permanently, in order to eliminate the connection setup overhead when a user connects to the site.

However, the persistent connection is not a practical assumption for HTTPS service because the connection sessions normally have the short lifetimes as stated in the previous section. More importantly, session cache and session ticket on the cloud CDN nodes suffer from spoofing attacks and session hijacking threats [42], which will severely degrade the security advantages obtained by the keyless SSL method from on-premise key custody.

Moreover, the website owner has many worries about the key leakages from the curious cloud administrator and malicious attacks. Therefore, the customers will insist on keeping custody of their private keys on their own server without any compromise on caching the keys on cloud to improve the performance.

2.5 Design Motivation

The key requirement is to improve the end-user SSL access performance by caching the keys on the CDN nodes, while ensuring the absolute security of SSL private keys.

Consequently, the system design policies include: (i) a key server distributes the SSL private key to each joining CDN node to serve the geographically nearby end-user requests. (ii) The distributed key hosted on the CDN node must be invulnerable to any security attack including the malicious hacker, the curious cloud infrastructure owner and the cloud administrator. (iii) Moreover, the key server should be capable to revoke the keys at any time. (iv) On the other hand, the CDN cloud infrastructure is authenticated by key server with anonymous attestation to keep the cloud based CDN infrastructure in safe unlinkability to the key server [49][24].

This implies the employment of trusted execution environment (TEE) to guarantee the integrity and confidentiality of secured computations and users' data on a remote node, against the malicious unprivileged or even privileged software, such as the operating system and hypervisor [29][43][7][62][63].

These requirements motivated the authors to propose the STYX scheme that ensures the security of the key provisioned on cloud CDN nodes. The design philosophy follows the attributes of Styx river in Greek mythology, that grants the power to a man and force him to keep the oath. It should be noticed that the performance of the STYX is not limited at achieving the key caching of Keyless SSL performance mitigation, but the STYX can even significantly outperform the direct HTTPS server's in terms of both throughput and latency by offloading the CPU-intensive workload to hardware-assisted accelerator through a secured mechanism.

3 STYX DESIGN

The overall architecture of the STYX is a hierarchical key distribution scheme with the optimizations on service performance for end-user requests by leveraging the cloud distributed infrastructure with trustful fidelity and hardware assistance. The STYX architecture is illustrated with the working procedure that mainly consists of three phases in Fig. 3.

3.1 Overall Architecture

The overall STYX architecture is composed of one centralized Key Distribution Center (KDC) and widely distributed Key Sub-Centers (KSC) hosted on CDN nodes. The key operation and management mechanism of STYX include three phases:

- **Phase 1**: A KSC constructs the Key Store module as a SGX application enclave to store and protect the SSL private key from any disclosure or modification. The centralized KDC authenticates the KSC and provisions the SSL private key securely by combining a general "SSL key request/allocate protocol" with the secured "EPID provisioning and attestation service" and "Elliptic Curve Diffie-Hellman" key exchange method.
- **Phase 2**: The acquired SSL private key in KSC's Key Store is accredited to Intel's "QuickAssist Technology" (QAT) [1] hardware through a secured SIGMA protocol [37] channel, in order to leverage the QAT device to accelerate the HTTPS services by offloading its CPU-intensive calculations.
- **Phase 3**: Each KSC can authenticate and service the end-user requests directly to achieve the exceptional performance, with the modified and optimized web service implementation, such as Nginx HTTPS server [46] and OpenSSL library on top of the QAT acceleration engine insides the cloud-based CDN nodes.

STYX Security: The STYX provides an industrial trusted key management prototype system with **invulnerability**, **anonymity**, **unforgeability** together with **revocation** capability. Generally speaking, the STYX can provide the security in leveraging the Intel SGX primitives and security attestation and authority techniques from several aspects:

(1) The SSL keys on the KSCs are protected by Key Store module powered by the SGX technology that is invulnerable to any attack and spoofing from outside hackers and cloud administrator.
(2) The SSL private keys are transmitted securely through a symmetric encryption channel established by EPID attestation service in conjunction with ECDH secret key exchange protocol.
(3) The KDC has full management capabilities to revoke a selected KSC's authentication by Key Store enclave revocation and/or SSL key revocation. Additionally, it has the revocation capacity to terminate the services for a group or the entire KSCs without impacting the non-revoked members.
(4) The STYX makes the KDC attest the KSCs based on anonymous attestation [10] mechanism using EPID technology. This means that the STYX provides unlinkability that KDC knows only what it is allowed to know about cloud's information. Thus, the STYX also protects the cloud infrastructure from any malicious customers who migrate the contents onto it, besides the protection on customer's SSL private key.

The STYX is implemented with the synthetical employments of modern Intel's trusted execution environment (TEE) technologies, such as SGX [3] [52] [60], EPID [11] and QAT [1]. The SGX was first issued in 2015 with Windows SDK, and its Linux SDK only became available in mid of 2016. To the authors' best knowledges, the STYX is the first prototype to push the Intel's TEE technology series into the area of enterprise applications. Consequently, STYX has the high demonstrative value along with the prominent advantages in terms of security guarantee and performance improvements for SSL private key management in CCDN application.

The technical implementations of STYX will be depicted in detail in the next section. This section introduces the technology independent design of STYX, including key management protocol and the functionalities of its major modules, in what follows.

3.2 Key Distribution Center (KDC)

The KDC is normally maintained and located by the customer (content owner of CCDN) in a secure place. The KDC is responsible for the private keys management and distribution on the largely distributed KSCs. It consists of the "*Key Provision and Auditing" module*, "*KSC Identity Authority*" module, and "*SSL Private Key Manager*" module.

Key Provisioning and Auditing (KPA) module is the main function in the KDC that deals with the incoming SSL key requests from the KSCs. The KPA will setup a secure channel using EPID based ECDH key exchange protocol, which makes the KDC and the KSC share a secret key (K_s). Then, the KPA will invoke the SSL Key Manager module to obtain a requested SSL private key (K_d) that will be transmitted securely by symmetric encryption with (K_s). In addition, the KPA will regularly audit the KSCs to keep them in loyal and trigger the revocation capacities to annul the authority of the KSC or revoke a provisioned SSL Private Key if necessary.

KSC Identity Authority (KIA) module runs a group of signature authentication algorithms using Intel Attestation Service. Once a KSC infrastructure is attested as an SGX platform, the KIA module will distribute an unforgeable EPID member private

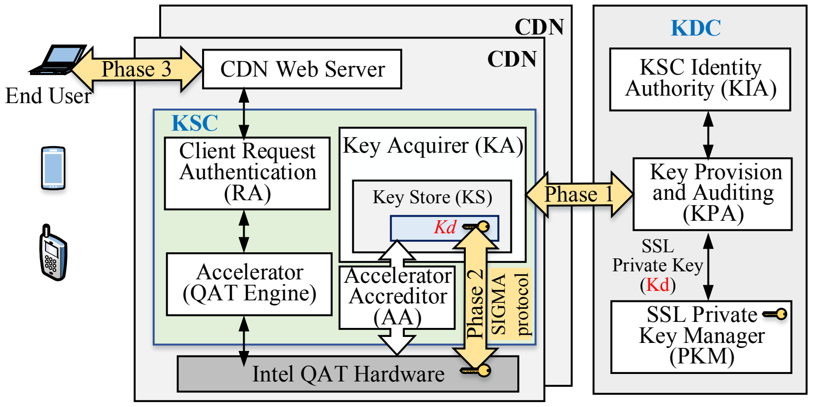


Figure 3: Overall architecture of STYX

key ($K_{EPID\_Private}$) to the KSC. Then the KSC will initiate an EPID attestation in conjunction with ECDH key exchange protocol [34][14][28][40] to setup a secured channel with shared secret for transmission encrypted data between the KDC and the KSC. The KIA module will also organize the signed signatures for all attested Key Stores on distributed KSCs, and keep the revocation control on one or a group of compromised KSCs.

SSL Private Key Manager (PKM) module performs the traditional SSL private key handshake protocol for a jointing KSC SSL key request. It will authenticate the KSC's SSL key request and select the SSL private key (K_d) for every joining KSC. At the same time, the SSL Private Key Manager module is also responsible for maintaining the revocation and reissuing control capacities for one or a group of SSL private keys and certificate files.

3.3 Key Sub-Center (KSC)

The KSC hosted on the CDN node needs to acquire a SSL Private Key from the KDC and keep it invulnerable of any security attack. The KSC provides high fidelity for private key with hardware assisted service in order to effectively handle the SSL/TLS connections from the clients without involving the KDC. A KSC consists of *Key Acquirer* module with SGX powered *Key Store*, "*Client Request Authority*" module and "*Accelerator Accreditor*" module.

Key Acquirer (KA) module is the main function of the KSC that is responsible for requesting the SSL private key from the KDC. It first makes the KSC infrastructure attested and setups a secure channel with the KDC for key delivery. The KA has a sub-module, called **Key Store module**, to protect the obtained SSL private key in an Intel SGX enclave [2].

Accelerator Accreditor (AA) module is responsible for securely accrediting the SSL private key from Key Store to the internal RAM of QAT hardware. It uses the SIGMA protocol [37] to setup a secure channel to protect the private key from snooping by adversary outside the SGX enclave in memory.

Client Request Authentication (RA) module employs hardware based Intel QAT accelerator [1] to increase the SSL/TLS handshake performance.

Performance Enhancement: The KSC can achieve the enhanced performance for HTTPS connections, since the CDN node can handle the web service with SSL/TLS connection using the SSL private key in Key Store without involving the KDC. Moreover, it can offload the CPU-intensive calculations, such as RSA calculation, to the QAT hardware through a secured mechanism to increase the service performance. Notice that this modern QAT hardware assisted performance enhancement engine enables the STYX to significantly outperform not only the Keyless SSL solution, but also the direct HTTPS server on the CDN node. These performance efficiencies will be evaluated in Section 5.

4 STYX IMPLEMENTATION

This section describes the detailed implementation of the STYX architecture. The STYX is a secured key management and hierarchical distribution protocol system that is implemented based on Intel SGX, Intel QAT, OpenSSL, and Nginx. The deployment coding and guidance are available on our lab website sdic.sjtu.edu.cn/projects/STYX with continuous updates.

The key management and distribution procedure are described in three designed phases. Table 1 lists the terminologies of employed keys that will be used in the STYX.

$K_{EPID\_Private}$	EPID member Private Key
K_{EPID_Public}	EPID Group Public key
K_s	Share secrete key BW KSC & KDC
K_d	SSL private key for delivery
$K_{Provision}$	Encrypted K_d from KDC to KSC
K_{Export}	Encrypted K_d from KS to QAT

Table 1: STYX terminology for key distribution

4.1 Phase 1: Key Provision and Management

Overall, Phase 1 implementation aims at sending the required SSL private key (K_d) from the KDC to the KSC through a secured channel, which is established by a symmetric encryption with a negotiated shared secret key (K_s) between the KSC's Key Store (KS) and the KDC's Key provisioning and auditing (KPA) module.

First, the Quoting Enclave of Intel SGX driver is leveraged that can access the EPID member private key provisioned in âĂIJjoinâĂİ procedure. Second, an application enclave is developed to implement the Key Store in order to protect the private key in the KSC with memory encryption technology. The Key Acquirer will setup a secure channel with the KDC to securely receive the private keys. At last, the KDC keeps the capability to revoke the provisioned SSL private key and KSC attestation when necessary.

Concretely, this procedure is a general key request/response procedure wrapped with Intel's Software Guard Extensions (SGX) attestation service [33] as well as ECDH key exchange method [14]. The detailed key distribution from the KDC to the KSC in Phase 1 includes five stages as shown in Fig. 4.

4.1.1 Setup: In this stage, the SSL Private Key Manager module in the KDC creates a set of SSL private keys and certificate files for the HTTPS applications that will be deployed on CCDN and the KSC Identity Authority module organizes an EPID group. In the STYX, the KSC Identity Authority is implemented by IAS (Intel Attestation Service) API [2] that provides restful API for users.

4.1.2 Join: This is **KSC Infrastructure Attestation** stage. When a KSC want to join in the KDC system, it will raise a join request to the KSC Identity Authority. The KSC Identity Authority will authenticate the KSC and assign an EPID member private key to it. In this procedure, the Intel SGX technology is leveraged. Since the CPU with Intel SGX has a pre-provisioned root key in manufacture phase demonstrating that it is a genuine Intel SGX CPU at a specific TEE. The Quoting Enclave of Intel SGX is leveraged. The Quoting Enclave is a unique platform enclave provided by Intel SGX driver that can access the EPID member private key. The stage follows the general EPID attestation procedure to allow the KDC to gain confidence that the KSC is a platform with SGX based trusted execution environment (TEE).

The stage results in that the Quote enclave on the KSC obtains the $K_{EPID\_Private}$ provisioned by the KSC Identity Authority in the KDC.

4.1.3 Attestation: The SGX application enclave is leveraged to implement the Key Store (KS) and the Key Acquirer will initiate the SSL private key request to the KDC. Intel SGX provides CPU-hardened "application enclaves" as protected execution areas even on compromised platforms. In order to ensure that the KDC provisions the requested SSL private key into a remote Key Store (KS), it is paramount to setup a secure channel to deliver the SSL private key.

The secured channel is implemented with Intel's SGX's attestation service [33][2][12] with Elliptic Curve Diffie-Hellman (ECDH) key exchange method [14][28][34].

The stage result of Attestation is KDC and KSC negotiating a share secrete key K_s. To this end, the Attestation Stage is divided in to seven sub-steps as follows.

①: The Key Acquirer raises an ***SSL Private Key Request*** to remote Key Provision and Auditing (KPA) module in the KDC.

②: The KPA receives the request and invokes the KIA to "***CHALLENGE***" the Key Store module in the KSC.

③: The Key Store module, as a SGX application enclave, generates a ***REPORT*** of itself that includes the production ID α, security version β, hash value of the enclave measured by the CPU γ, and an ephemeral public key (here is an ECDH public key, specifically) δ. Synthetically $REPORT = (\alpha, \beta, \gamma, \delta)$.

④: This report is forwarded to the Quoting enclave to be signed by using the EPID member private key $K_{EPID\_Private}$, and return a '***SIGNED REPORT***'.

⑤: The KPA module in the KDC receives the *SIGNED REPORT* from the Key Store and forwards it to the KIA that uses Intel Attestation Service to ***VERIFY the Signature***. The KIA module verifies that the KS (Key Store) is SGX powered TEE enclave with the information α in the *REPORT*.

⑥: If the signature verification succeeds, the KPA further checks the *REPORT* contents. Known that α, β, γ in the *REPORT* are used to verify the KS enclave, so the KPA will check the *REPORT* with other policies: if the production ID belongs to KDC, if the security version is aligned to KDC's security policy and most importantly, it will check if the hash value matches with the local record of the key protection enclave in the KDC (since the enclave in the KSC is developed and published by KDC, the KDC must maintain the hash value of each published enclave) to confirm there is no tempering during the enclave publication and launching in target platform.

⑦: *ECDH Retrieval*: The KDC confirms the key protection enclave (KS) in the KSC secure, then it retrieves the ephemeral public key δ from the report and negotiates a shared secret with the KPA enclave using the ECDH method[40][54][9][17]. At last, both the KDC and the KSC can derive the same encryption key (K_s).

Notice that the REPORT is generated by the KS module encapsulating only the necessary information of KSC, which is capable to protect the cloud infrastructure by means of an anonymous attestation to the KDC based EPID scheme [13].

4.1.4 Provision: In this stage, the SSL private key is delivered to the KSC. After establishing a secured channel, the KPA module invokes the SSL Private Key Manager (PKM) module to obtain the required SSL private key (denoted simply as K_d) for current requesting KSC and encrypts it with K_s to transmit through the channel as $K_{Provision}$.

$$K_{Provision} = ENC_{K_s}(K_d). \tag{1}$$

The Key Store receives the encrypted private key $K_{Provision}$, and decrypts it using the same encryption key (K_s) derived from the shared secret inside the enclave to obtain K_d.

$$K_d = DEC_{K_s}(K_{Provision}). \tag{2}$$

The stage result is that the SSL private key is delivered into the KS protected by Intel SGX memory encryption technology. Until now, the SSL private key is securely distributed from KDC to KSC and is protected in the SGX enclave powered Key Store module.

4.1.5 Revocation: This stage involves the revocation of the SSL private key and the Key Store. Another important feature of the KDC is the capability to revoke the provisioned SSL private keys on the KSC once a malicious behavior is detected. This is an important feature of STYX to ensure the KSCs be loyal to the KDC and will never abused by cloud administrator.

If the KSC is occupied by malicious attacker, the KDC must be able to revoke the KSC and refresh the private keys which may be stolen in that enclave. To this end, the KDC's KPA module can regularly audit the distributed KSCs, and has the capacity to withdraw

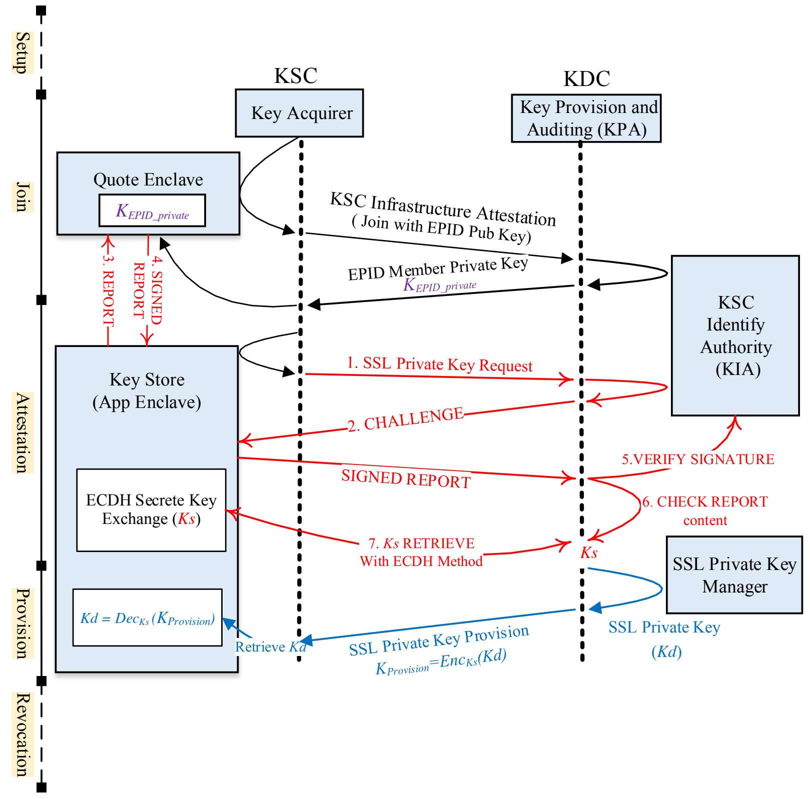


Figure 4: Key distribution from KDC to KSC in Phase 1

the authentication whenever necessary. Consequently, the STYX revocation service includes two levels: *Key Store Enclave Revocation* and distributed *SSL Key Revocation*ãĂĆ

SSL Key Revocation will mainly concern the SSL certificate reissues and revocations methods [61] [24]. This is a scope with various mature results to support this capacity and the details are omitted here. ***Key Store Revocation*** will use the EPID member management mechanism [10] that includes three revocation methods:

(1) EPID member private key based revocation: if the KDC finds the EPID member private key of KSC be published, the KDC can revoke the EPID member private key directly.
(2) Signature based revocation: if the KDC suspects that any KDC has been occupied but does not know its EPID member private key, the KDC can revoke the signature from that KSC.
(3) Group based revocation: if lots of members in a EPID group are revoked, the KDC can revoke the entire group and reassign EPID member private key to the one who should not be revoked.

Notice that the STYX only provides the revocation capacity here, and the customers can specify their useful policy according to their politic, economic and administrative interests.

4.2 Phase 2: QAT Accelerator

The KSC's implementation employs the Intel QuickAssist Technology (QAT) hardware based accelerator to offload the CPU-intensive operations such as the private key decryption. The QAT based accelerator leverages the typical usage of the QAT to offload the RSA operation from a web server during the SSL/TLS connection setup phase. Since the RSA operation [22] is extremely CPU-intensive workload, this offloading to QAT accelerator can greatly save the CPU utilization and increase the number of concurrent HTTPS connections for the web server.

4.2.1 Key Accrediting Module with SIGMA Protocol. In Phase 2, the Key Accreditor module is implemented to export the SSL private key from the Key Store enclave to QAT hardware device through an encrypted channel using SIGMA (SIGn-and-MAc) protocol [37].

SIGMA serves as the cryptographic basis for the signature-based modes of the standardized Internet Key Exchange (IKE) protocol. The Key Accreditor (KA) module and the QAT will exchange an ephemeral public key, based on the SIGMA's core cryptographic security that guarantee the independent encryption of the identities. The SIGMA channel makes the KS enclave and the QAT hardware share a secret with Diffie-Hellman exchange. Using the shared secret, they both can derive the same encryption key (K_c) independently.

Then, the key protection enclave encrypts the private key with K_c and exports it outside in DRAM as

$$K_{export} = ENC_{K_c}(K_d). \tag{3}$$

When the end-user sets up an SSL/TLS connection with the application service in the CDN node, the HTTPS service makes the private key request to the hardware accelerator along with the encrypted private key in the hardware accelerator. The encrypted SSL private key will be decrypted as:

$$K_d = DEC_{K_c}(K_{Export}). \tag{4}$$

In the entire end-user serving process, the KSC will not involve the remote the KDC. The Key Store protects the SSL private key by SGX enclave and the accredited QAT hardware can decrypt it making it inaccessible to the CDN node owner. Malicious application or roger administrators cannot retrieve the SSL private key into plaintext.

4.3 Phase 3: On CDN Software Stack

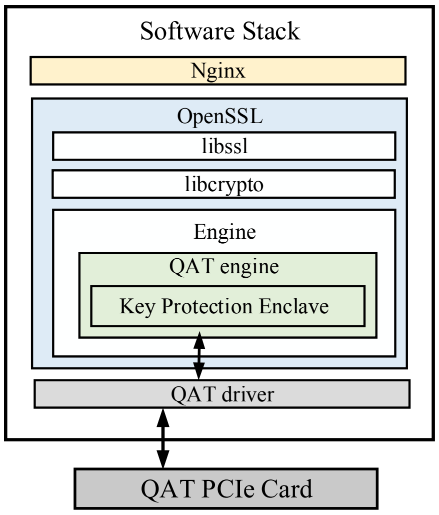


Figure 5: Software stack implementation with QAT engine, OpenSSL and Nginx

The entire software stack for KSC and KDC (excluding the Intel Attestation Service) is implemented based on OpenSSL and Nginx, as shown in Fig. 5.

Nginx provides an application level service (such as HTTPS) [46]; libssl is the SSL/TLS protocol stack library and libcrypto is the cryptography algorithm library provided by the OpenSSL. The OpenSSL is an open source project with general-purpose cryptography library that provides a robust, commercial-grade and full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols [45]. The OpenSSL has a common engine framework that can export the unified API to libcrypto library for crypto algorithm. All the proto of crypto algorithms should follow that framework to export their capability including software based and hardware based engines. The existing Nginx and OpenSSL API are leveraged and the proposed prototype implementation is transparent to the applications, Nginx and OpenSSL stack.

QAT engine is a hardware engine for Intel QAT device. There are standard engine API for applications that want to retrieve the SSL private key from a hardware accelerator, which is already supported in Nginx standard configuration file. So our implementation is transparent to the existing Nginx and OpenSSL stack. When Nginx starts service at the first time, it will retrieve the SSL private key from QAT engine. The SSL private key is missing for the first Nginx service and it will trigger the KA module to request a SSL private key from KDC as mentioned in Phase 2. After obtaining the SSL private key in KS module, Key Accreditor module will export the encrypted private key to QAT engine for accelerating Nginx services.

4.3.1 Web Server Software Stack Implementation. In order to make the Nginx web wervice engine be able to leverage the QAT hardware accelerator, the KSC employs the latest OpenSSL release (v1.1.0) for using the "Fibre mechanism" [57] that can handle the SSL/TLS connection asynchronously.

Asynchronous OpenSSL supports a parallel processing model at the cryptographic level for SSL/TLS protocols, which allows the cryptographic transforms to be processed on dedicated hardware engines or on separate logical cores. Consequently, this also enables the SSL/TLS protocol stack and applications to seamlessly run other tasks . Moreover, the asynchronous operation also enables the single-threaded applications to efficiently handle multiple SSL connections because individual flows are not blocked while using the hardware acceleration.

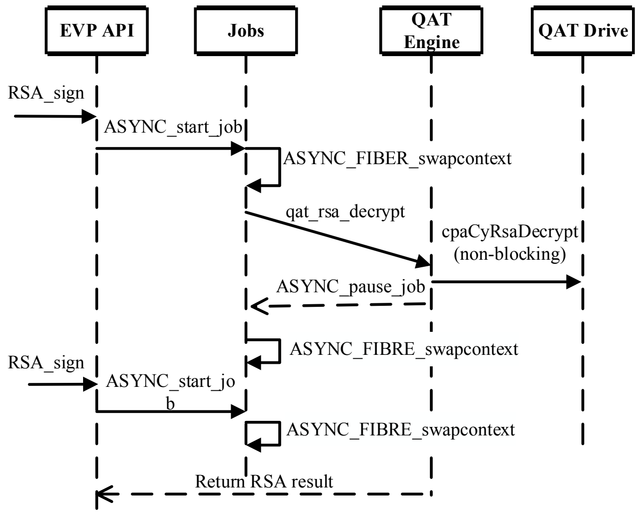


Figure 6: OpenSSL asynchronous mode

Fig. 6 shows the implementation of offloading RSA sign operation to the QAT engine with the asynchronous mode. In this mode, the host CPU can send a request to the hardware accelerator and return immediately to handle other requests. The CPU is notified in the post-processing flow once the hardware accelerator completes the task.

4.3.2 CDN Working Process. When a SSL/TLS connection is launched by a client, the Nginx will call into OpenSSL stack to do SSL/TLS handshake. During the handshake, the OpenSSL will offload the public key operation such as RSA decryption (which is a typical operation in AES128-SHA cipher suite) into the QAT engine along with the exported private key. Inside the QAT device, the exported private key will be decrypted using the shared encryption key (K_c in phase 2). The QAT engine will decrypt the payload using the SSL private key and will return to the OpenSSL stack.

In consideration of a stronger security scheme, a request authority module can be introduced into Key Store enclave of the KSC to authenticate each request using MAC code. This advanced feature is feasible with an further extension in the current implementation of the proposed STYX scheme.

5 PERFORMANCE EVALUATION

Section 4 has explained how the STYX can achieve the secured key management for CCDN application, in Phase 1 by SGX-based key distribution scheme. This security insurance will incur the major performance overhead of STYX in comparison with traditional HTTPS server. Actually, STYX intends to outperform both the Keyless SSL method and the HTTPS server deployed on CDN nodes by QAT based acceleration as aimed in the design motivation. In this section, we denote them by STYX with and without QAT acceleration to show the STYX's SGX based key distribution overhead and its QAT acceleration effects, respectively.

This section will first discuss the performance of the STYX scheme in comparison with other related key protection methods, including PKCS and Keyless SSL. Then the system efficiency enhancement achieved by STYX scheme will be evaluated in terms of reduced CPU utilization, increased throughput and decreased latency performance.

5.1 Experimental Setup

The experimental platform uses a Dell 7040MT desktop to deploy a CDN node and two Xeon servers to deploy a Key Distribution Center and a client, respectively. The CDN node, the KDC and the client are all connected through 10Gbps high speed Ethernet networking devices.

The hardware configurations of the CDN Node are: Intel®Core(TM) i7-6700 CPU@3.40GHz; 1 x 8Gib DDR4 3400MHz memory; 1 x Intel DH8950 PCIe QAT card; Intel SGX is enabled; and Ubuntu 14.04 LTS operating system.

The Key Distribution Center and the client are deployed on two servers, each of which is configured as: 1x Intel®Xeon E5-2699 v3@2.30GHz CPU; 8x8G DDR4 3400MHz memory and Fedora 16 operating system.

5.2 Qualitative Analysis Discussion

Scheme	Component	Security	Latency	CPS
Keyless SSL	OpenSSL/QAT	High	High	Low
PKCS#8	OpenSSL/QAT	Low	Low	High
SGX	SGX	High	Low	Low
SGX+QAT	SGX+QAT	High	Low	High

Table 2: Key protection schemes analysis

For the Keyless SSL scheme, each SSL connection requires an extra round trip from CDN node to the Key Distribution Center for private key decryption request, which introduces a high latency with low throughput of CPS (connections per second).

The use of PKCS#8 to protect the private key on the disk is a common scheme. The main problem is that the private key is decrypted before loading into the memory. Thus it may be attacked by memory snooping.

In this paper, the STYX uses SGX-based scheme to protect the private key on an untrusted CDN node. However, the enclave cannot efficiently handle heavy workload such as RSA decryption. Therefore, a hardware based accelerator (Intel *textcircledR* QAT) is introduced to provide a solution with high security level, high CPS performance and low latency as shown in Table 2.

5.3 Performance Measurement

In order to highlight the performance benefits of Intel's QAT engine in the proposed STYX scheme, the Nginx HTTPS output performance is evaluated with QAT engine enabled and disabled, denoted as "SGX only" and "SGX+QAT" scheme, respectively. All performance evaluation results are collected by Intel VTUNE Amplifier 2017 [31] (part of Intel®System Studio [30]). The TLS configuration is set with AES128-SHA cipher suite under TLS1.2 protocol. Each SSL/TLS connection requires one RSA private key decryption operation. The performance evaluation results, including latency, CPU utilization and connections per Second (CPS) in the Ngnix server, are collected on the CDN node.

5.3.1 CPU utilization measurement. Apache Bench (ab) is used as the practical end-user pressure benchmarking for HTTP server. The ab is configured with 20000 requests and 10 concurrency level. Moreover, one Nginx worker process is binded to one CPU core in the CDN node. The evaluated CPU utilization is presented in Fig. 7.

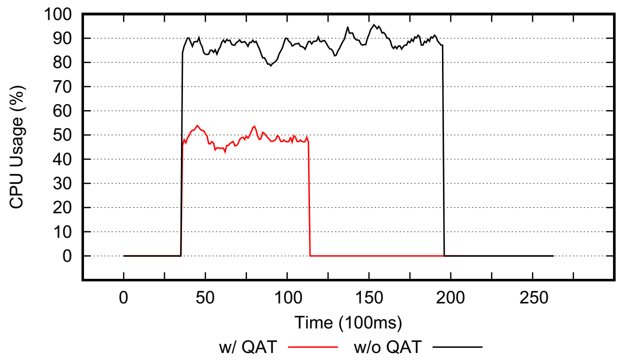
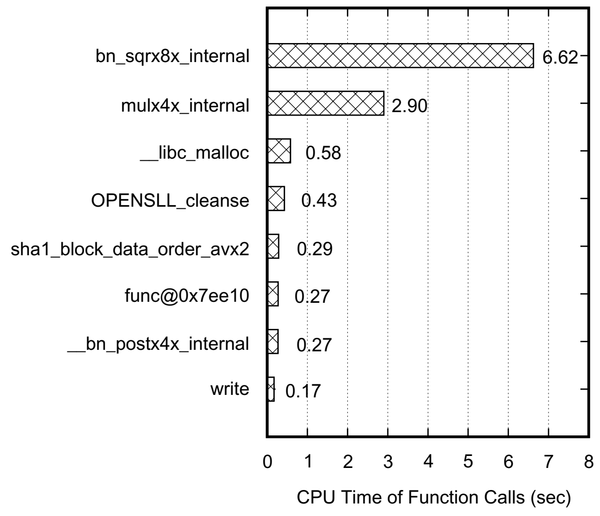
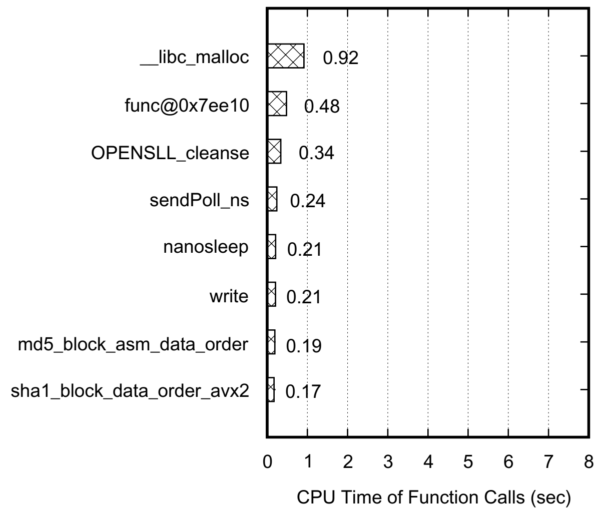


Figure 7: CPU utilization comparison with and without QAT acceleration.

Without QAT acceleration, the Nginx server costs 16.142 seconds with average CPU utilization of 90% to complete the 20000 HTTPS requests. With QAT acceleration, the running time cost of Nginx decreases to 7.784 seconds with only 50% CPU utilization compared to the case where the QAT is enabled. In a nutshell, QAT engine based accelerator can approximately save 1/2 CPU resources with 2x faster requests processing speed. The surface area of (*Running Time*)×(*CPU Usage*) in Fig. 7 denotes the system efficiency. The STYX with QAT acceleration has only a quarter of that without QAT acceleration.



(a) Hotspots for "STYX without QAT"



(b) Hotspots for "STYX with QAT"

Figure 8: Hotspots for STYX with and without QAT.

Next, the CPU hotspots are measured for the QAT enabled/disabled scenarios. The hotspot experiments are tested using the Intel VTUNE Amplifier performance profiler that can identify the most time consuming parts of the tested code and provide call stack information down to the source lines.

The evaluated results are plotted in Fig. 8. Without QAT acceleration, the top two hotspot functions are "*bn_sqrx8x_internal*" and "*mulx4x_internal*" that are part of the RSA decryption function, as shown in Fig. 8(a).

With QAT acceleration, the RSA related functions are eliminated from the hotspot list as disappeared in Fig. 8(b). The top hotspot function only consumes 0.92 second of CPU computing time indicating that the RSA operation overhead is offloaded to QAT hardware device and CPU resource is freed up for other workload services. Consequently, other CPU-intensive workloads now indirectly benefit from QAT acceleration by obtaining more CPU resource to achieve a better performance.

5.3.2 Connection Per Second (CPS). Since the HTTPS connections are normally with short session life-time, the CPS is evaluated as the system performance and is measured with following experimental scenario configuration:

- CDN node: Run 1, 2, 3, 4 Nginx worker processes, respectively. Each process is set a fixed bound affinity to an single physical core. Each Nginx worker listens to 443∼472 ports independently to break Linux kernel bottleneck.
- Client: Launch multiple processes to run ab benchmark. The ab is configured with 1000 concurrency level, 20000 requests, AES128-SHA SSL/TLS cipher suite and TLS1.2 SSL/TLS protocol.

The Keyless SSL solution provided by Cloudflare is not free and is only available to customers who pay the enterprise plan. It is considered that the keyless SSL service can be affected by the geographical location of an end user. Therefore, in the practical scenario, the performance upper bound of Keyless SSL is the HTTPS server directly deployed on CDN nodes. Hence, the STYX performance is only compared with the standard HTTPS Nginx server to demonstrate the STYX's performance advantages.

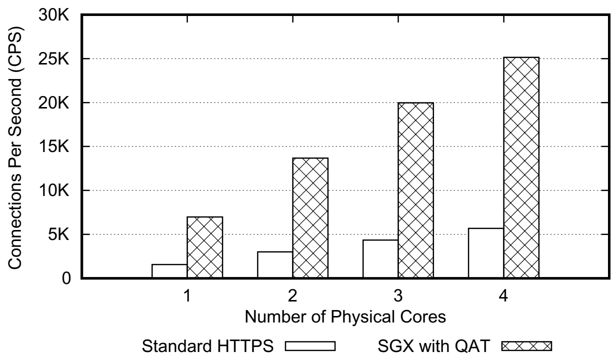


Figure 9: CPS Measurement Results

The Nginx server achieves about 25K HTTPS CPS with the QAT acceleration when running 4 Nginx works, , as shown in Figure 9. However, at the same testing scenario the standard HTTPS server can only achieve 5K CPS performance. In general, it is a 5 times throughput increment for SSL/TLS handshake performance.

This significant performance improvement of STYX is achieved by using QAT hardware based acceleration that offloads the CPU burden of RSA decryption operations. These CPU-intensive operations have been verified in Figure 8(a). Essentially, in HTTPS connections with the interference of SSL/TLS, more CPU resources are utilized for encryption and decryption operations causing less requests to be handled. In turn, it takes more time to accomplish all requests and results in a poor performance for web server applications and other CPU-intensive workloads.

5.3.3 Latency. Since the latency is the crucial performance metric for cloud based web applications, the Apache Bench (ab) benchmark is conducted to evaluate the latency performance. The main latency improvement of STYX is because that it saves an extra round trip between CDN node and original key server compared to the "Keyless SSL" scheme. The STYX provisions the private key into each KSC, so it does not need to involve key center for each client SSL connection. This will end up the most client SSL connections at the CDN node unlike Keyless SSL that needs an extra round trip to original key server to complete private key operation for each client SSL connection. However, this round trip time varies largely depending on the geographical distribution of CDN nodes.

Our measurement only covers the SSL connection latency on the CDN node that benefits from the QAT acceleration. The tested results are shown in Fig. 10, which shows that the latency increases linearly along with the increasing request number. The QAT acceleration enables STYX to significantly shorten the latency down to 0.5x in comparison with QAT disabled case when the number of requests varies from 2000 to 8000. It can be concluded that the latency-intensive application workloads can significantly benefit from the QAT acceleration of STYX.

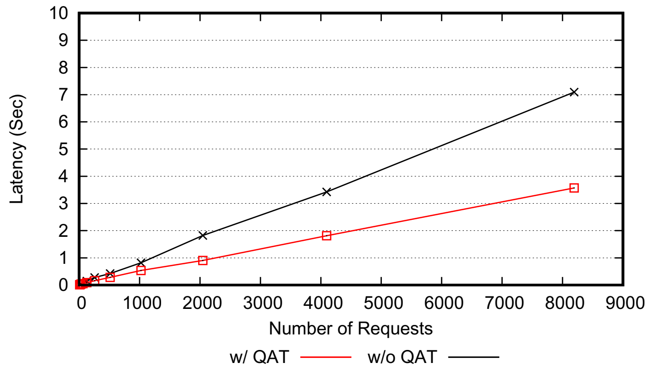


Figure 10: Apache Bench's latency

5.3.4 Overhead Analysis. The overhead of STYX is analyzed from its three phases for key management, respectively. In Phase 1, the STYX provisions the SSL private key from KDC to KSCs for deploying a CDN application, which is considered as the one-time execution outsides the runtime overhead. In phase 3, the STYX can provide an 5x higher throughput and 0.5x shorter latency performance because of the QAT acceleration, as evaluated in Fig. 9 and Fig. 10. These significant performance improvements are achieved when the client incurs the RSA decryption/sign request and the needed SSL private key is insides the internal RAM of QAT, so that it can be processed without any extra operations.

The only source of STYX's runtime overhead comes from the Phase 2, when the necessary SSL private key is missed in the RAM of QAT. This overhead includes the cost that the "Accelerator Accreditor" (a SGX enclave) exports encrypted SSL private key from the "Key Store" to QAT accelerator through the SIGMA channel as well as the cost for SSL private key decryption and caching in the RAM of QAT. The frequency of this SSL private key cache missing in QAT depends on the size of internal RAM of QAT and deployed applications on a CDN nodes.

According to the released parameters of QAT [4] , we can evaluate the overhead introduced by the SSL private key decryption. Considering the RSA 2048 decryption as an example, a CRT (which is a widely used RSA optimized implementation due to its better performance compared to the traditional RSA private key operation) based RSA private key that contains p, q, dp, dq, $qinv$ [58], the total key length is about 5120 bits. Suppose using AES with 128bits key to encrypt/decrypt this private key. According to the QAT production spec, its throughput of AES is 50Gbps, so the latency of the decryption of the private key will be 5120/50Gbps = 0.102us. On the other side, for RSA 2048, QAT can process about 40K request per second, for a single RSA decryption, we can expect the latency is 25us. So the overall overhead introduced by the private key decryption should be 0.102/25 = 0.4%. Note that this is only the theoretical calculation which omits the extra data moving and I/O operations. The internal firmware of QAT is not published, so our implementation did not cover the firmware change of QAT.

6 CONCLUSIONS AND FUTURE WORK

In this paper, the challenges in providing high performance and high scalability for TLS/SSL secured cloud based Content Delivery Network (CDN) application are investigated. Consequently, a three-phase hierarchical key management scheme, called STYX, is proposed. The proposed STYX scheme consists of a Key Distribution Server and numerous distributed Key Sub-Centers. The implementation of proposed STYX evolves synthesis employments of Intel's SGX with remote attestation service in combination with Elliptic Curve Diffie-Hellman key exchange algorithm, SIGMA protocol and Intel's QuickAssist Technology (QAT). The entire implementation process is divided in three phases. In Phase 1, the STYX provisions the SSL private key from KDC to KSC through secured channel constructed by Intel's SGX (Software Guard Extensions) with remote attestation service and the KSCs protect the SSL private key on the CDN nodes with the trusted execution environment (TEE) of SGX enclaves. In phase 2, the STYX establishes the acceleration engine by accrediting the SSL private key to QuickAssist Technology device, and offloads the CPU-intensive calculations in HTTPS service to QAT engine. In phase 3, the STYX enables the end-user to benefice from vastly geographically distributed CDN nodes to accomplish the HTTPS connection services with QAT accelerator without invoking the KDC with modified Nginx server and OpenSSL implementation on QAT acceleration engine.

In summary, the proposed STYX realizes a trusted key protection and a key transmission over the untrusted execution platform and untrusted network channel, respectively. The experimental results have demonstrated that the STYX not only provides absolute key protection, but also breaks the bottleneck of the performance and scalability caused by the customer's centralized key server for maintaining the SSL private key compared with modern Keyless SSL solutions. In future, the authors plan to extend the proposed STYX scheme to other on-cloud applications with secure key distribution and protection requirements, such as the virtual machine management, the encrypted database management and the remote data access schemes.

ACKNOWLEDGMENTS

The authors would like to thank the shepherd, Dr. Xuehai Qian, and the anonymous reviewers for their valuable comments.

This work is supported in part by the National Key Research & Development Program of China No. 2016YFB1000500, and National Natural Science Funds for Distinguished Young Scholar No. 61525204. Jian Li is the corresponding author.

REFERENCES

[1] Intel corp., intel quickassist technology. https://01.org/packet-processing/intel
[2] Intel corp., intel software guard extensions: Intel attestation service api. https://software.intel.com/sites/default/files/managed/3d/c8/IAS_1_0_API_spec_1_1_Final.pdf.
[3] Intel corp., intel software guard extensions (intel sgx). https://software.intel.com/en-us/sgx.
[4] Intel corp, intelÂő xeonÂő processor e5-2600 v2 product family and intelÂő communications chipset 89xx series,. https://www-ssl.intel.com/content/www/us/en/intelligent-systems/highland-forest/xeon-e5-2600-v2-large-scale-communications-brief.html.
[5] Amazon. Amazon cloudfront âĂŞ content delivery network (cdn). https://aws.amazon.com/cloudfront/.
[6] Armin, J., Foti, P., and Cremonini, M. 0-day vulnerabilities and cybercrime. In *Availability, Reliability and Security (ARES), 2015 10th International Conference on* (2015), IEEE, pp. 711–718.
[7] Baumann, A., Peinado, M., and Hunt, G. Shielding applications from an untrusted cloud with haven. *ACM Transactions on Computer Systems (TOCS) 33*, 3 (2015), 8.
[8] Blog, V. C. Analysis of "average session duration" in google analytics. https://www.visma.com/blog/analysis-reporting-average-session-duration-google-analytics/.
[9] Bresson, E., Chevassut, O., Pointcheval, D., and Quisquater, J.-J. Provably authenticated group diffie-hellman key exchange. In *Proceedings of the 8th ACM conference on Computer and Communications Security* (2001), ACM, pp. 255–264.
[10] Brickell, E., and Li, J. Enhanced privacy id: A direct anonymous attestation scheme with enhanced revocation capabilities. In *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society* (New York, NY, USA, 2007), WPES '07, ACM, pp. 21–30.
[11] Brickell, E., and Li, J. Enhanced privacy id from bilinear pairing. Cryptology ePrint Archive, Report 2009/095, 2009.
[12] Brickell, E., and Li, J. Enhanced privacy id from bilinear pairing for hardware authentication and attestation. *International Journal of Information Privacy, Security and Integrity 2 1*, 1 (2011), 3–33.
[13] Brickell, E., and Li, J. Enhanced privacy id: A direct anonymous attestation scheme with enhanced revocation capabilities. *IEEE Transactions on Dependable and Secure Computing 9*, 3 (May 2012), 345–360.
[14] Brown, R. Sec 1: elliptic curve cryptography. standards for efficient cryptography group (secg), 2016.
[15] Cangialosi, F., Chung, T., Choffnes, D., Levin, D., Maggs, B. M., Mislove, A., and Wilson, C. Measurement and analysis of private key sharing in the https ecosystem. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 628–640.
[16] Chandramouli, R., Iorga, M., and Chokhani, S. *Cryptographic Key Management Issues and Challenges in Cloud Services.* Springer New York, New York, NY, 2014, pp. 1–30.
[17] Choon, J. C., and Cheon, J. H. An identity-based signature from gap diffie-hellman groups. In *International Workshop on Public Key Cryptography* (2003), Springer, pp. 18–30.
[18] Cloudflare. How will keyless ssl affect performance? https://support.cloudflare.com/hc/en-us/articles/203243090-How-will-Keyless-SSL-affect-performance-.
[19] Cloudflare. Keyless ssl: The nitty gritty technical details. https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details/.
[20] Coppolino, L., DÂĂŹAntonio, S., Mazzeo, G., and Romano, L. Cloud security: Emerging threats and current solutions. *Computers & Electrical Engineering* (2016).
[21] Costan, V., and Devadas, S. Intel sgx explained. Tech. rep., Cryptology ePrint Archive, Report 2016/086, 2016. https://eprint. iacr. org/2016/086.
[22] Cramer, R., and Shoup, V. Signature schemes based on the strong rsa assumption. *ACM Transactions on Information and System Security (TISSEC) 3*, 3 (2000), 161–185.
[23] Durumeric, Z., Kasten, J., Adrian, D., Halderman, J. A., Bailey, M., Li, F., Weaver, N., Amann, J., Beekman, J., Payer, M., et al. The matter of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference* (2014), ACM, pp. 475–488.
[24] Fan, C. I., Wu, C. N., Hsu, J. C., Tseng, Y. F., and Chen, W. T. Anonymous credential scheme supporting active revocation. In *2014 Ninth Asia Joint Conference on Information Security* (Sept 2014), pp. 127–132.
[25] Google. https://support.google.com/analytics/answer/1006253?hl=en.
[26] Google. Google cloud cdn documentation. https://cloud.google.com/cdn/docs/.
[27] Grossklags, J., Christin, N., and Chuang, J. Secure or insure?: a game-theoretic analysis of information security games. In *Proceedings of the 17th international conference on World Wide Web* (2008), ACM, pp. 209–218.
[28] Hankerson, D., Menezes, A. J., and Vanstone, S. *Guide to elliptic curve cryptography.* Springer Science & Business Media, 2006.
[29] Hofmann, O. S., Kim, S., Dunn, A. M., Lee, M. Z., and Witchel, E. Inktag: Secure applications on an untrusted operating system. *SIGPLAN Not. 48*, 4 (Mar. 2013), 265–278.
[30] Intel. Corp., intel system studio 2017. https://software.intel.com/en-us/intel-system-studio.
[31] Intel. Corp., intel vtuneâĎć amplifier 2017. https://software.intel.com/en-us/intel-vtune-amplifier-xe.
[32] Jin, G., Song, L., Zhang, W., Lu, S., and Liblit, B. Automated atomicity-violation fixing. In *ACM SIGPLAN Notices* (2011), vol. 46, ACM, pp. 389–400.
[33] Johnson, S., Scarlata, V., Rozas, C., Brickell, E., and Mckeen, F. Intel software guard extensions: Epid provisioning and attestation services. *White Paper* (2016).
[34] Joux, A. A one round protocol for tripartite diffie–hellman. In *International Algorithmic Number Theory Symposium* (2000), Springer, pp. 385–393.
[35] Kaliski, B. Public-key cryptography standards (pkcs)# 8: Private-key information syntax specification version 1.2. *RFC 5208, IETF* (2008).
[36] Kim, S., Shin, Y., Ha, J., Kim, T., and Han, D. A first step towards leveraging commodity trusted execution environments for network applications. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks* (2015), ACM, p. 7.
[37] Krawczyk, H. Sigma: The 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike protocols. In *Annual International Cryptology Conference* (2003), Springer, pp. 400–425.
[38] Liang, J., Jiang, J., Duan, H., Li, K., Wan, T., and Wu, J. When https meets cdn: A case of authentication in delegated service. In *2014 IEEE Symposium on Security and Privacy* (May 2014), pp. 67–82.
[39] Litton, J., Vahldiek-Oberwagner, A., Elnikety, E., Garg, D., Bhattacharjee, B., and Druschel, P. Light-weight contexts: An os abstraction for safety and performance. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2016), OSDI'16, USENIX Association, pp. 49–64.
[40] Miller, V. S. Use of elliptic curves in cryptography. In *Conference on the Theory and Application of Cryptographic Techniques* (1985), Springer, pp. 417–426.
[41] Mutton, P. Half a million widely trusted websites vulnerable to heartbleed bug, 2014.
[42] Nikiforakis, N., Meert, W., Younan, Y., Johns, M., and Joosen, W. Sessionshield: Lightweight protection against session hijacking. In *International Symposium on Engineering Secure Software and Systems* (2011), Springer, pp. 87–100.
[43] Noubir, G., and Sanatinia, A. Trusted code execution on untrusted platform using intel sgx. *Virus Bulletin* (2016).
[44] Odelu, V., Das, A. K., and Goswami, A. A secure biometrics-based multi-server authentication protocol using smart cards. *IEEE Transactions on Information Forensics and Security 10*, 9 (Sept 2015), 1953–1966.
[45] OpenSSL. https://www.openssl.org/.
[46] Reese, W. Nginx: the high-performance web server and reverse proxy. *Linux Journal 2008*, 173 (2008), 2.
[47] Rescorla, E. Security holes... who cares? In *USENIX Security* (2003), Washington, DC.
[48] Rsa, A., and Note, L. T. Pkcs#8: Private-key information syntax standard, 1993.
[49] Sanatinia, A., and Noubir, G. Honey onions: a framework for characterizing and identifying misbehaving tor hsdirs. *arXiv preprint arXiv:1610.06140* (2016).
[50] Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G., and Russinovich, M. Vc3: Trustworthy data analytics in the cloud using sgx. In *Security and Privacy (SP), 2015 IEEE Symposium on* (2015), IEEE, pp. 38–54.
[51] Sidiroglou, S., Laadan, O., Perez, C., Viennot, N., Nieh, J., and Keromytis, A. D. Assure: automatic software self-healing using rescue points. *ACM SIGARCH Computer Architecture News 37*, 1 (2009), 37–48.
[52] Sinha, R., Rajamani, S., Seshia, S., and Vaswani, K. Moat: Verifying confidentiality of enclave programs. In *Proceedings of the 22nd ACM SIGSAC Conference*

on Computer and Communications Security (2015), ACM, pp. 1169–1184.

[53] Stebila, D., and Sullivan, N. An analysis of tls handshake proxying. In *2015 IEEE Trustcom/BigDataSE/ISPA* (Aug 2015), vol. 1, pp. 279–286.

[54] Steiner, M., Tsudik, G., and Waidner, M. Diffie-hellman key distribution extended to group communication. In *Proceedings of the 3rd ACM conference on Computer and communications security* (1996), ACM, pp. 31–37.

[55] Wang, Y. Public key cryptography standards: Pkcs. *arXiv preprint arXiv:1207.5446* (2012).

[56] Wikipedia. Content delivery network. https://en.wikipedia.org/wiki/Content_delivery_network.

[57] Wikipedia. Fiber. https://en.wikipedia.org/wiki/Fiber_(computer _science)#cite_note-flounder-1.

[58] Wu, C.-H., Hong, J.-H., and Wu, C.-W. Rsa cryptosystem design based on the chinese remainder theorem. In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference* (New York, NY, USA, 2001), ASP-DAC '01, ACM, pp. 391–395.

[59] Xie, W., and Wang, J. A trusted connection based scheme for ad hoc network. In *PROCEEDINGS OF 2013 International Conference on Sensor Network Security Technology and Privacy Communication System* (May 2013), pp. 34–38.

[60] Zhang, F., and Zhang, H. Sok: A study of using hardware-assisted isolated execution environments for security. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016* (2016), ACM, p. 3.

[61] Zhang, L., Choffnes, D., Levin, D., Dumitras, T., Mislove, A., Schulman, A., and Wilson, C. Analysis of ssl certificate reissues and revocations in the wake of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference* (New York, NY, USA, 2014), IMC '14, ACM, pp. 489–502.

[62] Zhao, M., Zhou, W., Gurney, A. J., Haeberlen, A., Sherr, M., and Loo, B. T. Private and verifiable interdomain routing decisions. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication* (2012), ACM, pp. 383–394.

[63] Zhao, M., Zhou, W., Gurney, A. J., Haeberlen, A., Sherr, M., and Loo, B. T. Private and verifiable interdomain routing decisions. *IEEE/ACM Transactions on Networking 24*, 2 (2016), 1011–1024.