DKS-PKI: A Distributed Key Server Architecture for Public Key Infrastructure

Abu Faisal(✉) and Mohammad Zulkernine

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

Abstract. Public key infrastructure (PKI) is the core of authentications performed in secure internet communications. The most popular PKI arrangement is the certificate authority (CA)-based PKI. The traditional security protocols use this CA-based PKI to provide server-side authentications. In this approach, the server-side uses its public key certificate that is the server's public key signed by a *trusted* CA. This CA-based PKI is not able to properly address the challenges associated with the growing demand of secure internet communications. We identified two major problems with this CA-based PKI to be used in secure communications. First, the mis-issuance of certificates or the disclosure of any such CAs' private keys can cause serious problems to the security of these internet communications. Second, revoking an issued public key certificate or any trusted CA's certificate is not a trivial task. In this paper, we proposed a distributed key server architecture (DKS-PKI) that provides a PKI arrangement that can solve the above mentioned problems. The proposed architecture offers registration/issuance, storage, distribution, and revocation of certificates in an efficient manner. It ensures transparency and accountability of the certificate issuers. All the registered/issued certificates and their sensitive identity information are verified and stored into a permissioned distributed storage system. The key-server nodes are responsible to make these issued certificates publicly accessible by means of their associated 256-bit unique identifiers (UIDs). We presented a thorough security analysis of the proposed architecture.

Keywords: DKS · PKI · Authentication · Certificate · Distributed · Issuance · Revocation · Verification

1 Introduction

In public key infrastructure (PKI), digital certificates are used to authenticate entities/participants (e.g., people or organizations) in secure internet communications. The most common format for these certificates is the X.509 standard. Each certificate contains the public key of the entity and some key information, such as domain name, organization name, validity period, and issuer's signature. The issuer's signature proves the trustworthiness of the presented information

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
V. R. Badarla et al. (Eds.): ICISS 2022, LNCS 13784, pp. 23–43, 2022.
https://doi.org/10.1007/978-3-031-23690-7_2

in the certificate. The most popular and widely accepted PKI for authentication is the certificate authority (CA)-based PKI. In CA-based PKI, the certificates are issued by the *trusted* certificate authorities (CAs). All the trusted CAs' certificates are already installed in all modern internet browsers and operating systems. Therefore, the entity certificates issued by these trusted CAs can be validated at any time.

The traditional security protocols (SSL/TLS/DTLS) use this CA-based PKI authentication approach to authenticate the server's identity. In these protocols, the client's identity authentication is optional and skipped most of the times. A basic example of the existing CA-based PKI is that a business owner wants to get a public key certificate for his/her online business. After validating the owner's identity, business information, and registered domain name, a trusted CA generates a public key certificate for the online business. The business owner stores the certificate in the server. Now, a client wants to establish a secure (SSL/TLS) communication channel with the server to send payment to the business. Therefore, the client needs to ensure the authenticity of the server. The server sends its public key certificate to the client. Since all trusted CAs' certificates are already installed into the system, the client can validate the server's authenticity using the received certificate.

In certificate authority (CA)-based PKI, the CAs are arranged in a hierarchical structure extending up to several levels (e.g., top-level/root CA and several levels of intermediate CAs). Any CAs except the root CA are called the intermediate CAs. The root CA certifies the next level of intermediate CAs, often called the regional CAs. These regional CAs certify the next levels of intermediate CAs and so on. Any of these intermediate CAs can generate the end-point certificates. Since the certificates are all signed, anyone can verify this chain of certificates (also called the chain-of-trust) tracing back to the root CA. Over the course, the number of such intermediate CAs has increased significantly. It gives rise to the possibility that the mis-issuance of certificates or the disclosure of any such trusted CAs' private keys can cause serious problems to the security of internet communications [5–7,11,13,17,20,21,23].

Moreover, these CAs have ample powers to maliciously or erroneously issue duplicate certificates or revoke the existing certificates without the owner's consent [5–7,11,13,17,20,21,23]. Therefore, this CA-based approach is not always able to address the challenges in secure internet communications. Furthermore, revoking an issued certificate before the end of its validity period is a difficult and cumbersome task. A certificate can be revoked for different reasons, such as the entity's private key or the CA's private key is compromised, or the CA improperly issued such certificates. To revoke such certificates, the issuing CA or any other trusted authority needs to generate and publish a certificate revocation list (CRL) [4]. This CRL is valid for a specific timeframe and needs to be updated and re-published in a timely manner.

The major problem with CRL is that the revoked certificates can still be accepted where the subjected certificate is not validated against the current CRL. There is no proper way to invalidate the revoked certificate. Also, the CRL has increased in size (up to megabytes) over time. Therefore, doing such validations

become cumbersome and cannot be enforced everywhere. To cope with this, an alternative online certificate status protocol (OCSP) [19] is implemented that requires less time and less network bandwidth. However, it has a little or no advantage over CRL. Because, a revoked certificate can still be accepted if the OCSP is not used or unavailable. In short, it requires tremendous efforts to stop rogue certificates in the existing CA-based PKI.

In this paper, we propose a distributed key server (DKS) architecture for public key infrastructure (PKI) that can solve these major problems in the existing CA-based PKI. The proposed architecture provides certificate registration/issuance, storage, distribution, and revocation mechanisms. It does not rely on such hierarchical intermediate CA structures where CAs cannot restrict their subordinate CAs from issuing rogue certificates. Instead, it utilizes distributed authority nodes to reach consensus for issuing or revoking any certificates. Each issued certificate is associated with a 256-bit (32 Bytes) unique identifier (UID). The issued certificates, their associated UIDs, and certificate owners' identity information are securely stored in a permissioned distributed storage system. In this way, this architecture ensures transparency and accountability of the certificate issuers (authority nodes).

The issued certificates and their UIDs are also securely transmitted and stored in the key-server nodes (KSNs) which are the client-facing nodes in this architecture. The KSNs manage the certificate distribution mechanism and participate in the authentication process of a secure communication. These key-server nodes (KSNs) extend the central key-server (CKS)-based authentication presented in the Graphene architecture [9,10]. This mechanism is able to solve the two major problems with the existing CA-based PKI authentication. Also, the revocation of any certificate is effective immediately as soon as the revocation transactions are approved by the network. No extra CRL or OCSP-based mechanisms are required.

The major contributions of this paper can be summarized as follows:

- A distributed key server architecture called DKS-PKI that provides public key infrastructure (PKI) for authentication in secure internet communications. This DKS-PKI can replace the existing certificate authority (CA)-based PKI.
- It utilizes a consensus-based distributed network of authority nodes, permissioned storage nodes, and publicly accessible key-server nodes to ensure that the certificate registration/issuance, storage, distribution, and revocation mechanisms are secure, transparent, and fault-tolerant. All the certificates (registered public keys) in DKS-PKI are accessible using their associated unique identifiers (UIDs) through the key-server nodes.
- This architecture solves the two major problems with the existing CA-based PKI and eliminates the need for having any hierarchical intermediate CAs that can issue any certificates to anyone. It ensures accountability of the certificate issuers and allows the system to detect any mis-issuance of certificates as well as any revocation of certificates promptly.

The rest of the paper is organized as follows. Section 2 discusses the most prominent research on different PKI approaches. Section 3 describes the

proposed distributed key server architecture for PKI in detail. Section 4 presents the evaluation of the proposed architecture. It provides the security analysis against the possible attack vectors in DKS-PKI. Then, it describes the implementation and the experimental environment. It also presents the performance analysis of DKS-PKI certificate distribution mechanism. Finally, Sect. 5 summarizes the paper.

2 Related Work

Public key infrastructure (PKI) offers the necessary authentication mechanism to guarantee the participants' identity and maintain integrity in any untrusted internet communications. In spite of different problems, the certificate authority (CA)-based PKI is the most popular arrangement for authentication in the present times. Researchers proposed different PKI approaches to establish authentication in internet communications. These approaches suffer from different security, deployment, and scalability issues. Some approaches [1,8,12,18,25] are attempted but not fully deployed due to different issues.

Web of trust [1] (WoT, also known as peer-to-peer PKI) is a decentralized approach driven by the peers in the network where one peer can generate/issue certificates for the other peers. However, it is very difficult for new users to get certificates from such peers. Also, all peers are not equally trusted to the other peers in the network. This may result in rejection of certificates even though they are valid. Moreover, WoT does not have any certificate revocation mechanism. This makes it almost impossible to revoke generated certificates before their validity periods expire.

Another PKI approach is proposed, called simple public key infrastructure (SPKI) [8,25] to bind privileges or rights (access control) to the public key. However, this approach is never finalized and later merged with the simple distributed security infrastructure (SDSI) [18] that deals with groups and group-membership certificates. These PKI approaches attempted to enhance the existing PKI approach. However, they are still in the draft stage and does not solve the stated problems with the CA-based PKI approach.

Alternative approaches, such as log-based PKI [12] and PKI Safety Net (PKISN) [22] are also proposed. The log-based PKI approach is basically an enhancement to the existing CA-based PKI approach. It follows the same CA-based certificate issuance and revocation mechanisms. However, it provides transparency and accountability by creating publicly available logs for the generated certificates. This approach greatly suffers from deployment challenges and certificate revocation issues [14,16]. Also, it cannot be guaranteed that each CA would play fair in the log generation.

The PKISN [22] also follows the same footsteps as in log-based PKI. However, they propose an approach to solve the certificate revocation problem when a trusted CA's private key is compromised. This approach claims that maintaining a log of issued certificates would help determine when the private key compromise happened and only the certificates issued after that event are deemed malicious

and require revocation. However, this approach as well suffers from the same deployment issues and depends on the CAs to maintain such logs.

To deal with this problems, Matsumoto and Reischuk [14,15] proposed an automated platform called "IKP" that incentivize certificate authorities (CAs) for properly issuing certificates and detectors for quickly reporting CA misbehaviors and unauthorized certificates. IKP uses Ethereum smart contracts to incentivize participating CAs and detectors. However, it again depends on a new third party called detectors. The detectors may not properly report any unauthorized certificates and gain advantage from the issuing CAs of such certificates. All CAs may not be a part of this incentivized approach and try to collude the participating CAs to maliciously issue certificates.

Similar to log-based PKI, Yakubov *et al.* [24] proposed a blockchain-based PKI that actually follows the same hierarchical CA-based PKI. Instead of generating logs for the generated certificates, this approach stores the generated certificates in the blockchain ledger. It offers transparency and accountability for the CAs' operations. However, it does not solve the major problems of the CA-based PKI discussed above.

In our prior work [9,10], we implemented a central key-server (CKS)-based authentication in our secure communication architecture called Graphene. In Graphene, the participants are authenticated using their public key certificates already registered with the CKS. This CKS-based authentication is able to solve the CA trust issues and the certificate revocation problem of the CA-based PKI.

The above discussed PKI approaches suffer from different problems that prevent their adoption into secure internet communications. Most of these approaches are enhancements over CA-based PKI and often suffer from deployment and/or scalability issues. They do not necessarily address the above mentioned problems of CA-based PKI. Also, the growing demand of internet communications require a distributed PKI approach that can decentralize the power of the CAs and handle the certificate mis-issuance problems in a robust way. The issued certificates and their sensitive identity information should be stored in a permissioned distributed storage system. Thus, these approaches may not be appropriate for establishing secure internet communications. Therefore, a distributed key server architecture is required that can address the above stated problems and offer a better, transparent, and accountable PKI mechanism for secure internet communications.

3 DKS-PKI Architecture

The proposed distributed key server (DKS) architecture provides public key infrastructure (PKI) for authentication in secure internet communications. It provides registration/issuance, storage, distribution, and revocation of public key certificates. These public key certificates are often referred to as "digital certificates" or simply "certificates". In DKS-PKI, these certificates are also addressed as "registered public keys" interchangeably and each certificate is associated with a 256-bit unique identifier (UID). This architecture consists of four types of nodes as shown in Fig. 1, namely – (*i*) authority nodes (ANs), (*ii*)

transaction repository nodes (TxRNs), (*iii*) broadcasting nodes (BNs), and (*iv*) key-server nodes (KSNs).

All four types of nodes play important roles in the certificate registration/issuance, storage, and revocation mechanisms. However, the certificate distribution mechanism that enables entities to authenticate each other in a secure communication, is provided by the KSNs. The proposed architecture does not use hierarchical intermediate CA structure for certificate issuance and revocation as observed in CA-based PKI. Instead, it uses authority nodes (ANs) that perform consensus to register/issue or revoke any certificates. This ensures transparency and accountability of the certificate issuers (ANs) in DKS-PKI. Unlike the CA-based approach where the top-level CAs cannot restrict the subordinate CAs from issuing certificates (original as well as duplicate) literally for any entities (people or organizations).

The key aspects of DKS-PKI architecture are discussed in detail in the following subsections.

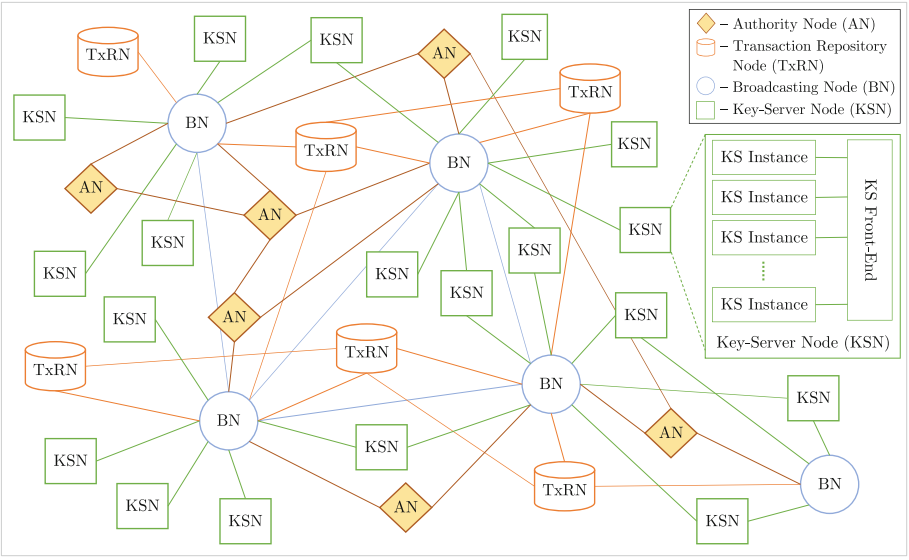

Fig. 1. The proposed distributed key server (DKS) architecture for PKI.

3.1 Overview

In the proposed architecture, the authority nodes (ANs), the transaction repository nodes (TxRNs), and the key-server nodes (KSNs) communicate to each other through the broadcasting nodes (BNs) only. However, the same type of nodes may have direct communication channels (e.g., AN–AN, TxRN–TxRN, BN–BN) between them for data synchronization. All node-to-node communications must be signed by the sender node. All the nodes need to be distributed and can subscribe to their nearest one or more BNs to provide better network stability. Only the authenticated nodes can subscribe to any BNs for receiving

specific messages from the other nodes. When the BNs receive an incoming message from a subscribed node, they publish this message to the other subscribed nodes only. Each AN validates certificate registration and revocation requests submitted to it. For each registration request, it generates a public key certificate and its associated 256-bit unique identifier (UID). Then, the AN generates a temporary transaction enclosing these information.

This 256-bit UID is generated by performing hash operation on the validated entity information, and the generated public key certificate. For any authentication, the generated certificates are not usable and the revoked certificates are not invalid until the ANs perform consensus to register/issue or revoke the subjected certificates into the DKS-PKI network. After validating registration/revocation requests, the ANs broadcast temporary transactions enclosing entity information and their signed public key certificates to the other ANs and the TxRNs. After successful consensus between the ANs, the TxRNs generate the sealed transactions from these temporary transactions.

Then, the TxRNs store these sensitive entity information, issued or revoked certificates, and their associated UIDs permanently. The TxRNs also transmit the issued certificates and their associated UIDs to the KSNs. The KSNs provide the certificate distribution mechanism that enables entities to authenticate each other using their associated UIDs. The KSNs store these issued certificates and their associated UIDs locally in the registered certificate store. In case of revocation, the KSNs receive the revoked UIDs from the TxRNs and move the corresponding certificates to the revoked certificate store. These KSNs need to be publicly available at all times and are responsible to manage certificate requests from anywhere in the world.

3.2 Node Operations

Nodes in this architecture operate on three types of transactions, such as register (REG), revoke (RVK), and copy (CPY) transactions. The consensus approach between the ANs uses two types of voting schemes, namely– vote in favor of approval, and vote in favor of rejection. The REG transactions are used to register/issue new certificates into the network. The RVK transactions are used to revoke any existing certificate(s) from the network. The CPY transactions are used to transmit the issued/revoked certificates from the transaction repository nodes (TxRNs) to the KSNs. Each REG transaction corresponds to only one certificate registration request and contains the registering entity information, the generated public key certificate, and the generated 256-bit unique identifier (UID). However, each RVK transaction may contain one or more certificate UID(s) for revocation and their reason(s) of revocation.

The REG and the RVK transactions are generated only by the ANs as temporary transactions (tx'). Each temporary REG/RVK transaction (tx') must be broadcasted to the other ANs and the TxRNs through the BNs. The other ANs verify (duplicity check, mis-issuance check, etc.) the incoming registration/revocation request. Then, the ANs perform consensus and cast their votes on the transaction either in favor of approval or rejection. The TxRNs act as the vote collectors (VCs) during the consensus process. Each transaction requires at least two-third votes in

favor of approval to be accepted. However, a single vote in favor of rejection would halt the consensus process and move the subjected transaction to the suspended transaction list. Next, the AN(s) who voted in favor of rejection must present valid reasoning for their actions. On failure to do so, the AN(s) are permanently removed from the network and the suspended transaction is moved to the pending transaction list. An AN may vote in favor of rejection if the registering entity's identity or domain information cannot be verified.

After that, the consensus process resumes with the remaining ANs in the network. After receiving at least two-third votes in favor of approval from the ANs, the temporary transaction (tx') is transformed into a sealed transaction (tx) by the TxRNs. The sealed transaction is signed by the TxRNs and contains the temporary transaction, the received voting information, and the hashed value of all these information. For certificate registration, the TxRNs generate a signed CPY transaction with the issued certificate and its associated UID, and broadcast it to the nearby KSNs. For certificate revocation, the signed CPY transaction only contains the revoked UID(s). Once the CPY transaction is broadcasted, the ANs are notified with a notice-of-approval. After that, the newly issued certificate is available and the revoked certificate(s) are removed for entity authentication. The registering entity receives its 256-bit UID from the corresponding AN.

3.3 Authoritative Signing Keys (ASKs)

The proposed architecture uses only *three* trusted authoritative signing keys (ASKs), such as root/master ASK (RASK), node ASK (NASK), and issuer ASK (IASK). These ASKs differ from the traditional certificate authorities (CAs). The ASKs do not belong to any specific authority or organization. Instead, they are used to maintain the trust value of the network nodes and prevent man-in-the-middle (MITM) attacks from happening at all steps of the node operations. Each ASK is essentially a keypair that consists of a public key certificate and its corresponding private key. The RASK is the master public-private keypair of this DKS-PKI architecture and securely stored offline.

The public key certificates in NASK and IASK are signed using the RASK private key. The NASK private key is only used to generate/issue the public key certificates for the broadcasting nodes (BNs), the transaction repository nodes (TxRNs), and the key-server nodes (KSNs). However, the IASK private key is used to generate the public key certificates for the ANs only. The NASK certificate and the IASK certificate play key roles in DKS-PKI. They are also called the *trusted* Node certificate and the *trusted* DKS certificate, respectively. These certificates must be installed (trusted) into these node systems, all modern browsers, and operating systems. They are used to validate the authenticity of the node-to-node communications and the entity certificates. In case of any private key disclosure, using different ASKs provides better control in separating the node systems (BNs, TxRNs, KSNs) and the entity certificate issuance mechanism.

The *trusted* DKS certificate is used to prove the trustworthiness of the entity certificates (also called the end-user certificates) issued by the ANs. Each entity certificate carries the entity information and its public key, issuer's (AN's) signature, 256-bit unique identifier (UID), and the AN's certificate. This AN's

certificate is signed by the IASK private key. Therefore, the AN's certificate can be validated using the IASK certificate, also referred to as the *trusted* DKS certificate in the end-user systems. Once the AN's certificate is validated using the *trusted* DKS certificate, it becomes trusted for the current authentication session. Then, this AN's certificate can be used to validate the entity certificates signed by this AN. This double validation reduces the number of trusted certificates in the end-user systems significantly.

3.4 Certificate Registration/Issuance and Storage

The authority nodes (ANs) receive public key registration applications from different types of entities (e.g., people, businesses, organizations). Each AN is responsible to validate the identity information associated with the applications submitted to them. After successful validation, the AN generates a certificate and a 256-bit unique identifier (UID) for the registering entity. This UID is generated by performing hash operations on the validated entity information and the generated certificate. Next, the AN generates a temporary REG transaction (tx'_{reg}) enclosing these information and broadcasts this tx'_{reg} to the other ANs and the TxRNs. The other ANs verify (duplicity check, mis-issuance check, etc.) this newly generated certificate and the entity information.

After verification, the ANs start casting their votes in favor of approval or rejection for this transaction. The TxRNs wait till they receive at least two-third votes in favor of approval for this transaction. Here, one key difference with the other consensus-based systems is that if a transaction receives a single vote in favor of rejection, that transaction is moved to the suspended transaction list. The AN(s) who voted in favor of rejection must present valid reasoning for casting such vote(s). On failure to do so, the AN(s) are permanently removed from the network and the consensus process resumes by moving the suspended transaction to the pending list.

Once the TxRNs receive two-third votes in favor of approval and there is no vote in favor of rejection, the TxRNs generate a sealed REG transaction (tx_{reg}) from the temporary transaction and store it permanently in the repository. The issued certificate is placed in the registered certificate store until the entity certificate is revoked or expired. Also, the TxRNs synchronize their current state to make sure that there is no discrepancies. After that, the TxRNs generate a CPY transaction (tx_{cpy}) with only the issued certificate and its associated UID and broadcast this transaction using the broadcasting nodes (BNs). The BNs push updates to the listening KSNs. Then, the TxRNs broadcast a notice-of-approval to the ANs. The registering entity receives its 256-bit UID from the corresponding AN. At this point, the issued certificate is registered into the network and can be accessed using its associated 256-bit UID.

3.5 Certificate Distribution

In DKS-PKI, the key-server nodes (KSNs) are responsible to manage the certificate distribution mechanism. This mechanism enables entity authentication in

secure communications using the proposed architecture. The issued certificates and their associated 256-bit unique identifiers (UIDs) are transmitted to these KSNs using CPY transactions and stored in their registered certificate store until they are revoked or expired. Similarly, when any certificate(s) are revoked or expired, they are moved to the revoked certificate store. The registering entities do not receive their public key certificates unlike CA-based PKI. Instead, they receive their 256-bit UIDs. In this proposed architecture, entities do not present their certificates to each other for authenticating themselves. Rather, they present their 256-bit UIDs.

These UIDs are used to request their associated registered certificates from the KSNs. To deal with these certificate requests, each KSN runs one or more key-server instance(s) (KSIs). Each KSI is essentially a Hypertext Transfer Protocol (HTTP) service. Therefore, these KSNs accept HTTP requests (e.g., /public-key/?id={UID}) from participating entities and respond with one of four HTTP responses, such as "200 OK", "410 Gone", "404 Not Found", and "400 Bad Request" where the response codes carry their usual meanings. When the requested KSN finds a registered/issued certificate with the requested UID, it responds with a "200 OK" HTTP response attaching the certificate to it. If the requested UID is associated with a revoked certificate, the KSN responds with a "410 Gone" HTTP response. A "404 Not Found" HTTP response is sent when the requested UID does not belong to any of the issued or revoked certificates. In case of any invalid/malformed request URI, the KSN responds with a "400 Bad Request" HTTP response.

Therefore, the availability of these KSNs is crucial for this PKI architecture to succeed in providing certificates for authentications during secure communications. The distributed design used for the KSNs would help improve the availability of these KSNs and reduce the impact of denial-of-service (DoS) or distributed denial-of-service (DDoS) attacks on these KSNs. Since these KSNs serve public key certificates, encrypting the data-in-transit is not necessary. However, all responses must be signed by the KSN's private key to prevent man-in-the-middle (MITM) attacks and accompany the KSN's certificate which is signed by the NASK private key as stated in Sect. 3.3. This KSN certificate can be validated using the *trusted* Node certificate. Then, the validated KSN certificate can be used to validate the response signature.

3.6 Certificate Revocation

In the proposed architecture, the revocation mechanism for the issued certificate(s) is straightforward. When one or more certificate(s) need to be revoked, the request is submitted to an authority node (AN). The requested AN validates the revocation request and generates a temporary RVK transaction (tx'_{rvk}) enclosing the associated UID(s) of the certificate(s) under revocation request and the reason(s) of revocation. Next, the AN broadcasts the transaction to the other ANs and the transaction repository nodes (TxRNs) through the broadcasting nodes (BNs). After receiving this transaction, the TxRNs wait for the votes in favor of approval or rejection from the ANs.

The other ANs verify (malicious/erroneous check) the certificate(s) and their reason(s) of revocation. Then, the ANs start casting their votes in favor of approval or rejection. If any AN(s) vote in favor of rejection, the consensus process halts and the transaction is moved to the suspended transaction list. The AN(s) who voted in favor of rejection must provide valid reasoning for such voting. Otherwise, they are removed from the network permanently and the consensus process resumes with the subjected transaction. Once the TxRNs receive at least two-third votes in favor of approval and no votes in favor of rejection for this transaction, the TxRNs generate a sealed RVK transaction (tx_{rvk}) from this temporary transaction and store it permanently.

After that, the revoked certificate(s) are moved to the revoked certificate store and the TxRNs synchronize their state to avoid any discrepancies. Next, the TxRNs generate a CPY transaction (tx_{cpy}) with the revoked UID(s) and broadcast it to the key-server nodes (KSNs) through the BNs. The KSNs move the revoked certificates and their associated UIDs to the revoked certificate store. A notice-of-approval is sent to the requesting AN. After that, anyone requesting a certificate associated with any of those revoked UIDs receives a HTTP "410 Gone" response from the KSNs.

3.7 Stored-Data Validation

In terms of DKS-PKI, the stored data incorporates the sealed transactions stored in the transaction repository nodes (TxRNs), the certificates and their associated 256-bit unique identifiers (UIDs) stored in the certificate stores (registered and revoked) of the TxRNs and the key-server nodes (KSNs). Each sealed REG transaction represents a single certificate registration/issuance. It contains the validated entity information, the generated public key certificate, the generated UID, the received voting information, and the hashed value of all these information. However, each sealed RVK transaction may contain one or more revoked UID(s), the reason(s) of their revocation, the received voting information, and the hashed value of all these information. Next, the sealed transaction is signed by the respective TxRN where it is stored and added to an append-only list of transactions. Each certificate store (registered/revoked) in the TxRNs, and the KSNs is essentially a list of {UID, certificate} object indexed by the corresponding UID.

To maintain integrity of the stored data, different nodes perform validation checks on the other nodes of the network. The authority nodes (ANs) perform validation check on the stored data in the TxRNs. The ANs pull randomly selected same portion of the transaction list from different TxRNs. Then, they perform integrity and consistency check on the data received from the TxRNs under consideration. If any TxRN(s) fail to comply with the validation process, they are suspended from the network. The suspended TxRN(s) are required to synchronize their stored data with the other active TxRNs. They remain suspended until they pass this integrity and consistency check. Similarly, the active TxRNs perform integrity and validation check on the KSNs. The KSNs also get suspended from the network if failed to comply with the validation check and need to pass the check to become active again.

4 Evaluation

4.1 Security Analysis

The proposed architecture and all the nodes need to be secured from any types of certificate mis-issuance, certificate cloning/tampering, and compromised private key attacks. It is equally important for all the nodes in this architecture to prevent themselves from man-in-the-middle (MITM), and denial-of-service (DoS) attacks. In this section, we identify the possible attack vectors and present a thorough security analysis of the proposed architecture.

Scenario-1 (Compromised Key-Server Node). The key-server nodes (KSNs) are solely responsible to provide the certificate distribution mechanism that allows entities to authenticate each other using their unique identifiers (UIDs) and registered certificates. Each valid certificate contains entity information, entity public key, authority node's signature, and authority node's certificate. However, a compromised KSN may attempt to tamper any registered certificate(s) and add duplicate or cloned certificates in the local storage. Since the KSNs' certificates are generated (signed) by a different authoritative signing key (ASK) than the ANs' certificates as stated in Sect. 3.3, the KSNs are not able to generate/forge any unauthorized or cloned/tampered certificates that can pass the validation check using the *trusted* DKS certificate.

Scenario-2 (Malicious Authority Node). A malicious authority node (AN) may try to mis-use its power to issue unauthorized or duplicate certificates. However, the AN itself cannot register any certificates to the transaction repository nodes (TxRNs) and the key-server nodes (KSNs) without broadcasting it through the broadcasting nodes (BNs). Also, the subjected certificates can only be accepted into the network after receiving *at least* two-third votes in favor of approval from the other ANs. This makes it fairly difficult to corrupt such significant amount of ANs in the proposed architecture.

Scenario-3 (Any TxRN Tampering a Certificate). No other nodes except the ANs can issue and/or revoke any certificates. The TxRNs store the sealed transactions, the issued and the revoked certificates, and their UIDs. Also, they act as the vote collectors (VCs) during the consensus process. If any TxRN is compromised and trying to tamper an approved certificate or publish a tampered certificate to the KSNs, it is readily detected by the other TxRNs and the KSNs. The TxRNs synchronize their sealed transactions. The KSNs receive the same CPY transactions from multiple TxRNs signed by the sender TxRNs. If there is any mismatch, they can report that to the other network nodes. Also, the stored-data validation process can track such discrepancies.

Scenario-4 (Any Private Key Is Lost or Stolen). Compromised private key(s) always cause serious issues in any PKI systems. Therefore, protecting

these private keys is of utmost importance. As described in Sect. 3.3, the certificates of the TxRNs, the BNs, and the KSNs are signed using the node ASK's (NASK) private key. The ANs' certificates are signed using the issuer ASK's (IASK) private key. The end-user certificates are signed by their issuing ANs' private keys. Therefore, if any end-user private key is compromised, that end-user must register for a new certificate. In case of any of the ANs' private keys are lost or stolen, the respective ANs must get their new certificates and re-generate (revoke-then-issue) all the previously issued end-user certificates using their new private keys. If any ASKs' private key is compromised, it would cause a large number of certificates to be re-generated. However, using different ASKs for node certificates and certificate issuance mechanism provides a better control over such situations.

Scenario-5 (In Case of Any Man-In-The-Middle). All the node-to-node communications and the KSNs' HTTP responses are signed using their respective private keys. The TxRNs', the BNs', and the KSNs' signatures can be validated using the *trusted* Node certificate and their own certificates. The ANs' signature can also be validated using the *trusted* DKS certificate and the ANs' own certificates. An adversary may try to perform man-in-the-middle (MITM) attacks on any node-to-node communications or the KSNs' HTTP services to gain access and tamper with any temporary transactions, consensus votings, and/or certificate transmissions. However, the authenticated communication approach can effectively prevent such attacks in this proposed architecture.

Scenario-6 (Any Types of Denial-Of-Service). The KSNs provide the certificate distribution mechanism using HTTP services that allow the entities to request each other's certificates for authentication. Also, the broadcasting nodes (BNs) enable the ANs, the TxRNs, and the KSNs to communicate with each other. The BNs use a publish-subscribe approach and only allow nodes that can prove their authenticity with valid certificates to subscribe for specific messages and publish selective messages to the network. An adversary may try to perform denial-of-service (DoS) or distributed denial-of-service (DDoS) attacks on the KSNs, and/or the BNs. This attack scenario can be prevented by increasing the number of KSNs and BNs and distributing them geographically.

4.2 Implementation

The proposed architecture utilizes a permissioned distributed network of four types of nodes, such as authority nodes (ANs), broadcasting nodes (BNs), transaction repository nodes (TxRNs), and key-server nodes (KSNs). Each type of nodes plays important roles in certificate registration/issuance, storage, distribution, and revocation mechanisms. However, to be fault-tolerant and avoid partial centralization problem, such distributed networks require all types of nodes to be deployed in large numbers. Increasing the number of nodes improves the network operations and the unbiasedness of the consensus process. However, deploying

such large number of nodes is a challenge on its own. To tackle this deployment challenge and implement the proposed architecture in a way so that its performance can be measured for the certificate distribution mechanism, we adopt some changes in the node operations.

The certificate registration and revocation mechanisms use consensus approach to register/issue and revoke certificates, respectively. The consensus is based on *at least* two-third votes in favor of approval of the transactions which is a proven Byzantine Fault Tolerant (BFT) approach for distributed systems. Therefore, we focused our implementation on the certificate distribution mechanism of the proposed architecture that enables entities to authenticate each other using their registered certificates and associated 256-bit unique identifiers (UIDs). This mechanism is solely dependent on the KSNs, and the already registered and revoked certificates in them. Therefore, we implemented the AN, the KSN, and a request generator (RG) that generates certificate requests to a KSN continuously.

These nodes are implemented using the Java programming language, Java Cryptography Architecture (JCA), socket programming, multi-threading, and HTTP communication protocol. The National Institute of Standards and Technology (NIST) recommendations are strictly followed at all steps of the implementation [2,3]. The AN implementation is used to generate the sample end-user certificates. These sample certificates are manually stored in the KSN's local file system indexed (renamed) by their UIDs. The KSN implementation provides HTTP services that manage certificate requests from the request generators (RGs). This KSN implementation supports all four possible types of request scenarios as stated in Sect. 3.5. Different types of request scenarios can cause different response times. However, the worst-case scenario for measuring the KSN performance is a complete certificate request-response for a valid UID. The RG is implemented as a single-threaded process to simulate continuous HTTP certificate requests to a specific request URI (/public-key/?id={UID}).

4.3 Experimental Environment

For the experiments, twelve Raspberry Pi 3 Model B+ (RPi) devices are configured with CentOS 8 (minimal version) and one-gigabit Ethernet connections as shown in Fig. 2. Each RPi device has a quad-core processor, and 1 GB LPDDR2 SDRAM. This means each RPi can easily run up to four processes or key-server instances simultaneously without any significant processing time overlap. Java 8 is used to develop the implementation and OpenJDK 11.0.3 is used to execute the experiments in CentOS 8 environment. The reason behind using RPi devices (in spite of being low in CPU and memory) is to deploy the node implementations in a more modular way. Using a faster processing multi-core CPU and larger memory machine would yield running multiple nodes in the same machine sharing the same network interface card (NIC) which does not properly represent the network design required for the certificate distribution mechanism of the proposed architecture.

All public-private keypairs are generated using 2048-bit RSA algorithm. The certificates are generated using 256-bit SHA2 hashing algorithm and RSA

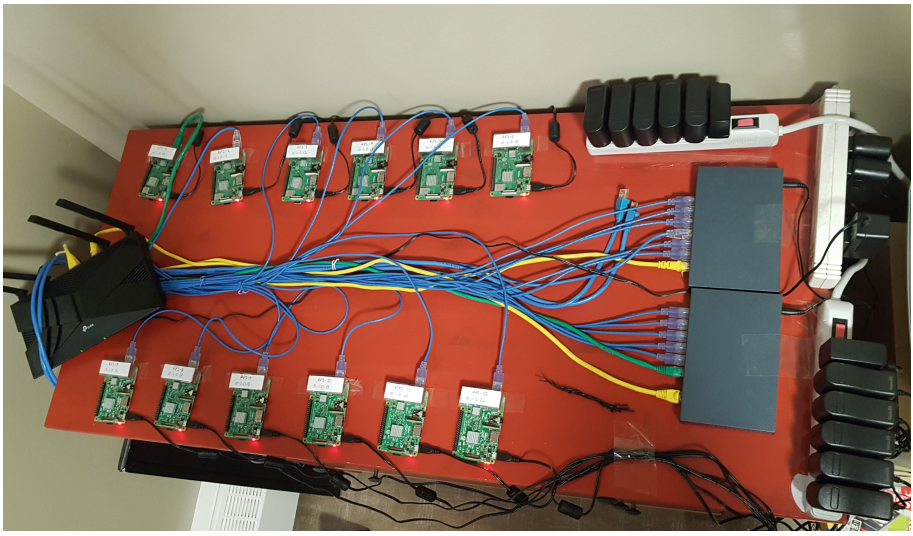

Fig. 2. Experimental environment of the DKS-PKI certificate distribution mechanism.

encryption with PKCS#1 v2.2 padding scheme [2,3]. Three authoritative signing keys (ASKs), such as root ASK (RASK), node ASK (NASK), and issuer ASK (IASK) as stated in Sect. 3.3 are generated using the above algorithms. The *trusted* Node certificate (NASK certificate) and the *trusted* DKS certificate (IASK certificate) are installed into all RPi devices. The KSN certificates are generated using the NASK private key. The AN certificate is generated using the IASK private key. Then, the AN and the KSN certificates and their respective private keys are installed in the same directory where these nodes are deployed.

First, the AN is deployed in a RPi device where all necessary public key certificates and its own private key are installed appropriately. Next, the AN generates 100 sample (test) 2048-bit RSA public-private keypairs, signs these public keys using its own private key, and attaches its own certificate at the end of these test certificates. The AN also generates 256-bit unique identifiers (UIDs) for each of these test certificates. After that, these test certificates are manually stored to a separate directory (presented as the registered certificate store) in each KSN local file system indexed (renamed) by their UIDs. Each KSN is deployed with four key-server instances to manage HTTP requests from the RGs. Therefore, The RPi devices running a KSN does not have any other processing load.

All devices are under the same local network that has no other traffic except the certificate requests generated from the RGs to the KSN(s). Different number of KSNs and RGs are executed in these twelve RPi devices in a distributed manner. The number of KSNs deployed is up to four and the number of RGs deployed is up to 25. Since the RGs are single-threaded processes, each RPi device can run up to four RGs simultaneously. We run up to 25 RGs distributed over eight RPi devices in a round-robin fashion that use three parameters (RG

ID, KSN IP, and KSN port). Each RG instance is executed one minute apart from the last executed RG instance on the next RPi device. For faster execution, we preloaded the RGs with 100 256-bit UIDs. Then, these UIDs are used randomly to generate valid certificate requests to the KSNs.

4.4 Performance Analysis

This section presents the performance analysis for the certificate distribution mechanism of the proposed DKS-PKI architecture. We evaluate the runtime performance and scalability of the key-server nodes (KSNs) in terms of the number of processed requests per second and the average request completion time (in milliseconds) with different number of request generators (RGs). We recorded the request completion time for each request at the RGs and calculated values for the above metrics. We also analyze the implementation with multiple KSNs to show the performance improvement in terms of the number of processed requests per second. Table 1 presents the specification of the experimental environment.

Table 1. Experimental environment specification.

Parameters	Values
Test environment	Raspberry Pi 3 Model B+ (RPi)
RPi processing unit	Cortex-A53 (ARMv8) 64-bit, SoC @ 1.4 GHz quad-core
RPi memory	1 GB LPDDR2 SDRAM
Number of RPi modules	12
Operating system	CentOS 8 (minimal version)
Runtime environment	OpenJDK v11.0.3
Network component	One Gigabit router, Two LAN (Gigabit) switches
Connection type	Ethernet
Network ping time	Less than 1 ms
Number of RGs	1–25

A single KSN is executed in one of the RPi devices. Three more RPi devices are reserved for running three more KSNs later. A new RG instance is executed every one minute in one of the remaining eight RPi devices in a round-robin fashion. Each RG generates a certificate request to the KSN with a valid UID and receives the response. If it is successful (200 OK), then it generates another certificate request with another valid UID and keeps doing this continuously until stopped. In case of a failed request, the RG reports the error and terminates itself. This single KSN experiment is executed with different number of RGs for at least 25 min from the secure shell (SSH) terminal. In this way, we can record the request processing times for one minute duration with different number of RGs sending concurrent requests to the single KSN. Here, our goal is to investigate the performance and scalability of the certificate distribution mechanism.

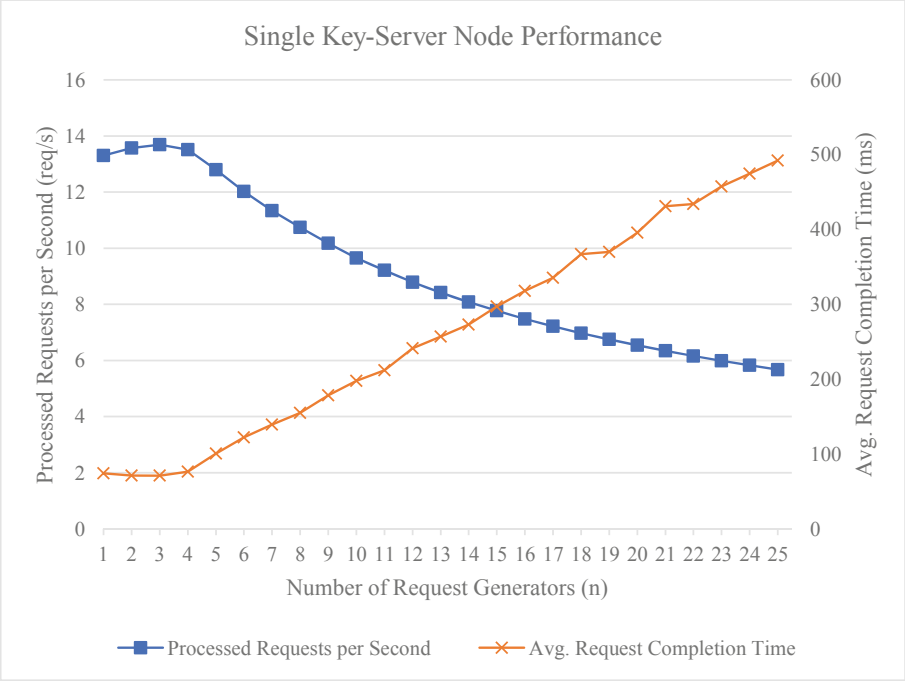

Fig. 3. Performance of the DKS-PKI certificate distribution mechanism using a single key-server node.

Figure 3 shows the runtime performance of the implementation in terms of the number of processed requests per second (viz., curve) and the average request completion time in milliseconds (viz., curve) with respect to different number of request generators (RGs) ranging from 1 to 25. The overall runtime performance of the certificate distribution mechanism is reasonable. The performance deteriorates with the increasing number of concurrent certificate requests generated by the RGs. For the first four RG instances, the number of processed requests per second (viz., curve) and the average request completion time (viz., curve) are steady and show similar pattern. The reason behind that the KSN is running four key-server instances. Therefore, the number of processed requests per second as well as the average request completion time remains steady.

After that, increasing the concurrent certificate requests (i.e., increasing the RG instances) decreases the number of processed requests per second exponentially and increases the average request completion time linearly. Three times increase in the concurrent certificate requests (12 RGs) results in a 36% decrease in the number of processed requests per second and 340% increase in the average request completion time. Similarly, increasing the concurrent certificate requests by six times (24 RGs) decreases the number of processed requests per second by 60% and increases the average request completion time by almost 700%.

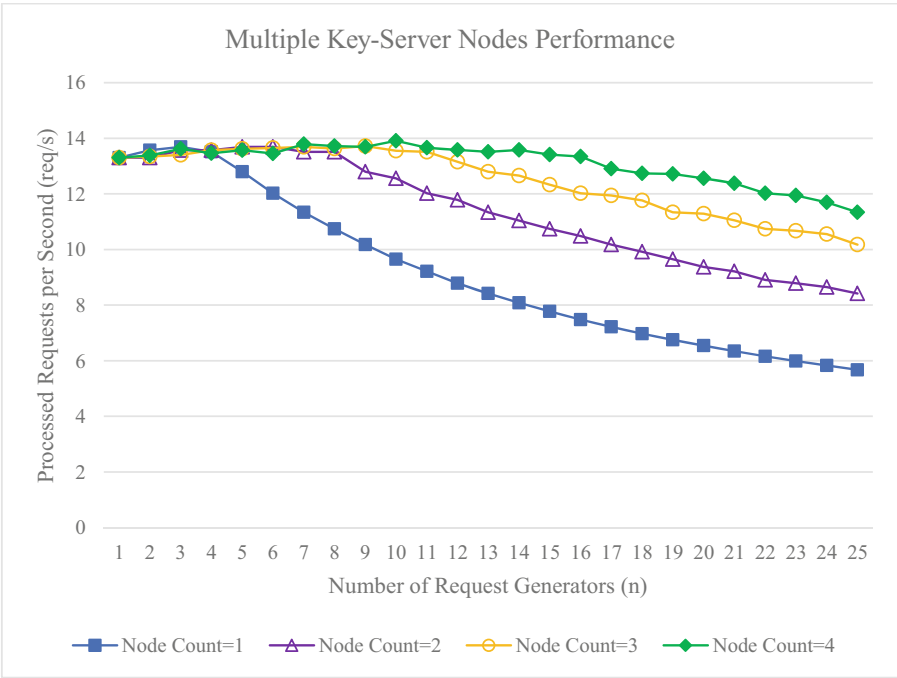

Fig. 4. Performance of the DKS-PKI certificate distribution mechanism using multiple key-server nodes.

Figure 4 presents the performance of the certificate distribution mechanism with multiple KSNs (up to four KSNs) with the increasing number of request generators (RGs). It is readily noticeable that the performance of the KSNs improved significantly with each new KSN. Since each KSN is running four key-server instances, the number of processed requests per second for two KSNs (viz., △ curve) remains steady up to eight RGs. Then, the performance starts deteriorating in the similar way. Similarly, for three KSNs (viz., ○ curve) and four KSNs (viz., ◆ curve) the performance remains steady up to 12 RGs and 16 RGs, respectively. Therefore, increasing the number of KSN improves the overall performance of the certificate distribution mechanism in DKS-PKI. Using faster processing multi-core CPUs and large memory machines would enhance the scalability of the network even further.

5 Conclusion

The traditional public key infrastructure (PKI) offers entity authentications using trusted third parties called certificate authorities (CAs). All these CAs are organized in a hierarchical structure (e.g., root CA, and several levels of intermediate CAs) and share ample power for issuing any certificate to anyone.

Therefore, this authentication mechanism is largely dependent on the trustworthiness of these CAs. Also, revoking any certificate in the existing CA-based PKI is a difficult and cumbersome job. It is hard to ensure that the revoked certificates cannot be used anymore. When any CA's private key is lost or stolen, this makes it even more complicated to revoke the CA's trusted certificate everywhere and its issued-certificates as well. To solve these problems, a distributed key server architecture for PKI (DKS-PKI) is proposed. This proposed architecture ensures transparency and accountability of the certificate issuers (authority nodes). Any authority node (AN) cannot issue certificates in the network without reaching consensus with the other ANs. This approach solves the certificate mis-issuance problem. Next, certificate revocation is effective as soon as the authority nodes reach a consensus on the revocation transaction. After that, the revoked certificates are separated from the rest of the issued certificates in the TxRNs and the KSNs. This solves the certificate revocation issue as well. In case of any private key disclosure, only the certificates issued by that private key need to be revoked and re-issued. Overall, the performance of the certificate distribution mechanism shows reasonable results with the increasing number of certificate requests. The distributed network of ANs, BNs, TxRNs, and KSNs helps this architecture to prevent DoS/DDoS attacks on any node and improves the availability of the PKI network for authentication.

References

1. Aberer, K., Datta, A., Hauswirth, M.: A decentralised public key infrastructure for customer-to-customer e-commerce. Int. J. Bus. Process Integr. Manag. **1**, 26–33 (2005)
2. Barker, E., Roginsky, A.: Transitioning the use of cryptographic algorithms and key lengths, March 2019. https://doi.org/10.6028/NIST.SP.800-131Ar2
3. Barker, E.B., Dang, Q.H.: SP 800–57 Pt3 R1. Recommendation for Key Management, Part 3: Application-Specific Key Management Guidance, January 2015. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf. Accessed 04 Dec 2021
4. Boeyen, S., Santesson, S., Polk, T., Housley, R., Farrell, S., Cooper, D.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008. https://doi.org/10.17487/RFC5280. https://www.rfc-editor.org/info/rfc5280
5. Charette, R.: DigiNotar Certificate Authority Breach Crashes e-Government in the Netherlands, September 2011. https://spectrum.ieee.org/riskfactor/telecom/security/diginotar-certificate-authority-breach-crashes-egovernment-in-the-netherlands. Accessed 25 July 2022
6. ComputerWorld.com: To punish Symantec, Google may distrust a third of the web's SSL certificates, March 2017. https://www.computerworld.com/article/3184573/to-punish-symantec-google-may-distrust-a-third-of-the-webs-ssl-certificates.html. Accessed 24 July 2022
7. Constantin, L.: French intermediate certificate authority issues rogue certs for Google domains, December 2013. https://www.computerworld.com/article/2486614/french-intermediate-certificate-authority-issues-rogue-certs-for-google-domains.html. Accessed 25 July 2022

8. Ellison, C.: SPKI Requirements. RFC 2692, September 1999. https://doi.org/10.17487/RFC2692. https://www.rfc-editor.org/info/rfc2692
9. Faisal, A., Zulkernine, M.: Graphene: a secure cloud communication architecture. In: Zhou, J., Deng, R., Li, Z., Majumdar, S., Meng, W., Wang, L., Zhang, K. (eds.) ACNS 2019. LNCS, vol. 11605, pp. 51–69. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29729-9_3
10. Faisal, A., Zulkernine, M.: A secure architecture for TCP/UDP-based cloud communications. Int. J. Inf. Secur. **20**(2), 161–179 (2020). https://doi.org/10.1007/s10207-020-00511-w
11. Hoogstraaten, H.: Black tulip report of the investigation into the diginotar certificate authority breach. Technical report, Fox-IT BV, August 2012. https://doi.org/10.13140/2.1.2456.7364. Accessed 25 July 2022
12. Laurie, B., Langley, A., Kasper, E.: Certificate Transparency. RFC 6962, June 2013. https://doi.org/10.17487/RFC6962. https://rfc-editor.org/rfc/rfc6962.txt
13. Leyden, J.: 23,000 HTTPS certs will be axed in next 24 hours after private keys leak, March 2018. https://www.theregister.co.uk/2018/03/01/trustico_digicert_symantec_spat/. Accessed 25 July 2022
14. Matsumoto, S., Reischuk, R.M.: IKP: turning a PKI around with blockchains. Cryptology ePrint Archive, Paper 2016/1018 (2016). https://eprint.iacr.org/2016/1018
15. Matsumoto, S., Reischuk, R.M.: IKP: turning a PKI around with decentralized automated incentives. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 410–426 (2017). https://doi.org/10.1109/SP.2017.57
16. Matsumoto, S., Szalachowski, P., Perrig, A.: Deployment challenges in log-based PKI enhancements. In: Proceedings of the Eighth European Workshop on System Security, EuroSec 2015. Association for Computing Machinery, New York (2015). https://doi.org/10.1145/2751323.2751324
17. Mozilla.org: CA/Symantec Issues. https://wiki.mozilla.org/CA/Symantec_Issues. Accessed 24 July 2022
18. Rivest, R., Lampson, B.: SDSI - a simple distributed security infrastructure, August 1996. https://people.csail.mit.edu/rivest/sdsi10.html. Accessed 24 July 2022
19. Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, D.C.: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960, June 2013. https://doi.org/10.17487/RFC6960. https://www.rfc-editor.org/info/rfc6960
20. SecurityIntelligence.com: Symantec's SSL Certificate May Get Cut Off by Chrome, March 2017. https://securityintelligence.com/news/symantecs-ssl-certificate-gets-cut-off-by-chrome/. Accessed 25 July 2022
21. Symantec: Test Certificates Incident Final Report v3, October 2015. https://bug1214321.bmoattachments.org/attachment.cgi?id=8852862. Accessed 24 July 2022
22. Szalachowski, P., Chuat, L., Perrig, A.: PKI safety net (PKISN): addressing the too-big-to-be-revoked problem of the TLS ecosystem. In: 2016 IEEE European Symposium on Security and Privacy (EuroS P), p. 407–422 (2016). https://doi.org/10.1109/EuroSP.2016.38
23. Williams, O.: Google to drop China's CNNIC Root Certificate Authority after trust breach, April 2015. https://thenextweb.com/insider/2015/04/02/google-to-drop-chinas-cnnic-root-certificate-authority-after-trust-breach/. Accessed 25 July 2022

24. Yakubov, A., Shbair, W.M., Wallbom, A., Sanda, D., State, R.: A blockchain-based PKI management framework. In: NOMS 2018–2018 IEEE/IFIP Network Operations and Management Symposium, pp. 1–6, April 2018
25. Ylonen, T., Thomas, B., Lampson, B., Ellison, C., Rivest, R.L., Frantz, W.S.: SPKI Certificate Theory. RFC 2693, September 1999. https://doi.org/10.17487/RFC2693. https://www.rfc-editor.org/info/rfc2693