Computer Networks 175 (2020) 107268

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

TwinPeaks: An approach for certificateless public key distribution for the internet and internet of things

Eunsang Cho[a], Jeongnyeo Kim[b], Minkyung Park[a], Hyeonmin Lee[a], Chorom Hamm[a], Soobin Park[a], Sungmin Sohn[a], Minhyeok Kang[a], Ted Taekyoung Kwon[a,*]

[a] Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul, 08826, Korea
[b] Electronics and Telecommunications Research Institute, 218 Gajeong-ro, Yuseong-gu, Daejeon, 34129, Korea

ARTICLE INFO

Keywords:
Public key infrastructure
Certificateless public key cryptography
Public key distribution
Internet of things

ABSTRACT

The current public key infrastructure (PKI) has thorny issues like the overhead of certificate revocations and the consequence of fraudulent certificates. To address such issues, we propose TwinPeaks, which is an infrastructure to distribute public keys of named entities on the Internet and the Internet of Things (IoT). TwinPeaks leverages certificateless public key cryptography (CL-PKC), where a key generation center (KGC) cannot know the private key of its member, and hence its compromise will not result in member key leakage. By extending CL-PKC, the public key of an entity becomes dependent on any combination of its networking parameters; thus TwinPeaks can thwart spoofing attacks systematically. With TwinPeaks, the public key of every named entity is distributed online while addressing the PKI's vulnerabilities.

TwinPeaks has public key servers, which constitute the domain name system (DNS)-like hierarchical tree structure. For each parent-child link in the tree, the parent node serves as a key generation center (KGC), and its child nodes set up their own public/secret key pairs by interacting with the KGC as proposed in CL-PKC. In this way, every named entity (e.g., a domain name) has its own public/secret key pair. Thus, a public key of an entity will be provided to a user by its key server as the DNS response is returned to the user by its DNS server.

TwinPeaks removes certificates and hence has no revocation overhead. Instead, each named entity should keep/update its networking parameters and public key up-to-date in its DNS server and key server, respectively. By making its public key depend on both its Internet protocol (IP) address and domain name, the compromise of a single entity (e.g., a DNS or key server) cannot lead to successful impersonation. TwinPeaks achieves scalable distribution of public keys since public keys can be cached long term. We also show that TwinPeaks can be applied to the IoT environments by extending the naming scheme.

1. Introduction

Authenticating the other endpoint is a basis for making secure communications over the Internet. For authentication in the Internet, a certificate-based public key infrastructure (PKI) is the de facto standard and thus a certificate authority (CA) has a crucial role which attests the certificate as a binding between an entity and its public key. Therefore the CAs' certificate operations (i.e., issuance and revocation) should be performed flawlessly. However, over the years, we have witnessed many high-profile attacks on the PKI due to its vulnerabilities in terms of systems, operations, and practices. For instance, at least two Comodo resellers (GlobalTrust and InstantSSL) issued fake certificates of famous sites like mail.google.com and login.yahoo.com [1]. DigiNotar was spoofed to issue at least 513 fraudulent certificates including Google domain names to the Iranian government, some of which were used for overseeing Google emails [2]. Two Taiwanese CAs' private keys were compromised, which were used by Stuxnet developers to sign their malware [3].

The outcome of aforementioned attacks reveals the inherent problems of the PKI. First, there is no systematic limitation on certificate issuance. Therefore any CA can issue a certificate for any entity. Second, trusting a wrong endpoint is possible if a single entity (e.g. CA) is compromised. Third, a compromise of a CA brings a painful recovery process (e.g., revocation/reissuance of certificates). Fourth, the overhead of maintaining revoked certificates keeps on increasing (e.g., the length

* Corresponding author.
E-mail addresses: escho@mmlab.snu.ac.kr (E. Cho), jnkim@etri.re.kr (J. Kim), mkpark@mmlab.snu.ac.kr (M. Park), hmlee@mmlab.snu.ac.kr (H. Lee), crhamm@mmlab.snu.ac.kr (C. Hamm), sbpark@mmlab.snu.ac.kr (S. Park), smsohn@mmlab.snu.ac.kr (S. Sohn), mhkang@mmlab.snu.ac.kr (M. Kang), tkkwon@snu.ac.kr (T.T. Kwon).

https://doi.org/10.1016/j.comnet.2020.107268
Received 24 December 2018; Received in revised form 31 March 2020; Accepted 11 April 2020
Available online 17 April 2020

of Certifiate Revocation Lists (CRLs) and/or the size of Online Certificate Status Protocol (OCSP) data). Fifth, the current practice of verifying certificates (and public keys therein) is dependent on the implementations of individual browsers and web servers [4]. Last but not least, usability studies report that users often ignore certificate warnings, which indicates the possible vulnerability to simple certificate interception attacks [5]. Such PKI issues make us question whether we should continue using the PKI for authenticating endpoints. We believe the PKI should be changed fundamentally to address the above issues. Prior studies of augmenting the PKI systems/operations without changing the PKI itself might lead to inefficient, ineffective, or interim solutions [6].

To provide secure services in Internet of Things (IoT) environments, the use of public key cryptography is highly desirable. However, IoT raises the following issues in addition to the PKI problems aforementioned. First, using the PKI in the IoT may results in inappropriate practices such as (i) hard-coded root certificates inside the IoT device, or (ii) bypasses of verification of certificate chains due to the resource limitations. Second, some IoT devices cannot interoperate with others from different manufacturers or service providers without any backend assistance [7]. Some devices rely on their manufacturers or service providers for the authentication and secure channel setup by delegating PKI-related operations to remote servers. It may incur additional delays, which results in more resource consumption compared to direct and local communications.

As a building block of the infrastructure, TwinPeaks leverages certificateless public key cryptography (CL-PKC). A desirable feature of CL-PKC is that the public and secret keys of an Internet entity are generated as a function of any arbitrary input string. For instance, an input string to make the keys of a server can contain its fully qualified domain name (FQDN), say www.example.com. In this way, we can make the public (and secret) key be dependent on any networking parameters such as FQDN and IP address; thus TwinPeaks can thwart spoofing attacks.

Our Contribution. We propose a new infrastructure, *TwinPeaks*, to provide public keys of all the named entities (i.e., domain names) considering the above issues in the PKI. TwinPeaks makes the public key of a named entity depend on its public information, such as its domain name and IP address. In this case, a successful masquerading attack requires to compromise both the public key and the public information of a target. Thus, a single point of compromise cannot lead to spoofing a user to trust a fake endpoint (for a given domain name). Another merit of such generation of public keys is that it can help thwart DNS poisoning attacks if the domain name (or its endpoint) uses its public key for secure communications. TwinPeaks removes the CA hierarchy (and hence the whole PKI), but instead introduces a DNS-like hierarchical structure of public key servers. We also provide the IoT authentication scenarios using TwinPeaks. By extending the identity (ID) naming scheme, TwinPeaks can handle authentication of numerous IoT devices and can allow for direct and local communications across devices from different manufacturers or service providers.

2. Related work

Numbers of previous research usually focused on the problems regarding the CA and certificate issues. **Certificate Transparency** [8] is a log-based certificate validation approach. A certificate log is maintained by using a Merkle Hash Tree (MHT), which operates in append-only mode to validate certificates through the history of certificates. **Sovereign Keys** [9] is another proposal which utilizes a timeline server to attest the correctness of certificates based on the log of the timeline server. **Accountable Key Infrastructure** [10] proposes to use the combination of the Integrity Log Server and Validators with a checks-and-balances approach. It provides a public key validation infrastructure that aims to reduce compromise and unavailability periods. **BlockPKI** [11] is a blockchain-based PKI that enables an automated, resilient, and transparent issuance of digital certificates. With a blockchain as a distributed append-only ledger, all the operations related to certificate management are logged in the blockchain for resiliency and transparency. It brings Automated Certificate Management Environment (ACME) [12] into the blockchain and its smart contract to automate issuance. However, these schemes suggest yet another trusted third parties to keep monitoring and archiving the certificates typically by using append-only storage. Introducing more watchers/notaries could relieve the risk of PKI failures but these approaches would make the PKI landscape more complex.

There are DNS-centered approaches such as **DNSSEC** [13–15] and **DANE** [16,17]. Both of these IETF working groups are initiated for securing IP addresses and public keys of domain names (in their DNS entries), which is crucial for secure connections over the Internet. DNSSEC and DANE utilize the CAs for the authentication of the IP address and the public key of a domain name, respectively. DANE leverages DNSSEC in the sense that a domain owner indicates the designated CA for its certificate. If the client has a query-response exchange with a DNSSEC-enabled DNS server, it can verify the authenticity of the response through the CA by inquiring the certificate. As DANE relies on DNSSEC, it inherits the CA dependency from DNSSEC. If a domain owner in DANE uses self-signed certificates, he or she can achieve the CA independency, but a certificate forgery may happen while being undetected.

TwinPeaks [18] suggests an infrastructure distributing certificateless public key for the Internet, proposed as an alternative to the PKI. However, it did not encompass IoT service scenarios in which devices may play a server role without a fixed and public IP address. In this paper, we extend TwinPeaks [18] in two aspects: (i) supporting IoT environments with backward-compatibility, and (ii) elaborating its operations and security analysis.

3. Design rationale

Considering the problems of the current PKI systems and operations, we conclude that a fundamentally new infrastructure should be designed.

Detection (Easy/Instant Detection of Fraudulent Certificates/Public Keys): When a CA is compromised or spoofed, it is likely to issue a fraudulent (but seemingly valid) certificate. The central problem is that the current revocation mechanism (i.e., CRL or OCSP) cannot help detect the fraudulent certificate since it looks valid until somebody detects and reports it to the CA [19]. Moreover, fraudulent certificates are not easy to detect and may even go undetected [20,21]. Spoofing of any named entity should be instantly and systemically detected.

Responsibility (Due Positioning of Responsibility in Case of Compromise): In the current PKI, even if a CA is compromised by its negligence of operations (e.g., out-of-date software patch), the damage of a fraudulent certificate (i.e., trusting wrong endpoints) goes to the entity of the spoofed domain name and/or its clients. We seek to achieve a goal that the entity responsible for compromises should take the liability. That is, we make a named entity (e.g., a website) in charge of countermeasures against certificate- or public key-related attacks, and hence it will take the responsibility in case of compromises.

Separation (of the Source and the Owner of the Public Key): With the current PKI, a client contacts a server and receives its certificate from the server itself. Thus, if the server is compromised, the attacker may have replaced its certificate by a fraudulent one. The client has no other choice but to believe the server if the fraudulent certificate is verified by the PKI. We believe it is safer to separate the source of the public key and the server verified by the public key. In this way, the attacker should compromise both the source and the server of the public key, which makes public key-related attacks more difficult.

Scoping (of Certificate Issuance): In the current CA practice, any CA can issue a certificate for any entity. That is, the compromise of any CA can result in a forged certificate for any entity. Also, a certificate (or a public key therein) of an entity could be issued even without contacting the entity itself. We believe the unlimited range of certificate issuance by any CA is one of the root causes of the PKI problems. Thus, we limit

the scope of issuing a public key of a named entity systematically by constructing a certain structure.

Scalability (in terms of Number of Named Entities): The Internet has been and will be growing in terms of the volume of traffic and also the number of connected devices (say, IoT). Accordingly, the number of named entities that are to be authenticated will increase. Also, the cost to obtain/verify certificates for devices is not trivial with the current PKI. Hence, the new infrastructure should be able to distribute public keys in a scalable and inexpensive fashion.

Deployability (of A New Infrastructure): The business model underlying the current PKI is somewhat a misfit. For instance, a CA cannot charge a relying party (*i.e.*, a client) for the cost of certificate management like revocation [22]. The new infrastructure should be designed in such a way that its deployment can take into account relations/incentives among/to interest parties. That is, the relevant stakeholders find it easy/practical to accommodate the new infrastructure. In the same vein, legacy systems or business relations should be reused as much as possible. The proposed infrastructure leverages the current DNS hierarchy, so that DNS operators and Internet service providers (ISPs) can participate in deploying the infrastructure easily with new opportunities for public key businesses.

4. Background: Certificateless public key cryptography (CL-PKC)

To design a new infrastructure that distributes public keys for the above criteria, we adopt the certificateless public key cryptography (CL-PKC), which is a core mechanism of TwinPeaks. Let us first explain the original CL-PKC scheme [23] and advanced CL-PKC schemes [24–26] in this section, and then detail how we bring CL-PKC for substantiating TwinPeaks in Section 5.

In traditional public key cryptographic algorithms (e.g., RSA and ElGamal) that underpin the most of the PKI systems, the public/private key pair is generated from some random values, which has nothing to do with the method of verifying the binding between the public key and the entity's identity. That is why there is a requirement for a certificate (and the PKI) to verify the binding, which entails the PKI issues explained earlier.

To solve this fundamental problem, there have been many studies to tackle the authenticity of public keys from a different perspective. Identity-based public key cryptography (ID-PKC) is proposed and substantiated to address this issue [27], where an entity's public key is derived from its identity value (i.e., any unique string related to its identity like an email address). Thus, finding the public key of an entity is simple and there is no possibility of fraudulent certificates. The key advantage of ID-PKC is that anybody who knows the public parameters and the identity of a member can derive the public key of the member. Thus, the PKI is not needed any more.

However, ID-PKC has its own drawbacks. The private key generator (PKG) which is a trusted third party (TTP) has a master secret key, which is used to derive the private key (along with the public parameters) of each member. Hence, the PKG knows the private keys of all the members, so-called the key escrow problem. For instance, the PKG can forge the signature of any member. What is worse, if the PKG is compromised, the private keys of all the members may be revealed.

To tackle the dependency on the PKG, Al-Riyami and Paterson (AP) proposed the original CL-PKC formulation [23,28]. A CL-PKC system also has a TTP, which is called a key generation center (KGC). Like ID-PKC, the KGC has a master secret key; however, the KGC cannot know the private keys of the members. Instead, the KGC (securely) supplies a member with a partial private key. The member generates its own private key from both the partial private key and its own secret. The CL-PKC scheme is based on ID-PKC in the sense that the partial private key (of a member) is derived from any arbitrary string related to the identity of the member. The member also generates its public key from the KGC's public parameters, and its own secret.

Let us now briefly describe the main algorithms of the AP scheme [23]. Discussing the mathematical details of the algorithms goes beyond the scope of this paper, please refer to [23] for the details. Suppose that the KGC is assumed to set up a system with member m whose identifier is given by ID_m. The member m can obtain its key pairs by the seven algorithms below by itself or by interacting with the KGC.

Setup: In the KGC, this algorithm takes a security parameter k, and returns the system parameters `params` and `master-key`. The system parameters include the message space $\mathcal{M}$, and ciphertext space $\mathcal{C}$, and other cryptographic parameters.

Partial-Private-Key-Extract: In the KGC, this algorithm takes `params`, `master-key`, and a member identifier $ID_m \in \{0, 1\}^*$ as input, and returns a partial private key D_m. The partial private key should be delivered to m over a secure channel.

Set-Secret-Value: In member m, this algorithm takes `params` and ID_m as input, and outputs its own secret value x_m with randomness.

Set-Private-Key: In member m, this algorithm takes `params`, D_m, and x_m as input, and returns the private key S_m. That is, the value x_m is used to transform D_m into the final private key S_m. Thus, the KGC cannot know the private key of member m.

Set-Public-Key: In member m, this algorithm takes `params` and x_m as input, and returns the public key P_m.

Encrypt: This algorithm takes `params`, a message M, P_m and ID_m as input, and returns either a ciphertext C or the null symbol $\perp$ when it fails.

Decrypt: In member m, this algorithm takes `params`, C and S_m as input, and returns either M or $\perp$ when it fails.

Due to the computational complexity of the bilinear pairing, however, schemes using the AP formulation have been questioned about the costly operations. Baek, Safavi-Naini and Susilo (BSS) [24] proposed a certificateless public key encryption scheme without the bilinear pairing by relaxing the AP formulation.

In the original AP formulation, the public key is generated in member m at any time without the intervention of the KGC. The BSS formulation modified key generation algorithms to generate public key only after obtaining the partial keys from the KGC. Later, Lai and Kou (LK) [25] formulation gives more restrictions on the generation of the public key by using a key generation protocol between member m and the KGC. These alterations lead the public key and the identity of member m more linked than the AP formulation.

As a building block of TwinPeaks described in Section 5.2, a proposed scheme of Yao, Han, and Du (YHD) [26] is chosen and is used for the implementation of TwinPeaks evaluation. YHD proposed certificateless public key encryption and signature schemes based on ECC, which is following the BSS formulation. The encryption scheme is constructed 5 steps as followings:

Setup: In the KGC, this algorithm takes a security parameter l and parameters for an elliptic curve as input, and it returns `master-private-key` and the system parameters `params` including `master-public-key`.

Partial-Key-Extract: In the KGC, this algorithm takes `params`, `master-private-key` and ID_m as input, returns a partial private key d_m and a partial public key R_m. The partial key (d_m, R_m) should be delivered to m over a secure channel.

Key-Generate: In member m, this algorithm takes `params`, (d_m, R_m), and ID_m as input, and returns the private key S_m and the executive public key X_m. A random secret value x_m is chosen and used to transform d_m into the final private key S_m. Thus, the KGC cannot know the private key of member m. Also, the actual public key P_m is a point multiplication for a scalar S_m and a base point G within `params` ($P_m = S_m \times G$). In turn, P_m is equivalent to a point addition of `master-public-key` and a point multiplication of hash of ID_m and X_m ($P_m = P_{pub} + H(ID_m) \times X_m$ where P_{pub} is `master-public-key`). Therefore the public key P_m is linked with ID_m and `master-public-key`, and P_m can prove its authenticity by itself.

Encrypt: This algorithm takes `params`, a message M, X_m, and ID_m as input, and returns a ciphertext C.

Decrypt: In member m, this algorithm takes `params`, C and S_m as input, and returns either M or $\perp$ when it fails.

Note that the **Encrypt** algorithm takes ID_m as input (in addition to a message to send), and thus there is an additional linkage with ID_m when a client encrypts a message for a server (i.e., member m). Compared to RSA- or ECC-based public keys, we believe such linkage is important since if the destination of the encrypted message does not know the private key which is also linked with ID_m, it cannot decrypt the message, and hence the masquerading attack will be much more difficult. Therefore, the YHD scheme provides the binding between an entity and its public/private keys without a certificate of RSA- or ECC-based public key.

However, CL-PKC is designed for operations only in a single domain. Also, an input string for key generation (of an Internet server) is not elaborated for realistic attacks and countermeasures in CL-PKC. We apply CL-PKC into TwinPeaks for Internet-scale operations and networking vulnerabilities in order to replace PKI. TwinPeaks is to provide the public key of every named entity or server.

5. How TwinPeaks works

TwinPeaks is a new infrastructure that distributes public keys considering the above criteria. We first explain the overall design of TwinPeaks, followed by how the public key of a named entity (e.g., a website with a domain name) is distributed to a relying party (i.e., a client). We then explain how CL-PKC [23,24,26] is modified to substantiate the new infrastructure for providing public keys.

5.1. TwinPeaks design

For each named entity, TwinPeaks introduces a corresponding key server that provides its public key online. The key servers collectively construct a DNS-like hierarchy, which is called *a key server hierarchy*.

Instead of the PKI, TwinPeaks key servers take a role of public key distribution. The followings look into the key servers, the construction of the key server hierarchy, and the step-by-step operation of TwinPeaks.

5.1.1. Key server

A key server distributes public keys of its corresponding entities. The named entity registers its name, public key, and other required information to the key server. For example, a www.example.com web server registers its FQDN, a corresponding public key, and an IP address to the key server of example.com domain. The registered information contains the public information such as a part of DNS or Blockchain entries.

A key server also registers its information to the another key server. When the information of the key server A is registered to the key server B, we call the key server A as a *child* and the key server B as a *parent*. For example, example.com key server registers its information to `.com` key server, just as the DNS-like hierarchy.

A key server can be working as a KGC, or have its dedicated KGC. The registered named entity or key server should be a member of the KGC of the key server.

5.1.2. DNS and key server hierarchy

TwinPeaks consists of two parts: one is the public information, such as the legacy DNS hierarchy, and the other is the key server hierarchy (as shown in the left and right part of Fig. 1, respectively). While Fig. 1 shows only a single root in the key server hierarchy in detail, the TwinPeaks architecture supports multiple roots. A country, a regional alliance, or a large organization can construct its own root key server to avoid the issues of a single root of trust (which might be problematic as in DNSSEC and RPKI).

Note that every link in the DNS hierarchy reflects the current business or administrative relation already established among different DNS servers (or domains). Thus, it is not too difficult for a domain to deploy another key server in addition to its DNS server. Recall that a parent DNS server and its child usually set up a security association for secure dynamic updates of DNS entries. Thus, it should be easy to securely distribute a partial private key from a KGC (i.e., a parent) to its member (its child) in the key server hierarchy.

Each root key server has its own public key parameters and serves as a KGC on its own, which operates independently of other root key servers. At the next lower level in the hierarchy, a *top level key (TLK) server* corresponds to a top level DNS server in the DNS. Usually, a TLK server can be a member of each and every root key server, separately. That is, if there are n root key servers, a TLK server generates up to n independent key pairs.

However, we allow a TLK server not to be a member of some root key servers due to political and distrust issues. Thus, the TLK server will have its public/private key pair for each root key server which it is associated with. Also, every relying party is assumed to pre-load the public keys, IP addresses, and public parameters of multiple root key servers. (This is similar to the current practice that browsers and operating systems include the certificates of trusted root CAs.) Henceforth, for the sake of exposition, we will assume only a single root key server by default.

5.1.3. Message types

There are two kinds of messages for obtaining a public key in TwinPeaks: one is for query, and the other is for response. The new messages are only for communicate with key servers, and the ordinary DNS query-response remains unchanged.

The *query* message (*key request message*) consists of ID components, such as a domain name, a IP address, other DNS entries which are required, and the cryptographic hash of itself. The packet length is variable because the length of ID (and also its components) is not predefined.

The *response* message (*key response message*) consists of ID components as the query message does, and the public key which is corresponding to the requested ID follows.

5.1.4. In-depth operations

Suppose a client wishes to establish a secure connection to the server with the domain name of www.a.com. Note that there is a preconfiguration of the public key, the IP address, and public parameters of a root key server on the client (Step 1 in Fig. 1). As the server www.a.com is a member of the domain (`a.com`), the public key of www.a.com has been registered with the `a.com` key server (Step 2). Similarly, the public key of `a.com` is registered with the `.com` TLK server, which in turn registers its public key with the root key server.

After the client looks up the IP address of www.a.com as public information from the DNS (Step 3a to 3c),[1], the client starts to resolve the public key by sending a *key request message* to the root key server. The root key server then replies with a *key response message*, which contains the public key, the IP address, the public parameters of the TLK server (`.com`), and the signature of the root. This process will be iterated downward until the client obtains the public key of www.a.com. In general, public key lookup is similar to the iterative handling of DNS queries along the DNS hierarchy (Step 4a to 4c).

Finally, with the obtained public key and IP address, the client can assure the authenticity by sending an encrypted message using the public key to the IP address. If the server www.a.com can decrypt the message with its private key, it can reply the authentic response back to the client. In contrast to the CAs' burden of key verification in the PKI, the client and server in TwinPeaks interact for public key verification. As a side benefit, TwinPeaks is more resilient to version rollback attacks since a client does not have to send messages in plaintext.

Additionally, the key server hierarchy is also used for issuance of partial keys when establishing the secure channel with KGC. A member of the domain can obtain the public key, the IP address, and the

[1] The local recursive DNS resolver is omitted in Fig. 1 for simplicity.

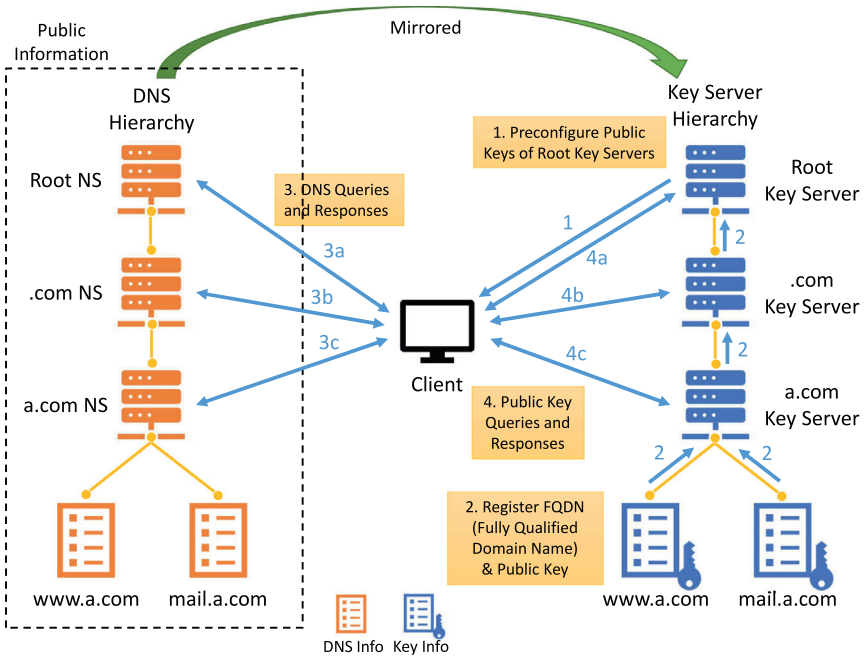

Fig. 1. Overview of the TwinPeaks operations is shown. To connect to website www.a.com, a relying party retrieves its IP address from the DNS hierarchy on the left, and its public key from the key server hierarchy on the right.

public parameters of the corresponding key server by looking up key servers from the root iteratively. Note that the key server works as a KGC on its own or has a dedicated KGC, the secure channel to the corresponding KGC can be established using the retrieved public key.

5.2. CL-PKC as a building block

CL-PKC is an algorithm for public-key-based encryption and decryption as RSA and ECC, not an infrastructure as the PKI. To bring the algorithm into the Internet, in TwinPeaks, some modifications need to be made in consideration with the concept of autonomous domains. Thus we need to extend environments of CL-PKC for operations in multiple domain environments of the Internet.

In order to work through independent domains with TwinPeaks, we adopt the DNS hierarchy partly because two linked nodes (say, parent and child) in the DNS hierarchy usually have security associations already. For instance, a parent DNS server and a child DNS server usually share a secret for secure dynamic updates of DNS entries [29].

At the top of the hierarchy, there is a root key server that corresponds to a root DNS server. From the root key server, the next lower level key server is a TLK server that corresponds to a top level domain (TLD) DNS server. In this case, the root key server works as the KGC of a domain, and the TLK server is a member of the domain. Thus, the root key server and the TLK server perform the CL-PKC algorithms in Section 4 so that the TLK server has its own private/public key pair.

The same relation/interaction will be iterated as shown in Step 2 of Fig. 1 between the TLK server and the second level key (SLK) server (say, example.com key server). Then we assume that the public keys of named entities (e.g., websites) that belong to the same domain are registered with the SLK server. That is, the key server of example.com maintains the up-to-date public keys of all of its members in the domain.

Another modification of the algorithm is related to the ID naming scheme of the CL-PKC algorithm. As explained in [23], ID_m is any string about member m's identity. We concatenate the domain name, the IP address of the member, and the hash of public parameters of the entity's key server (against the forgery of the public parameters). Altogether, the entity m's identity for the secret value is given by ID_m = `domain-name||IP-address`[2]$||h($`params`$)$, where $h(\cdot)$ is a hash function. In case its domain name or IP address is changed, the member should change its public key due to the binding of ID_m and its public key.

6. Operation and deployment of TwinPeaks

6.1. Public key update

The public key with ID_m in TwinPeaks may be used long term depending on the policy. However, we need to focus on the conditions which bring enforced public key updates such as node compromises or IP address changes by modifying its identifier ID_m.

6.1.1. Update cases

First, The key server compromise comes from the following cases. (i) In case of the compromise of the parent key server, the parent key server (as a KGC) should re-establish its master key and public parameters. Then the key server (as a child) should re-build the public/private key pair with the parent. (ii) In case of the compromise of the key server itself, the key server (as a KGC) should update its master key and public parameters. Note that compromise of a key server does not affect all the descendants but only its children.

Next, in case of the compromise of a named entity, it regenerates its secret value, and re-establishes its public/private key pair with its key server.

Finally, in case of the IP address change of a key server or a named entity, its secret value is required to be updated.[3] Then its public/private key pair is also changed. Thus, the new public key should be registered with its parent node in the key server hierarchy. Note that its new IP address should also be registered with its parent node.

6.1.2. Update as implicit revocation

While we use the term *'update'* for the case in which an old public key is discarded and a new key is issued in TwinPeaks, the *'update'*

[2] Here, `IP-address` is an IPv4 or IPv6 address, which is public and routable.

[3] Recall that the IP address is an element of the ID in TwinPeaks.

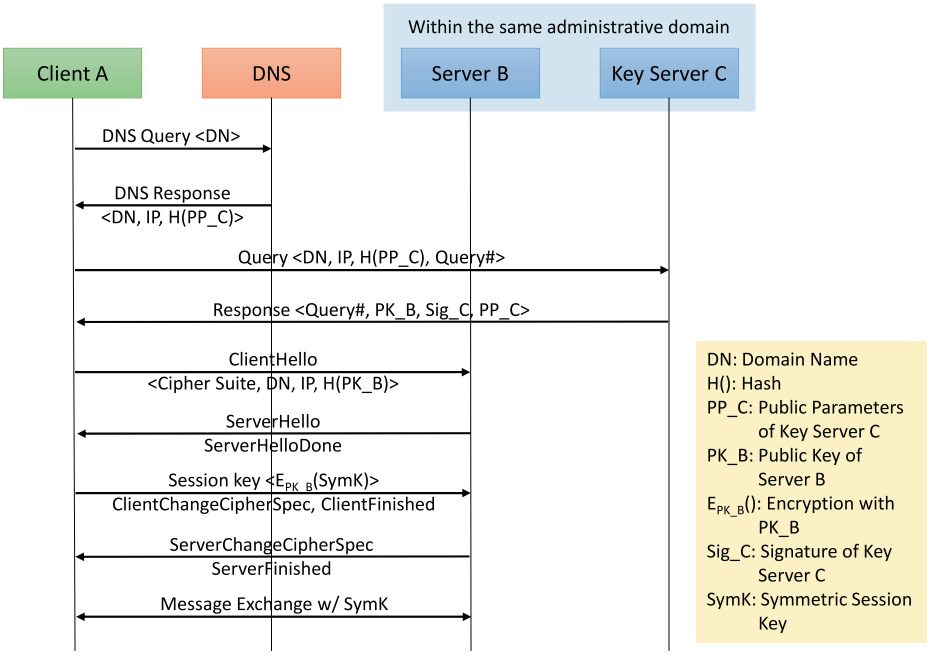

Fig. 2. Operations of the TLS variant for TwinPeaks are illustrated. To connect to server B, relying party A retrieves B's IP address from the DNS, and B's public key from the key server hierarchy (only the last key server is depicted for brevity). After that, the relying party starts simplified TLS handshakes.

mechanism can be deemed roughly equivalent to *'revocation'* in the PKI. Periodically updating public keys would help enhance the overall security in TwinPeaks. Sometimes, periodically updating a public key is possible without changing its ID, when the node wants to update its key periodically or occasionally for stronger security. In that case, there are no differences with explanations above. The updated key will be distributed from the key server immediately.

Just as the IP address of a domain name can be cached in the DNS, so the public key of a server can be cached in TwinPeaks. With caching, there should be some mechanism to notify key updates, which is detailed in Section 6.2.

6.2. Public key caching

TwinPeaks can achieve scalable distribution of public keys of named entities since public keys can be cached like DNS resource records. As a local DNS resolver caches recent DNS responses, a local key resolver of TwinPeaks can cache recent responses from key servers. The cacheable responses are not only the public key of the node, but also the public keys of the key servers along the key server hierarchy. With the caching mechanism, retrieving a public key can be accelerated, and the load of the key server hierarchy will be relieved significantly.

The key resolver incurs no additional communication overhead for caching operation. For caching, all the required data can be obtained by observing key request and response messages. The additional cacheable data includes the ID related to the public key, the public parameters used for public key generation, and the signature of the key server of the node are cached along with the public key.

If the public key of the node is updated, the client can figure out the update from responses from the node itself (say, 'outdated public key' in *ServerHello* in Fig. 2, to be detailed later). Then the client will ask the key resolver to invalidate the cached entry, and to retrieve the new public key again.

6.3. Deployment

6.3.1. Island approach

TwinPeaks can be deployed incrementally, which is important since ISPs cannot deploy a new network infrastructure in a coordinated fashion. Even if TwinPeaks has the DNS-like hierarchy, it does not have to mirror the DNS hierarchy in the same fashion. It can be deployed in the form of islands since a "local" root key server can be located at any level in the DNS hierarchy.

6.3.2. TLS Variant

Another merit of TwinPeaks is that the existing security protocol (e.g., TLS 1.2) can be substantially simplified with the new infrastructure. In the following, we will illustrate how TLS can be simplified for the secure connection setup. Note that most of the TLS protocol is retained, while the handshake part is substantially simplified with TwinPeaks.

We change the TLS handshake procedure for the operations in TwinPeaks as shown in Fig. 2. The major difference is that the certificate exchange and verification in TLS are removed since the client already retrieves the public key of the server during the key resolution. Note that the time to verify the certificates from CAs (e.g., OCSP servers) is also eliminated. When a client initiates a TLS handshake, it sends the domain name, the IP address, and the hashed value of the server's public key and public parameters in the *ClientHello* message (in addition to the original fields like the list of cipher suites). Next, the server, on receipt of the message from the client, can confirm the correctness of the information in the message. If it is wrong, the server can identify which field has a problem, and depending on the problem(s), it can notify the relevant entities (like the client or the key server) of the identified problem(s). If the information is correct, the server and the client continue to exchange TLS messages to derive a shared session key.

6.3.3. Consideration of the Internet practice

We have discussed basic operations of TwinPeaks so far, however, the current Internet has more complex practices.

Multiple IP Addresses for a Domain Name: A single domain name can be mapped to multiple IP addresses, e.g., load balancing, or use of content delivery network (CDN) services, etc. In TwinPeaks, a number of public key for a domain name is allowed, thus a new public key can be generated for each IP address. If the DNS returns multiple IP addresses for a requested domain name, the client chooses a single IP address for the Internet communication. In the same way, the client can choose a matched public key with the chosen IP address. In this case, the corresponding key server has the burden of managing the public keys for all the IP addresses of the same domain name.

HTTP Redirection: In the web, CDNs rely on application-level rerouting or DNS-level rerouting to forward the request to a close replication point. In case of application-level rerouting, it can be handled regardless of TwinPeaks. For example, a URL address in the HTTP GET request can be redirected to another URL. Then the client will perform another resolution for the newly-acquired URL. However, DNS-based request rerouting requires some extension to TwinPeaks. In many cases, DNS-based request routing utilizes a CNAME resource record in the DNS for redirection [30]. Suppose that a.com is redirected to CNAME a.cdn1.com at the DNS level. Then the `a.com` key server can indicate the corresponding CDN provider's key server.[4] After that, the client compares the CNAME resource record of a.com (which is actually a.cdn1.com) and the delegation information from the key server, and then verifies whether the two responses match.

Cloud-based Services: Number of Internet services are based upon cloud services because of the elasticity of resources. However, scale-out of service or migrating resources (e.g., virtual machine (VM) images) may result in frequent IP address changes. In this case, most of IP address changes occur within data center networks and the public IP addresses which are typically used for front-end servers are remained unchanged. Thus clients can connect to the same public IP addresses for secure communication without frequent changes of the public keys related to the cloud-based services.

7. Applying TwinPeaks into IoT environments

IoT is becoming popular in our daily lives. However, IoT devices and services are designed and developed for individual situations and scenarios without standard ways of authentication. It makes it difficult to interconnect devices/services from different manufacturers/providers.

Security technologies of the Internet may be applicable to the IoT landscape. However, there are issues regarding the authentication in IoT. For example, using PKI in the IoT results in inappropriate usage cases such as (i) hard-coded root certificates inside the IoT device, or (ii) bypasses of verification of certificate chains due to the communication and computation limits of IoT devices.

Addressing two reasons aforementioned, some hub devices like smart speakers, such as Amazon Echo [31] or Google Home [32], use back-end authentication [7] and interconnection through OAuth [33] token exchanges. This token-based authentication causes long-haul communications, which result in large delays and user intervention for every step of back-end authentication. Also, each hub device has its own specification for the authentication and interconnection, which leads to additional burdens to other manufacturers/providers.

TwinPeaks can help devices authenticate and interconnect across each other for IoT services. It eliminates the communication overhead of verification along the CA. Instead of the CA overhead, the verification of a TwinPeaks public key is done with the known ID of the IoT device. Also a TwinPeaks public key has a longer validity period and thus it is more cache-friendly compared to the PKI public key and certificate.

7.1. IoT Scenario

Suppose that a house is configured with a smart hub device such as a smart speaker, and we would like to add a new device such as a robot vacuum cleaner into the house. To make the hub and the new device set up a secure association, each device needs to authenticate the other.

For the authentication, the current practice is that the authentication messages are exchanged between the remote servers. Each device has been registered at the server of its domain, and the server handles authentication and interconnection between the two domains. This back-end model incurs large delays, and thus it may result in service disruptions in IoT devices even if they are closely located.

[4] The CNAME resource record from the DNS can be used to verify the delegation (by the key server) to its corresponding CDN provider's key server.

Instead of the back-end model, the direct authentication between devices will reduce the delay substantially. In the Internet, we have witnessed a PKC-based authentication is viable for the direct authentication. Compared to the current PKI approaches, TwinPeaks has benefits over the scoping and scalability as described in Section 3 in the IoT environments.

7.2. ID Naming

In order to apply TwinPeaks into the IoT, the ID system needs to be extended since some IoT devices do not have their own public IPv4 addresses. If a device has a public IPv4 address, ID_m in Section 5.2 can be used without any changes. Otherwise, when a device has a private IPv4 address or it is intermittently connected to the Internet, a unique value of the IoT device (e.g., generated from MAC address, UUID, and etc.) could be used for the ID construction.

When we use a unique value as an ID, the entity m's identity for the secret value is given by $ID_m = \texttt{domain-name}||\texttt{unique-value}||h(\texttt{params})$. Thus, if its domain name or unique value is changed, m should change its public key.

7.2.1. Examples of unique value construction

Globally unique IPv6 address from the MAC address: If a public IPv4 address is not allocated to an IoT device, the use of a globally unique and routable IPv6 address is one of the options. For example, a globally unique IPv6 address can be generated from the modified EUI-64 format IID (interface identifier) with the given subnet prefix by SLAAC (stateless address autoconfiguration), when the device is connected to the Internet through an IPv6-capable router [34].

Unique local address of IPv6: Use of unique local address, which is non-routable, is also viable approach because ID_m has a scope of unique value as the domain name. The pseudo-random global ID algorithm of RFC

[35] can be applied to construct the unique value for ID_m.

Personal IPv6 address: In a person-centric perspective, the 'personal IPv6 address' for consumer-based business model [36] is an another candidate for the unique value of an IoT environment. Ganchev and O'Droma [36] proposed a locally-routable address scheme with NAT translation at the egress router of access network providers for global routing (if supported). The personal IPv6 address is constructed as 3 parts: class prefix, consumer ID, and assignable sub-address/ID. If an IoT network uses a consumer ID, an IoT device can have an assignable sub-address as a natural transition from person-terminal concept of [36].

7.3. TwinPeaks-based authentication in IoT

TwinPeaks-based authentication in IoT is performed in five steps as depicted in Fig. 3. There are three entities involved: IoT devices, key servers, and a public registry such as the DNS. Suppose there are two IoT devices to be interconnected and they belong to two different IoT domains. Let us describe the authentication process as an IoT device A authentications the other one B. Cross-authentication can be done if both of the two devices execute the process.

Initialization: A key server within the IoT domain of device A provides the required parameters for the public key generation to IoT device A. We assume that every IoT device has its authentication hint, which is information that uniquely identifies the device itself, to be detailed in Section 7.4. And the authentication hint of device A is collected by the key server X. (Device B belongs to another domain, which performs the same initialization with the key server Y, independently.)

Discovery: We assume device A can find device B by soliciting B's ID and hash of the public key via local area wireless communications such as WiFi, Bluetooth, and ZigBee.

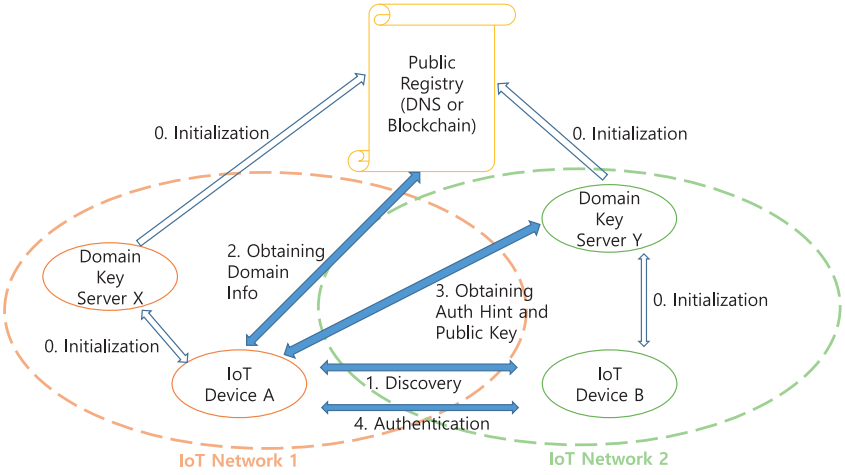

Fig. 3. Overview of TwinPeaks-based operations for IoT Authentication.

Obtaining Domain Information: From the obtained ID of *B, A* can figure out the domain of *B*. And then the contact point of *B*'s key server *Y* would be obtained via a public registry, such as the DNS.

Obtaining Authentication Hint and Public Key: Device *A* verifies the given ID of device *B* with the authentication hint from *B*'s key server *Y*. *A* also obtains the public parameters of key server *Y* and the public key of *B*.

Authentication: With the public key, the given ID, and the public parameters of communicating peer, each device verifies the authenticity of the other device by exchanging encrypted messages or signatures.

7.4. Authentication hint and device profile

An authentication hint will be exchanged in a form of a device profile. The authentication hint includes the data for identifying its IoT device, e.g., the date and time of its initialization, the MAC address, the manufacturer-provided values, and/or the randomly-generated value at the device bootstrapping. These values construct the unique ID of the device. Thus the device could be authenticated via the domain name and the valid authentication hints, by verifying the unique value construction procedure.

Also, the device profile contains the information related to automated negotiation of available security levels. For example, types of asymmetric cryptography, types of symmetric cryptography, and their parameters.

For authentication and encryption, the common subset of the two profiles will be used. The negotiation could fail, if the common subset is empty, and thus the baseline profile is required. If there are two or more common candidate profiles, the choice will be made based on the policy of the IoT domain or the local administrator. The default policy could be choosing the strongest level of security. In order to compare the candidate profiles, each cryptographic scheme provides the security level of the equivalent security level of symmetric cryptography.

7.5. Public registry

TwinPeaks relies on DNS for a public registry. For the TwinPeaks-based application on IoT, DNS is also preferred to lookup the domain and its key server information timely, when an ID_m with the non-routable unique value is used. However, DNS may not scale when the number of domain names grows enormously, in case that the routable IP address, such as the globally unique IPv6 address, is used with ID_m.

Other than DNS, the blockchain (e.g., Namecoin [37]) may be a candidate solution for storing the domain name and its key server information of the IoT devices with benefits of scalability and the following reasons. The data lookup within the blockchain takes the overhead of constantly tracing transaction records, compared to DNS. However, the blockchain provides a tamper-evident history of the device-related information, and thus it is robust against the forgery of the information. If the IoT device cannot afford to have a computational power for looking up the blockchain records, the device may delegate the lookups to its domain key server which is a trusted entity.

8. Security analysis

8.1. Threat analysis

8.1.1. Threat model

An adversary may launch the following attacks to masquerade a website. First, he or she may wish to compromise either a key or a DNS server. If he or she compromises both servers, then the attack is a success. Second, he or she may make poisoning attacks on a DNS or a key resolver.

Adversaries can be internal for the former attacks, and can be external for both kinds of attacks. An internal adversary has proper control over either a key server or a DNS server, but he or she wants to alter the public key or DNS entries for obtaining access to the client-server communication illegally. An external adversary has no legal control over the infrastructure, but he or she tries to gain control of either a key server or a DNS server or to disguise a malicious host as an authentic one.

For the former type of attacks, it is assumed that an adversary can successfully compromise only one entity: either a DNS server or a key server. We assume that the operation and management of the DNS hierarchy and the key server hierarchy are independent (as the DNS and CA hierarchies are independent). He or she could start an attack inside the domain (of the website) or manage to gain the access to the domain by an advanced persistent threat (APT) [19]. For the latter type of attacks, he or she may be located around the target client.

Also, we assume that Internet routing delivers a packet to its destination IP address correctly; IP prefix hijacking is thwarted by BGPSEC and so on. Note that TwinPeaks focuses on authentication and public key distribution. That is, we can rely on the existing mechanisms like DNSSEC and TLS to thwart man-in-the-middle attacks and down-grade attacks.

8.1.2. Attack and analysis

Compromised key server: Suppose an adversary wishes to masquerade a famous website with identifier ID_A. Assume that he or she compromises the key server that is responsible for the public key of the website ID_A, and forges its public key as P'_A with parameters $\texttt{params}_A$. As the DNS server is not compromised, the client's ciphertext $C_1 =$

$Encrypt(\texttt{params}_A, m_1, P'_A, ID_A)$, which is encrypted by the forged public key P'_A, will be delivered to the authentic web server, where m_1 is the client's message. Now, the authentic web server cannot decrypt C_1 correctly with the authentic public key P_A, and hence the client figures out that the given public key is not valid.

Compromised DNS server: Likewise, if the corresponding DNS entry is forged for a given website, or a DNS poisoning attack is successful (but the key server is not compromised), then the client has the wrong IP address and wrong identifier ID'_B but the valid public key P_B of the website with parameters $\texttt{params}_B$ from the key server hierarchy. The client's ciphertext $C_2 = Encrypt(\texttt{params}_B, m_2, P_B, ID'_B)$, encrypted by the valid public key, will go to a wrong destination, possibly a spoofing host, where m_2 is the client's message. However, the spoofing host cannot decrypt C_2 correctly since it does not have the authentic private key S_B. Note that public key poisoning attacks are not possible since the client will know the public key of each key server along the key server hierarchy, and hence it will check the digital signature of each key response message.

Therefore, an impersonation attack in TwinPeaks is possible only if both the DNS server and the key server are compromised.

Compromised root key server: Root key servers are much more important than lower level key servers since the resolution of a public key starts from a root key server. An adversary may try to compromise the root servers to forge the entire hierarchy. Some approaches can mitigate attacks or help defenses as follows.

Recall that a structure of TwinPeaks is multi-rooted in a global scale. If a root key server is compromised, we assume that the adversary obtains the secret key of the root key server. It is hard to distinguish whether the root key server performs the normal or malicious operations. This vulnerability can be alleviated with cross-verification among multiple root key servers, which means that the client sends a key request message to multiple root key servers. By comparing the responses from the multiple root key servers, the evidence of one or more compromised root key servers could be found. However, if all the root servers are compromised, the attack is succeeded against the countermeasure of cross-verification.

Also, root-targeting attacks could be neutralized via out-of-band distribution of the public keys of root servers. This approach is already in use in the form of pre-distribution of root certificates in PKI, such as software distribution of web browsers and operating systems.

Compromised root DNS server: Similarly with the case of compromised root key server, the root DNS servers are very important and there is a risk to be attacked. However, in practice, they are hard to be compromised because they are well distributed and sustainable due to techniques of anycast, caching, and redundancy.

8.2. Certificateless validation of a public key

The validation of a public key of a named entity is performed in TwinPeaks by two mechanisms: (i) encryption (and decryption) with the pair of public and private keys including the form of signature, and (ii) the responses from the DNS and the public key servers. Recall that the generation of a public/private key pair is linked with the identifier of the entity.

As ID_m depends on its networking parameters, such as domain name and IP address, the correctness of the responses from the public registry (or the DNS) and the key servers are the basis of the validation. If one of the elements mentioned above is incorrect, the decryption of the ciphertext cannot be done succesfully. Note that the certificate validation is done by the issuer CA in the PKI, whereas the (key) validation in TwinPeaks depends on the client and on the network infrastructure. Dependency on multiple entities for validation of a public key in TwinPeaks helps detect invalid public keys and figure out the cause of the validation failures. In addition, modified TLS handshake messages can help explain the cause of invalidation (see Section 6.3.2).

9. Evaluation

We compare TwinPeaks with the current PKI and IETF DANE [16,17] from qualitative and quantitative perspectives. Numerical experiments are conducted to show the feasibility and performance of TwinPeaks in the Internet environments, where a client verifies the authenticity of a server. The TLS variant explained in Section 6.3.2 is used for comparison purposes. With the PKI and DANE, the original TLS 1.2 is used for establishing secure connections.

9.1. Qualitative comparison

TwinPeaks eliminates certificates, and hence removes the overhead of certificate issuance and revocation. Instead, a domain has the responsibility of keeping the IP addresses and public keys of its members up-to-date in its DNS and key servers, respectively. Each key server can issue keys only to its members; thus the problem of the issuance of certificates to arbitrary entities in the PKI is mitigated.

The IETF DANE standards address some of the PKI's problems by incorporating the certificate of an entity into its DNS record. As DANE incorporates many of current PKI practices, the PKI-related problems like the overhead of revocation still remain. However, DANE can solve the CA dependency if named entities decide to use self-signed certificates as TLSA records within the DNS entry. Furthermore, the usage of self-signed certificates in DANE can help limit the scope of key issuance to the sub-domains of the current DNS entry.

While DANE may mitigate the risk of fraud certificates, the compromise of the DNS server can result in spoofing attacks. In TwinPeaks, the public keys are stored in separate physical servers from DNS servers, which makes the compromise of the DNS server independent of that of the key server. DANE as well as the PKI cannot take advantage of long-term caching since certificates are to be fetched from the servers due to the TLS, and if certificates are issued by CAs, they need to be validated anyway. However, in DANE, only self-signed certificates (or its hash values) can be cached.

We summarize the above comparison as well as the prior survey in the literature. [38] surveyed and evaluated number of proposals for enhancing the PKI. Based on [38] and the aforementioned issues, qualitative comparison of TwinPeaks with the PKI and DANE is shown in Table 1.

9.2. Quantitative comparison

In order to examine the feasibility of the proposed architecture, we evaluate TwinPeaks with PKI and DANE. The evaluation pursue the fairness by comparing different architectures, especially the point of occurrence in key distribution varies. We use the TLS handshake procedure and assume the DNSSEC deployment in comparing the three schemes because DANE requires DNSSEC.

9.2.1. Evaluation environments

To mirror a globally distributed network infrastructure, two cloud service providers which are located in North America and East Asia are used. Therefore the latencies among the client, the server (or website), the DNS servers, key servers, and CAs reflect global-scale Internet environment by setting up physically-distant VMs. The VMs that run TwinPeaks, PKI, and IETF DANE are chosen as one of *medium-ranged* recommendations of the cloud service providers; each VM has 2 virtual cores and approximately 4 GB of main memory. If we use entry-level VMs (which have lower priority), unstable performances like throughput instability and delay variations are reported [39]. By choosing the medium-ranged VMs, the side effect of virtualization that affects network performance is reduced.

A hierarchical structure is applied to connect DNS servers and key servers, respectively. Three levels of DNS servers are deployed to emulate the current DNS structure, which are the root, the TLD (e.g., `.com`),

Table 1
Comparison of TwinPeaks, PKI, and IETF DANE schemes is summarized. Properties that are fully/already supported are denoted by •, while ones partially fulfilled are denoted by ◦. Note that • is always better than ◦, which in turn is better than blank for each criterion.

		PKI (baseline)	DANE (key pinning at DNS)	TwinPeaks
Security Properties Offered against Certificate Forgery, CA Compromises, etc.				
	Detects MITM		•	•
	Detects Local MITM		•	•
	Updatable Pins		•	•
	Responsive Revocation			•
	Intermediate CAs Visible			•
	Restricts Scope of Key Issuance		•	•
	Reduces Key Retrieval Overhead with Cache		◦	•
	No Revocation Overhead		◦	•
	Easy Recovery of Spoofed Certificate/Key		•	•
Evaluation of Impact on HTTPS/TLS				
Security & Privacy	No New Trusted Entity	•	◦	
	No New Traceability	•	•	•
	Reduces Traceability	•	•	•
	No New Authentication Tokens	•		•
Deployability	No Server-Side Changes	•	◦	
	Deployable without DNSSEC	•		•
	No Extra Communications	•	•	
	Internet Scalable	•	•	•
Usability	No False-Rejects	•	•	•
	No New User Decisions	•	•	•

and the second-level domain (e.g., example.com) name servers. A client is located in East Asia, and hence the root name server is also located in East Asia to reflect the practice of IP anycasting. The TLD name server is located in the Western coast of North America since we assume that a generic TLD name server may be located there. The second-level domain name server is located in East Asia again, assuming that the client tries to set up a secure connection with a local content provider. The locations of the corresponding key servers are the same as those of the DNS servers, respectively.

We use `BIND` for the authoritative DNS server configuration without recursion, allowing DNSSEC and DANE features. `BIND` is also used for a local DNS resolver, which is modified to indicate our root server and to allow recursive DNS queries. OpenSSL, an open-source SSL/TLS implementation, is used for the CA and its OCSP responder, which are located in East Asia. Other required entities (such as the TwinPeaks key servers, the server, and the client) are prototyped with the following libraries. Bouncy Castle cryptographic APIs are used for the RSA-, ECC-based computations and TLS connection establishment. JPBC2 is used for pairing-based cryptography (PBC).

9.2.2. Evaluation methods

To compare TwinPeaks with the PKI and DANE, we measure the resolution delay, which is the interval from the starting time of the DNS query of a client to the starting time of a TLS session setup. DNS server hierarchy is configured as three-level domains, and it is mirrored by key server hierarchy. For PKI, an OCSP responder is set up for a CA operation of public key verification. The local key resolver is co-located with the client for simplicity.

We consider two cases with respect to caching. First, we make all the caches cleaned to measure the elapsed time with no cache hits. In this case, all the required communications and computations take place. Next, the client makes a connection (i.e., to the same server) again to find out the effectiveness of caching for each scheme. For both cases, we do not activate (application level) content caching (e.g., web cache).

We carry out the experiments with the key length changed to observe the performance difference between RSA/ECC-based PKI, RSA-based DANE, and CL-PKC-based TwinPeaks. The matching key lengths between RSA, ECC and CL-PKC are determined based on the literature [40] as shown in Table 2. We average the experimental results over 10 runs for each setting.

There are number of approaches in implementing CL-PKC [28]. Due to the high complexity of bilinear pairing, we prefer approaches without pairing, e.g., the BSS formulation [24], the LK formulation [25], for comparison. Thus we adopt the YHD scheme [26] based on ECC in evaluation. Later, we also compare the two implementations of CL-PKC, which are denoted by TwinPeaks (PBC) and TwinPeaks (ECC). Note that PBC and ECC-related parameters are chosen based on [40–42] for the computational efficiency and flexibility.

9.3. Numerical results

We measure the DNS resolution delay, authentication delay, other TLS connection setup delay, traffic for TLS connection setup, caching effects on key resolution, comparison of TwinPeaks implementations with PBC and ECC, and feasibility check for the IoT. Prior to the establishment of secure connections, the client will access its local DNS resolver in order to obtain the IP address corresponding to the given domain name (i.e., the server). In case of the PKI and DANE, only the DNS resolution is required before starting to set up a TLS connection. The TLS connection setup (i.e., TLS handshake) in the PKI and DANE includes authentication and symmetric key generation. In contrast, TwinPeaks requires the explicit key resolution after the DNS resolution. Then, the TLS connection setup in TwinPeaks only includes symmetric key generation since the authentication (of the server) corresponds to the key resolution. For the experiments below, the simplified TLS variant for TwinPeaks is used (see Section 6.3.2).

9.3.1. DNS Resolution delay

1

All the comparison settings require DNS resolution before establishing a TLS connection. Therefore there is no difference across the three schemes. For comparison with other delay components, we measure the DNS resolution delay as follows. We assume that all the DNS entries are signed by DNSSEC, and the local DNS resolver is DNSSEC-enabled; 2048-bit Zone Signing Keys and 4096-bit Key Signing Keys are used for DNSSEC. Even though TwinPeaks does not rely on DNSSEC, DNSSEC is employed for conservative comparison purposes. In the DNS resolution of DANE, the hash value of the server's certificate (not the full certificate) is used for faster resolution.

Table 2
Parameters of RSA and CL-PKC (elliptic-curve-based) algorithms are shown as the key size increases (which corresponds to security levels). Unit is in bits.

Security Level (~ symmetric crypto)	RSA Key Size	CL-PKC (EC) Key Size ($\log r$)	Size of Finite Field $\mathbb{F}_q$ ($\log q$)	Size of Extended Field $\mathbb{F}_{q^k}$ ($\log q^k$)	ρ ($\log q/\log r$)	Embedding Degree k
80	1024	160	512	1024	3.2	2
112	2048	224	1024	2048	4.6	2
128	4096[a]	256	2048	4096	8	2

[a] As shown in [40], RSA-equivalent key sizes are different depending on the references. For example, NIST recommends RSA 3072-bit key size and Lenstra recommends RSA 4440-bit key size for 128-bit security level. We use 4096-bit key size for RSA in this case.

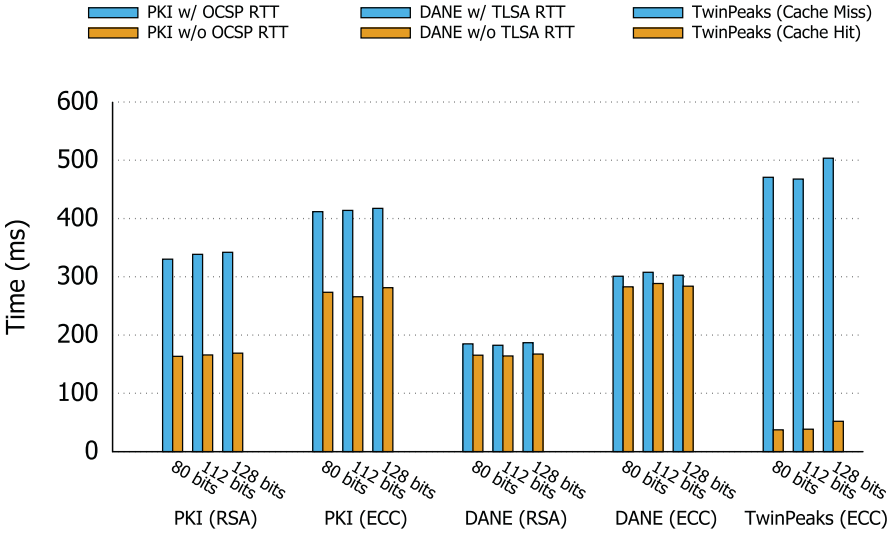

Fig. 4. Average authentication delays of the PKI, DANE, and TwinPeaks are analyzed (in ms) during/before the TLS process.

The measured DNS resolution delay is significantly reduced in case of cache hits within the DNS resolver. The average DNS resolution delays are 168.5 ms for the cache hit and 1,382.7 ms for the cache miss, which shows over 8 times speedups with cache hits. This gap includes the networking latencies between entities and delays for DNSSEC operations.

9.3.2. *Authentication delay*

In TwinPeaks, the client obtains the public key of the domain name from the key server hierarchy; note that the DNS retrieval and the public key retrieval are performed in sequence. Due to the key resolution in TwinPeaks, the authentication (i.e., certificate verification) in the TLS is removed (Section 6.3.2). Thus we compare the authentication delays of the three schemes in terms of certificate verification in both PKI and DANE, and key resolution in TwinPeaks.

The measured authentication delays before/during the TLS process is shown in Fig. 4. The authentication delay is defined as the interval from the TLS initiation to the end of server authentication (i.e., right before sending a secret for the shared key setup at the client) for PKI and DANE. Overall, the measured results show small differences across different security levels, which means the most of the time is consumed for networking.

Like the DNS resolution, the key resolution time of TwinPeaks (ECC) is significantly reduced with cache hits in the key resolver. In TwinPeaks, the average measured time for key resolution is 37.3 ms with cache hits and the 80-bit security level. In comparison to cache-miss cases of 470.9 ms, it is over 12 times speedups with cache hits.

Before analyzing the authentication delay of the other schemes, let us detail the settings for Fig. 4. OCSP and CRL are used for validating certificates in the PKI, which might incur delays due to visits to CAs (compared to DANE and TwinPeaks). In this experiment, OCSP is adopted for certificate validation in the PKI and we consider the two following cases. (i) The client visits the OCSP server only once; this case corresponds to the 'w/ OCSP RTT' part in Fig. 4. (ii) In some servers, they adopt OCSP stapling, where an OCSP response message (for the server's certificate) signed by its CA is sent by the server during the TLS handshake. Then the client does not have to visit the OCSP server; this case corresponds to 'w/o OCSP RTT' part in Fig. 4. Similarly, there are two cases as to DANE. (i) The client visits the DNS server once again to retrieve the TLSA record; this case corresponds to the 'w/ TLSA RTT' part in Fig. 4. (ii) The client has the information (e.g., CRL preloading) that can verify the authenticity of the server's certificate without visiting the CA; this case corresponds to the 'w/o TLSA RTT' part in Fig. 4.

The authentication delays of PKI (RSA and ECC) in cases of visiting the CA are comparable to those of TwinPeaks (ECC) with cache misses. When we see the 2nd case of PKI (i.e., not visiting the CA), TwinPeaks (ECC) with cache hits shows substantial gains in terms of the authentication delay.

In DANE results, the authentication delays of the 2nd case are slightly less than those of the 1st case (visiting the CA once). The reason why the two cases shows small difference is that the difference comes from the DNS resolution for DANE authentication (TLSA) which takes less than 20 ms regardless of settings. Again, TwinPeaks (ECC) with cache hits shows better performance compared with DANE in the 2nd case.

Notably, the authentication delay of the PKI is not necessarily increased as the security level increases. The reason is that the networking delays for an OCSP response are long and somewhat unstable, and hence the time to verify signatures takes a small portion (the 'PKI w/o OCSP RTT' part in Fig. 4). DANE has an advantage to the PKI with regard to the server authentication since the TLSA record can be cached in the local DNS resolver. That is, DANE requires the client to visit the DNS to retrieve the TLSA record once; the two entities (i.e., the client and the local DNS resolver) are co-located in East Asia, which explains the smaller RTT than that of the PKI in Fig. 4. Right after receiving the

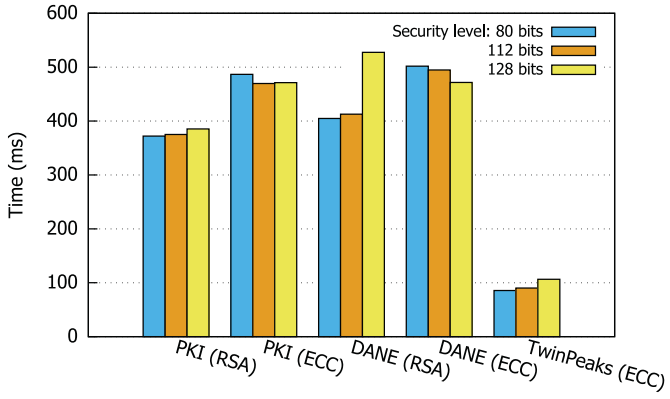

Fig. 5. Average of post-authentication TLS connection setup delays of the three schemes are compared (in ms).

server certificate, the client in DANE can compare the TLSA record with (the hash of) the received certificate.

The key resolution delay of TwinPeaks (ECC) in case of cache hits is much shorter than the authentication delays of the other schemes. Also, it achieves less than 503.5 ms delay even with cache misses since ECC [26] helps reduce the complexity of signature verification.

We perform the key resolution after the DNS processing in TwinPeaks is finished in the experiments. However, note that the key resolution delay can partially overlap with the DNS resolution delay, which will reduce the delay. In Section 10.1, performing two resolutions in parallel is discussed.

9.3.3. Post-authentication TLS connection setup delay

We next measure the other delay of TLS connection setup (except authentication), which is mainly the time to set up a shared key between the client and the server. The TLS variant for TwinPeaks is slightly simplified from the original TLS (See Section 6.3), which results in shorter delay. While the TLS for TwinPeaks can be further simplified (e.g., encrypting *ClientHello* with the server's key), we retain most of the TLS messages in the experiments for fair comparison.

Fig. 5 shows the average delay of the shared key setup. All the settings are the same as for authentication delay. Note that the key setup delay is not necessarily increased as the security level becomes stronger since the networking latencies take a significant portion. Even though TwinPeaks shows the shortest delay compared to other schemes, it does not mean that TwinPeaks is more efficient than others in the shared key setup. It comes from the simplification of key setup in Fig. 2. It is expected that TwinPeaks shows similar tendency with other ECC-based schemes unless TwinPeaks uses the simplified key setup.

9.3.4. Traffic generated during TLS connection setup

Fig. 6 compares the amount of traffic generated during the TLS connection setup (including authentication and the shared key generation). Note that the traffic between the client and the key server is for public key resolution with cache hits. For the PKI and DANE, RSA-based implementations generate more traffic than ECC-based ones. It implies that the smaller key size of ECC has a benefit in terms of the traffic amount. Even with the public key resolution traffic, TwinPeaks shows the smallest traffic for the TLS connection setup among the three schemes.

Overall, TwinPeaks achieves the shortest delay in terms of the key resolution (i.e., authentication) and the minimum traffic. These advantages result mostly from the caching-friendly resolution and ECC-based cryptographic operations. From now on, we investigate (i) caching effects on key resolution and (ii) the effect of two implementations of TwinPeaks.

9.3.5. Caching effects on key resolution

Basically, the local key resolver of TwinPeaks operates like the local DNS resolver when looking at the lookup sequence along the hierarchical tree. However, the caching operations of the two resolvers are not exactly the same, mainly due to TTL (Time-to-Live) settings of DNS. Public keys of TwinPeaks can be cached until the public key or public parameters are updated, which implies potentially long term caching of public keys. Obviously, Jung et al. [43] showed that increasing TTL values leads to increasing hit rates of the DNS cache. Thus, using the cache hit rate of DNS can be viewed as a lower bound of the cache hit rate of public keys in TwinPeaks.

The cache hit rates of DNS are known as very high. In KAIST traces [43], the hit rates are in the range of 70% to 90% in most settings. Also in NLANR traces [44], the hit rates ranges from 70% to 96% approximately. Thus, we calculate the expected delay of key resolution t_{avg} with the hit rate r_{hit} in the range of 70% to 95% using the following equation:

$$t_{avg} = r_{hit} \cdot t_{hit} + (1 - r_{hit}) \cdot t_{miss} \quad (1)$$

where t_{hit} and t_{miss} are the delays with cache hits and misses, respectively.

Fig. 7 shows the expected delay of key lookup to see the cache effectiveness using Eq. (1). The lookup delays of TwinPeaks (ECC) lie between 59.0 ms and 167.2 ms. In comparison to the DNS resolution with DNSSEC in Section 9.3.1, all of the lookup delays of TwinPeaks (ECC) are less than those of the DNSSEC case with cache hits, except when TwinPeaks (ECC) is tested with 70% hit rate and the 128-bit security level. In most of the cases, TwinPeaks (ECC) shows competitive performance in key resolution if caching is effective.

9.3.6. Comparison of two implementations of TwinPeaks

Fig. 8 compares the delays of cryptographic primitives in two implementations of TwinPeaks: PBC and ECC. In the ECC algorithm of CL-PKC, the computation time increases slightly with increasing security levels and all the primitives are finished less than 38 ms. In contrast, the time for the PBC algorithm are measured longer than the ECC algorithm and the measured time grows exponentially. Note that the y-axis of Fig. 8 is given in log-scale.

In order to discuss the cause of soaring, let us explain the parameters of the PBC implementation briefly since those of the ECC implementation are from [41]. As shown in Table 2, the key size for CL-PKC (both PBC and ECC) is two times the number of bits of the security level, which is the subgroup size r [42]. The extension field size q^k should be equal to the RSA key size, and the embedding degree k is 2 with the chosen elliptic curve [45]. Thus, we choose q to be 512, 1024, 2048 bits for the 80, 112, 128 security levels, respectively. It makes the parameter ρ ($= \log q / \log r$) increase, which leads to slow computations of PBC [42].

9.4. Feasibility check for the IoT

In order to confirm the feasibility of TwinPeaks for the IoT environments, the prototype of TwinPeaks for IoT is implemented with C and OpenSSL instead of Java and Bouncy Castle, considering the performance limitations of IoT devices. The prototyping device is Raspberry Pi (RPi) 3, with Raspbian OS. We set two RPis to play the roles of a client and a server, respectively. We call the client as a challenger which tries to authenticate the other device, the server. The ECC key size is set to 256 bits, which indicates 128-bit of the security level as described in Section 9.2.1. We assume that DNS resolution and authentication delays are similar to the measured results in Section 9.3, where we deploy the DNS and key servers as described in Section 9.2.1.

We see that the networking delay takes the most of the time for the authentication delay, but the signature verification may take different amounts of time with the IoT device. It is measured as 14.68 ms on

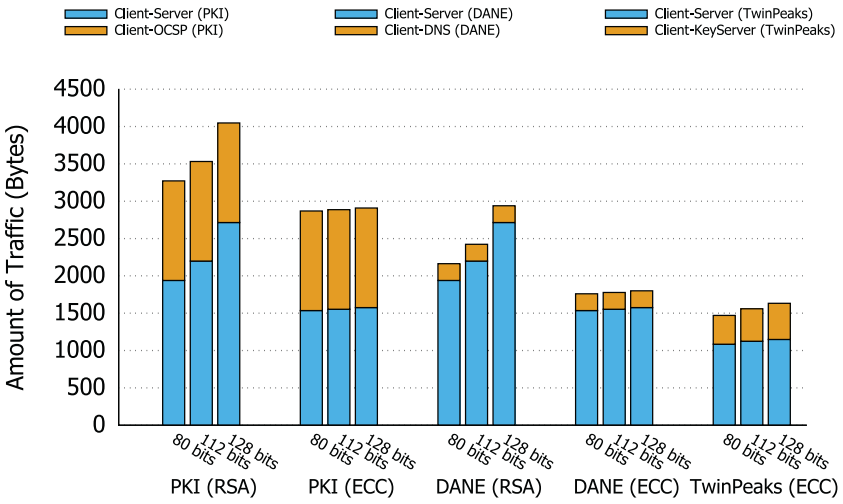

Fig. 6. Amounts of traffic generated by TLS connection setup of the three schemes with cache hits are compared (in Bytes).

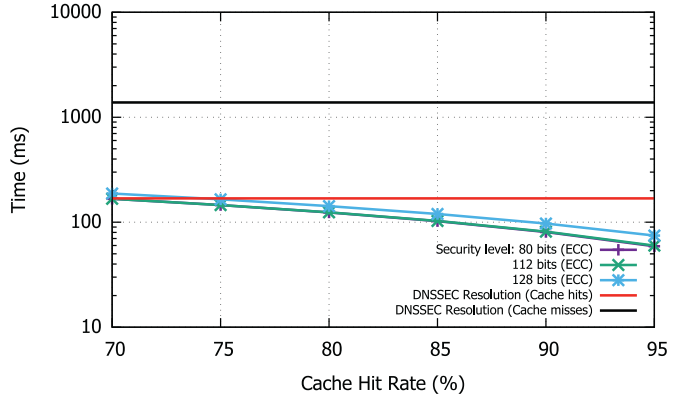

Fig. 7. Expected delays of the key resolution are plotted as the cache hit rate increases (in ms).

average at the tested device, which is comparable to the 'Cache Hit' result of TwinPeaks (ECC) in Fig. 4.

For the post-authentication delay, we assume a simple procedure for the IoT environment. An ECDHE key exchange is performed for establishing a common symmetric key. It takes 284.01 ms at the client and 211.89 ms at the server. Within the key exchange, the time for encryption takes 0.02 ms at the client and the time for decryption takes 0.03 ms at the server. The result shows that the post-authentication delay in the IoT environments takes longer than that of the 'TwinPeaks (ECC)' result in Fig. 5; however, it is comparable to those of the other schemes.

10. Discussions

10.1. DNS and key resolutions in parallel

There are two kinds of resolutions in TwinPeaks: an IP address and a public key, which are performed one by one in the vanilla TwinPeaks. The reason for the sequential resolution of IP addresses (from the DNS) and keys (from the key server hierarchy) is that the response of DNS resolution (i.e., an IP address) is one of input components for key resolution. Thus the entire resolution consists of repetitive request-response exchanges along the DNS and key server hierarchies, and cache hits will reduce the number of exchanges.

While we adopt the sequential resolution in the experimental settings, a significant portion of the key resolution process can be performed in parallel with the DNS resolution. Suppose a client wishes to

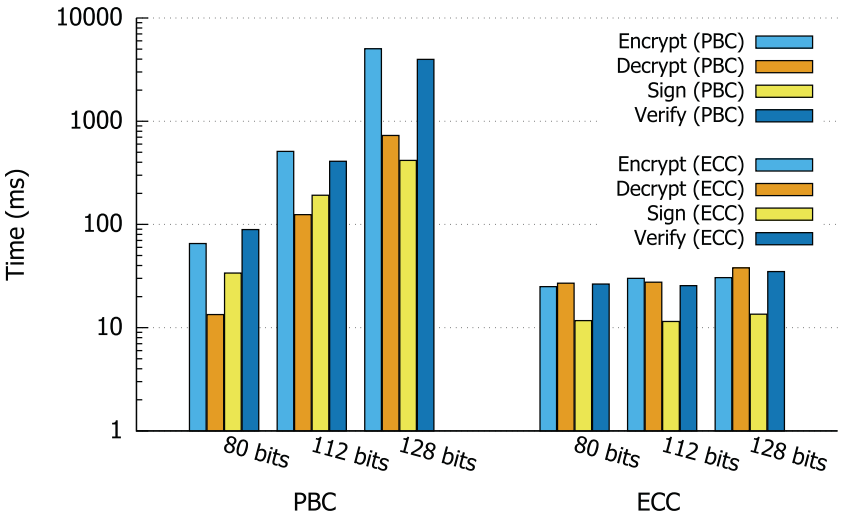

Fig. 8. Delays for cryptographic primitives of two implementations of TwinPeaks are plotted (in ms).

obtain the IP address and the public key of a server mail.example.com. It can request the IP address of mail.example.com to the DNS resolver, and the public key and public parameters of the key server in charge of `example.com` to the key resolver simultaneously. Note that the key resolver will contact the key server of `.com` to obtain these data. Then the client requests the key resolver to obtain the public key of mail.example.com by sending its IP address, the domain name, and the hash of the public parameters just received. The reason for sending the IP address of mail.example.com is that a single domain name may have multiple IP addresses as aforementioned. Finally, the key resolver contacts the key server of `example.com` to retrieve the public key of mail.example.com.

11. Conclusions

We proposed TwinPeaks, a new infrastructure of providing public keys to address the problems of the PKI like CA compromises and certificate revocation. TwinPeaks took a new approach by removing certificates and hence the PKI. Instead, it distributes public keys online by constructing a DNS-like hierarchical structure of public key servers. For the IoT devices, TwinPeaks is applicable by extending the naming scheme. TwinPeaks can thwart spoofing attacks as well as the single point of compromise by making each named entity generate its public/private key pair as a function of its domain name and IP address. We also explained how the compromise of a single node like a public key server or a DNS server in TwinPeaks cannot lead to successful impersonation attacks. We compared TwinPeaks with the current PKI and DANE from both the qualitative and the quantitative perspectives. For the latter comparison, we implemented TwinPeaks, PKI, and DANE on a cloud service-based testbed that spans over North America and East Asia. Comprehensive experiments showed that TwinPeaks can achieve less delays and smaller traffic than the other schemes.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Eunsang Cho: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Visualization, Project administration, Funding acquisition. **Jeongnyeo Kim:** Resources, Funding acquisition. **Minkyung Park:** Software, Validation, Investigation, Data curation, Writing - review & editing, Visualization. **Hyeonmin Lee:** Software, Validation, Investigation. **Chorom Hamm:** Validation. **Soobin Park:** Validation. **Minhyeok Kang:** Validation. **Ted Taekyoung Kwon:** Conceptualization, Methodology, Writing - review & editing, Supervision, Funding acquisition.

Acknowledgement

This work was supported by Institute for Information & communications Technology Promotion (IITP) (grant no. 2018-0-00231) grant funded by the Korea government (MSIT) (No.2018-0-00231, Development of context adaptive security autonomous enforcement technology to prevent spread of IoT infrastructure attacks). Also, this research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) (grant no. 2017R1A6A3A11032626) funded by the Ministry of Education (2017R1A6A3A11032626). The ICT at Seoul National University provides research facilities for this study.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.comnet.2020.107268.

References

[1] P. Eckersley, Iranian hackers obtain fraudulent HTTPS certificates: how close to a web security meltdown did we get?, 2011. URL https://www.eff.org/deeplinks/2011/03/iranian-hackers-obtain-fraudulent-https.
[2] T. Sterling, Second firm warns of concern after dutch hack, 2011, URL http://news.yahoo.com/second-firm-warns-concern-dutch-hack-215940770.html.
[3] N. Falliere, L. Murchu, E. Chien, W32.Stuxnet Dossier, 2011, URL http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.
[4] A. Delignat-Lavaudz, M. Abadiy, A. Birrelly, I. Mironovy, T. Wobbery, Y. Xie, Web PKI: closing the gap between guidelines and practices, NDSS 2014, 2014.
[5] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, L.F. Cranor, Crying wolf: an empirical study of SSL warning effectiveness, in: USENIX security symposium, Montreal, Canada, 2009, pp. 399–416.
[6] S. Farrell, Not reinventing PKI until we have something better, IEEE Internet Comput. 15 (5) (2011) 95–98, doi:10.1109/MIC.2011.120.
[7] H. Tschofenig, J. Arkko, D. Thaler, D.R. McPherson, Architectural considerations in smart object networking, 2015, (RFC 7452). 10.17487/RFC7452URL https://rfc-editor.org/rfc/rfc7452.txt.
[8] B. Laurie, A. Langley, E. Kasper, Certificate Transparency, 2013, (RFC 6962). 10.17487/RFC6962URL: https://rfc-editor.org/rfc/rfc6962.txt.
[9] P. Eckersley, The sovereign keys project, 2012, URL: https://git.eff.org/?p=sovereign-keys.git;a=blob;f=sovereign-key-design.txt;hb=master.
[10] T.H.-J. Kim, L.-S. Huang, A. Perrig, C. Jackson, V. Gligor, Accountable key infrastructure (AKI): A proposal for a public-key validation infrastructure, in: WWW 2013, Republic and Canton of Geneva, Switzerland, 2013, pp. 679–690.
[11] L. Dykcik, L. Chuat, P. Szalachowski, A. Perrig, BlockPKI: an automated, resilient, and transparent public-key infrastructure, in: 2018 IEEE International Conference on Data Mining Workshops (ICDMW), 2018, pp. 105–114, doi:10.1109/ICDMW.2018.00022.
[12] R. Barnes, J. Hoffman-Andrews, D. McCarney, J. Kasten, Automatic certificate management environment (ACME), 2019, (RFC 8555). 10.17487/RFC8555URL: https://rfc-editor.org/rfc/rfc8555.txt.
[13] S. Rose, M. Larson, D. Massey, R. Austein, R. Arends, DNS security introduction and requirements, 2005a, (RFC 4033a). 10.17487/RFC4033URL: https://rfc-editor.org/rfc/rfc4033.txt.
[14] S. Rose, M. Larson, D. Massey, R. Austein, R. Arends, Resource records for the DNS security extensions, 2005b, (RFC 4034b). 10.17487/RFC4034URL: https://rfc-editor.org/rfc/rfc4034.txt.
[15] S. Rose, M. Larson, D. Massey, R. Austein, R. Arends, Protocol modifications for the DNS security extensions, 2005c, (RFC 4035c). 10.17487/RFC4035URL: https://rfc-editor.org/rfc/rfc4035.txt.
[16] R. Barnes, Use cases and requirements for DNS-based authentication of named entities (DANE), 2011, (RFC 6394). 10.17487/RFC6394URL: https://rfc-editor.org/rfc/rfc6394.txt.
[17] P.E. Hoffman, J. Schlyter, The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA, 2012, (RFC 6698). 10.17487/RFC6698URL: https://rfc-editor.org/rfc/rfc6698.txt.
[18] E. Cho, M. Park, T.T. Kwon, TwinPeaks: a new approach for certificateless public key distribution, in: 2016 IEEE Conference on Communications and Network Security (CNS), 2016, pp. 10–18, doi:10.1109/CNS.2016.7860465.
[19] R. Oppliger, Certification authorities under attack: a plea for certificate legitimation, Internet Comput. IEEE 18 (1) (2014) 40–47, doi:10.1109/MIC.2013.5.
[20] L.S. Huang, A. Rice, E. Ellingsen, C. Jackson, Analyzing forged SSL certificates in the wild, in: 2014 IEEE Symposium on Security and Privacy, 2014, pp. 83–97, doi:10.1109/SP.2014.13.
[21] C. Brubaker, S. Jana, B. Ray, S. Khurshid, V. Shmatikov, Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations, in: 2014 IEEE Symposium on Security and Privacy, 2014, pp. 114–129, doi:10.1109/SP.2014.15.
[22] P. Gutmann, PKI: It's not dead, just resting, Computer 35 (8) (2002) 41–49, doi:10.1109/MC.2002.1023787.
[23] S. Al-Riyami, K. Paterson, Certificateless public key cryptography, in: C.-S. Laih (Ed.), Advances in Cryptology - ASIACRYPT 2003, LNCS, vol. 2894, Springer, 2003, pp. 452–473, doi:10.1007/978-3-540-40061-5_29.
[24] J. Baek, R. Safavi-Naini, W. Susilo, Certificateless public key encryption without pairing, in: J. Zhou, J. Lopez, R.H. Deng, F. Bao (Eds.), Information Security, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 134–148.
[25] J. Lai, W. Kou, Self-generated-certificate public key encryption without pairing, in: T. Okamoto, X. Wang (Eds.), Public Key Cryptography – PKC 2007, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 476–489.
[26] X. Yao, X. Han, X. Du, A light-weight certificate-less public key cryptography scheme based on ECC, in: ICCCN 2014, 2014, pp. 1–8, doi:10.1109/ICCCN.2014.6911773.
[27] D. Boneh, M. Franklin, Identity-based encryption from the weil pairing, in: J. Kilian (Ed.), Advances in Cryptology — CRYPTO 2001, LNCS, vol. 2139, Springer, 2001, pp. 213–229, doi:10.1007/3-540-44647-8_13.
[28] A.W. Dent, A survey of certificateless encryption schemes and security models, Int. J. Inf. Secur. 7 (5) (2008) 349–377, doi:10.1007/s10207-008-0055-0.

[29] D.E.E. 3rd, O. Gudmundsson, P.A. Vixie, B. Wellington, Secret key transaction authentication for DNS (TSIG), 2000, (RFC 2845). 10.17487/RFC2845URL: https://rfc-editor.org/rfc/rfc2845.txt.
[30] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, J. Wu, When HTTPS meets CDN: a case of authentication in delegated service, in: IEEE S&P 2014, 2014, pp. 67–82, doi:10.1109/SP.2014.10.
[31] Amazon echo, echo plus, and echo dot, URL: https://www.amazon.com/b/?ie=UTF8&node=9818047011.
[32] Google homeURL: https://store.google.com/product/google_home.
[33] W. Denniss, J. Bradley, OAuth 2.0 for Native Apps, 2017, (RFC 8252). 10.17487/RFC8252URL: https://rfc-editor.org/rfc/rfc8252.txt.
[34] D.S.E. Deering, B. Hinden, IP Version 6 addressing architecture, 2006, (RFC 4291).URL https://rfc-editor.org/rfc/rfc4291.txt10.17487/RFC4291.
[35] B. Haberman, B. Hinden, Unique Local IPv6 Unicast Addresses, 2005, (RFC 4193). 10.17487/RFC4193URL: https://rfc-editor.org/rfc/rfc4193.txt.
[36] I. Ganchev, M. O'Droma, New personal IPv6 address scheme and universal CIM card for UCWW, in: 2007 7th International Conference on ITS Telecommunications, 2007, pp. 1–6, doi:10.1109/ITST.2007.4295898.
[37] NamecoinURL: https://www.namecoin.org/.
[38] J. Clark, P. van Oorschot, SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements, in: Security and Privacy (SP), 2013 IEEE Symposium on, 2013, pp. 511–525, doi:10.1109/SP.2013.41.
[39] G. Wang, T. Ng, The impact of virtualization on network performance of amazon EC2 data center, in: INFOCOM, 2010 Proceedings IEEE, 2010, pp. 1–9, doi:10.1109/INFCOM.2010.5461931.
[40] S.D. Galbraith, K.G. Paterson, N.P. Smart, Pairings for cryptographers, Discrete Appl. Math. 156 (16) (2008) 3113–3121 https://doi.org/10.1016/j.dam.2007.12.010.
[41] FIPS PUB 186-4. Digital signature standard (DSS), National Institute of Standards and Technology (NIST) (2000).
[42] D. Freeman, M. Scott, E. Teske, A taxonomy of pairing-friendly elliptic curves, J. Cryptol. 23 (2) (2010) 224–280, doi:10.1007/s00145-009-9048-z.
[43] J. Jung, E. Sit, H. Balakrishnan, R. Morris, DNS performance and the effectiveness of caching, Netw. IEEE/ACM Trans. 10 (5) (2002) 589–603, doi:10.1109/TNET.2002.803905.
[44] C.E. Wills, H. Shang, The Contribution of DNS Lookup Costs to Web Object Retrieval, Technical Report, Worcester Polytechnic Institute, 2000.
[45] M. Scott, Computing the tate pairing, in: A. Menezes (Ed.), Topics in Cryptology — CT-RSA 2005, Lecture Notes in Computer Science, vol. 3376, Springer Berlin Heidelberg, 2005, pp. 293–304, doi:10.1007/978-3-540-30574-3_20.

Eunsang Cho received his B.S. and Ph.D. in computer science and engineering from Seoul National University. He is currently a postdoctoral researcher at Seoul National University. His research interests include network security, Internet-of-Things, blockchain, content-centric, and peer-to-peer networking.

Jeongnyeo Kim received her MS degree and Ph.D. in Computer Engineering from Chungnam National University, Korea, in 2000 and 2004, respectively. She studied at computer science from the University of California, Irvine, USA in 2005. Since 1988, she has been a principal member of engineering staff at the Electronics and Telecommunications Research Institute (ETRI). Her research interests include mobile security, secure operating system, network security and system security.

Minkyung Park received her B.S. degree in computer science from Korea Aerospace University, Korea, in 2014. She is currently working toward her Ph.D. degree at the School of Computer Science and Engineering, Seoul National University. Her research interests include network security, privacy, anonymity, and blockchain.

Hyeonmin Lee received his B.S. degree in computer science and engineering from Seoul National University. He is currently working toward his Ph.D. degree at Seoul National University. His research interests include network security.

Chorom Hamm received her BS degree in Computer Engineering from Soongsil University, South Korea, in 2006. She also got a master of Science in Information Technology from Carnegie Mellon University, USA, in 2011. She worked for Samsung Electronics and SK as a software engineer in South Korea, and she is currently attending a Ph.D course in School of Computer Science and Engineering at Seoul National University since 2018. Her research interests include mobile computing, security, and context-awareness.

Soobin Park received her BS degree in Computer Engineering from Hanyang University, Korea, in 2013. She has been a software engineer at the Samsung Electronics since 2013, and she is attending a master's course in School of Computer Science and Engineering at Seoul National University since 2017. Her research interests include network security, blockchain, and system security.

Sungmin Sohn received her B.S. degree in computer science and engineering from Ewha Womans University. She is attending a master's course in School of computer science and engineering at Seoul National University since 2017. Her research interests include network security and web security.

Minhyeok Kang received his B.S. in computer science and engineering from Seoul National University, Korea, in 2017. He is currently an M.S. student at Seoul National University. His research interests include network security.

Ted "Taekyoung" Kwon received the BS, MS, and PhD degrees from Seoul National University (SNU) in 1993, 1995, and 2000, respectively. He is a professor with the Department of Computer Science and Engineering, Seoul National University. Before joining SNU, he was a postdoctoral research associate at the University of California Los Angeles and City University New York. During his graduate program, he was a visiting student at the IBM T.J. Watson Research Center and at the University of North Texas. He was a visiting professor at Rutgers University in 2010. His research interest lies in future Internet, network security, and wireless networks.