공학박사 학위논문

A New Certificateless Public Key Distribution and Lightweight Secure Communication

새로운 무인증서 공개키 배포 방법과 경량 보안 연결 방법

2017년 2월

서울대학교 대학원

전기 · 컴퓨터공학부

조 은 상

A New Certificateless Public Key Distribution and Lightweight Secure Communication

새로운 무인증서 공개키 배포 방법과 경량 보안 연결 방법

지도교수 권 태 경

이 논문을 공학박사 학위논문으로 제출함

2016년 10월

서울대학교 대학원

전기 · 컴퓨터공학부

조 은 상

조은상의 공학박사 학위논문을 인준함

2016년 12월

위 원 장	김 종 권
부위원장	권 태 경
위 원	박 근 수
위 원	최 성 현
위 원	박 민 호

Abstract

A New Certificateless Public Key Distribution and Lightweight Secure Communication

Eunsang Cho
Department of Electrical Engineering & Computer Science
The Graduate School
Seoul National University

Authenticating the other endpoint and protecting the data communication are the basic and important ways of secure communication. As the penetration of the Internet to the everyday life is getting accelerated, e.g. Internet of Things (IoT), the demand of secure communications increases. However, the aforementioned two ways have been threatened due to the problems of the Public Key Infrastructure (PKI) and the constrained resources of IoT devices. Therefore, this dissertation focuses on enhancing authentication regarding public key distribution and data protection considering resource-limited IoT devices.

First, the current PKI has problems like certificate revocations and fraudulent certificates. To address such issues, we propose TwinPeaks, which is a new infrastructure to distribute public keys of named entities online. TwinPeaks leverages certificateless public key cryptography (CL-PKC), which we extend to make the public key of an entity depend on any combination of its networking parameters; thus TwinPeaks can mitigate spoofing attacks systematically. TwinPeaks needs public key servers, which constitute a hierarchical tree like Domain Name System (DNS). For each parent-child link in the tree, the parent and the child interact in such a way that every named entity has its own public/secret key pair. TwinPeaks removes certificates and hence has no revocation overhead. Instead, each named entity should keep/update its IP address and public key up-to-date in its DNS server and key server, respectively. TwinPeaks also achieves scalable distribution of public keys since public keys can be cached long term without elevating security risks.

Next, the IoT will be the norm in the foreseeable future. However, the security problem in the Internet will be worsened in IoT services considering the constrained resources of IoT devices. We propose a delegation-based DTLS/TLS framework (D2TLS) for cloud-based IoT services. D2TLS aims to achieve mutual authentication and to lower the burden of setting up secure connections significantly while keeping the private keys of IoT devices secret. Leveraging the session resumption in the DTLS/TLS standard and introducing a security agent, D2TLS achieves these goals with the modifications only within the IoT domain. That is, cloud and PKI systems need no change to deploy D2TLS. Numerical results show that D2TLS can achieve

better performance in terms of delay and energy consumption than making a DTLS/TLS connection in standalone mode.

Keywords: Public Key Infrastructure (PKI), Certificateless Public Key Cryptography (CL-PKC), Transport Layer Security (TLS), Datagram Transport Layer Security (DTLS), Delegation, Session Resumption, Internet of Things (IoT)

Student Number: 2008-20981

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The Internet is now connecting various devices from resource-limited wireless sensors to high-performance servers in the cloud. As the penetration of the Internet to the human life is getting accelerated, the demand of secure communications increases.

In practice, the Public Key Infrastructure (PKI), the Transport Layer Security (TLS) protocol, the Datagram Transport Layer Security (DTLS) protocol are widely used in order to secure communications over the Internet. The PKI is the de facto standard for authenticating named entities in the Internet. The TLS protocol provides privacy and data integrity by data encryption and a keyed message authentication code, and it generally relies on the PKI for the authentication of communicating peers. The DTLS protocol is similar to the TLS protocol, but for datagram-based applications.

Over the years, numbers of high-profile attacks on the PKI have been reported [1, 2, 3]. These attacks aim the core infrastructure of the PKI and neutralize the trust to a certificate authority (CA) as well as a certificate. By issuing a fake certificate which is signed with a genuine CA certificate, the authentication process passes the fake certificate and it does not assure the correct binding between an entity and its public key anymore. This problem is due to the inherent problems of the PKI discussed in Chapter 2. Some approaches have been proposed to address the problems of the PKI [4, 5, 6]. However, the approaches are mainly to introduce more watchers/notaries as another trusted third parties. To this end, a novel infrastructure for public keys, TwinPeaks [7], is proposed to overcome the aforementioned problems.

More recently, the Internet of Things (IoT) broaden the Internet connectivity to virtually every device. The problem is a majority of IoT devices may be constrained in their capabilities. Therefore the security preparation for IoT has less priorities compared to its basic operations. However, an IoT devices and its communication should be secured because the overall security level could be defined as the weakest point. Hence a lightweight DTLS/TLS framework for a resource-limited IoT environment is required to accelerate adoption of secure communications.

1.2 Research Contributions

Contributions of this dissertation are summarized as follows.

- A proposal of a new infrastructure, TwinPeaks, to provide public keys of all the named entities to address the vulnerabilities in the PKI by extending certificateless public key cryptography

- A prototype implementation and a performance evaluation on a cloud-based testbed of TwinPeaks and other schemes including the PKI
- Measurements of IoT products in order to analyze the communication patterns of cloud-based IoT services
- A proposal of a delegation-based DTLS/TLS framework (D2TLS) for cloud-based IoT services by leveraging the session resumption functionality without hand-over of the private key of IoT device
- Prototype implementations for two IoT devices and a performance evaluation of D2TLS in terms of delay, energy consumption, and space requirements

1.3 Organization of Dissertation

The remainder of this dissertation is organized as follows. In Chapter 2, the issues and the vulnerabilities of the PKI are investigated, and then a new public key infrastructure, TwinPeaks, is proposed based on the explored design rationales. In Chapter 3, the communication traffic of two IoT products are measured in order to analyze the communication patterns, and then a delegation-based DTLS/TLS framework (D2TLS) for cloud-based IoT services is proposed. Chapter 4 concludes this dissertation.

Chapter 2

TwinPeaks: A New Approach for Certificateless Public Key Distribution

2.1 Introduction

Authenticating the other endpoint is a basis for making secure communications over the Internet. The Public Key Infrastructure (PKI) is the de facto standard for authenticating named entities in the Internet. For instance, most of the e-commerce services require two endpoints to set up Transport Layer Security (TLS) connections, which rely on the PKI. The central element of the PKI is a certificate that assures the binding between an entity and its public key. This binding is attested by a digital signature of a certificate authority (CA). Thus, the trust to CAs is crucial in making secure communications over the Internet, and hence 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 (e.g., [1, 2, 3]).

The inherent problems of the PKI and the aftermath of its compromise can be summarized as follows. First, any CA can issue a certificate for any entity. Second, the compromise of a single entity (say, CA) may lead to trusting a wrong endpoint (e.g., typing the id/password of a user into a spoofing server). Third, if a CA is compromised, its recovery is a very painful process (e.g., revoking/reissuing its certificates). Fourth, the overhead of the maintenance and distribution of revoked certificates keeps on increasing (e.g., the length of Certificate 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 [8]. Last but not least, usability studies report that users often ignore certificate warnings, which indicates the possible vulnerability to simple certificate interception attacks [9]. 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 [10].

We propose a new infrastructure, *TwinPeaks*, to provide public keys of all the named entities (i.e., domain names) to address the above issues and vulnerabilities in the PKI. TwinPeaks first makes the public key of a named entity depend on its public information, such as its domain name and IP address. That is, if its public information is changed, its public key should

also be changed. In this case, a successful impersonation attack requires to change/compromise both the public key and the public information of a target entity. 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.

2.2 Design Rationale

We take into account the following issues in designing the new infrastructure.

Detection: Fraudulent certificates are not easy to detect and may even go without being detected [11, 12]. Spoofing of any named entity should be instantly and systemically detected. This is why we consider a certificateless public key approach. With the proposed approach, a named entity (e.g., a website) or its counterpart can detect the compromise of its public key (to be detailed later).

Responsibility: 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 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: 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: 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. Thus, we limit the scope of issuing a public key of a named entity systematically by constructing a particular structure.

Scalability: The number of connected devices (say, Internet of things) will be soaring. 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: 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 [13]. The new infrastructure should be designed in such a way that its deployment can take into account relations/incentives among/to interest parties. 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.

2.3 Certificateless Public Key Cryptography (CL-PKC)

In traditional public key cryptographic algorithms (e.g., RSA and ElGamal) that underpin most of the PKI systems, the public/private key pair is generated from some random information, which has nothing to do with the method of verifying the binding between the public key and the entity's identity. Thus it is impossible to figure out the identity from its public key; in other words, any entity can be associated with any public key. 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 [14], 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. In ID-PKC, the private key of a member is generated from a trusted third party (TTP), called a private key generator (PKG). The PKG first sets up public parameters (for cryptographic operations), which should be made public. 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 PKG 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 proposed the CL-PKC scheme [15]. 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 her own secret.

Let us now briefly describe the main algorithms of CL-PKC [15], some of which are to be extended for TwinPeaks in the next section. Discussing the mathematical details of the algorithms goes beyond the scope of this paper, please refer to [15] 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 her key pairs by the seven algorithms below by herself 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 an identifier for member $ID_m \in \{0,1\}^*$ 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.

There are also **Encrypt** and **Decrypt** algorithms to change a plaintext into a ciphertext and vice versa. 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 generic RSA- or ECC-based public keys, we believe this is important since if the destination of the encrypted message does not know the private key, it cannot decrypt the message, and hence fraud identity attacks will be much more difficult.

2.4 How TwinPeaks Works

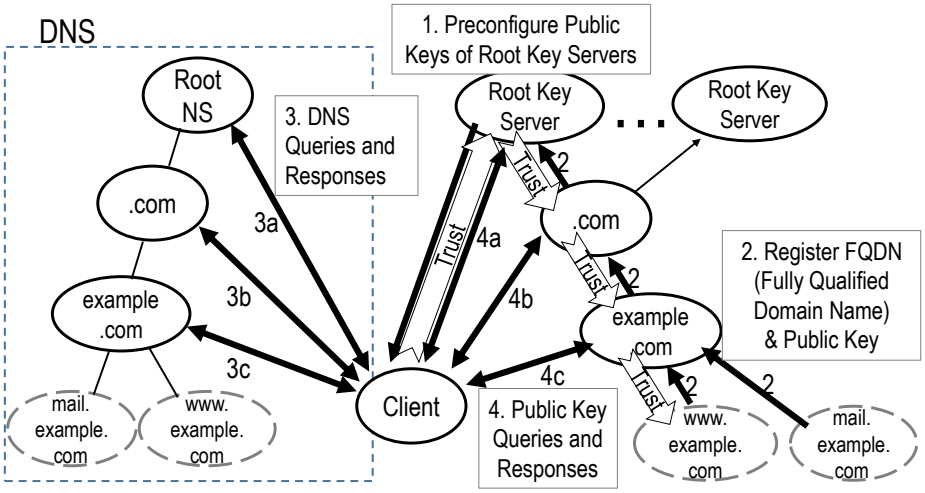
To design a new infrastructure that distributes public keys for the above criteria, we adopt the CL-PKC [15]. We first explain the overall design of TwinPeaks, followed by how the public key of a named entity (e.g., a website with a domain name) is distributed to a relying party (i.e., a client).

We then explain how CL-PKC is extended to substantiate the new infrastructure for providing public keys. Finally, how/when a public key is to be updated/revoked is discussed.

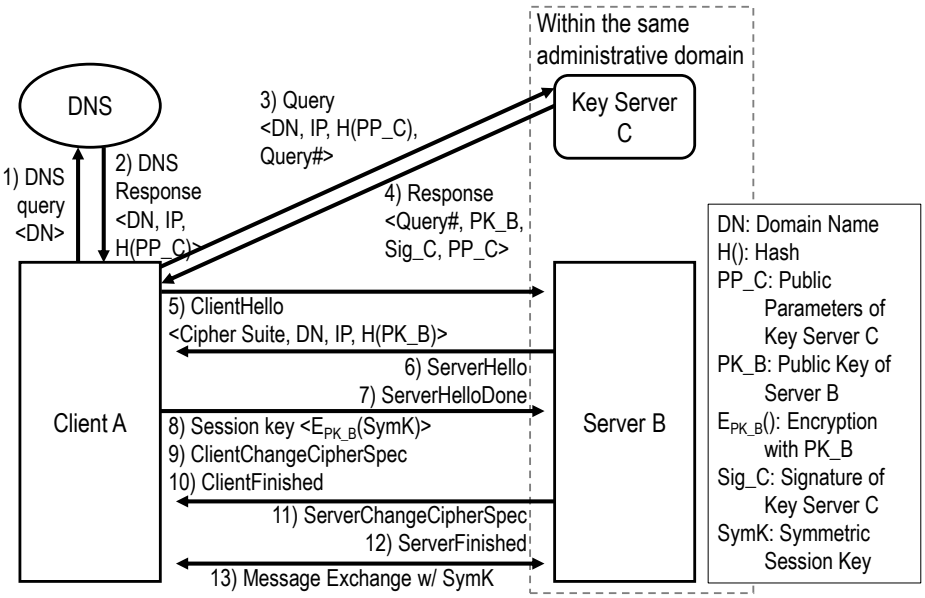
2.4.1 TwinPeaks Overview

TwinPeaks eliminates certificates and hence the PKI. Instead, for each named entity, there is a corresponding key server that provides its public key online. As mentioned above, the key servers collectively construct a DNS-like hierarchy, which is called *a key server hierarchy*. 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.

TwinPeaks consists of two parts: one is the legacy DNS hierarchy and the other is the key server hierarchy (as shown in Fig. 2.1(a)). While Fig. 2.1(a) 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). 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) Overview



(b) TLS variant

Figure 2.1 Overview of TwinPeaks operations and the TLS variant for TwinPeaks

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.

Suppose a relying party wishes to set up a secure connection with `www.example.com`. The client has already pre-configured the public key, the IP address, and public parameters of a root key server (Step 1 in Fig. 2.1(a)). The server `www.example.com` is a member of the domain (`example.com`), and hence the public key of `www.example.com` has been registered with the `example.com` key server (Step 2). Similarly, the public key of `example.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.example.com` from the legacy DNS (Step 3), the client sends 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 until the client obtains the public key of `www.example.com`. Overall, the client

looks up the public key of `www.example.com` from the key servers just like the iterative handling of DNS queries along the DNS hierarchy (Step 4).

Finally, the client sends a message encrypted with the public key of `www.example.com` to its IP address. Then `www.example.com` will decrypt the message by its private key and reply back to the client. In this way, the client can assure the authenticity of `www.example.com`. As a side benefit, TwinPeaks is more resilient to version rollback attacks since a client does not have to send messages in plaintext. This last step moves the burden of public key verification to the client and the server (in contrast to burdens on CAs in the PKI), which we believe is a natural transition considering the interest of the relevant stakeholders.

2.4.2 CL-PKC extension

CL-PKC is designed for operations within a single domain.[1] However, there are many autonomous domains in the Internet. Thus we first need to extend CL-PKC for operations in multiple domain environments. A named entity is assumed to be a member of a particular domain. For instance, `Walmart` is a domain (or `walmart.com` for ease of explanation), while `www.walmart.com` and `news.walmart.com` are the members of the domain. TwinPeaks proposes that: *just as each domain can be assumed to have its own DNS server, so each domain should have its own (public) key server.*

Then how can we make independent domains interwork with TwinPeaks? We adopt the DNS hierarchy partly because two linked nodes (say, parent

[1]Here, we assume an administrative domain, which means a group of entities that are governed by a single authority. Thus, the entities typically share the same operational settings like security parameters.

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 [16].

At the apex of the hierarchy, there is a root key server that corresponds to a root DNS server. There will be multiple root servers in reality, but we assume a single root key server for the sake of exposition. At the next lower level of the hierarchy, there is a TLK server that corresponds to a top level domain (TLD) DNS server. The root key server plays the role of 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 so that the TLK server has its own private/public key pair. The same relation/interaction will be iterated 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 (public) key server of `example.com` maintains the up-to-date public keys of all of its members in the domain. Thus, a key server for `www.example.com` does not have to exist individually.

Another CL-PKC extension is related to the **Set-Secret-Value** algorithm [15]. As explained in [15], ID_m is any string about member m's identity. We first concatenate the domain name and the IP address of the member for the identity. Then we need to add the hash of public parameters of the entity's key server (to counter the forged public parameters). Finally, the entity m's identity for the secret value is given by ID_m = `domain-name`||`IP-address` $||h($`params`$)$, where $h(\cdot)$ is a hash function. Thus, if its domain name or IP address is changed, the member should change its public key. Notice that this

is also useful to thwart DNS spoofing attacks, to be detailed in Section 2.5.1.

2.4.3 Public Key Update

In TwinPeaks, the public key of a named entity may be used long term. However, the public key of an entity should be changed in case of node compromises or IP address changes. Let us discuss the node compromise first. A key server should update its public key in the following cases. (1) If its parent key server is compromised, the parent key server (as a KGC) should set up its master key and public parameters again, and then the key server (as a child) should re-establish the public/private key pair with the parent key server. (2) If the key server itself is compromised, then 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, and updating the public keys of non-compromised (child) key servers is not so urgent since their private keys are not revealed.[2]

Next, if the IP address of a key server is changed, its secret value needs to be changed.[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.

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'* mechanism can be deemed roughly equivalent to 'revocation' in the PKI. Periodically updating public keys would help enhance the overall security in TwinPeaks.

[2]The private key of a node is kept only within the node itself in CL-PKC.
[3]Recall that the IP address is an element of the ID 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.

2.4.4 Public Key Caching

Any public key can be cached around a client in TwinPeaks. 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.

Note that a response contains some additional information as well as the node's public key itself. That is, 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. All of these data can be obtained only by looking at key request/response messages. Thus the key resolver incurs no additional communication overhead for the caching operation.

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. 2.1(b), 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.

2.4.5 Deployment: Islands & TLS Variant

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. For example, the key server for `building1.com` may serve as a root key server. Assume that a client visiting `Building1` pre-loads the public parameters and the public key of the root key server. If the client needs to obtain the public key of `printer1.floor2.bulding1.com`, it can traverse the key server hierarchy from `building1.com` to `floor2.bulding1.com`, which gives the public key of `printer1.floor2.bulding1.com` to the client.

Another merit of TwinPeaks is that the existing security protocol (e.g., TLS) 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. 2.1(b). 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.

2.5 Security Analysis

2.5.1 Threat Analysis

Threat Model

An adversary may launch the following attacks to impersonate a website. First, she may wish to compromise either a key or a DNS server.[4] Second, she may make poisoning attacks on a DNS or a key resolver.

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). 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) [17]. For the latter type of attacks, 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.

[4] If she compromises both servers, then the attack is a success.

That is, we can rely on the existing mechanisms like DNSSEC and TLS to thwart man-in-the-middle attacks and down-grade attacks.

Attacks and Countermeasures

TwinPeaks thwarts both types of attacks as follows. Suppose an adversary wishes to masquerade a famous website. Assume that she compromises the key server that is responsible for the public key of the website, and forges its public key. If the DNS server is not compromised, the client's message encrypted by the forged key will be delivered to the authentic website. However, the website cannot decrypt the message correctly, and hence the client figures out that the public key is not valid.

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 but the valid public key of the website from the key server hierarchy. The message encrypted by the valid public key will go to a wrong destination, possibly a spoofing host. However, the spoofing host cannot decrypt the message correctly since it does not have the private key. 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. Thus, an impersonation attack in TwinPeaks is possible only if both the DNS server and the key server are compromised.

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 one of 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 should assume that the adversary obtains the secret key of the root key server. 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, a single compromised root key server may be easily detected.

2.5.2 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, 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.

The identifier depends on its domain name and IP address, and thus the correctness of the responses from the DNS and the hierarchy of public key servers are the bases for the validation. If one of the elements described above is incorrect, the ciphertext will not be decrypted. 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. Modified TLS handshake messages can help explain the cause(s) of invalidation (see Section 2.4.5).

2.6 Evaluation

2.6.1 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 [18, 19]. 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. Qualitative comparison of TwinPeaks with the PKI and DANE is summarized in

Table 2.1 Comparison of TwinPeaks, PKI, and DANE schemes.

	TwinPeaks	PKI	DANE
Single root dependency	Multiple Roots	Multiple Roots (CAs)	Single root
Scoping of key issuance	Strict	Limited	Strict
Revocation overhead	No overhead	Heavy overhead	Depends on portion of self-signed certs
Infrastructure overhead	DNS and key servers	DNS and CRL/OCSP	DNS and CRL/OCSP
Key retrieval overhead	Caching	No caching	Caching self-signed certs only
Recovery of spoofed certificate/key	Easy	Difficult	Easy
Condition of impersonation attacks	Compromise of both DNS and key servers	Compromise of CA or RA	Compromise of DNS server

Table 2.1.

2.6.2 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.

Evaluation Environments

We use two cloud service providers located in North America and East Asia to mirror a globally distributed network infrastructure. By setting up

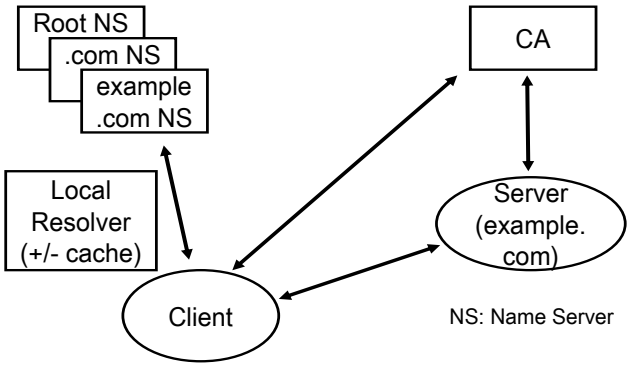
geographically-distant virtual machines, the latencies among the client, website (or server), DNS servers, key servers, and CAs are reflected in the evaluation.

The DNS servers and key servers are hierarchically connected, respectively. Three levels of DNS servers are deployed to emulate the current DNS structure (and their geographical positions). A client is located in East Asia, and we assume the root name server is also located in East Asia due to 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 in the US. 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.

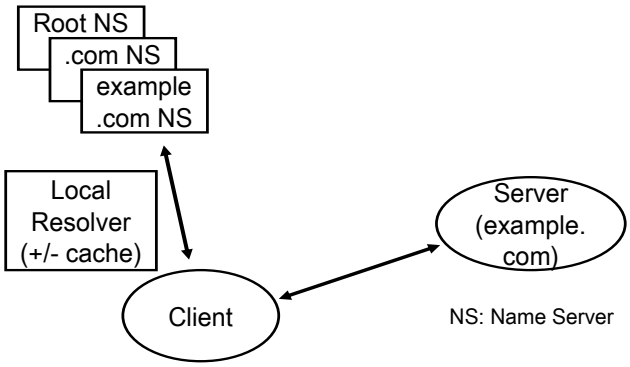
We use `BIND` for the authoritative DNS server without recursion, allowing DNSSEC and DANE features. `BIND` is also used for a local DNS resolver, which is modified to indicate the root server and to allow recursive DNS queries. OpenSSL is used for the CA and its OCSP responder, both of which are located in East Asia. Other entities (e.g., key servers) are prototyped with Bouncy Castle.

Evaluation Methods

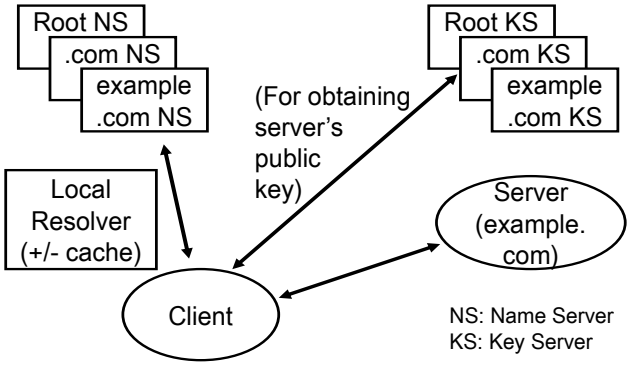
To compare TwinPeaks with the PKI and DANE, we measure the resolution delay, which is the interval from the starting time of the DNS query of a client to the starting time of a TLS session setup. The node configurations are shown in Fig. 2.2. The local key resolver is co-located with the client in



(a) PKI



(b) DANE



(c) TwinPeaks

Figure 2.2 Experimental settings for PKI, DANE, and TwinPeaks

Table 2.2 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	RSA Key Size	CL-PKC (EC) Key Size ($\log r$)
80	1024	160
112	2048	224
128	4096 [5]	256

Fig. 2.2 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. For 80-bit, 112-bit, and 128-bit of security levels, we use 1024-bit, 2048-bit, and 4096-bit of RSA keys. Also, 160-bit, 224-bit, and 256-bit of ECC and CL-PKC keys are used for each of the security levels. The matching key lengths between RSA, ECC and CL-PKC are determined based on the literature [20] as shown in Table 2.2. We average the experimental results over 10 runs for each setting.

There are two approaches in implementing CL-PKC: pairing-based cryptography (PBC) [15] and ECC [21]. The former one is inefficient due to the

[5] As shown in [20], 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.

high complexity of bilinear pairing. Thus we adopt the latter in evaluation.

2.6.3 Numerical Results

We measure the DNS resolution delay, authentication delay, other TLS connection setup delay, and traffic for TLS connection setup. Prior to the connection setup, the client will request its local DNS resolver to obtain the IP address of the domain name (i.e., the server). In 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 2.4.5).

DNS Resolution Delay

All require DNS resolution before establishing a TLS connection. Hence 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 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.

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.

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 2.4.5). 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.

Fig. 2.3 shows the measured authentication delays. 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 is significantly reduced with cache hits in the key resolver. As to 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.

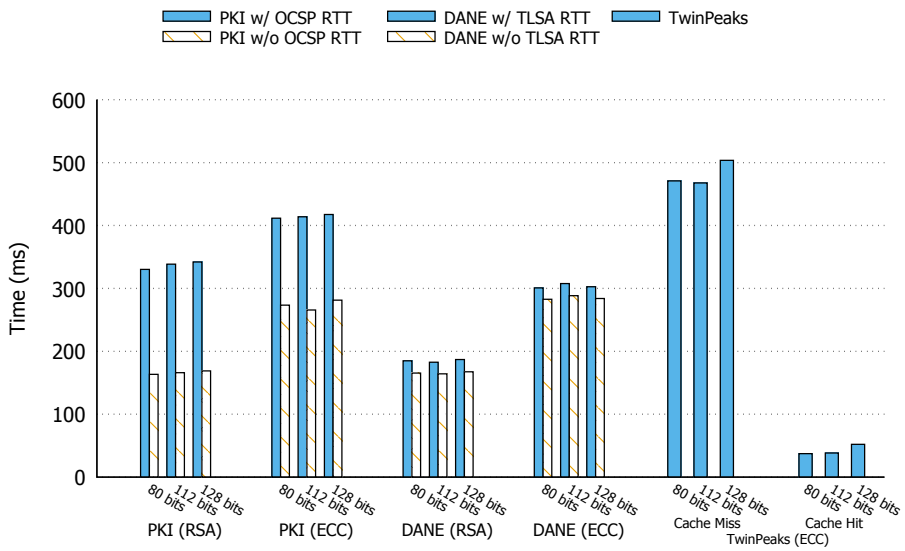


Figure 2.3 Average authentication delays of the PKI, DANE, and TwinPeaks are analyzed (in ms) during/before the TLS process.

Before analyzing the authentication delay of the other schemes, let us detail the settings for Fig. 2.3. 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. (1) The client visits the OCSP server only once; this case corresponds to the 'w/ OCSP RTT' part in Fig. 2.3. (2) 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. 2.3. Similarly, there are two cases as to DANE. (1) The client visits the DNS server once again to retrieve the TLSA record; this case corresponds to the 'w/ TLSA RTT' part in Fig. 2.3. (2) 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. 2.3.

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

For the results of DANE, 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 with cache hits shows better performance compared with DANE in the 2nd case.

Note that the authentication delay of the PKI is not necessarily increased as the security level increases. The reason is that the networking delays among two data centers (in East Asia and the US) 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. 2.3). 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. 2.3. Right after receiving the server certificate, the client in DANE can compare the TLSA record with (the hash of) the received certificate.

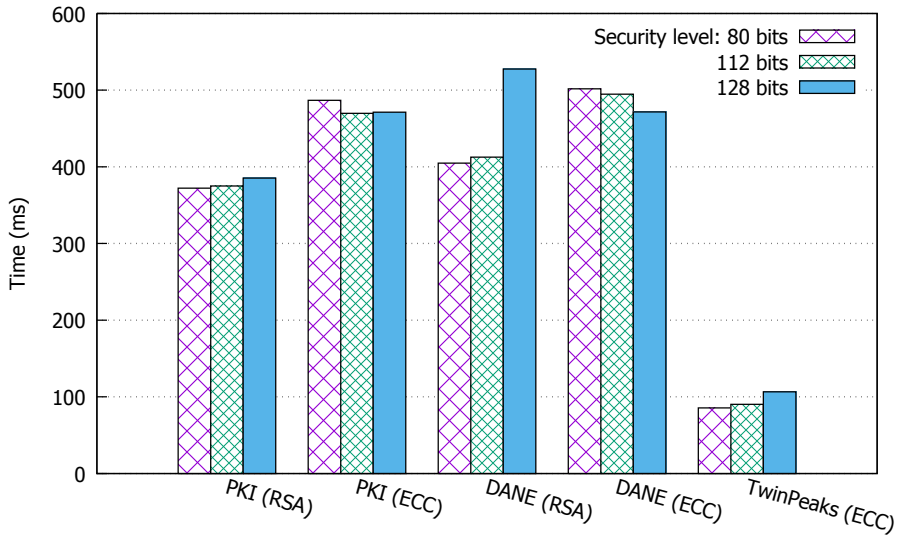


Figure 2.4 Average of TLS connection setup delays (except authentication) of the three schemes are compared (in ms).

The key resolution delay of TwinPeaks 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 [21] 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.

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 2.4.5), which results in shorter delay.

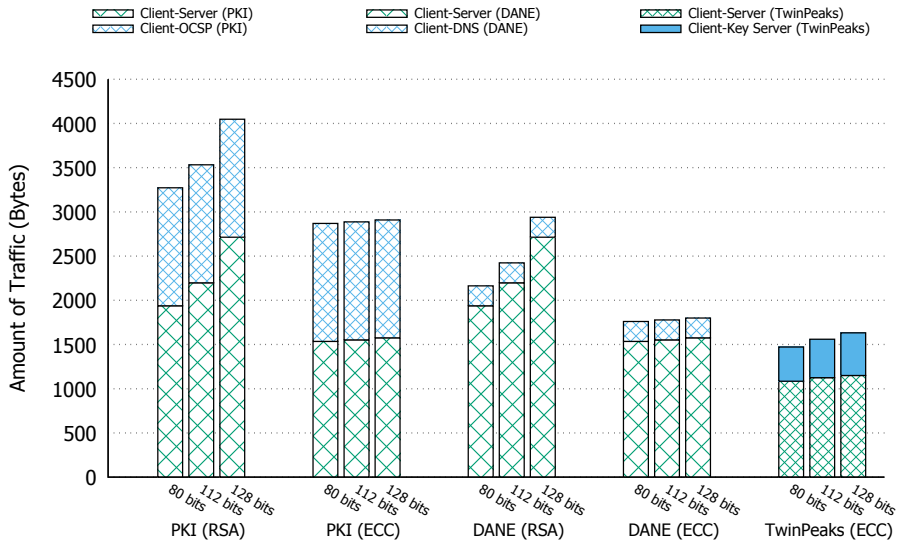


Figure 2.5 Amounts of traffic generated by TLS connection setup of the three schemes are compared (in Bytes).

Fig. 2.4 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. 2.1(b). It is expected that TwinPeaks shows similar tendency with other ECC-based schemes unless TwinPeaks uses the simplified key setup.

Traffic Generated during TLS Connection Setup

Fig. 2.5 compares the amount of traffic generated during the TLS connection setup. 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.

2.7 Discussions

Scalable Distribution of Public Keys. TwinPeaks can achieve scalable distribution of public keys of named entities since public keys can be cached like DNS resource records. The valid time of a cached public key can be long since a relying party will verify both the public key every time it sets up connectivity with the corresponding server. This long term caching will significantly reduce the burden of key distribution, considering that the number of secure communications as well as the number of IP devices is likely to increase.

Multiple IP Addresses. A popular domain name like `google.com` has to use multiple IP addresses. Basically, in TwinPeaks, a new public key is to be generated for each IP address even if it is mapped to the same domain. If a domain name is resolved to multiple IP addresses, the client chooses a single IP address for the communication. The chosen IP address is included in *the key request* message to the key server of the domain name. Then the

key server will reply with *the key response* message including the public key for the chosen IP address. For the popular domains, the corresponding key server has the burden of managing the public keys for all the IP addresses of the same domain name.

HTTP Redirection. To deliver web content efficiently, content delivery networks (CDNs) rely on application level rerouting or DNS level rerouting to forward the request to a close replication point. As for application level rerouting, it can be done independently of TwinPeaks. For instance, a URL address in the HTTP GET request can be redirected to another URL. In that case, the client will perform resolution for the new URL. However, since TwinPeaks is coupled with the DNS resolution, DNS-based request routing requires some extension to TwinPeaks as follows. In many cases, DNS-based request routing utilizes a CNAME resource record in the DNS for redirection [22]. Suppose that `www.example.com` to redirected to CNAME `example.cdn1.com` at the DNS level. A response from the key server (of `example.com`) indicates the corresponding CDN provider's key server[6]. Thus, the client compares the CNAME resource record of `www.example.com` (which is actually `example.cdn1.com`) and the delegation information from the key server, and then verifies whether the two responses match.

Cloud-based Services. Cloud-based Internet services provide the elasticity of computing resources. However, migrating resources (e.g., virtual machine images) may result in frequent IP address changes. However, most of IP address changes occur within data center networks and the public IP addresses are typically used for front-end servers, to which clients are con-

[6]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.

nected for secure communications. Thus, TwinPeaks can support both CDN and cloud-based services while maintaining the simple binding between public keys and domain names (and their IP addresses).

DNS and Key Resolutions in Parallel. There are two kinds of resolutions in TwinPeaks: an IP address and a key, which are performed one by one in the vanilla TwinPeaks. The reason for 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 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 mentioned in the above. Finally, the key resolver contacts the key server of `example.com` to retrieve the public key of `mail.example.com`.

Public Keys for Clients. TwinPeaks targets the server public key (and its certificate) based on a domain name and an IP address within the ID_m. Furthermore, TwinPeaks could be extended for a client public key by using a unique ID string and its IP address. For instance, an ID_m of a user client could be constructed from a set of an e-mail address and a home IPv6 address of Mobile IPv6, which both indicates the identity of the user client.

When a server receives the client information, i.e. its e-mail address, the server tries to resolve the corresponding key server. As we use an e-mail address in the form of `username@example.com`, the ID_m shows the corresponding key server of the user client key as the domain name part of the e-mail address, `example.com`. For the IP address acquisition of the user client, a new type of DNS resource record (RR) is required. The new DNS RR indicates the directory server which converts an e-mail address into its corresponding home IPv6 address. Therefore, by asking the DNS and the directory server, the server could resolve the corresponding address. The server contacts the key server and in turn asks for the public key with the e-mail address and the home IPv6 address. The verification could be done with the ID_m and the corresponding public key. Note that the home IPv6 address is also the source address of the client packets, because reverse tunneling is the default behavior of Mobile IPv6.

2.8 Related Work

A growing body of research tries to address the PKI problems, where the focus is usually on the CA and certificate issues. Certificate Transparency [4] is a log-based certificate validation approach. A certificate log is maintained

by using a Merkle hash tree, which operates in append-only mode to validate certificates through the history of certificates. Sovereign Keys [5] 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 [6] 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. However, these schemes suggest yet another trusted third parties to keep monitoring and archiving the certificates typically by using append-only storages. Introducing more watchers/notaries could relieve the risk of PKI failures but these approaches would make the PKI landscape more complex.

Aside from the PKI, there are DNS-centered approaches such as DNSSEC and DANE [19]. Both of these IETF working groups seek to secure 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, she 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, she can achieve the CA independency, but a certificate forgery may go undetected.

Chapter 3

D2TLS: Delegation-based DTLS for Cloud-based IoT Services

3.1 Introduction

The Internet of Things (IoT) is poised to provide connectivity to virtually every device, so that the devices can be monitored and/or controlled usually in an automatic fashion. While the IoT is expected to help realize convenient and smart lives by tapping devices that have not been online, the security issues in the current Internet may be worsened due to the limited capabilities of the devices. The careful design of a trustworthy networking framework will be essential for secure IoT services.

The connectivity and networking settings for the IoT services will be diverse. Some "high-end" IoT devices directly provide sensory data in response to remote requests/queries like servers in conventional client-server mode. However, a vast majority of IoT devices may be constrained in their

capabilities; they may periodically go into sleep mode (e.g., to save energy), the transmission rate is too slow (e.g., ZigBee and LoRa), or the computing power is not enough to handle multiple requests from clients. In such cases, perhaps their sensory data will be stored to reverse proxy nodes, which will reply to the requests on behalf of the devices.[1] However, proxy-based IoT services have the following problems: (i) an IoT device may have to share its security information (e.g., its private key) with the reverse proxy if it allows the proxy to set up a security association with remote clients [23], and (ii) the sensory data is also shared between the device and the reverse proxy, which indicates another vulnerability by violating the end-to-end principle.

Recently, the cloud computing proliferates and hence cloud-based IoT services have become widespread. Suppose we have a smart home monitoring system that consists of door locks, fire detectors, temperature sensors, motion sensors, and so on. Making these individual sensors operate like servers is likely to be expensive, energy-inefficient and/or slow solutions. It is much more practical for such sensors to send data to cloud systems depending on their capability and availability (say, transmit their sensory data when they are awake), and the cloud will service the queries from clients. To summarize, the cloud plays the role of a server on behalf of the resource-constrained sensors. Note that the proxy mode IoT services break the end-to-end communications model, while the cloud-based IoT services retain it.

Even though cloud-based IoT networking and services are becoming the norm as envisioned in [24], we believe a security framework for such settings is not well provisioned yet. Recently, Symantec published a report on the

[1] We assume a reverse proxy keeps the data of the corresponding devices, and is located within the same administrative domain.

insecurity of 50 different smart home devices [25]. While 68 percent of the tested devices rely on cloud services, their control and data messages may be exchanged over untrusted networks. In particular, around 19 percent of the tested devices communicate without encryption, e.g., Transport Layer Security (TLS), and none of the devices provides mutual authentication. The report argues that strong encryption and mutual authentication be required even for resource-limited IoT devices, and it recommends efficient cryptographic methods such as elliptic curve cryptography (ECC).

Standards developing organizations (SDOs) seek to develop new networking platforms or frameworks for IoT services; however, as to security, they usually adopt the current Internet security framework like TLS and Datagram Transport Layer Security (DTLS). For instance, the DICE working group in the IETF proposes to use TLS/DTLS profiles [26] for IoT deployments with CoAP [27], which is a lightweight version of HTTP. The ETSI OneM2M consortium also suggests to use the TLS or DTLS [28]. Using the DTLS implies that the current authentication mechanisms like the Public Key Infrastructure (PKI) may have to be used in IoT environments. However, due to typical IoT constraints like the energy budget, hardware capability, and/or low speed communications, IoT devices may not be able to handle the PKI and certificates timely. While the DTLS is lighter than TLS, it may still incur substantial overhead on IoT devices, which will be investigated in this paper.

In the same vein as the above Symantec report [25], we claim that mutual authentication be crucial for secure IoT services. Considering the limited capabilities of IoT devices, we propose a delegation-based DTLS/TLS framework (D2TLS) for cloud-based IoT services. The central idea of D2TLS

is to leverage the session resumption functionality in the certificate-based DTLS/TLS standards [29, 30]. D2TLS also introduces a security agent to relieve a device of the setup burden of a DTLS/TLS connection; that is, it sets up a secure connection between a device and a cloud system on behalf of the device. D2TLS allows only the device to keep its private key (unlike [23, 31]), while performing mutual authentication of two endpoints.

3.2 Related Work

The overhead of the full TLS handshake in DTLS/TLS imposes a serious workload on the two endpoint in general. To help mitigate the overhead, TLS provides the session resumption mechanism by which the same negotiated secret can be resumed if the two endpoints already have set up a DTLS/TLS connection. Thus, it is conceivable for a resource-constrained IoT device to delegate the first full handshake to some other entity, while the device performs only the task of taking over the security context for the DTLS/TLS session by leveraging the session resumption.

[23] introduced a delegation server for IoT devices; the delegation server is responsible for the first full handshake on behalf of the devices. Right after the first handshake is done, the delegation server transfers the session ticket to the corresponding IoT device, which then resumes the DTLS session. Their scheme is proposed for highly resource-constrained devices. Thus, the whole handshake process is done by the delegation server, which can be called "full delegation." During the delegation process, all the secret data are transferred via pre-configured out-of-band secret exchanges. The weakest point of this scheme comes from the full delegation. Trusting the delegation server by

sharing the private keys of the IoT devices means granting full permissions for the security-related activities of the devices to the delegation server.

[31] targets medical sensor networks, where the delegation process is performed at a gateway of the sensor network. In this scheme, the medical sensors work as servers, and hence remote clients connect to the sensors directly. The sensors assumed in their work are also highly resource-constrained, and thus they delegate the first DTLS handshake to the gateway like [23]. However, [31] is different from [23] in the following points. First, the gateway is within the same administrative domain as the origin sensors; thus, it is easier to secure the communications between them compared to [23], where the delegation server in [23] can be located outside the domain. Next, only the essential secret data is transferred to the sensor since it may not have the capability of keeping all the session-related information. However, [31] has the same problem as [23] since the private key of a device is shared with the gateway.

In cloud systems, there is a key escrow problem, which means that the private key of an origin server is shared with a replicated server running in the cloud since the cloud server should make TLS connections with the clients. CloudFlare, a cloud service provider, devises a scheme in which the cloud server does not need to know the private key of the origin server, so-called Keyless SSL [32]. CloudFlare introduces a dedicated key server, which holds the private key of the origin server. The dedicated key server is controlled by the origin server. By introducing the key server, Keyless SSL achieves the authentication of the original server during TLS sessions even if the cloud server sets up TLS connections with the clients.

3.3 Measurement of IoT Products

In order to analyze the communication patterns of cloud-based IoT services, we measured two IoT products: (i) a smart home monitoring system, and (2) a smart watch. The analyses of the measurements will help design the D2TLS framework.

3.3.1 Smart Home Monitoring System

A smart home monitoring system can be used for multiple purposes. For example, it works as a surveillance system against unexpected intrusions. The system can check whether the doors and windows are open or closed, and whether there are any moving objects in our home in the family's absence. Another example is a baby monitor. A video stream is monitored and analyzed all day long and the parents can know whether their baby is staying OK or is requesting for their help. The temperature and humidity of the baby's room is monitored and controlled.

We measure the whole traffic between sensor devices and a gateway, which constitutes a Xiaomi's smart home system. The communication interface of the system is IEEE 802.15.4. We use a TI CC2531 packet sniffer to capture the traffic in the IEEE 802.15.4 network. The Xiaomi system consists of a door (open/closed) detector, a temperature/humidity monitor, a motion detector, a push button, and a smart gateway. All the sensor devices are connected to the gateway via IEEE 802.15.4, while the gateway has connectivity to the cloud system via IEEE 802.11 as well as IEEE 802.15.4 for the home network. A user can access the sensory data in the cloud system using a smartphone app. Note that there is no encrypted traffic; all the packets carry the plaintext

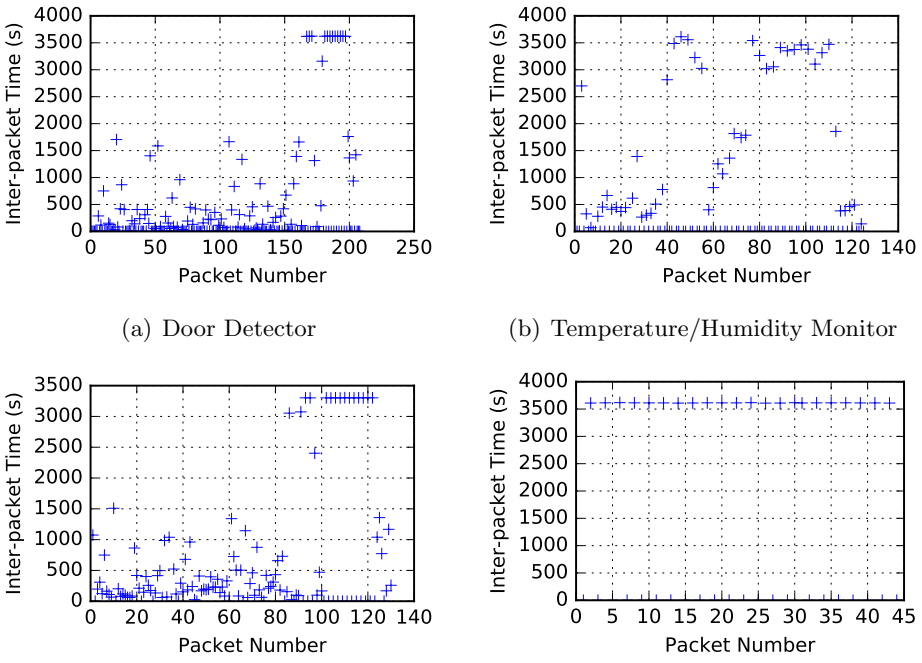
Table 3.1 Number of packets to be analyzed for each sensor device of the Xiaomi home network is shown; the packets are captured over 24 hours.

	Door Detector	Temperature/Humidity Monitor	Motion Detector	Push Button
Number of Packets	209	126	131	45

payload.

In our measurement analyses, we focus on the distribution of the lengths and the frequency of communication sessions between the sensor devices and the gateway (towards the cloud) for the following reason. Compared to consecutively transmitted UDP datagrams (which helps to keep a single session) or unencrypted TCP sessions, DTLS/TLS sessions require more resources of IoT devices for creating and maintaining sessions. In particular, a session creation consumes more resources than in-session communications due to the heavy computations of public-key operations in the DTLS/TLS handshake process. Thus, we believe the overhead of DTLS/TLS sessions is dependent on their lengths and frequency.

Figure 3.1 shows the inter-packet times of the traffic for each device in seconds. Since the TLS/DTLS protocol protects packets in application layer, we filtered out IEEE 802.15.4 beacon and ACK messages from 6,622 captured packets, and then we used 511 packets for analysis. Table 3.1 shows the number of packets analyzed for each device. Figures 3.1(a) and 3.1(c) show that most of data points are distributed close to 0 seconds, which indicates that a vast majority of packets are generated consecutively. These successive packets can be considered as traffic belonging to a single session when we apply the concept of TLS/DTLS sessions. Therefore, to identify separate



(a) Door Detector

(b) Temperature/Humidity Monitor

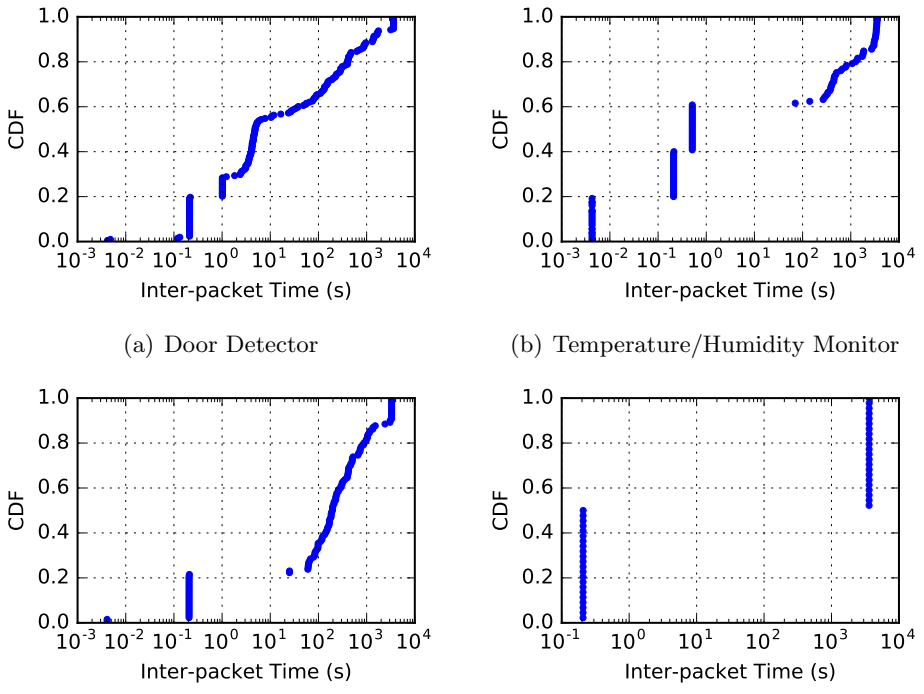
(c) Motion Detector

(d) Push Button

Figure 3.1 Inter-packet times of the 4 devices in the Xiaomi home network are plotted over the 24 hours as time goes on.

sessions, we should note the data points of long inter-packet times. Figure 3.2 shows the distributions of inter-packet times of the four devices in CDF. At least, approximately 30% of inter-packet times are longer than 100 s for every graph, which means that there are many separate sessions conceptually within the 24 hours of measurements.

Let us now estimate the lengths and frequency of separate sessions from the above measurements of the inter-packet times. Figure 3.3 shows the estimated frequency of sessions as we vary the threshold (of inter-packet time) to separate different sessions. As there is no fixed/standard value for the



(a) Door Detector

(b) Temperature/Humidity Monitor

(c) Motion Detector

(d) Push Button

Figure 3.2 The CDFs of inter-packet times of the 4 devices in the Xiaomi home network are plotted.

threshold of session separation, we use the widely used session timeout values of HTTP and application servers in general Internet services. The timeout values are set to 60s, 600s, 1800s, and 3600s, which are adopted from the minimum and default values of Microsoft IIS servers, the default values of Apache HTTP servers and Apache Tomcat servers, respectively. If we set the threshold to 60s, there are 101 sessions in the motion detector. Obviously, the numbers of estimated sessions decrease sharply as the threshold increases across the four sensors, except for the push button. Considering the constrained resources in IoT environments, the session timeout values of IoT

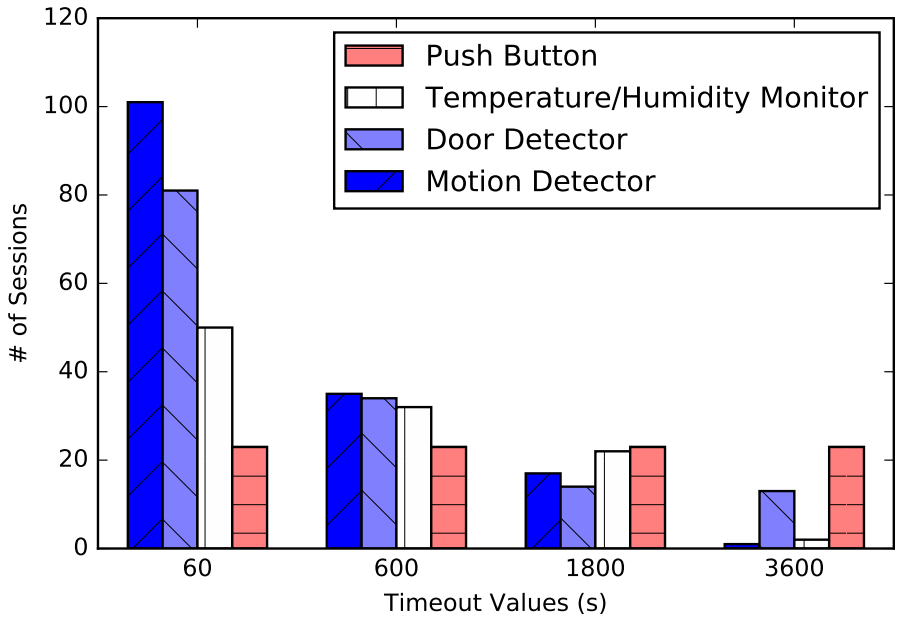
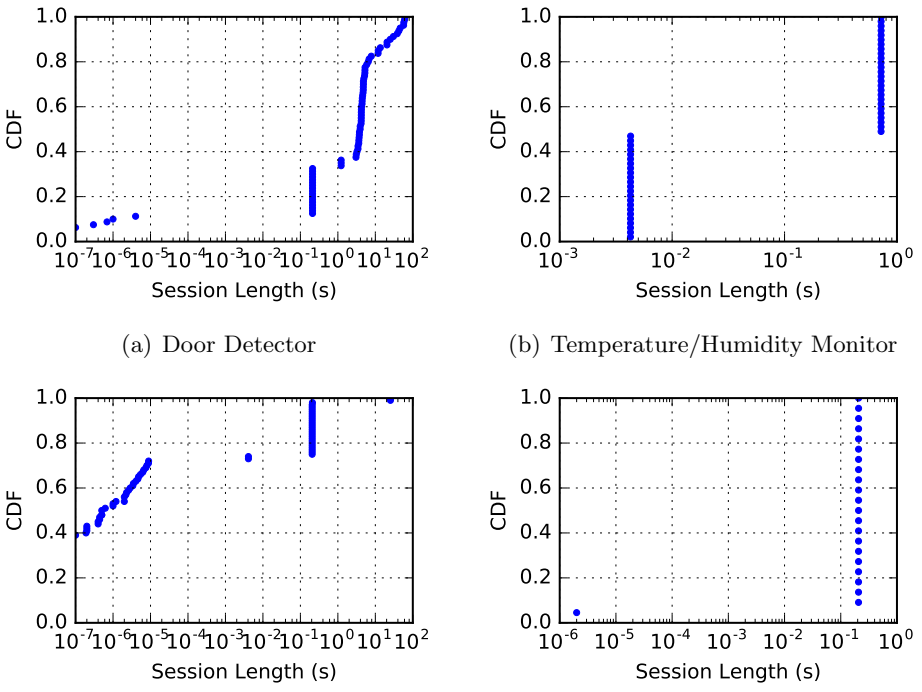


Figure 3.3 Frequency of the sessions of the Xiaomi home network for 24 hours is analyzed.

devices may as well be shorter than those of HTTP servers in the general Internet environments. Therefore, we believe that the durations of sessions in IoT environments are likely to be close to that of the minimum timeout value (60s) in the graph.

Figure 3.4 shows the session length distribution when the session timeout value is set to 60s. The means of session lengths are 8.40s, 0.39s, 0.56s, and 0.20s in the four sensors, respectively. However, most of the session lengths are clustered at a few distinct values smaller than 1s. It reveals that a majority of the sessions are short, and hence the overhead of setting up a DTLS/TLS connection for each session can incur a heavy workload.



(a) Door Detector

(b) Temperature/Humidity Monitor

(c) Motion Detector

(d) Push Button

Figure 3.4 The CDFs of session lengths of the 4 sensors in the Xiaomi home network for 24 hours are plotted.

3.3.2 Smart Watch

A smart watch is a wearable device that is usually connected to a smartphone, which in turn is connected to cloud services via its WiFi or 3G/LTE interface. For example, smart watches using the Android Wear platform require the Internet connection through smartphones to Google services.

We measure the traffic between a smart watch and a smartphone using Motorola's Moto360 smart watch, which uses Bluetooth. We rely on the Bluetooth HCI snoop feature in the Android OS, so that we can capture all the traffic between the smart watch and the smartphone. The measurement

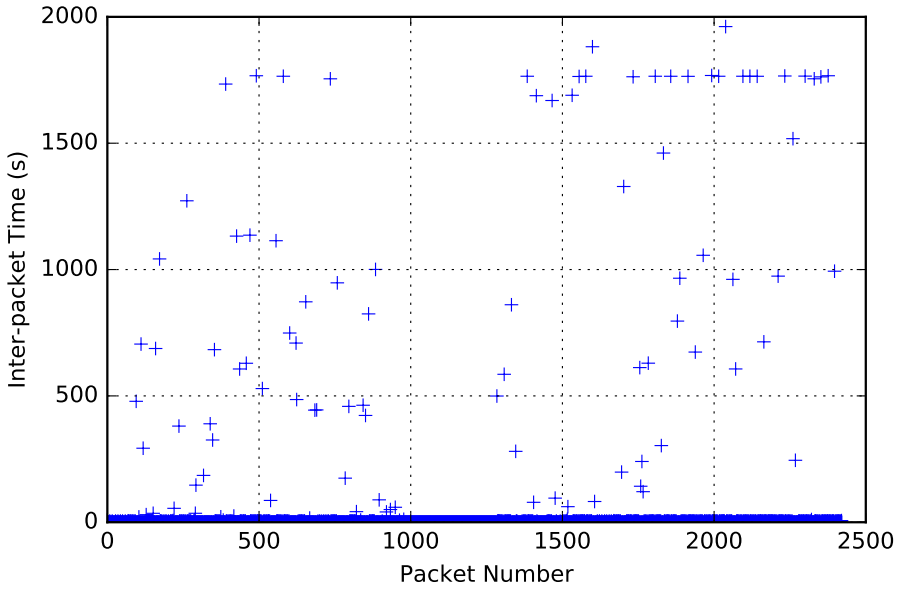


Figure 3.5 Inter-packet times of a smart watch for 24 hours are plotted in unit of seconds as time goes by.

is again focused on the lengths and frequency of communication sessions as described in Section 3.3.1.

Figure 3.5 shows the inter-packet times in seconds during the measurement for 24 hours. The total number of captured packets is 15,202. Since TLS/DTLS encrypts packets at application layer, we filter out messages not containing application data. That is, we exclude signaling messages in Bluetooth, which leaves us 2,422 packets for analysis. As shown in Figure 3.5, most of data points are concentrated close to 0s, and the other data points are somewhat uniformly scattered between 0s and 1800s, compared to Figure 3.1. Figure 3.6 shows a similar tendency with Figure 3.2, but the concentration level of successive packets is stronger since almost 80% of inter-packet times are shorter than 1s. Note that there are a number of applications running in-

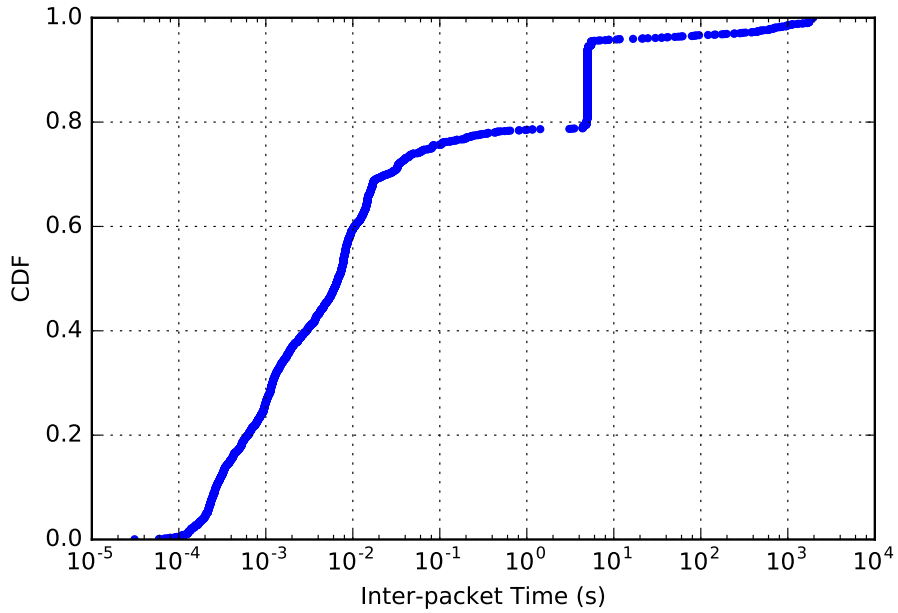


Figure 3.6 Inter-packet times of a smart watch for 24 hours are shown in CDF.

side the smart watch, which was factory-resetted and synchronized before the measurement. For experimental purposes, only a Google account is activated and there is no app installed other than pre-installed ones.

Let us analyze the lengths and frequency of distinct communication sessions. Figure 3.7 shows the estimated session frequency by varying the threshold of inter-packet time to identify separate sessions. The same timeout values in Section 3.3.1 are set for session separation. Maximum 88 sessions are identified with the 60s threshold.

Figure 3.8 shows the session length distribution when the session timeout value is set to 60s. The mean of session lengths is 29.25s, but about half of the session lengths are concentrated at a single value: approximately 35s. It

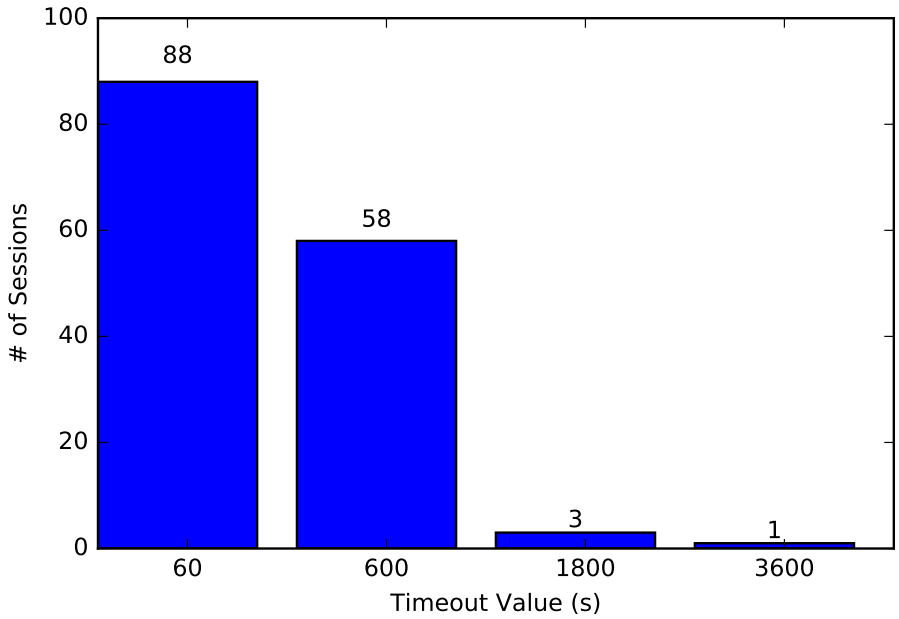


Figure 3.7 Estimated session frequency of a smart watch for 24 hours is plotted as the timeout value changes.

shows that the other half of the sessions are in short duration, but they are still longer than those of sessions of the sensors in the Xiaomi's smart home system.

3.4 Delegation-based DTLS (D2TLS)

The measurements with real IoT systems based on cloud services in Section 3.3 reveal that there are many short sessions. For instance, in case of condition-triggering sensors, the sensory data would be sent only when an activity/phenomenon of interest is observed. That is, the packet sizes of the sensory data and session durations are usually short.

As mentioned earlier, heavy computations occur when a DTLS[2] connec-

[2]Even though D2TLS can be applied to TLS as well, we proceed with DTLS for sake of

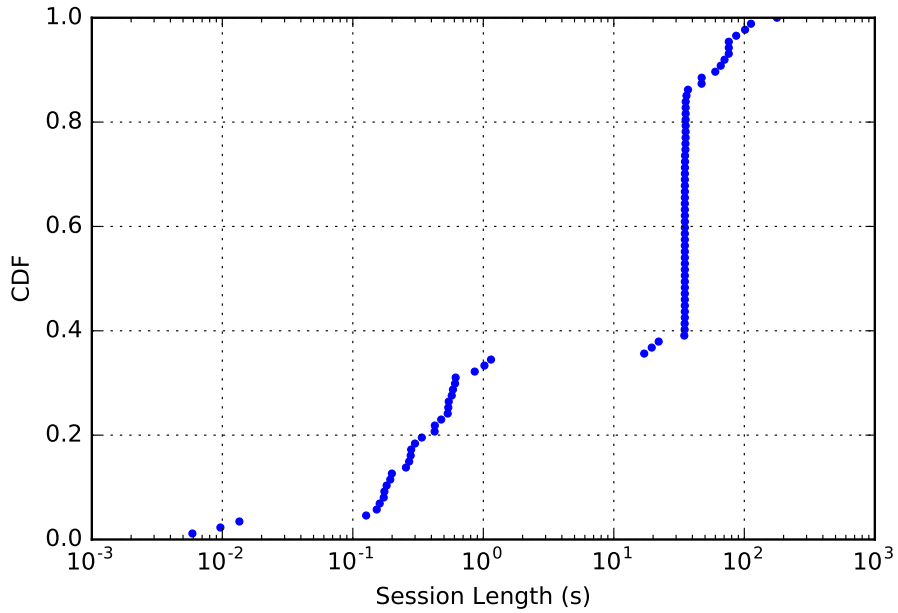
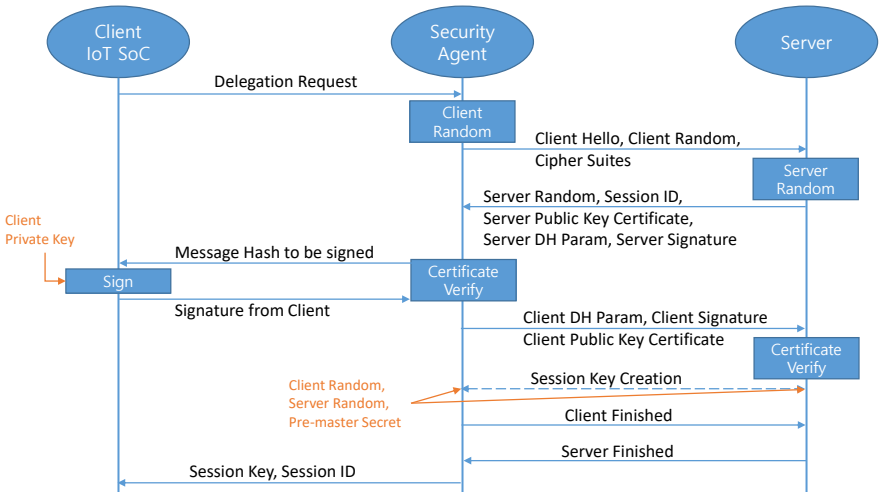


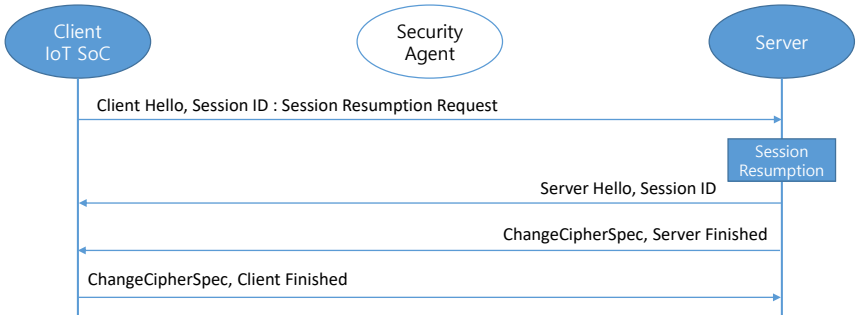
Figure 3.8 Session lengths of a smart watch for 24 hours are plotted in CDF when the threshold of session separation is 60s.

tion is created during its handshake process. Our measurement study, however, reveals that the communication patterns of cloud-based IoT services have intermittent and short sessions. Thus, if we have to set up a DTLS connection for each session, the burden of a session creation would be huge. To lower the burden on IoT devices, we leverage the session resumption feature in DTLS and introduce an entity for delegation—a security agent. However, unlike [23, 31], D2TLS blocks the security agent from accessing the privates keys of IoT devices, and hence there is no key escrow problem.

exposition in the remainder of this paper.



(a) Delegation-based full DTLS handshake



(b) End-to-end session resumption

Figure 3.9 Message flows for the D2TLS framework and the session resumption for the cloud-based IoT services are depicted.

3.4.1 D2TLS Framework

The goals of the D2TLS framework are (1) to make the private key of an IoT device not shared with other entities, and (2) to lower the burden of handling the DTLS handshake on the device significantly. We assume that the mutual authentication is to be achieved in IoT environments, since automated machine-to-machine communications will be prevalent since human

supervision is almost infeasible (i.e., typing id/pwd is infeasible). Moreover, we try to retain the current DTLS standard as much as possible.

As shown in Figure 3.9(a), the client part of a DTLS handshake is handled at the security agent (on behalf of the device), which is assumed to be located within the same IoT domain as the device. It is assumed that there is a secure channel between the device and the security agent before the message flow in Figure 3.9(a) starts. At first, the IoT device, who wishes to establish a secure session with a remote server in the cloud, sends a delegation request message to the security agent. Then the agent initiates a certificate-based full DTLS handshake with the remote server on behalf of the IoT device. As D2TLS performs mutual authentication, the security agent should create a signature for ECDHE (which is shown in Figure 3.9(a)) during the handshake process. For this, the security agent needs the private key of the IoT device. At this moment, the agent forwards the related information to the IoT device and then the IoT device decrypts or signs the given information. After the device processes the information and replies back to the agent, the agent resumes the handshake. Other parts in the current DTLS standard remains unchanged. Finally, the agent hands over the session context to the IoT device for the session resumption.

With D2TLS, the IoT device needs to carry out only the decryption or signing with its private key in terms of computations. All the other tasks are performed by the security agent. As for the remote server, the whole handshake process is exactly the same as the current DTLS standard. The session resumption is not changed from the standard. Modifications are made at the security agent and the IoT device. Therefore the modifications are

confined within a local IoT domain, which implies the easy deployability of D2TLS.

3.4.2 End-to-End Secure Connection

The end-to-end principle is one of the basic tenets of the Internet. It is also applied to security-related protocols and mechanisms; the notable example is the authentication of two endpoints, which is a crucial point addressed in this paper. Based on the authentication, DTLS provides confidentiality and data integrity.

There are two ways of resuming a DTLS session. One is the abbreviated handshake [30] and the other is the session resumption without server-side state [33]. The former requires both of the endpoints maintain the session state, and hence the session is resumable only with the session ID. The latter issues a session ticket including its session state, and the session is resumable with the session ticket, which removes the overhead of maintaining the session information (on the server side). An IoT device in D2TLS takes the former approach and works as a client in cloud-based IoT environments, which means the number of connections is limited by the number of the counterpart servers in the clouds. We believe it is not a significant burden that maintains a small number of connections on an IoT device. Also, the session ID is much smaller than the session ticket, which is suitable for low bitrate IoT networking.

As shown in Figure 3.9(b), the IoT device sends a ClientHello using the session ID to the server. After matching its session cache with the session ID, the server replies back a ServerHello with the same session ID if the server is willing to re-establish the connection. If the session ID is not matched, the server replies back a ServerHello with a new session ID, and the IoT device

discards this resumption attempt and proceeds to a new session creation with the security agent as described in Section 3.4.1. Other parts of the current DTLS standard remain unchanged.

Overall, the D2TLS framework makes the IoT device keep its private key, and lowers the burden of setting up a secure connection significantly. By leveraging the session resumption, D2TLS holds the end-to-end communication/security model. The processing overhead for keeping the private key will be investigated in the evaluation later.

3.5 Security Considerations

Let us now discuss the attack vectors in D2TLS. Recall that D2TLS improves the level of security over other delegation-based ones (e.g., [1]), by solving the key escrow problem and employing mutual authentication.

Compromised delegation server: If an adversary compromised a delegation server, this adversary could act as an IoT device and try to establish a secure connection with the server in the cloud. Because the mutual authentication is required for D2TLS, the private key of the IoT device is essential for the connection establishment. However, the adversary does not have the private key of the IoT device. Therefore she cannot set up a D2TLS connection as an IoT device by herself. The adversary might gain access to the stored security contexts which were established previously. In order to prevent this attack, the security contexts should not be stored at the delegation server after a handshake.

If an IoT device requests a new delegation to the compromised delegation server, the adversary could establish a new secure connection with the

signature from the IoT device. In this case, the attack could be noticed by the IoT device. When the IoT device notices a problem, the session should be made invalid immediately. That is, the adversary communicates with the server directly and does not send the session context back to the device. Or the adversary sends an incorrect security context back to the IoT device. The IoT device now realizes the delegation is not correctly performed through the delegation server.

The operator of the IoT domain must pay attention to the delegation server, in order to prevent and counteract the compromise. If the compromised delegation server connects to the cloud server only when the IoT device does not, it would be helpful to keep monitoring the communication patterns of the delegation server towards the outside of the domain.

Compromised cloud server: If an adversary compromises a server in the cloud, the adversary could gain access to security contexts which were established previously. Then the adversary could resume any connection with the IoT device, because the session resumption process does not require re-authentication by default. Also the private key of the server could be taken by the adversary in this case. However, this attack could occur regardless of delegation. To mitigate the problem, the lifetime of the session should be made shorter. Thus there is a tradeoff between the security against the cloud compromise and the D2TLS connection setup overhead, which is also the same for TLS/DTLS cases. It is also advised that the security contexts of inactive sessions be encrypted.

Compromised IoT device: If an adversary compromised an IoT device, the adversary could gain access to the private key of the IoT device. Therefore

the compromised device should be sanitized and the new public and private key pair should be re-generated. Again, this attack could exist regardless of delegation.

Eavesdropping and the lifetime of a session: The recommendation of session ID lifetimes is upto 24 hours with the RFC of TLS 1.2 [30]. The reason of the recommendation is that there is a possibility of attacks which obtain a master key. With the master key, the adversary could impersonate the compromised party until the corresponding session ID is expired.

If an adversary performs passive eavesdropping, the key recovery could be possible with high complexity. For the key exchange, D2TLS uses ECDHE which is known as one of the secure exchange. The recent paper which revealed a risk of RSA Diffie-Hellman recommended use of ECDHE [34]. After the exchange, the passive adversary tries to recover a AES key of the session. Even though there are a lot of attacks with AES, the best known attack [35] reduces the computational complexity of 128-bit AES down to $2^{126.18}$ without impractical assumptions (e.g., related-key attacks). Compared to the complexity of brute-force attack, 2^{128}, the attack is not practical for a short lifetime of a session.

If an adversary executes active eavesdropping, the attack could be successful within a short time period. We already discussed active compromises of the communicating parties. Some side-channel attacks could be done without the full compromise of one of the parties, e.g., cache attacks. The best known attack [36] shows the 128-bit AES key recovery after observing 100 encryptions, which requires just few minutes with a 1.5GHz Intel Pentium M machine. Considering the side-channel attack, the lifetime of a session should

Table 3.2 Comparison of IoT Devices under Test

SoC	Microcontroller	RAM	Flash Memory Storage	Radio Transceiver
TI CC3200	ARM Cortex-M4 @ 80MHz	256KB	1MB	IEEE 802.11b/g/n, 2.4GHz
TI CC2538	ARM Cortex-M3 @ 32MHz	32KB	512KB	IEEE 802.15.4, 2.4GHz

be managed short enough, based on the communication patterns.

3.6 Evaluation

We compare D2TLS with the current DTLS handshake performed by a standalone device.

3.6.1 Evaluation Environments

D2TLS is implemented for experimental evaluation with the main goal of providing a secure end-to-end connection setup between IoT devices (which are clients) and cloud servers with the delegation. The test environment consists of IoT clients, a security agent, a cloud server, and an OCSP server[3].

To investigate the feasibility and performance of D2TLS, we use two products for IoT clients: TI CC3200 SoC and TI CC2538 SoC as shown in Table 3.2. The experiments are mainly conducted with TI CC3200 devices, whereas some of the experiments are conducted with TI CC2538 devices in order to show the feasibility of D2TLS with lower hardware specifications. The security agent is co-located with a gateway to the Internet. The agent

[3]An online certificate status protocol (OCSP) server keeps track of the validity of certificates in charge.

is a machine with Intel Core i5-4690 CPU at 3.5GHz clock speed and 8GB of RAM. The agent is connected to a TI CC3200 device through a WiFi AP. Or the agent uses a USB interface to connect to a TI CC2538 node, which in turn is connected to another TI CC2538 node via IEEE 802.15.4.

We consider two networking environments between IoT device/security agent and cloud/OCSP server: LAN and WAN. In LAN settings, the cloud server and OCSP server are running in the same machine equipped with Intel Xeon E3-1245v3 CPU operating at 3.4GHz and 16GB of RAM. The IoT clients and the security agent are located in the same room; the cloud server and OCSP server are located in the other building in the same campus network in east Asia. The average RTT between the security agent to the cloud/OCSP server is 1.366ms.

In WAN environments, the cloud server and OCSP server are running is a virtual machine (VM), which is chosen as one of high-performance recommendations of the cloud service provider. The VM has 8 virtual cores of Intel Xeon E5-2666v3 and 15 GB of main memory. The locations of the IoT clients and security agent are same. However, the location of the VM is in a data center in the US across the Pacific ocean. The average RTT between the security agent to the cloud/OCSP server is 167.888ms.

The settings to implement the certificate-based DTLS handshake are as follows. We use OpenSSL 1.0.1p for creating the certificates and the OCSP server application. All the entities with this experiment are created and configured using WolfSSL 3.7.0 except for CC2538 node. Since the binary size of compiled WolfSSL is bigger than the available RAM size of CC2538, we use tinyDTLS 0.8.2 and the relic toolkit only for the CC2538 device. The

Table 3.3 Delay of a full DTLS handshake at an IoT device with 256-bit ECC key (in ms) is shown in LAN environments.

Setting		Handshake			Session Resumption
		Client Total	Reduction	Signature	
TI CC3200	DTLS	2,920	70.9%	-	57
	D2TLS	851		414	
TI CC2538	DTLS	63,161	73.5%	-	3,902
	D2TLS	16,748		9,903	

ciphersuite used for evaluation is ECDHE-ECDSA-AES128-CCM-8. Each of the results is averaged by five measurements, unless stated otherwise.

3.6.2 Delay

As shown in Table 3.3, the delay of a full DTLS handshake in LAN environments is compared from two viewpoints: (1) with and without delegation, and (2) two different IoT devices.

For the TI CC3200 device, the full handshake time is compared between (i) DTLS (TI CC3200 sets up a DTLS connection by itself) and (ii) D2TLS (the security agent sets up a DTLS connection, which TI CC3200 takes over). The delay of performing a handshake in D2TLS consists of a delegation request from the device to the agent, a delegated handshake at the agent, a signature operation at the IoT device, a transmission of the session context to the device, and a session resumption from the IoT device to the remote server as shown in Figures 3.9(a) and 3.9(b). The full handshake in D2TLS takes 851ms on average, while the one in DTLS takes 2,920ms on average. Among the delay of the handshake in D2TLS, signature-related operations take 414 ms, which means the burden of signing operations are about a half of the total time. The delay for the session resumption of TI CC3200 in D2TLS takes

Table 3.4 Delay of a full DTLS handshake at an IoT device with 256-bit ECC key (in ms) is shown in WAN environments.

Setting		Handshake			Session Resumption
		Client Total	Reduction	Signature	
TI CC3200	DTLS	3,581	48.6%	-	537
	D2TLS	1,840		415	

only 57ms. The delay of the full handshake in D2TLS is substantially reduced by 70.9% due to the delegation handshake, and the session resumption takes reasonably short time at the TI CC3200 device.

For a TI CC2538 device, the full handshake in DTLS takes 63,161ms. The full handshake in D2TLS takes 16,748ms and the session resumption takes 3,902ms on average. Recall that TI CC2538 has very limited resources. However, compared to TI CC3200 results, the initial handshake time with TI CC2538 is not so long, in consideration of the session resumption time. Notice that the reduction ratio (73.5%) of the full handshake time of TI CC2538 is higher than that of TI CC3200 (70.9%). It means that the delegation works well for the resource-limited devices. Even though we did not carry out experiments, TI CC2538 has a crypto module capable with ECC. The signature generation time could be reduced with the crypto module, which is discussed in Section 3.7.2.

The delay of a full DTLS handshake in WAN environments with the TI CC3200 device is shown in Table 3.4. The full handshake delays are compared between D2TLS and DTLS. The handshake delay is increased by 661 ms for DTLS, and 989 ms for D2TLS. The larger increase in D2TLS comes from the session resumption, which does not take place in DTLS. The delay for the session resumption is 537 ms, which is increased by 480 ms compared to LAN

Table 3.5 Energy consumption of a DTLS/D2TLS handshake in LAN settings at IoT devices with 256-bit ECC key is shown.

Setting		Delay (ms)	Power (mW)	Energy Consumption (mJ)
TI CC3200	DTLS	2,920	234	683
	D2TLS	851	276	235
	Session Resump.	57	364	21
TI CC2538	DTLS	63,161	573	36,161
	D2TLS	16,748	591	9,898
	Session Resump.	9,903	592	5,863

settings. Even though the increment of the handshake delay is larger, D2TLS outperforms DTLS about two times faster with the TI CC3200 device.

3.6.3 Energy Consumption

We measure the average power consumption of the IoT device by using a Monsoon power monitor. As shown in Table 3.5 in LAN settings, D2TLS consumes slightly more power than DTLS in TI CC3200. However, DTLS consumes approximately 2.9 times of energy than D2TLS, since the delay of DTLS is around 3.4 times longer than that of D2TLS. The third row in Table 3.5 shows the delay/power/energy incurred during the session resumption part in the full handshake in D2TLS. The session resumption consumes only 21mJ of energy, and it is about 3% and 9% of energy for D2TLS and DTLS, respectively. The power consumption of the session resumption is higher than the average power consumed during the full handshake process. However, it takes a small portion in terms of time in the full handshake, and hence the session resumption is energy-efficient.

The average power and energy consumption of TI CC2538 during DTLS and D2TLS handshakes show similar tendency with ones of TI CC3200 as

Table 3.6 Energy consumption of a DTLS/D2TLS handshake in WAN settings at IoT devices with 256-bit ECC key is shown.

Setting		Delay (ms)	Power (mW)	Energy Consumption (mJ)
TI CC3200	DTLS	3,581	234	838
	D2TLS	1,840	249	458
	Session Resump.	537	285	153

Table 3.7 Flash Storage and RAM Consumption for a full handshake in DTLS and in D2TLS with 256-bit ECC key at IoT Devices are shown in bytes.

Setting	Flash Usage (text+data)	RAM Usage (data+bss)	text +data +bss	Max. Dynamic Memory Usage
TI CC3200 (DTLS)	116,832	47,372	161,892	21,493
TI CC3200 (D2TLS)	99,144	47,388	144,204	15,237
TI CC2538 (DTLS)	77,330	14,253	91,033	1,960
TI CC2538 (D2TLS)	73,550	14,849	87,849	1,148

shown in Table 3.5. Note that, the session resumption takes about 59% of D2TLS energy consumption due to its large delay.

In WAN settings, the measured power consumption is slightly reduced compared to the results in LAN settings, except for DTLS as shown in Table 3.6. The IoT device may go into the idle/sleep state more frequently when it does not have loads for computations or communications, and that is why D2TLS and the session resumption consume less power than DTLS because they are lightweight in terms of computations and communications.

3.6.4 Code Size and Memory Requirements

Considering the resource constraints of IoT devices, the sizes of compiled binary and memory usage should be investigated. As described in Table 3.2, TI CC3200 has 256KB of RAM, and TI CC2538 has only 32KB of RAM. Their flash memory sizes are larger than RAM sizes, respectively. Therefore the major limitation comes from RAM usage for both static/predefined and dynamic memories.

As shown in Table 3.7, D2TLS at TI CC3200 takes less flash and dynamic memories, compared to DTLS. The compiled code of D2TLS is reduced since it requires less public-key related operations. TI CC3200 adopts Energia OS, which uses its dynamic memory mainly for heap space. To find out the maximum usage of dynamic memory, we change the size of the minimum heap space that can makes the program run. The total memory requirement of D2TLS is 62,625 bytes, especially considering 256KB of RAM.

The memory usage of TI CC2538 cannot be compared with those of TI CC3200 since different DTLS modules and microcontrollers are used. The compiled code of D2TLS is reduced due to the same reason of the TI CC3200 case. TI CC2538 adopts Contiki OS, which has no memory allocator and heap space but it has a stack space only. The maximum dynamic memory is measured by memory dump at the end of the handshake, by comparing zeroed memory dump before the handshake. The total memory requirement of D2TLS is 15,997 bytes, which fits into 32KB of RAM.

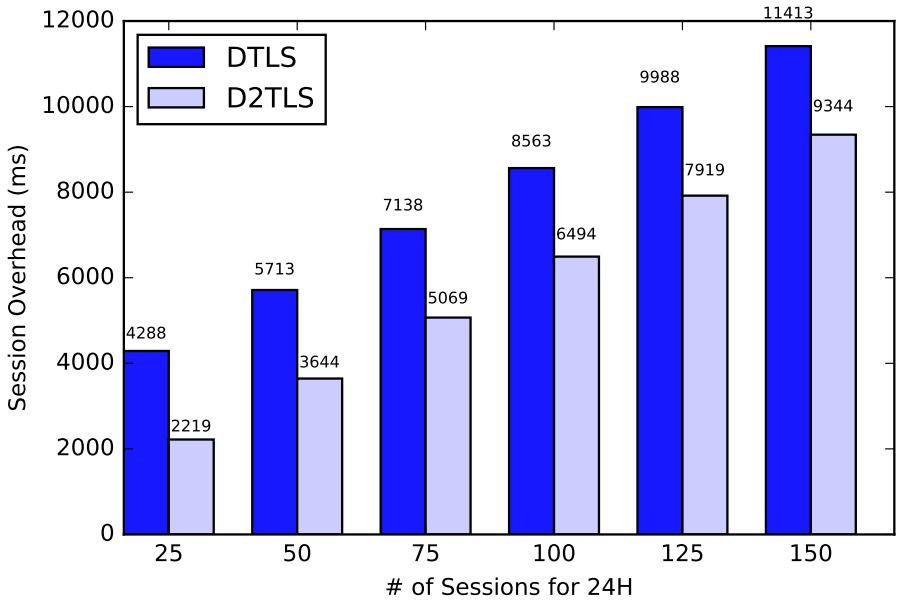


Figure 3.10 Session overhead varying frequency of the sessions for 24 hours is plotted (in ms).

3.6.5 Expected Session Overhead varying Frequency and Lifetime of a Session

So far, we evaluate the one-time session overheads with D2TLS and DTLS. The expected session overhead for 24 hours could be calculated based on our measurement results in Section 3.3.

The aggregate session overhead varying frequency of the sessions for 24 hours is plotted in Figure 3.10. The session overhead means the sum of delays of session creation (i.e., establishment) and resumption. The lifetime of session ID is set to 24 hours and the delays for calculation are measured from TI CC3200 with the LAN configuration. For every case, D2TLS has lower overhead than DTLS due to the shorter handshake delay of D2TLS. However, the relative reduction in overhead is getting smaller as the session frequency

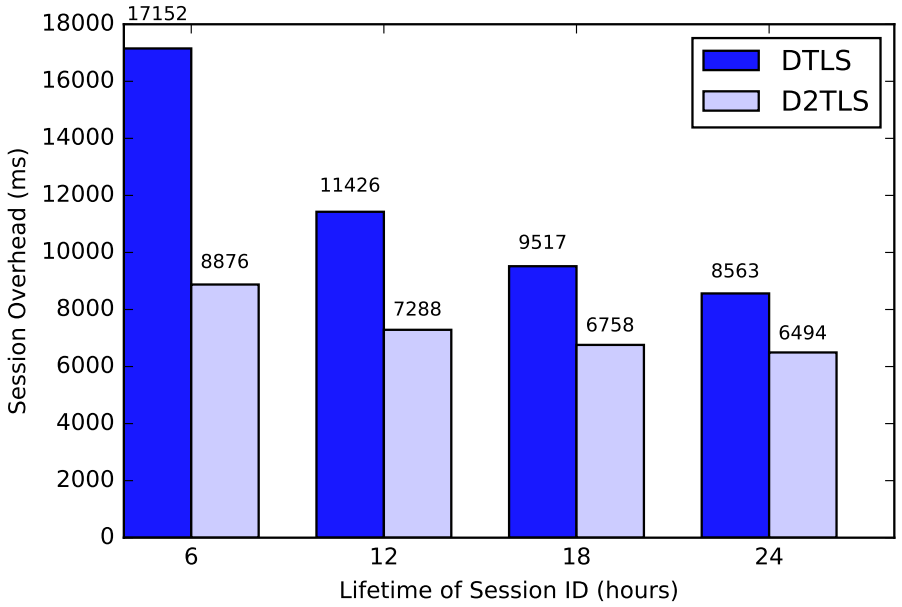


Figure 3.11 Session overhead varying lifetimes of a session ID for 24 hours is plotted (in ms).

grows, because the difference of the overheads comes from the handshake delay which is fixed and is related to the lifetime of session ID.

Figure 3.11 shows the aggregate session overhead varying lifetimes of a session ID for 24 hours. The session frequency is set to 100 and the delays are measured from TI CC3200 with the LAN configuration. The reduction of the session overhead of D2TLS over DTLS grows as the lifetime of a session ID decreases, because the shorter lifetime brings the more session creation which is substantial to the aggregate overhead. With 6 hours of session ID, the overhead of D2TLS is approx. 48% lower than one of DTLS.

Overall, the aggregate overhead could be reduced with D2TLS over DTLS. Especially the reduction of the overhead grows with the shorter lifetime of a

session ID. It is advised to use the short lifetime of a session ID in consideration with the attack to the session master key. However the short lifetime incurs more handshake overhead, being an obstacle in terms of efficiency. D2TLS reduces the first full handshake overhead of a DTLS session, thus it is beneficial for the short lifetime. Therefore D2TLS is suitable for the short lifetime of a session ID with the repeated session resumptions in terms of efficiency and security.

3.7 Discussion

3.7.1 IoT device as a Server

For a vast majority of cases, cloud-based IoT services will require an IoT device to operate as a client. While D2TLS in this paper assumes an IoT device to be a client, it can be extended to the case in which an IoT device serves as a server. If the device operates as a server, a remote client needs to know the server address information. The IETF CoRE working group suggests some resource discovery mechanisms [27, 37] for an IoT device to act as a server, which can be used in the D2TLS framework.

Using any resource discovery mechanisms, the remote client can find out the IP address of the device/server. The problem is that the device cannot perform a full DTLS handshake timely by itself. Our idea is to use Mobile IPv6. We assume there is a home agent in the same subnet as a device. When the home agent receives a ClientHello message destined to the device, it informs the remote client of the IP address of a security agent by sending a Binding Update message in Mobile IPv6. In this way, the client is ready to receive messages from the security agent. The home agent also delivers

Table 3.8 Execution time of sign/verify operations at TI CC2538 is measured with 256-bit ECC key (in s).

Setting	Signature Time	Verification Time
Software-only	9.126	20.534
Hardware-accelerated	0.34	0.70

the ClientHello message to the security agent as well as to the device. Then the device sends a delegation request message in Figure 3.9 if it wishes to proceed. Once the security agent receives the delegation request from the device, it will process the ClientHello and continue the full handshake with the remote client. The rest of the handshake and session resumption will be the same.

While the session resumption in D2TLS is made by the session ID only as described in Section 3.4, the session ticket extension [33] of the session resumption in DTLS/TLS can be considered for server mode operations of an IoT device. This extension allows the client to hold its session context, while the server does not need to maintain previous sessions. Therefore it is suitable for the case of an IoT device acting as a server.

3.7.2 Hardware-assisted IoT Security

We measure the execution times of cryptographic operations at TI CC2538, which is the most expensive part in D2TLS in terms of computational cost. Section 3.6.2 indicates that signing operations at TI CC2538 take significant time. The signature generation time costs approximately 59% of the total delay, which is about 10s. However, the device has a dedicated handware crypto module, which can expedite public key operations including RSA and ECC.

Table 3.8 shows that the execution times for sign/verify operations on a message of 256 bytes for software-only and hardware-accelerated implementations, respectively. For the signature generation time, the hardware-accelerated case costs only 0.34s, which is approximately 3.7% of that of the software-only one. The signature verification time of the hardware-accelerated implementation is also reduced to approximately 3.4% of that of the software-only one. It demonstrates that an IoT device that has a dedicated hardware crypto module can process signature-related operations much faster in D2TLS.

Nevertheless the dedicated crypto hardware does not reduce the benefits of D2TLS. A majority of IoT SoCs with a crypto hardware module currently supports the AES acceleration only, not ECC nor RSA. Even if an ECC-capable crypto hardware is adopted with IoT SoCs, D2TLS can still leverage the higher performance of the delegation server in terms of communication and computation overheads. In other words, the hardware acceleration enhances both DTLS/TLS and D2TLS.

Chapter 4

Conclusion

We first propose TwinPeaks, a new infrastructure of providing public keys to address the PKI problems like CA compromises and certificate revocation. TwinPeaks removes certificates and hence the PKI. Instead, it distributes public keys online by constructing a DNS-like hierarchical structure of public key servers. 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 explain 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 compare TwinPeaks with the current PKI and DANE from both qualitative and quantitative perspectives. For numerical results, we implement TwinPeaks, PKI, and DANE on a cloud service-based testbed that spans over two continents. Comprehensive experiments show that TwinPeaks can achieve less delays and smaller traffic than the other schemes.

We next propose a delegation-based DTLS/TLS framework (D2TLS) for cloud-based IoT services. D2TLS allows for a resource-limited IoT device to set up a DTLS/TLS connection with minimum computations; most of the DTLS/TLS handshake is performed a third party in D2TLS. Unlike prior schemes that make the private key of a device shared with the third party, D2TLS solves the key escrow problem. Our measurement studies investigate the communication patterns of two cloud-based IoT products, which reveal that most of the IoT service sessions are short and intermittent. Thus, creating a secure connection for each session will incur substantial overhead. To lower the burden of setting up a DTLS/TLS connection, D2TLS leverages the session resumption and introduces a security agent, which makes a DTLS/TLS connection on behalf of the device. Numerical experiments are conducted with two IoT devices of different hardware capabilities, which reveals that D2TLS can achieve better performance in terms of delay and energy consumption in comparison to running DTLS/TLS in standalone mode. In particular, the execution times of cryptographic operations in IoT devices can vary significantly depending on their capabilities and networking environments.

Bibliography

[1] P. Eckersley, "Iranian hackers obtain fraudulent HTTPS certificates: How close to a web security meltdown did we get?" March 2011. [Online]. Available: https://www.eff.org/deeplinks/2011/03/iranian-hackers-obtain-fraudulent-https

[2] N. Falliere, L. Murchu, and E. Chien, "W32.stuxnet dossier," February 2011. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

[3] T. Sterling, "Second firm warns of concern after dutch hack," July 2011. [Online]. Available: http://news.yahoo.com/second-firm-warns-concern-dutch-hack-215940770.html

[4] A. Langley, E. Kasper, and B. Laurie, "Certificate transparency," RFC 6962, June 2013.

[5] P. Eckersley, "The sovereign keys project," June 2012. [Online]. Available: https://git.eff.org/?p=sovereign-keys.git;a=blob;f=sovereign-key-design.txt;hb=master

[6] T. H.-J. Kim, L.-S. Huang, A. Perrig, C. Jackson, and 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.

[7] E. Cho, M. Park, and T. T. Kwon, "Twinpeaks: A new approach for certificateless public key distribution," in *2016 IEEE Conference on Communications and Network Security (CNS)*, October 2016, pp. 39–47.

[8] A. Delignat-Lavaudz, M. Abadiy, A. Birrelly, I. Mironovy, T. Wobbery, and Y. Xie, "Web PKI: Closing the gap between guidelines and practices," in *NDSS 2014*, 2014.

[9] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. Cranor, "Crying wolf: An empirical study of SSL warning effectiveness," in *USENIX Security '09*, 2009.

[10] S. Farrell, "Not reinventing PKI until we have something better," *Internet Computing, IEEE*, vol. 15, no. 5, pp. 95–98, September 2011.

[11] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, "Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations," in *IEEE S&P 2014*, 2014.

[12] L.-S. Huang, A. Rice, E. Ellingsen, and C. Jackson, "Analyzing forged SSL certificates in the wild," in *IEEE S&P 2014*, 2014.

[13] S. Farrell, "PKI: it's not dead, just resting," *IEEE Computer*, vol. 35, no. 8, pp. 41–49, August 2002.

[14] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Advances in Cryptology — CRYPTO 2001*, ser. LNCS, J. Kilian, Ed. Springer, 2001, vol. 2139, pp. 213–229.

[15] S. Al-Riyami and K. Paterson, "Certificateless public key cryptography," in *Advances in Cryptology - ASIACRYPT 2003*, ser. LNCS, C.-S. Laih, Ed. Springer, 2003, vol. 2894, pp. 452–473.

[16] P. Vixie, O. Gudmundsson, D. Eastlake, and B. Wellington, "Secret key transaction authentication for DNS (TSIG)," RFC 2845, May 2000.

[17] R. Oppliger, "Certification authorities under attack: A plea for certificate legitimation," *Internet Computing, IEEE*, vol. 18, no. 1, pp. 40–47, Jan 2014.

[18] R. Barnes, "Use cases and requirements for DNS-based authentication of named entities (DANE)," RFC 6394, October 2011.

[19] P. Hoffman and J. Schlyter, "The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA," RFC 6698, August 2012.

[20] S. D. Galbraith, K. G. Paterson, and N. P. Smart, "Pairings for cryptographers," *Discrete Applied Mathematics*, vol. 156, no. 16, pp. 3113 – 3121, 2008.

[21] X. Yao, X. Han, and X. Du, "A light-weight certificate-less public key cryptography scheme based on ECC," in *ICCCN 2014*, Aug 2014, pp. 1–8.

[22] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu, "When HTTPS meets CDN: A case of authentication in delegated service," in *IEEE S&P 2014*, May 2014, pp. 67–82.

[23] R. Hummen, H. Shafagh, S. Raza, T. Voig, and K. Wehrle, "Delegation-based authentication and authorization for the ip-based internet of things," in *2014 Eleventh Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, June 2014, pp. 284–292.

[24] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645 – 1660, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13000241

[25] M. B. Barcerna and C. Wueest, "Insecurity in the internet of things," *Symantec*, 2015. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/insecurity-in-the-internet-of-things.pdf

[26] T. Fossati and H. Tschofenig, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things," RFC 7925, Jul. 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc7925.txt

[27] C. Bormann, K. Hartke, and Z. Shelby, "The Constrained Application Protocol (CoAP)," RFC 7252, Jun. 2014. [Online]. Available:

https://rfc-editor.org/rfc/rfc7252.txt

[28] "oneM2M Security Solutions," *oneM2M Partners*, August 2014. [Online]. Available: http://onem2m.org/images/files/deliverables/TS-0003-Security_Solutions-V-2014-08.pdf

[29] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347, Jan. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6347.txt

[30] T. Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, Aug. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5246.txt

[31] S. R. Moosavi, T. N. Gia, E. Nigussie, A. M. Rahmani, S. Virtanen, H. Tenhunen, and J. Isoaho, "Session resumption-based end-to-end security for healthcare internet-of-things," in *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on*, Oct 2015, pp. 581–588.

[32] "Announcing keyless SSL™: All the benefits of cloudflare without having to turn over your private SSL keys," *CloudFlare, Inc.*, 2014. [Online]. Available: https://blog.cloudflare.com/announcing-keyless-ssl-all-the-benefits-of-cloudflare-without-having-to-turn-over-your-private-ssl-keys/

[33] J. A. Salowey, "Transport Layer Security (TLS) Session Resumption without Server-Side State," RFC 5077, Jan. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5077.txt

[34] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann, "Imperfect forward secrecy: How Diffie-Hellman fails in practice," in *22nd ACM Conference on Computer and Communications Security*, Oct. 2015.

[35] A. Bogdanov, D. Khovratovich, and C. Rechberger, *Biclique Cryptanalysis of the Full AES*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 344–371. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25385-0_19

[36] D. Gullasch, E. Bangerter, and S. Krenn, "Cache games – bringing access-based cache attacks on aes to practice," in *2011 IEEE Symposium on Security and Privacy*, May 2011, pp. 490–505.

[37] C. Bormann, Z. Shelby, P. V. der Stok, and M. Koster, "CoRE Resource Directory," Internet Engineering Task Force, Internet-Draft draft-ietf-core-resource-directory-08, Jul. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-ietf-core-resource-directory-08

초록

상대방을 인증하고 데이터 통신을 보호하는 것은 안전한 통신을 위한 기본적이고 중요한 수단이다. 일상생활에서 인터넷 보급이 점차 가속화됨에 따라, 안전한 통신에 대한 요구가 증가하고 있다. 그러나, 앞서 언급한 두 가지 수단은 PKI의 문제점과 IoT 기기의 제한된 자원 등의 이유로 위협받고 있다. 그러므로 본 학위논문은 공개 키 배포를 고려한 인증, 그리고 자원이 제한된 IoT 기기를 고려한 데이터 보호를 개선하는데 초점을 둔다.

첫째로, 현재의 PKI는 인증서의 폐기 및 위조 인증서 등의 문제를 가지고 있다. 이와 같은 이슈를 해결하고자, 온라인 상의 이름 붙은 개체의 공개 키를 배포하는 새로운 기반구조인 TwinPeaks를 제안한다. TwinPeaks는 무인증서 공개 키 암호(CL-PKC)를 활용하며, 개체의 공개 키가 네크워크 파라미터의 조합에 기반하도록 CL-PKC를 확장하였다. 따라서 TwinPeaks는 시스템적으로 스푸핑 공격에 대응할 수 있다. TwinPeaks는 공개 키 서버가 필요한데, 이 서버는 DNS와 유사한 계층 트리를 구성한다. 트리의 각 부모-자식 링크에서, 모든 이름 붙은 개체가 자신의 공개/비밀 키 쌍을 가지고 있는 것과 같은 방식으로 부모와 자식이 상호작용한다. TwinPeaks는 인증서를 제거하였으므로 따라서 폐기에 따르는 오버헤드가 없다. 그 대신, 각 이름 붙은 개체는 자신의 IP 주소와 공개 키가 각각 DNS 서버와 키 서버에 항상 최신으로 유지/관리될 수 있도록 해야 한다. TwinPeaks는 또한 확장성있는 공개 키 배포를 달성하였는데, 이는 보안 위험을 높이지 않으면서도 공개 키를 장기간 캐시할 수 있기 때문이다.

다음으로, 사물인터넷(IoT)는 예측가능한 미래에 일반화 될 것이다. 그러나 IoT 기기의 제한된 자원을 고려할 때 인터넷의 보안 문제는 IoT 서비스에서 더

욱 악화될 것이다. 클라우드 기반 IoT 서비스를 위해 위임 기반의 DTLS/TLS 프레임워크(D2TLS)를 제안한다. D2TLS는 IoT 기기의 개인 키를 외부에 공개하지 않으면서도 상호 인증을 달성하며, 보안 연결을 생성할 때의 부담을 큰 폭으로 줄이는 것을 목표로 한다. DTLS/TLS 표준의 세션 재개를 활용하고 보안 에이전트를 도입함으로써, D2TLS는 IoT 도메인 내부만을 변경하면서도 이러한 목표를 달성한다. 즉, 클라우드와 PKI 시스템에는 D2TLS를 도입할 필요가 없다. D2TLS는 DTLS/TLS 연결을 직접 다루는 경우와 비교하여 지연시간과 에너지 소비 측면에서 더 나은 성능을 달성하였음을 수치 결과로 보였다.

주요어: 공개 키 기반구조(PKI), 무인증서 공개 키 암호(CL-PKC), 전송 계층 보안(TLS), 데이터그램 전송 계층 보안(DTLS), 위임, 세션 재개, 사물인터넷(IoT)

학번: 2008-20981