iTLS: Lightweight Transport-Layer Security Protocol for IoT With Minimal Latency and Perfect Forward Secrecy

Pengkun Li, Jinshu Su, *Senior Member, IEEE*, and Xiaofeng Wang

***Abstract*—Enabling end-to-end secure communication is essential for many Internet-of-Things (IoT) application scenarios. Transport-layer security (TLS) and datagram TLS (DTLS) are the de-facto protocols for communication security in the IP-based IoT. However, the current authentication approaches of TLS are confronted with the heavy overhead and security issues in the resource-constrained IoT scenario. On the other hand, the identity-based cryptography (IBC) becomes an attractive cryptographic solution for the IoT. Unfortunately, the current IBC-based proposals exist the problem of either high communication latency or low level of security. In this article, we propose the first lightweight secure transport protocol called iTLS, which delivers protected data in the first flight with perfect forward secrecy, and provides implicit mutual authentication without certificates. iTLS dynamically generates identity-based early keys before receiving a server response, allowing clients to send the encrypted data without additional round trips. Furthermore, it employs the ephemeral secret ticket to obtain an ephemeral server key in the previous connection. Therefore, the early key established afterward can provide full forward secrecy. Design and implementation in the form of extension make iTLS fully compatible with TLS 1.3 and easy to be converted to a DTLS version. Our evaluation shows that iTLS reduces the network traffic overhead by at least 61.2%, and handshake latency on an ideal network by at least 60% compared to the certificate-based TLS. The results demonstrate iTLS achieves strong adaptability on the low-power and lossy IoT networks.**

***Index Terms*—Authentication, end-to-end security, identity-based cryptography (IBC), Internet of Things (IoT), secure transport protocol.**

I. Introduction

THE INTERNET of Things (IoT) plays an increasingly considerable role in all aspects of our daily life. It has been used in various application domains, such as factory automation, medicine of healthcare, and smart city/home [1]. Naturally, the data collected in many scenarios are sensitive and may be delivered through the untrusted network infrastructure, e.g., the Internet. Therefore, end-to-end communication security is a critical requirement for these IoT application scenarios, which can protect the sensitive data even if the underlying network infrastructure is not under the user's control [2]–[4]. Current IoT security solutions include standards-based proposals. Among them, the transport-layer security (TLS) protocol and its datagram-oriented variant datagram TLS (DTLS) protocol are the de-facto protocols for communication security in the IoT [5], [6], which can provide encryption, authentication, and data integrity for information exchange. The Internet Engineering Task Force (IETF) has defined a TLS/DTLS 1.2 profile that offers communication security for the IoT applications [7]. The newest version 1.3 of TLS provides considerable performance improvement and smaller message sizes that can further advance the IoT communication security [8].

Although the standardized Internet security protocols bring many advantages in terms of interoperability and deployment, typical deployment scenarios of IoT limit the feasibility of TLS because of highly constrained devices in low-power and lossy networks of the IoT [3], [6]. The TLS protocol supports authentication using the symmetric preshared key (PSK) or public-key certificates. PSK-based authentication consumes a small number of computational resources and bandwidth. However, considering the large-scale deployment of IoT devices, there exist key management and scalability issues when using the PSK. In addition, the PSK established out of band is vulnerable to attacks due to the IoT devices' uncontrolled deployment environment and restricted security features. Generating the PSK also needs sufficient entropy to ensure security, which is usually not guaranteed in practice. Finally, the data encrypted by the PSK, especially sent in the zero round-trip time (0-RTT) mode in TLS 1.3, cannot provide forward secrecy [8], [9].

Certificate-based authentication can solve many challenges where particularly PSK-based solutions fall short, so a number of IoT systems choose certificates as the authentication method [5], [10]. However, the long certificate chain processing, as well as the revocation list checking, bring significant overhead in terms of resource consumption, including computing, memory, bandwidth, and energy. In addition, due to the low-packet delivery rate and computational performance, certificate transmission and verification also induce a considerable increase in the connection latency of the protocol. Furthermore, the use of small packets (e.g., the maximum frame size for IEEE 802.15.4 is 127 B) will result in the fragmentation of large certificate messages, which may

Manuscript received December 23, 2019; revised March 25, 2020; accepted April 9, 2020. Date of publication April 15, 2020; date of current version August 12, 2020. This work was supported by the National Key Research and Development Program of China under Grant 2017YFB0802300 and Grant 2018YFB0204301. *(Corresponding author: Jinshu Su.)*

The authors are with the College of Computer, National University of Defense Technology, Changsha 410073, China (e-mail: lipengkun14@nudt.edu.cn; sjs@nudt.edu.cn; xf_wang@nudt.edu.cn).

Digital Object Identifier 10.1109/JIOT.2020.2988126

cause specific attacks [11]. Meanwhile, the above problems is exacerbated due to the fragment losses and the need for retransmissions. Although both hardware-related and software-related improvements have been considered by many works [7], [12]–[16], the above problems have not been effectively solved.

Identity-based cryptography (IBC) is a public-key cryptography mechanism that allows the user to take its identity as the public key. Thanks to its low-bandwidth overhead and high scalability, the IBC-based end-to-end security solutions for IoT gained considerable attention. However, as we discuss in the related work, exiting solutions either have the problem of complex communication procedures and high communication latency or cannot provide enough security as the traditional end-to-end security scheme. Some proposals employ the identity-based noninteractive key agreement (IBNIKA) protocol [17], [18] to establish pairwise keys between devices [19]–[22], benefitting from extremely low-bandwidth overhead. However, this approach provides a lower level of security, especially the shared key established is not forward secret. Compromising the long-term private key of either party can result in the compromise of the shared key. Also, the assailant who compromises a device's private key can impersonate any legitimate entity to establish communications with the device. On the other hand, some proposals adopt the identity-based encryption (IBE) or the identity-based signature (IBS) instead of certificates for authentication and apply the Diffie–Hellman (DH) key exchange protocol to establish shared keys [20], [23]–[25], thereby achieving the authenticated key agreement. While this approach provides higher security, the DH parameters exchange imposes an additional round trip before sending any sensitive data. Furthermore, few IBC-based solutions consider compatibility with the standardized protocols, which is not conducive to the protocol deployment and the interoperability between IoT and the Internet.

In this article, we extend our previous work [26] to propose iTLS, the first lightweight secure transport protocol for IP-based IoT, which can deliver the protected data in the first handshake flight with perfect forward secrecy, and provide implicit mutual authentication without using certificates. The iTLS authenticates communication parties while establishing the shared key by integrating the identity-based key agreement protocol into TLS. Therefore, the entity only needs to be configured with the identity-based private key and requires few computing resources used to calculate the shared secret to establish encrypted communications. The iTLS also introduces little bandwidth and energy consumption because no large certificates messages need to be transmitted. What is more, iTLS comes with minimal connection latency through a zero round-trips handshake, which also provides perfect forward secrecy for the 0-RTT data. Specifically, the main contributions of iTLS are as follows.

First, iTLS designs the identity-based 0-RTT cryptographic handshake that employs dynamic early keys (IDEK) to minimize the security handshake latency for all connections. The iTLS handshake allows the client to dynamically generate the IDEK, i.e., an authenticated shared key with the server, based on the identities of the communication parties before receiving a reply from the server. IDEK enables the client to send the data immediately following the initial handshake message, i.e., "early data," without sharing a static PSK with the server in advance, thereby establishing the encrypted connection with no additional round trips.

Second, iTLS introduces the ephemeral secret ticket mechanism to provide perfect forward secrecy and replay protection for the early data. After each connection is established, the server sends a ticket that is associated with an ephemeral secret determined by itself. This ticket is leveraged to generate the IDEK for the subsequent connection. Establishing the IDEK with the ephemeral cryptographic key corresponding to the ticket can provide forward secrecy for the early data. Rejecting reduplicative tickets on the server side can also mitigate the replay attack on the early data [9], [27].

Finally, the extension-based design and implementation make the iTLS fully compatible with the TLS 1.3 protocol and can facilitate its deployment with the assurance of interoperability with the Internet. Moreover, a design in the form of the extension also makes the iTLS easy to be converted to DTLS-oriented versions [28] that can provide security for datagram-based applications, e.g., the CoAP protocol [29].

We implement the iTLS protocol based on an IoT-oriented open TLS library and evaluate the performance of iTLS on networks with different latencies, packet loss rates, and bandwidth. We compare iTLS with the TLS 1.3 protocol, and the experimental results show that iTLS reduces the network traffic overhead by at least 61.2% and the full handshake latency by at least 60% compared to the certificate-based TLS protocol at 128-b symmetric-key security. The 0-RTT model of iTLS handshake makes establishing secure connections using iTLS at least 1.5 times faster than using the certificate-based TLS. In addition, the connection performance of iTLS is similar to that of the PSK-based TLS, especially in low-power and lossy networks.

The remainder of this article is organized as follows. Section II presents some prerequisites about iTLS, including an overview of TLS 1.3 and the identity-based authenticated key agreement (IBAKA) protocol. In Section III, we detail the iTLS design. The security of iTLS is analyzed in Section IV. Then, Section V implements and evaluates iTLS. Finally, we discuss the related work and conclude in Sections VI and VII, respectively.

II. Prerequisites

A. TLS 1.3: Overview

TLS 1.3 comprises two primary components, a handshake protocol and a record protocol [8]. The handshake protocol negotiates cryptographic algorithms and parameters, establishes shared keying material, and authenticates the communicating parties. The record protocol carries the handshake messages and application-layer data to be transmitted and divides traffic up into a series of records, each of which is independently protected with the parameters established by the handshake protocol.

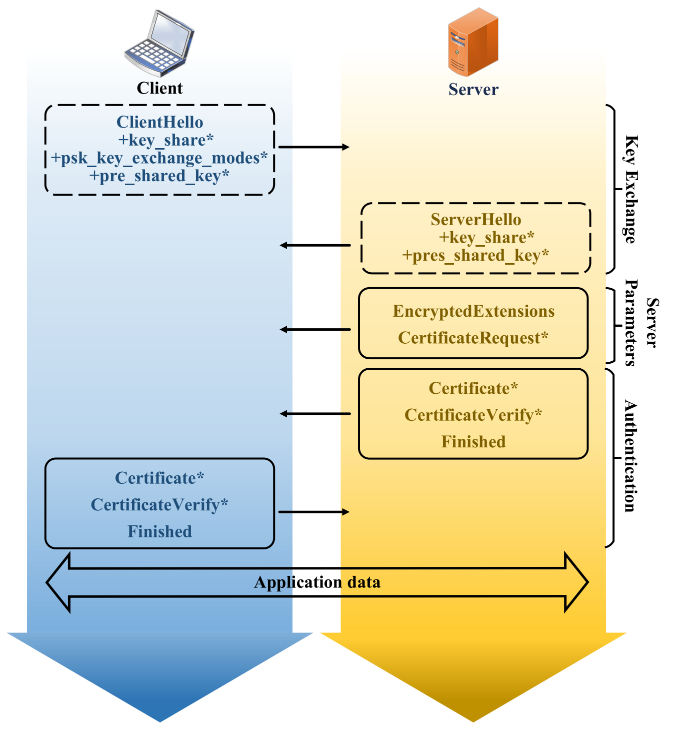


Fig. 1. Full TLS 1.3 handshake. "+" indicates extensions sent in previously noted messages; "*" indicates optional or situation-dependent messages/extensions that are not always sent; and solid line box indicates messages encrypted.

The handshake protocol is the core of the TLS protocol. TLS 1.3 uses a new handshake protocol compared to the previous versions. This handshake protocol reduces the handshake latency from the original two RTT to one RTT. A basic full TLS 1.3 handshake is shown in Fig. 1.

The TLS client sends the ClientHello message to initialize the handshake, which contains its supported cryptographic algorithms and parameters. After receiving and processing the ClientHello, the server determines the appropriate cryptographic parameters and responds with a ServerHello message to indicate the negotiated connection parameters. The client and server complete the key exchange and establish the shared secret key in this phase. TLS 1.3 supports three key exchange modes: 1) (EC)DHE (DH exchange over either finite fields or elliptic curves); 2) PSK-only; and 3) PSK with (EC)DHE. If (EC)DHE key establishment is in use, a "key_share" extension is included in the ClientHello and ServerHello messages, which contains the client and server's Diffie–Hellman key shares, respectively; if PSK key establishment is in use, a "pre_shared_key" extension is used by the client and the server. This extension contains a set of PSK labels in the ClientHello, and the PSK identity chosen by the server in the ServerHello. The (EC)DHE and PSK can be used together, in which case both extensions are contained in the ClientHello and ServerHello messages.

After sending the ServerHello, the server then sends the EncryptedExtensions message which contains extensions that are not required to determine the cryptographic parameters, and the CertificateRequest message if client authentication is desired.

The client and the server send authentication messages to each other in the authentication phase. If they choose the (EC)DHE mode to exchange key shares, certificate-based authentication is needed. The server and client send Certificate and CertificateVerify messages to authenticate themselves. The former message contains the endpoint's certificate chain and the latter one contains a signature over the handshake transcript using the private key corresponding to the public key presented in the Certificate. Finally, they exchange the Finished message to complete the handshake, which contains a MAC over the entire handshake. If the PSK-based key exchange is used, endpoints do not send the Certificate and CertificateVerify messages, since they authenticate each other via the PSK.

Note that, except for the ClientHello and ServerHello messages, all other handshake messages are encrypted by the shared key established after exchanging the ClientHello and ServerHello. As a contrast, only the Finished message is encrypted in the previous versions.

Compared with the older versions, the other major change in TLS 1.3 is to support the 0-RTT model that allows the client to send the application data to the server on the first flight when they share a PSK [8]. With this model, the ClientHello message contains an "early_data" extension to indicate that the client will send the early data in the 0-RTT model. Then, the client can immediately send the application data encrypted with the PSK following the ClientHello message. If the server determines to accept the early data, the server chooses the first PSK presented in the list of PSKs offered by the client and adds the "early_data" extension to the EncryptedExtensions message. Then, the server also uses the PSK to decrypt the early data. In the 0-RTT model, endpoints authenticate each other via the PSK, too. After the handshake is completed, the connection is protected under the new keying material established via the PSK or (EC)DHE.

B. Identity-Based Authenticated Key Agreement

In 1985, Shamir [30] first proposed the concept of IBC. The main idea of IBC is to use the user's unique identifier as its public key. The corresponding private key is generated by a trusted authority, called the private-key generator (PKG), using secret knowledge only possessed by the PKG. Boneh and Franklin [31] proposed the first fully functional solution for IBE schemes using bilinear pairing in 2001. Following this article, numerous feasible IBAKA protocols based on bilinear pairing have been proposed [32]–[35]. There are also some pairing-free IBAKA schemes proposed to avoid the computational overhead of the pairing [36]–[39]. Generally, a two-party IBAKA can be described using three algorithms.

Setup(k): Given a security parameter k as the input, this algorithm outputs the system public parameters SPP and the master secret key msk.

KeyExtract(SPP, msk, ID$_i$): This algorithm extracts a private key from an arbitrary bit string. It takes an identity ID_i, the msk, and SPP as the input, and then outputs a secret key sk_i for ID$_i$.

KeyAgreement(SPP, sk$_i$, es$_i$, ID$_p$, EK$_p$): Before establishing the shared key, the communication parties generally choose an ephemeral secret es and compute the corresponding ephemeral public key EK, respectively, and then exchange their ephemeral public keys. This algorithm takes SPP, the private key sk_i of ID_i and its ephemeral secret es_i, the peer's identity ID_p and its ephemeral public key EK_p as the input, and outputs a secret shk shared with ID_p for ID_i. es and EK can be $\emptyset$, then this is an identity-based two-party noninteractive key exchange scheme [17], [18].

In this article, we chose the pairing-based IBAKA algorithm proposed in [34] to better illustrate iTLS because of its high computational performance and the security properties it holds. Here, we briefly review the basic facts of bilinear pairing and several related hardness assumptions. The details can be found in [32].

Bilinear Pairing: Let G denote an additive group and G_T denote a multiplicative group, and both of them are cyclic groups of prime order q. Let P denote a generator of G. A bilinear pairing is a map $e : G \times G \rightarrow G_T$, which has the following properties.

1) *Bilinear:* For all $Q, R \in G$ and all $a, b \in \mathbb{Z}_q$, we have $e(aQ, bR) = e(Q, R)^{ab}$.
2) *Nondegenerate:* $e(P, P) = 1$.
3) *Computable:* For all $Q, R \in G$, $e(Q, R)$ is efficiently computable.

Assumption 1 [Bilinear DH (BDH)]: For $a, b, c \in \mathbb{Z}_q^*$, given (P, aP, bP, cP), computing $e(P, P)^{abc}$ is hard.

Assumption 2 [Computational DH (CDH)]: For $a, b \in \mathbb{Z}_q^*$, given (P, aP, bP), computing abP is hard.

Assumption 3 [Elliptic Curve Discrete Logarithm (ECDL)]: Given (P, aP), extracting $a \in \mathbb{Z}_q^*$ is hard.

III. iTLS Design

This section describes the details of the iTLS protocol. iTLS utilizes IBC to establish an inherent binding between the public key and the entity presenting the public key. By using the IBAKA to authenticate communication parties while establishing the shared key, iTLS eliminates the transmission and verification of certificates during the connection establishment. The iTLS also generates the early key dynamically using the identities of the communication parties, which is used to protect the data sent on the first flight. Moreover, iTLS associates the ticket with an ephemeral server key and uses it in the early key generation procedure of the next connection to provide forward secrecy and replay protection.

The iTLS consists of two phases. In the initialization phase, a PKG is initialized and generates the private keys for entities. Then, the communication participants use the iTLS handshake protocol to establish secure connections between each other. This section first introduces the basic iTLS handshake protocol, where a new "identity_share" extension is presented for key establishment and authentication. Then, we extend the basic protocol to support the 0-RTT model, which adopts IDEK for early data transmission and ephemeral secret tickets for early data security enhancement.

A. Initialization

Before using iTLS for secure communication, a trusted PKG is needed to initialize cryptographic system parameters and generate private keys for communication participants. The PKG is usually run by the administrator for a specific IoT ecosystem. First, the PKG is initialized through the *Setup* algorithm of the chosen IBAKA scheme, which generates the system public parameters and the master secret key at a specified security level. Specifically, in this article, the PKG performs the following steps in the initialization phase.

The PKG selects two groups G and G_T, a bilinear pairing $e : G \times G \rightarrow G_T$, and the generator P as described in Section II. It then randomly selects $s \in \mathbb{Z}_q^*$ as the master secret key and computes the master public key $P_{\text{pub}} = sP$. It also chooses a hash function $H : \{0, 1\}^* \rightarrow G$ to map an arbitrary string to a point on G. Finally, the PKG stores the master secret key s secretly and publishes the system parameters $SPP =< G, G_T, e, P, P_{\text{pub}}, H >$.

After the system parameters initialization is completed, the communication participants can register their own identities with the PKG and obtain the corresponding private keys. The identity can be the unique identifier of a device or the identifier of an IoT platform. The PKG adopts the *KeyExtract* algorithm of the specified IBAKA to generate the corresponding private key for an entity. In this article, the PKG maps the entity's identity ID_i to a point on G via H and calculates the private key as $sk_i = sH(\mathrm{ID}_i)$. $PK_i = H(\mathrm{ID}_i)$ is used as the public key corresponding to the identity.

This article does not discuss in detail the specific implementation of PKG and how it assigns the private key to the device in IoT. The administrator can run a PKG server in its administration domain and configure the device with the URI of the PKG. Then, the device can fetch and update the public system parameters and the private key based on the configured URI. There are usually two types of strategies to initialize a private key for a device. One is by the offline type. Before deploying the device, the administrator can preload each device with a unique identity, the public system parameters, and the associated private key. The other approach is to initialize the device online dynamically. This approach requires the device to carry through an authentication procedure with the PKG, and a cryptographic channel to distribute the private key. After obtaining the private key for the first time, subsequent key updates can be done via the secure channel established by iTLS.

Compared to the certificate-based authentication where the private key is generated locally, the private key used for iTLS communication is generated by the PKG. The iTLS protocol may face a security risk of the private-key disclosure due to improper management of the PKG, which can easily lead to man-in-the-middle attacks and interception of the traffic in the system (see Section IV). Therefore, a stronger key management system shall be adopted by the PKG when using iTLS. For example, the PKG can adopt a hardened security device, e.g., the trusted platform module (TPM) [40], to protect the master secret, and should update the master secret and public system parameters periodically. In addition, iTLS requires all entities to register with the same PKG for encrypted communication. In typical IoT networks with closed groups, such as wireless sensor networks, the trusted PKG should be played by the network administrator. However, in networks with multiple administrative domains, hierarchical and cross-domain IBAKA algorithms should be considered by the administrator.

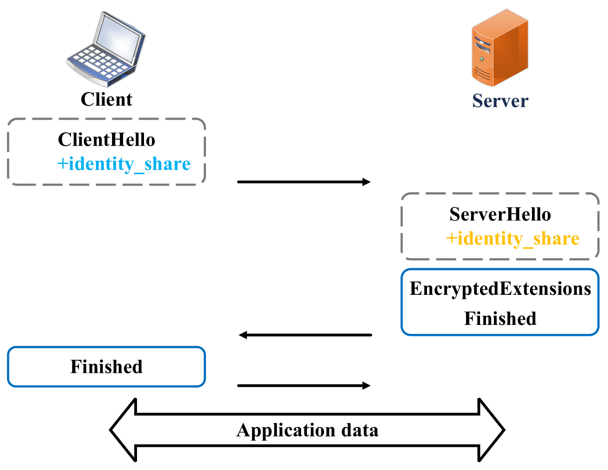


Fig. 2. Basic full iTLS handshake. "+" indicates newly added extension and blue box indicates messages protected using keys derived from handshake_secret.

After registering the identity with the PKG and equipped with the associated private key, each entity can use iTLS to establish secure connections with others. The iTLS adopts a new identity-based cryptographic handshake protocol and the same record-layer protocol as TLS 1.3. Next, we introduce the iTLS handshake protocol.

B. Handshake Protocol

The basic iTLS handshake, shown in Fig. 2, is similar to that of TLS 1.3 with two major changes are as follows.

1) A new extension is used by ClientHello and ServerHello to exchange identity information.
2) Remove the component in the handshake related to the certificate.

The "identity_share" extension is added to iTLS. The client and the server send this extension in the ClientHello and ServerHello messages to exchange their identities and cryptographic parameters used for establishing the shared secret. The format of the "identity_share" extension is provided below. The *identity* field contains the endpoint's identity that represents its public key.

The *trusted_authority* field comprises the identifier of the PKG where the endpoint registers with the identity and the related algorithm information. The information should include the IBAKA algorithm used, which can be represented by an object identifier, and the system public parameters generated by the PKG, the structure of which is specified by the IBAKA algorithm. In order to reduce the number of bytes carried in this field, a hash of the algorithm information is generally put into the trusted_authority field.

Finally, the endpoint usually chooses a random ephemeral secret and calculates an ephemeral public key for the establishment of the shared secret. The ephemeral public key is carried in the *key_exchange* field, whose content is determined by the specified IBAKA algorithm

```
struct {
    TrustedAuthority trusted_authority;
    Identity identity;
    opaque key_exchange;
} IdentityShare.
```

Fig. 2 shows the basic full iTLS handshake.

The client sends the ClientHello message containing an "identity_share" extension. This extension indicates to the server that the client supports iTLS and further tells the server the client's identity ID_C, the corresponding PKG that distributes the private key for this identity, and the ephemeral public key the client provides for key agreement. According to the IBAKA algorithm selected in this article, the client selects an ephemeral secret $x \in \mathbb{Z}_q^*$ at random and calculates the ephemeral public key $EK_C = xP$.

The server processes the ClientHello message and checks the carried "identity_share" extension. It needs to validate the client's identity and whether the PKG presented in the extension is trusted. There are two cases where the PKG received is untrusted. One case is that the server has not registered with the same PKG and does not have the corresponding private key. The other case is that the PKG parameters presented in ClientHello are different from those known to the server, which usually happens when PKG updates parameters. Under these circumstances, the server can either continue the handshake by responding with a HelloRetryRequest message if the server supports other authentication methods, or abort the handshake with an alert directly.

If the server has registered with the same PKG, then it responses a ServerHello message with an "identity_share" extension which contains the server's identity ID_S, the PKG information, and the ephemeral public key $EK_S = yP$, where $y \in \mathbb{Z}_q^*$ is the random ephemeral secret selected by the server.

With the identity information in the "identity_share" extension of the ClientHello, the server can calculate the shared secret with its private key sk_S through the *KeyAgreement* algorithm of the corresponding IBAKA. In this article, the server computes the shared secret via the following formula:

$$\text{shared\_secret} = yEK_C \| e\big(EK_C + H(\mathrm{ID}_C), yP_{\text{pub}} + sk_S\big).$$

As same as the key derivation schedule mechanism of TLS 1.3 [8], the server extracts the handshake secret from the shared secret through the HMAC-based key derivation function (HKDF) [41]

$$\text{handshake\_secret} = HMAC_{ek}(\text{shared\_secret})$$

where the Hash function used for HMAC is the cipher suite hash algorithm selected by the server, and the HMAC key ek is the key derived from the early key that is described in the next section. Then, the server can send EncryptedExtensions and Finished messages which are encrypted by the handshake traffic key derived from the handshake secret.

After receiving the ServerHello message, the client must verify that the PKG presented in the "identity_share" extension is the same as that in the "identity_share" extension of the original ClientHello message. With the server's identity and ephemeral public key offered in the ServerHello, the client can also compute the shared secret with its private key sk_C

$$\text{shared\_secret} = xEK_S \| e\big(xP_{\text{pub}} + sk_C, EK_S + H(\mathrm{ID}_S)\big).$$

By the bilinearity of the pairing, the client and the server can obtain the same secret

$$\begin{aligned}\text{shared\_secret}&\\ &= xEK_S \| e\big(xP_{\text{pub}} + sk_C, EK_S + H(\text{ID}_S)\big)\\ &= xyP \| e(xsP + sPK_C, yP + PK_S)\\ &= xyP \| e(xsP, yP)e(xsP, PK_S)e(sPK_C, yP)e(sPK_C, PK_S)\\ &= xyP \| e(P, P)^{xys}e(P, PK_S)^{xs}e(PK_C, P)^{ys}e(PK_C, PK_S)^{s}\\ &= yxP \| e(xP, ysP)e(xP, sPK_S)e(PK_C, ysP)e(PK_C, sPK_S)\\ &= yxP \| e(xP + PK_C, ysP + sPK_S)\\ &= yEK_C \| e\big(EK_C + H(\text{ID}_C), yP_{\text{pub}} + sk_S\big).\end{aligned}$$

The client also extracts the handshake secret from the shared secret using HKDF and responds with the Finished message protected by the handshake traffic key.

At this point, the client and the server complete the handshake. Both of them can derive the traffic keying material from the shared secret to protect the application-layer data through authenticated encryption with associated data (AEAD) functions [42]. It should be noted that iTLS eliminates the exchange of Certificate and CertificateVerify messages between the server and the client. They have completed implicit mutual authentication after only verifying the Finished messages (see Section IV).

The iTLS achieves the compatibility with the original protocol by having the ClientHello message carry the "identity_share" extension along with an empty "key_share" extension. If the server supports iTLS, the ServerHello message replied to the client only contains "identity_share" extension; if not, it ignores the "identity_share" extension contained in the ClientHello, and responds with a HelloRetryRequest message. Then, the client and the server fall back to a full TLS 1.3 handshake.

C. 0-RTT Model

The iTLS allows the client to send the application data in its first flight of messages ("early data") whenever establishing a secure connection with the server. Unlike TLS 1.3 that requires communication parties to share a static PSK in advance, iTLS instead generates the identity-based dynamic early key (IDEK) though IBAKA in each connection to protect the early data.

As shown in Fig. 3, the client includes an "early_data" extension in the ClientHello message to indicate that it will send the early data following the ClientHello message. Then, it calculates the IDEK through the IBNIKA using its private key and the server's identity as the input parameters

$$IDEK = HMAC_0(e(sk_C, H(\text{ID}_S)))$$

where the Hash function for HMAC is the hash algorithm defined in the first cipher suite in the client's cipher suites list, and the HMAC key is set to a string of zeros with the output length of the hash algorithm in bytes. The early traffic key derived from the IDEK is used to encrypt the early data through the AEAD algorithm.

After receiving the ClientHello message, the server extracts the client's identity from the "identity_share" extension in ClientHello and can also compute IDEK

$$IDEK = HMAC_0(e(H(\text{ID}_C), sk_S)).$$

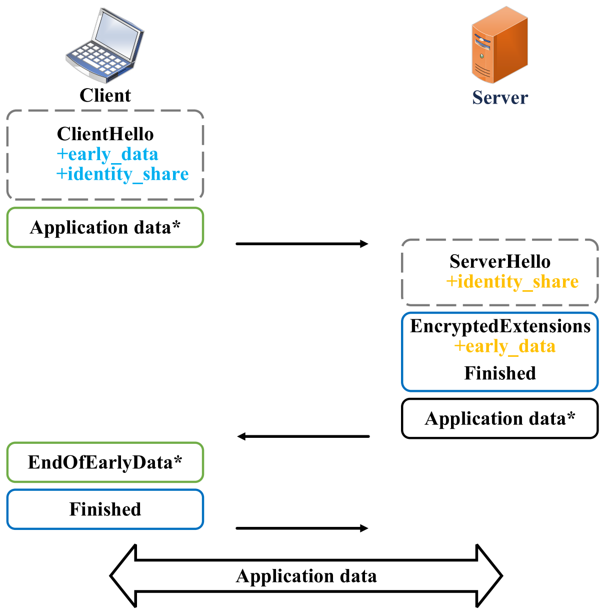


Fig. 3. iTLS 0-RTT handshake. "+" indicates extensions sent in the previously noted message and "*" indicates optional or situation-dependent messages that are not always sent. Green box indicates messages protected using keys derived from IDEK. Blue box indicates messages protected using keys derived from handshake_secret.

From the bilinearity of the pairing, we can get the equation

$$e(sk_C, H(\text{ID}_S)) = e(H(\text{ID}_C), H(\text{ID}_S))^s = e(H(\text{ID}_C), sk_S).$$

Thus, the server can accept and decrypt the early data with the IDEK. In order to accept the early data, the server must have selected the first symmetric cipher suite offered in the ClientHello's cipher suites field. It also returns its own "early_data" extension in the EncryptedExtensions message, indicating that it intends to process the early data.

After receiving the Finished message from the server, if the server has accepted the early data, the client needs to send an EndOfEarlyData message to the server before sending the Finished message. This message indicates that all the 0-RTT data have been transmitted. The rest of the handshake is the same as the iTLS 1-RTT handshake shown in Fig. 2. The client and the server establish the new session keys to encrypt the handshake messages and the normal application data by exchanging the "identity_share" extensions.

The IDEK described above provides protection analogous to TLS 1.3 with the PSK model. However, the security properties for the early data encrypted using this IDEK are weaker than those for other kinds of data (the handshake messages and application data). First, the early data are not the forward secret. If the long-term private key of the client or the server is compromised after completing the handshake, an attacker who has stored the early data in advance can compute the early secret and decrypt the early data. Second, the early data can be replayed arbitrarily. The attacker simply duplicates a flight of 0-RTT data, the server cannot determine whether it is a new connection or a replay.

The iTLS employs the ephemeral secret ticket to enhance the 0-RTT data's security. After establishing the connection successfully, the server sends the client a NewSessionTicket message that carries an "identity_share" extension and an "early_data" extension. The former extension contains an ephemeral public key offered by the server, which will be involved in establishing the IDEK in the subsequent connection. The server determines the ephemeral key according to the IBAKA algorithm that the current handshake uses. In this article, the server chooses a random secret r and calculates $[EK_S]_{ek} = rP$ as the ephemeral public key for establishing the early key. The latter extension is the same as the extension in TLS 1.3, which contains an integer to indicate the maximum amount of 0-RTT data that the client is allowed to send when using this ticket and ephemeral server key.

The server then stores the ephemeral secret r and the identity information of the client (the identity and PKG information) in a database with the ticket in the NewSessionTicket message as the lookup key. For the constrained server, it can directly use the encrypted and integrity-protected r and the client's identity information as the value of the ticket, thereby avoiding keeping per-client state.

After receiving the NewSessionTicket, the client caches the ticket and the associated ephemeral server key along with the server's identity information. The identity information contains the server's identity and PKG information, which is carried in the "identity_shared" extension in the NewSessionTicket message. Since the client has determined that the server registers with the same PKG based on the successful handshake, it only needs to cache a hash of the PKG information. In the next connection, the client can use this hash value to determine whether the PKG has updated the system parameters, so as to determine whether the ticket can still be used. Therefore, the ephemeral secret ticket introduces very little memory overhead for constrained devices.

When establishing a new connection with the same server subsequently, the client includes this ticket in the "early_data" extension within the ClientHello message. With the server's ephemeral public key $[EK_S]_{ek}$ and the ephemeral secret x randomly selected by the client, the client calculates the IDEK through the corresponding IBAKA

$$IDEK = HMAC_0\big(x[EK_S]_{ek} \| e\big(xP_{\text{pub}} + sk_C, [EK_S]_{ek} + PK_S\big)\big).$$

The client's ephemeral public key $EK_C = xP$ is sent to the server by carrying it in the "identity_share" extension of the ClientHello.

The server retrieves its ephemeral secret r from the received ticket. Combined with the client's identity information within the "identity_share" extension of the ClientHello, the server can also compute the IDEK

$$IDEK = HMAC_0\big(rEK_C \| e\big(EK_C + PK_C, rP_{\text{pub}} + sk_S\big)\big).$$

The ephemeral secrets offered by both sides of the communication guarantee that the IDEK provides perfect forward secrecy. In addition, the server can reject reduplicative tickets to mitigate the replay attack. After accepting the early data, the server erases the ephemeral secret from the local database or stores the value of the received ticket until the ticket expires and rejects duplicates during storage.

The new format of the "early_data" extension is shown as

```
struct {
    select (Handshake.msg_type) {
        case new_session_ticket:
                uint32 max_early_data_size;
        case client_hello: opaque ticket;
        case encrypted_extensions: Empty;
    }
} EarlyDataIndication.
```

IV. Security Analysis

In this section, we describe several important security properties the iTLS provides.

We consider an attacker to be an active network attacker, which means it has complete control over the communication channel between the communicating parties. Thus, the attacker can eavesdrop, replay, insert, delete, or modify all messages exchanged between the parties, and further, may attempt to impersonate either party to communicate with the other party. We also consider an attacker that has the capacity of tampering with the IoT devices to extract their long-term private keys. We do not consider the Denial-of-Service (DoS) attack since iTLS is a connection-oriented protocol. Nevertheless, the "cookie" extension supported by TLS 1.3 can also be used to provide a measure of DoS protection for the nonconnection-oriented version [8].

It is assumed that the PKG is trusted, secure, not compromise, and will not launch any active attacks. Since all the private keys are derived from the master key held by the PKG, an untrusted PKG or an attacker who compromises the master key can arbitrarily mount a successful MitM attack. We also assume that the AEAD cipher for symmetric encryption is secure, and the hard problems described in Section II cannot be solved with an efficient algorithm. In addition, the security of iTLS is relevant to the IBAKA protocol chosen to establish the shared key during the handshake. The formalized security proof of the IBAKA protocol chosen in this article can be found in [32].

End-to-End Security: Since iTLS has the same record layer as the TLS 1.3 protocol, all the application data transmitted through iTLS is divided up into a series records, and all the records are encrypted and integrity protected by AEAD with the shared key established during the handshake. Moreover, iTLS ensures that the shared key established can only be known by the communication parties. Thus, only the communication endpoints can encrypt and decrypt the data, and further, only the endpoints can authenticate the data transmitted from the peer. Any third party on the communication path (such as the gateway) or attacker that eavesdrops and manipulates the network cannot acquire the plaintext data, modify existing data, or inject false data. Thus, end-to-end security can be realized through iTLS.

Mutual Authentication: iTLS provides implicit mutual authentication without using certificates. During the handshake, after receiving and validating the Finished message from the server, the client can confirm that the server has computed the same shared secret as that calculated using the server's public key. Given the assumption that the PKG is trusted and well behaved (it will not impersonate the server), the client can confirm the server's authenticity since only the entity that possesses the server's long-term private key can calculate the same shared secret. Similarly, upon receiving and validating the Finished message sent by the client, the server can also verify the client's authenticity.

Perfect Forward Secrecy: This property ensures that compromising the long-term private keys of one or both communication parties cannot break the security of the session key established in the previous handshake, and therefore the security of the protected data transmitted beforehand. In iTLS, the calculation of the shared session key depends not only on the long-term private key of the participating parties but also on the randomly selected endpoints' ephemeral secrets. Even if an attacker compromises the private keys, he also needs to compute xyP from $EK_C = xP$ and $EK_S = yP$, which is a CDH problem and is independent of the long-term private keys. Thus, an attacker who knows the private keys cannot get previous session keys. In the 0-RTT mode of iTLS, the server sends the ephemeral public key in a previous connection through the ephemeral secret ticket. The client can generate the early key with forward secrecy by including the ticket value in the "early_data" extension in the ClientHello.

Uniqueness of the Session Keys: Since the endpoints' ephemeral private keys are randomly selected in each connection, any two distinct handshakes produce distinct and independent session keys. Even the early key and the ordinary session key produced by the same handshake are distinct because they are calculated from different ephemeral server keys. Therefore, even though one session key is compromised, the attacker cannot obtain any useful information on other session keys.

Key Compromise Impersonation Resistance: In a mutually authenticated connection, compromising the long-term private key of one participant should not break that participant's authentication of its peer. It means that an attacker who compromises a device's private key cannot impersonate arbitrary legitimate entities to establish a connection with that device successfully in subsequent handshakes. In iTLS, if the attacker compromises the client's long-term private key, he impersonates the server to establish a connection with the client. He knows the client's private key sk_C, the ephemeral public key EK_C sent by the client, and the ephemeral secret y' he chooses. To calculate the shared secret $y'EK_C||e(EK_C + PK_C, y'P_{\text{pub}} + sk_S)$, he must calculate $e(EK_C, sk_S) = e(P, PK_S)^{xs}$. However, the server's private key is not known, and calculating xsP from $EK_C = xP$ and $P_{\text{pub}} = sP$ is a CDH problem.

PKG Escrow: In contrast to classical public-key cryptography that the key generated by the user, the private key in IBC is generated by the PKG. PKG escrow indicates that the PKG can recover the session key for a completed handshake. For the iTLS protocol described in this article, even though the PKG knows the communication parties' private keys and the master secret s, it cannot recover the session key yet, since it cannot calculate xyp from xP and yP, which is a CDH problem. However, PKG escrow is desirable under certain circumstances, especially in certain closed group applications. In these situations, other IBAKA protocols that support PKG escrow can be implemented in iTLS [35].

Resistance to Man-in-the-Middle Attack: An attacker subverts the communication steam in order to masquerade as the server to the client and the client to the server. In iTLS, when an attacker receives respective EK_C and EK_S from the client and the server, he can construct arbitrary EK'_C and EK'_S to replace EK_C and EK_S, and send them to the server and the client, respectively. For example, the attacker can randomly select x' and y', and replace $EK_C = xP$ with $EK'_C = x'P$ and $EK_S = yP$ with $EK'_S = y'P$. Then, the shared secret the client computes is $xEK'_S||e(xP_{\text{pub}} + sk_C, EK'_S + PK_S) = y'xP||e(P_{\text{pub}}, y'xP)e(P, PK_S)^{xs}e(PK_C, y'sP)e(PK_C, PK_S)^s$. However, though the attacker knows xP and y', he cannot compute the shared secret without the server's private key. The attack can also substitute EK_C and EK_S with $EK'_C = x'P - PK_C$ and $EK'_S = y'P - PK_S$, respectively. Thus, he can calculate $e(xP_{\text{pub}} + sk_C, EK'_S + PK_S) = e(xP_{\text{pub}} + sk_C, y'P) = e(P_{\text{pub}}, y'EK_C)e(PK_C, y'P_{\text{pub}})$. However, he still cannot compute the shared key, since he needs to compute $xEK'_S = x(y'P - PK_S)$, which needs him get x from xP. This is the ECDL problem.

Resistance to Replay Attack: The keying material used for encrypting the ordinary 1-RTT data is determined by both the client and the server, thus providing replay protection for the 1-RTT data. However, the early traffic key does not depend on the ServerHello, and therefore the 0-RTT data are vulnerable to the replay attack [27]. An attacker could simply record the ClientHello message and the 0-RTT data and duplicates the flight to mount a replay attack. The iTLS prevents this type of attack by guaranteeing that the ticket associated with the ephemeral public key used for the early secret establishment is accepted at most once. Nevertheless, a network attacker can also take advantage of the client retry behavior to arrange for the server to receive multiple copies of an application message [9]. This type of attack cannot be prevented at the transport layer and must be dealt with by the application.

V. Implementation and Performance Evaluation

We implemented iTLS[1] based on the WolfSSL library,[2] which is a lightweight open-source TLS/SSL implementation targeting at IoT and currently supports the TLS 1.3 protocol. As we choose the pairing-based IBAKA protocol in iTLS in this article, we also introduced the PBC library to implement the pairing operations [43]. The PBC library is a free portable C library that allows the rapid prototyping of pairing-based cryptosystems. We also modified the demo server and client applications that are part of the WolfSSL to support the iTLS protocol for the experiment. Our implementation was tested and run on the Linux 4.10 kernel.

We experimentally evaluate the performance of iTLS, specifically on network traffic overhead and connection latency. The experimental results are shown by comparing the

[1]https://github.com/PengkunLi-nudt/iTLS

[2]wolfSSL. https://www.wolfssl.com/products/wolfssl/

TABLE I
TRAFFIC OVERHEAD COMPARISON AT 112-B SECURITY (BYTES)

	TLS 1.3			iTLS
	RSA Certificate	ECC Certificate	PSK	
Client	2630	1482	237	501
Server	2603	1454	147	423
Total	5233	2936	384	924

TABLE II
TRAFFIC OVERHEAD COMPARISON AT 128-B SECURITY (BYTES)

	TLS 1.3			iTLS
	RSA Certificate	ECC Certificate	PSK	
Client	3397	1529	237	627
Server	3370	1501	147	549
Total	6767	3030	384	1176

performance of iTLS and TLS 1.3. We configure TLS 1.3 to use PSK, RSA certificate, and elliptic curve cryptography (ECC) certificate, respectively. RSA certificate-based TLS uses RSA for authentication and DHE for key exchange, while ECC certificate-based TLS uses the elliptic curve digital signature algorithm (ECDSA) for authentication and ECDHE for key exchange. Both the client and the server are set to be authenticated in certificate-based TLS. We compare the performance of iTLS and TLS at the 112- and 128-b symmetric key security, respectively. In different security levels, the length of involved cryptographic keys is chosen according to the recommendations of NIST [44]. Finally, the cipher suites for both iTLS and TLS 1.3 select "TLS_AES_128_CCM_SHA256" for symmetric key and hashing operations.

Our experiment was run on two virtual machine instances within a computer with a 2.30-GHz Intel i5-6300HQ quad-core processor and 4-GB memory. We run the client application on one virtual machine and run the server application on the other one. Each virtual machine instance consumed a processor core and 1-GB memory.

A. Network Traffic Overhead

Our results from comparing network traffic generated by iTLS and TLS 1.3 are listed in Table I and Table II. These results provided indicate the number of bytes of the records generated during the full handshake, so these bytes do not contain the IP protocol header and the TCP header. Table I shows the results measured with 112-b security level, at which the RSA and ECC certificates used by TLS are 1322 and 814 B in size, respectively. Each certificate is accompanied by a minimal certificate chain of length 2. Since the client and server do not need to transmit Certificate and CertificateVerify messages, iTLS reduces the traffic overhead by 82.3% and 68.5% compared to TLS using RSA and ECC certificates, respectively. At the security level of 128-b (RSA certificate size is 1667 B and ECC certificate size is 839 B), the traffic overhead decreases by 82.6% and 61.2%, respectively (shown in Table II). It should be noted that large certificate messages have to be divided into multiple packets when transmitting over a link with limited packet sizes, such as IEEE 802.15.4. These small packets result in more network traffic overhead. Sending fewer bytes also means less energy consumption, which has significant implications for constrained devices. Therefore, iTLS is more suitable for the IoT scenarios than certificate-based TLS in terms of communication overhead and energy consumption. However, iTLS establishes a secure connection with more traffic overhead than the PSK-based TLS. These extra bytes are all introduced by the "identity_share" extension in the ClientHello and ServerHello messages.

B. Connection Latency

We evaluate the connection performance of iTLS with two kinds of experiments. The first experiment is designed to evaluate the full handshake latency for iTLS. In this experiment, the client establishes encrypted connections with the server using the basic full handshake of iTLS and TLS, respectively. The handshake latency is measured on the server side as the time from receiving the ClientHello message until the handshake is considered complete. We conduct 500 measurements for each type of handshake and compute the average handshake latency.

To study the impact of iTLS's 0-RTT model on connection latency, we design the second experiment in which the client sequentially connects to the server, sends a 28-B request, receives a 58-B response, and terminates the connection. The client establishes a secure connection with the server using iTLS's 0-RTT handshake, TLS's 0-RTT handshake with PSK, and TLS's 1-RTT handshake with certificates, respectively. We measure the number of such operations completed in 1 min for each type of connection.

We performed the experiments under a variety of network latencies, packet loss rates, and bandwidth using Linux's *netem* interface. In addition, all the measurements were performed on links with the maximum transmission unit (MTU) of 1500 (the Ethernet MTU) and 127 (the maximum frame size for IEEE 802.15.4), respectively. The MTU of 127 is adopted to simulate an IEEE 802.15.4 link for better evaluating iTLS performance on the IoT networks, such as wireless sensor networks.

1) Full Handshake Latency: Fig. 4 shows the comparison of the full handshake latency between iTLS and TLS 1.3 on a network with zero latency, no packet loss, and unlimited bandwidth. As we can see, in the ideal network environment, iTLS creates connections faster than TLS 1.3 that uses certificate-based authentication. At the security level of 112 b, the iTLS handshake is about three times faster than certificate-based TLS. At 128-b security, the performance of iTLS handshake increases by 78% and 60% compared to the RSA certificate-based TLS and the ECC certificate-based TLS, respectively. The handshake latency is mainly caused by the processing and transmitting of handshake messages. While the time to transmit messages is negligible in the ideal network environment, iTLS eliminates the processing and verification of certificates comparing with the certificate-based TLS. In addition, the average handshake latency for iTLS is higher than that for PSK-based TLS, which also does not need to exchange and process the Certificate and CertificateVerify messages. This is

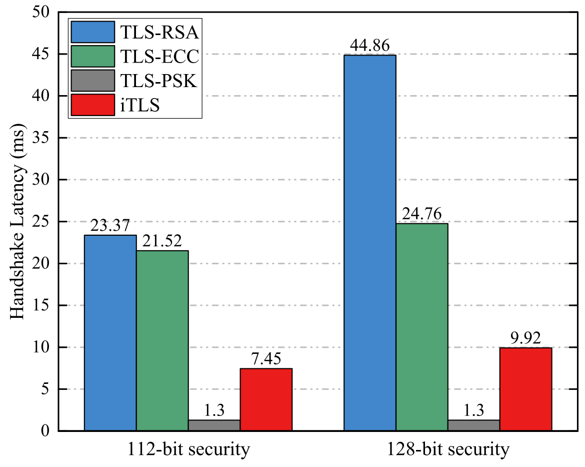


Fig. 4. Time spent completing a full handshake with different security in an ideal network environment. Data shown are measured on 1500 MTU link, data with MTU 127 are similar.

because iTLS needs to use the time-consuming pairing operation to calculate the shared secret during the handshake. In contrast, TLS uses the PSK directly as the shared key.

Fig. 5 displays the handshake latency for iTLS and TLS 1.3 according to the network latency. The average time each handshake takes is plotted in log scale. With increasing network latencies, the time to transmit messages gradually dominates during the handshake, so the average handshake latency for both iTLS and TLS is increasing. However, the handshake performed by iTLS always takes less time than the certificate-based TLS, especially on the link with MTU of 127. The handshake latency for iTLS is slightly higher than that for TLS with PSK mode, and these two latencies become equal as the network latency increases. In addition, the gap between the handshake latencies of certificate-based TLS on 127 MTU link and on 1500 MTU link enlarges as network latency increases, while the handshake latencies for iTLS and PSK-based TLS have slightly changed with different MTU. At the network latency of 256 ms, the handshake of iTLS, PSK-based TLS, RSA certificate-based TLS, and ECC certificate-based TLS performed on the link with MTU of 1500 takes 527.18, 514.10, 623.42, and 598.39 ms at 128-b security, respectively. With MTU of 127, the times are 530.58, 515.93, 3094.78, and 2083.92 ms, respectively. The reason is that the large certificate messages require to be divided into more fragments, and thus taking more time to transmit on the lower MTU link.

Fig. 6 shows the impact of packet loss on the handshake performance of iTLS and TLS 1.3. The time that iTLS performs a full handshake is always close to the time the PSK-based TLS takes to perform a handshake in a packet loss network environment. What is more, the handshake latency for the iTLS increases more slowly than that for the certificate-based TLS as the packet loss rate rises. For example, at 128-b security, when the packet loss rate reaches 17.5%, the iTLS, PSK-based TLS, RSA certificate-based TLS, and ECC certificate-based TLS handshake performed on the 1500 MTU link takes 304.41, 290.27, 876.63, and 769.41 ms, respectively. With MTU of 127, the gap between the handshake latencies of certificate-based TLS and iTLS is greater. The average handshake times are 658.74, 980.40, 12740.72, and 6787.71 ms, respectively. These results are attributed to the avoidance of the delivery of large certificate messages, which could result in many retransmissions for the lost packets. This indicates that the connection latency of iTLS that does not use certificates is less affected by the packet loss.

Fig. 7 displays the full handshake latency for iTLS and TLS on networks with different bandwidth. The iTLS achieves better connection performance than the certificate-based TLS in low-bandwidth networks because of the lower traffic overhead. With 64-kb/s bandwidth, the average handshake latencies for iTLS, PSK-based TLS, RSA certificate-based TLS, and ECC certificate-based TLS are respective 87.36, 68.75, 993.73, and 508.80 ms on 1500 MTU link, and respective 204.26, 87.82, 1713.03, and 1004.22 ms on 127 MTU link at 112-b security level. At the security level of 128 b, the times are, respectively, 81.70, 68.75, 1235.95, and 511.05 ms with MTU of 1500, and respective 183.47, 87.82, 2231.25, and 1281.36 ms with MTU of 127. In addition, when the bandwidth achieves 256 kb/s, the handshake latency for iTLS is close to the handshake latency on the ideal network. This indicates that iTLS is well suited for the low-bandwidth network.

2) 0-RTT Connection Performance: Fig. 8 shows the connection performance ratios between 0-RTT iTLS versus certificate-based 1-RTT TLS and 0-RTT iTLS versus PSK-based 0-RTT TLS. Each connection contains a TCP handshake, a handshake for secure channel establishment, and an application data exchange. On the link with MTU of 1500, the performance of the iTLS is at least 1.5 times efficient than that of the certificate-based TLS. This protocol performance ratio approaches the number predicted by counting the round trips inherent in each type of connection at high latencies. The 0-RTT model of iTLS facilitates the full connection process to complete in two round trips with one for TCP handshake, while certificate-based TLS requires three round trips to perform a full connection. With a lower MTU, the transmission of large certificates further lowers the connection performance of the certificate-based TLS, while iTLS is almost unaffected. Thus, as shown in Fig. 8(c) and (d), iTLS is 2.5–4.0 times faster than RSA certificate-based TLS, and about 2.0–3.5 times faster than ECC certificate-based TLS on 127 MTU link. Although PSK-based 0-RTT TLS also completes the data exchange in two round trips, the connection based on iTLS takes more time because of the IDEK calculation. However, the time to compute the early key is negligible with high network latencies. Thus, the performance ratio of iTLS versus PSK-based TLS gradually approaches 1.0 as the network latency increases.

VI. Related Work

The existing research results have demonstrated that ECC and pairing-based cryptography can be well suited for the resource-constrained devices [21], [45]. Many works apply IBC for key management in IoT and WSN. These schemes use IBC to set up the pairwise keys between nodes and provide security for end-to-end communication by using the symmetric keys to protect the data. Szczechowiak and Collier [46]

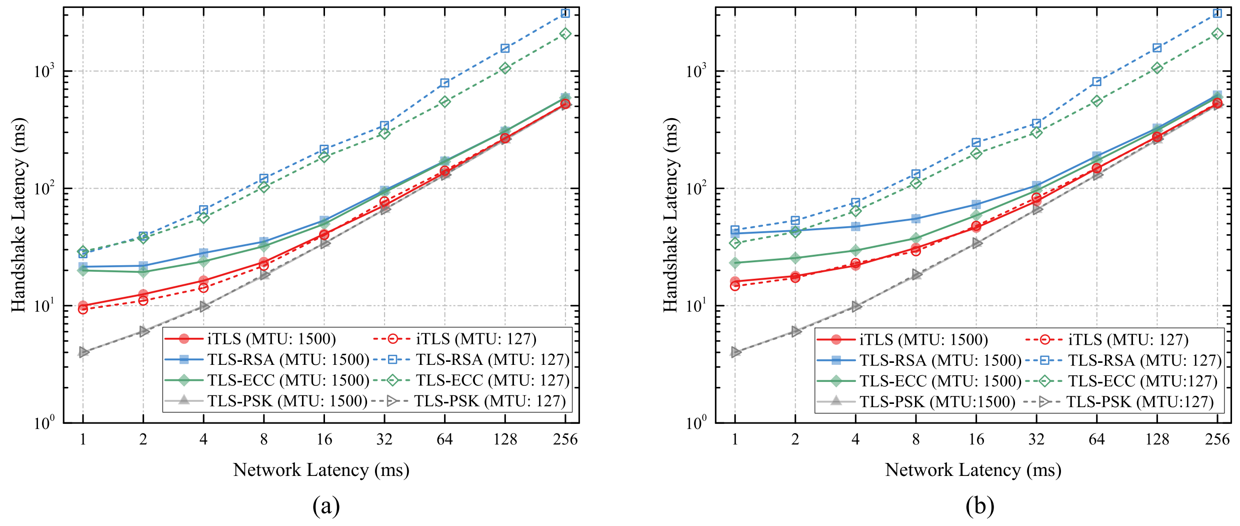


Fig. 5. Comparison of full handshake latency for iTLS and TLS 1.3 on networks with different latencies. (a) At 112-b security. (b) At 128-b security.

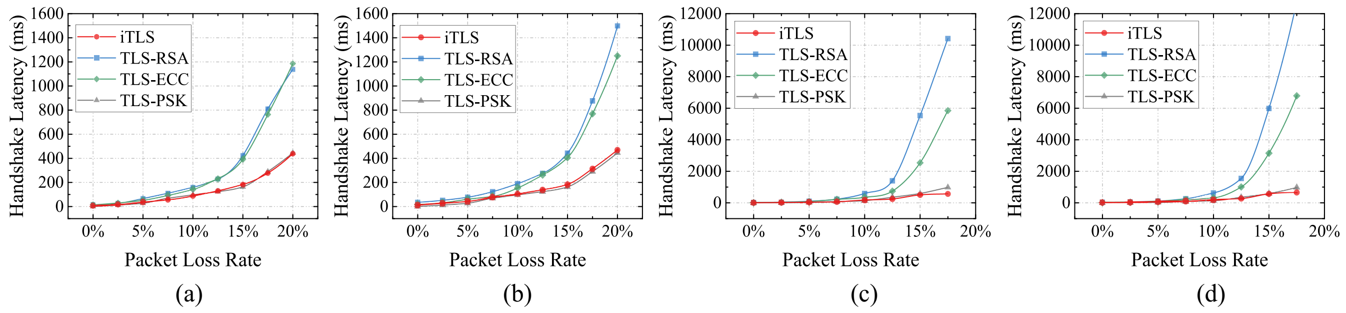


Fig. 6. Comparison of full handshake latency for iTLS and TLS 1.3 on networks with different packet loss rates. (a) With 112-b security and MTU of 1500. (b) With 128-b security and MTU of 1500. (c) With 112-b security and MTU of 127. (d) With 128-b security and MTU of 127.

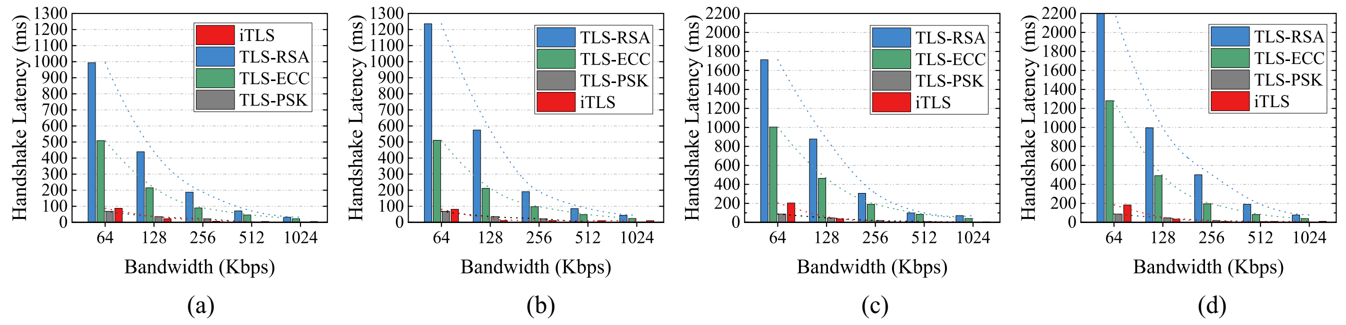


Fig. 7. Comparison of full handshake latency for iTLS and TLS 1.3 on networks with different bandwidth. (a) With 112-b security and MTU of 1500. (b) With 128-b security and MTU of 1500. (c) With 112-b security and MTU of 127. (d) With 128-b security and MTU of 127.

proposed TinyIBE, a simple authenticated key distribution mechanism using IBE for heterogeneous WSN. This solution provides only one-way authentication and no forward secrecy. Since pairing-based IDNIKA allows two nodes to establish a shared secret without exchanging any messages, Oliveira *et al.* [21] used IDNIKA to bootstrap the security in WSN and proposed TinyPBC, an efficient implementation of pairing primitives for resource-constrained sensor nodes. Boujelben *et al.* [22] also proposed a key management scheme for heterogeneous WSN based on IDNIKA. However, the IDNIKA protocol cannot defend against the key compromise impersonation attack or provide forward secrecy. Cakulev *et al.* [25] combined the IBE and ECDH and proposed an identity-based authenticated key exchange scheme that does not suffer from the key escrow problem. This scheme encrypts the exchanged ECDH parameters using IBE to establish the authenticated symmetric key, thus providing forward secrecy. Qin *et al.* [47] introduced the Bloom filter into the identity-based key management scheme to reduce the memory overhead of the sensor node for storing other nodes' identities and public keys. Aiming at better efficiency, some pairing-free authenticated key agreement protocols are proposed for IoT [37]–[39], [48]. To tackle the key escrow issue, there are also a few identity-based escrow-less authentication schemes

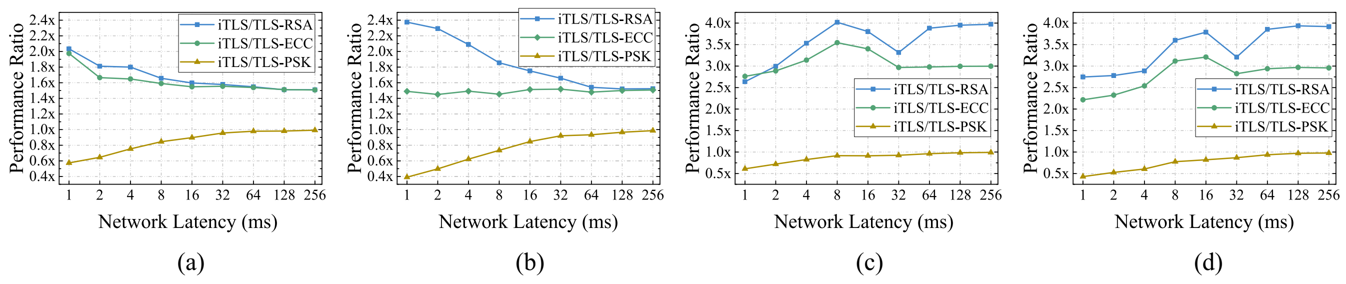


Fig. 8. Connection performance ratios between iTLS with 0-RTT model and TLS 1.3. (a) With 112-b security and MTU of 1500. (b) With 128-b security and MTU of 1500. (c) With 112-b security and MTU of 127. (d) With 128-b security and MTU of 127.

for IoT by adopting the certificate-less paradigm [49]–[54]. However, many of these schemes are accompanied by complex processing and high latency.

Some research works embed IBC into the network layer to provide end-to-end authentication and secure communication for the IoT. Markmann *et al.* [23] proposed a federated end-to-end authentication mechanism for constrained devices based on IBC. The entire IPv6 address of a device is used as its identity, and the powerful border gateway of each subnetwork acts as the TA to assigns IP addresses and the corresponding private keys to devices in this domain. The messages transmitted are authenticated by signing them with a specific IBS algorithm. To enable end-to-end authentication between nodes from different subnetworks, the subnet-ID equal to the cryptographic hash of the TA public system parameters is embedded into the IPv6 address. The embedded hash binds the TA to the subnetwork, preventing the replacement of the public parameters without changing the identity as part of a man-in-the-middle attack. However, this authentication mechanism does not provide confidentiality nor perfect forward secrecy, thus an ECDH key exchange is needed to establish a symmetric-shared secret between two parties.

Wang *et al.* [19] proposed the self-trustworthy and secure Internet protocol (T-IP) for authenticating and encrypting network-layer communications. T-IP also uses the IP address as the public key and assigns the corresponding IBC-based private key to the host, thus providing a self-trustworthy IP address. Based on the T-IP address, two hosts calculate the symmetric session key through the IDNAKA protocol to encrypt the communication. The self-trustworthy also provides the secure feature of source accountability and prevents the spoofing source address. Therefore, T-IP can achieve an authenticated and secure data transmission without spending time on credential-based authentication or key negotiation. However, T-IP cannot provide forward secrecy, since it adopts IDNAKA to establish the session key. In addition, the mechanisms operating at the IP layer generally must be implemented in the operating system kernel.

Similar to our work, some proposals integrate IBC into the TLS protocol to accommodate the constrained nature of IoT and WSN. Mzid *et al.* [20] proposed two modified TLS handshake protocols based on IBC for IP-based WSN. They incorporate IBC for authentication to reduce the management overhead of certificate-based solutions in the traditional TLS and improve the TLS handshake performance in terms of latency and energy consumption. The first proposed TLS handshake still uses ECDH for key exchange but uses the IBS algorithm to sign the ECDH ephemeral public key for authentication. This handshake protocol skips the transfer of certificates, thus reduces the number of messages exchanged during the handshake. The second proposal goes further by using the IBNIKA protocol instead of ECDH to establish the shared key. Bilinear pairing allows two nodes to establish the shared session key without any interaction. This further reduces the number of handshake messages sent. However, the session key established based on the pairing does not satisfy the forward secrecy property. Both handshake protocols are designed based on TLS 1.2, thus the secure connection is established with two round trips. In addition, the design of these two protocols does not consider compatibility with the original protocol.

Wang *et al.* [24] extended the raw public-key mechanism to support the IBS algorithm. The raw public key is used for authentication in TLS/DTLS to simplify certificate exchange. With the raw public-key mechanism, only entities' public keys are exchanged during the TLS/DTLS handshake, instead of transmitting certificates or the full certificate chains. However, the binding between the public key and the entity presenting the key is established through an out-of-band mechanism, which might be challenging in the IoT networks. Wang *et al.* used an IBC-based public key as the raw public key to simplify this binding. To support IBS algorithms, the entity's identity as the raw public key is carried by the certificate message along with the corresponding IBS algorithm identifier and the PKG public system parameters. The CertificateVerify message contains a signature over the handshake context using the IBS algorithm. Since the TLS 1.3 handshake flow is not modified in this way, the shared session key is still established by the (EC)DHE key exchange, which also needs 1 round trip to complete the handshake. However, this mechanism does not support the 0-RTT model.

The existing works proposed solutions based on the standardized protocols to secure the end-to-end communication in the IP-based IoT. Raza *et al.* [55], [56] defined an IPsec extension and implemented compressed IPsec for 6LoWPAN to provide end-to-end secure communication between IP-based sensor networks and the Internet. Kivinen [57] proposed a minimal implementation of the Internet Key Exchange version 2 (IKEv2) protocol for constrained devices, which is a component of IPsec that performs mutual authentication and establishes security associations. However, this minimal IKEv2 can only interoperate with a full IKEv2 implementation using shared secret authentication. SSNAIL provides security for IP-based WSN based on ECC-enable SSL [58], which uses certificates for authentication. Kothmayr *et al.* [12] proposed an IoT security architecture based on the DTLS protocol using RSA. To support RSA in sensor networks, the solution employs TPM assistance on sensor nodes. The IoT entities perform a fully authenticated handshake by exchanging their X.509 certificates. Though this solution provides higher security, it is complex and expensive with respect to deploying a hardware accelerator to every sensor, especially for a large number of sensors. Raza *et al.* [13] proposed a 6LoWPN header compression mechanism for DTLS, which considerably reduces the number of transmitted bytes and thus reduces the energy consumption. However, the proposed solution does not propose backward compatibility with the actual DTLS standard.

Some solutions use the way of delegating to reduce overhead resulting from certificates in the resource-constrained IoT. Granjal *et al.* [14] proposed an architecture that supports end-to-end security at the transport layer with the DTLS handshake being mediated by a border router. The border router uses certificates for authentication to communicate with the Internet host and negotiates the session with the constrained device through the PSK security mode. All the expensive operations (ECC computation, key agreement, etc.) are delegated to the border router. Hummen *et al.* [15] proposed a similar approach. The full TLS/DTLS handshake is delegated to a rich-resource entity, e.g., the gateway or the device's owner. The session state established by the full handshake is transferred to the constrained device to help it resume the session. Santos *et al.* [16] proposed a DTLS-based security architecture for the IoT, which allows to delegate the authentication process to a trusted third-party device. However, the delegating method generally has intricate handshake produce and cannot provide high-level security.

VII. Conclusion

In this article, we have proposed iTLS, a lightweight TLS protocol for end-to-end secure communication in the IP-based IoT. Compared with the existing solutions, iTLS is more prominent in three aspects: 1) iTLS minimizes connection latency with the fact that it delivers data in the first flight of messages with the protection under the IDEK, and that it requires no transmission and verification of certificates by providing mutual authentication through IBAKA; 2) iTLS provides perfect forward secrecy for all protected data, including early data, by exchanging the ephemeral server keys in the previous connection; and 3) iTLS is easy to deploy because of its full compatibility with the standardized protocol. Security analysis performed on iTLS showed that the protocol can achieve enhanced end-to-end security. Performance analysis demonstrated the efficient connection performance of iTLS and its strong adaptability to the low-power and lossy IoT networks. In Future work, we prepare to implement its datagram-oriented version, iDTLS, based on the version 1.3 of the DTLS protocol [28]. We believe that iDTLS can achieve more significant performance improvement for DTLS since the retransmission mechanism of DTLS for lost handshake packets operates on all packets in a flight.

References

[1] L. Atzori, A. Iera, and G. Morabito, "Understanding the Internet of Things: Definition, potentials, and societal role of a fast evolving paradigm," *Ad Hoc Netw.*, vol. 56, pp. 122–140, Mar. 2017.

[2] T. Heer, O. Garcia-Morchon, R. Hummen, S. L. Keoh, S. S. Kumar, and K. Wehrle, "Security challenges in the IP-based Internet of Things," *Wireless Pers. Commun.*, vol. 61, no. 3, pp. 527–542, 2011.

[3] P. I. R. Grammatikis, P. G. Sarigiannidis, and I. D. Moscholios, "Securing the Internet of Things: Challenges, threats and solutions," *Internet Things*, vol. 5, pp. 41–70, Mar. 2019.

[4] G. Liu, W. Quan, N. Cheng, H. Zhang, and S. Yu, "Efficient DDoS attacks mitigation for stateful forwarding in Internet of Things," *J. Netw. Comput. Appl.*, vol. 130, pp. 1–13, Mar. 2019.

[5] M. Ammar, G. Russello, and B. Crispo, "Internet of Things: A survey on the security of IoT frameworks," *J. Inf. Security Appl.*, vol. 38, pp. 8–27, Feb. 2018.

[6] K. T. Nguyen, M. Laurent, and N. Oualha, "Survey on secure communication protocols for the Internet of Things," *Ad Hoc Netw.*, vol. 32, pp. 17–31, Sep. 2015.

[7] H. Tschofenig and T. Fossati, "Transport layer security (TLS)/datagram transport layer security (DTLS) profiles for the Internet of Things," IETF, RFC 7925, Jul. 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc7925.txt

[8] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," IETF, RFC 8446, Aug. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8446.txt

[9] C. MacCarthaigh. (May 2017). *Security Review of TLS1.3 0-RTT.* [Online]. Available: https://github.com/tlswg/tls13-spec/issues/1001

[10] R. Hummen, J. H. Ziegeldorf, H. Shafagh, S. Raza, and K. Wehrle, "Towards viable certificate-based authentication for the Internet of Things," in *Proc. 2nd ACM Workshop Hot Topics Wireless Netw. Security Privacy*, 2013, pp. 37–42.

[11] R. Hummen, J. Hiller, H. Wirtz, M. Henze, H. Shafagh, and K. Wehrle, "6LoWPAN fragmentation attacks and mitigation mechanisms," in *Proc. 6th ACM Conf. Security Privacy Wireless Mobile Netw.*, 2013, pp. 55–66.

[12] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle, "DTLS based security and two-way authentication for the Internet of Things," *Ad Hoc Netw.*, vol. 11, no. 8, pp. 2710–2723, 2013.

[13] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, "Lithe: Lightweight secure COAP for the Internet of Things," *IEEE Sens. J.*, vol. 13, no. 10, pp. 3711–3720, 2013.

[14] J. Granjal, E. Monteiro, and J. S. Silva, "End-to-end transport-layer security for Internet-integrated sensing applications with mutual and delegated ECC public-key authentication," in *Proc. IEEE IFIP Netw. Conf.*, 2013, pp. 1–9.

[15] R. Hummen, H. Shafagh, S. Raza, T. Voig, and K. Wehrle, "Delegation-based authentication and authorization for the IP-based Internet of Things," in *Proc. 11th Annu. IEEE Int. Conf. Sens. Commun. Netw. (SECON)*, 2014, pp. 284–292.

[16] G. L. dos Santos, V. T. Guimarães, G. da Cunha Rodrigues, L. Z. Granville, and L. M. R. Tarouco, "A DTLS-based security architecture for the Internet of Things," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, 2015, pp. 809–815.

[17] R. Sakai, K. Ohgishi, and M. Kasahara, "Cryptosystems based on pairing," in *Proc. Symp. Cryptography Inf. Security*, 2000, pp. 26–28.

[18] Y. Chen, Q. Huang, and Z. Zhang, "Sakai–Ohgishi–Kasahara identity-based non-interactive key exchange revisited and more," *Int. J. Inf. Security*, vol. 15, no. 1, pp. 15–33, 2016.

[19] X. Wang, H. Zhou, J. Su, B. Wang, Q. Xing, and P. Li, "T-IP: A self-trustworthy and secure Internet protocol," *China Commun.*, vol. 15, no. 2, pp. 1–14, 2018.

[20] R. Mzid, M. Boujelben, H. Youssef, and M. Abid, "Adapting TLS handshake protocol for heterogenous IP-based WSN using identity based cryptography," in *Proc. IEEE Int. Conf. Wireless Ubiquitous Syst.*, 2010, pp. 1–8.

[21] L. B. Oliveira *et al.*, "TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks," *Comput. Commun.*, vol. 34, no. 3, pp. 485–493, 2011.

[22] M. Boujelben, H. Youssef, R. Mzid, and M. Abid, “IKM—An identity based key management scheme for heterogeneous sensor networks,” *J. Commun.*, vol. 6, no. 2, pp. 185–197, 2011.

[23] T. Markmann, T. C. Schmidt, and M. Wählisch, “Federated end-to-end authentication for the constrained Internet of Things using IBC and ECC,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 603–604, 2015.

[24] H. Wang, Y. Yang, X. Kang, and Z. Cheng, “Using identity as raw public key in transport layer security (TLS) and datagram transport layer security (DTLS),” Internet Eng. Task Force, Fremont, CA, USA, Internet-Draft draft-wang-tls-raw-public-key-with-ibc-11, Oct. 2019. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-wang-tls-raw-public-key-with-ibc-11

[25] I. Broustis, V. Cakulev, and G. Sundaram, “IBAKE: Identity-based authenticated key exchange,” IETF, RFC 6539, Mar. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6539.txt

[26] P. Li, J. Su, and X. Wang, “ITLS/IDTLS: Lightweight end-to-end security protocol for IoT through minimal latency,” in *Proc. ACM SIGCOMM Conf. Posters Demos*, 2019, pp. 166–168.

[27] M. Fischlin and F. Günther, “Replay attacks on zero round-trip time: The case of the TLS 1.3 handshake candidates,” in *Proc. IEEE Eur. Symp. Security Privacy (EuroS&P)*, 2017, pp. 60–75.

[28] E. Rescorla, H. Tschofenig, and N. Modadugu, “The datagram transport layer security (DTLS) protocol version 1.3,” Internet Eng. Task Force, Fremont, CA, USA, Internet-Draft draft-ietf-tls-dtls13–34, Nov. 2019.

[29] Z. Shelby, K. Hartke, and C. Bormann, “The constrained application protocol (CoAP),” IETF, RFC 7252, Jun. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7252.txt

[30] A. Shamir, “Identity-based cryptosystems and signature schemes,” in *Proc. Workshop Theory Appl. Cryptograph. Techn.*, 1984, pp. 47–53.

[31] D. Boneh and M. K. Franklin, “Identity-based encryption from the weil pairing,” in *Proc. Annu. Int. Cryptol. Conf.*, 2001, pp. 213–229.

[32] L. Chen, Z. Cheng, and N. P. Smart, “Identity-based key agreement protocols from pairings,” *Int. J. Inf. Security*, vol. 6, no. 4, pp. 213–241, 2007.

[33] M. Hölbl, T. Welzer, and B. Brumen, “An improved two-party identity-based authenticated key agreement protocol using pairings,” *J. Comput. Syst. Sci.*, vol. 78, no. 1, pp. 142–150, 2012.

[34] Q. Yuan and S. Li, “A new efficient ID-based authenticated key agreement protocol,” in *Proc. IACR Cryptol. ePrint Archive*, vol. 2005, 2005, p. 309.

[35] S. Wang, Z. Cao, Z. Cheng, and K.-K. R. Choo, “Perfect forward secure identity-based authenticated key agreement protocol in the escrow mode,” *Sci. China F Inf. Sci.*, vol. 52, no. 8, pp. 1358–1370, 2009.

[36] X. Cao, W. Kou, and X. Du, “A pairing-free identity-based authenticated key agreement protocol with minimal message exchanges,” *Inf. Sci.*, vol. 180, no. 15, pp. 2895–2903, 2010.

[37] S. H. Islam and G. Biswas, “A pairing-free identity-based two-party authenticated key agreement protocol for secure and efficient communication,” *J. King Saud Univ. Comput. Inf. Sci.*, vol. 29, no. 1, pp. 63–73, 2017.

[38] S. Bala, G. Sharma, and A. K. Verma, “PF-ID-2PAKA: Pairing free identity-based two-party authenticated key agreement protocol for wireless sensor networks,” *Wireless Pers. Commun.*, vol. 87, no. 3, pp. 995–1012, 2016.

[39] R. Yasmin, E. Ritter, and G. Wang, “Provable security of a pairing-free one-pass authenticated key establishment protocol for wireless sensor networks,” *Int. J. Inf. Security*, vol. 13, no. 5, pp. 453–465, 2014.

[40] Trusted Computing Group. (Mar. 15, 2013). *TPM 2.0 Library Specification*. Accessed: Aug. 2019. [Online]. Available: https://trustedcomputinggroup.org/resource/tpm-library-specification/

[41] D. H. Krawczyk and P. Eronen, “HMAC-based extract-and-expand key derivation function (HKDF),” IETF, RFC 5869, May 2010. [Online]. Available: https://rfc-editor.org/rfc/rfc5869.txt

[42] D. McGrew, “An interface and algorithms for authenticated encryption,” IETF, RFC 5116, Jan. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5116.txt

[43] B. Lynn. (2019). *PBC Library: The Pairing-Based Cryptography Library*. [Online]. Available: https://crypto.stanford.edu/pbc/

[44] E. Barker, *Recommendation for Key Management Part 1: General (Revision 4)*, document NIST SP 1-156, NIST, Gaithersburg, MA, USA, Sep. 2015.

[45] P. Szczechowiak, A. Kargl, M. Scott, and M. Collier, “On the application of pairing based cryptography to wireless sensor networks,” in *Proc. 2nd ACM Conf. Wireless Netw. Security*, 2009, pp. 1–12.

[46] P. Szczechowiak and M. Collier, “TinyIBE: Identity-based encryption for heterogeneous sensor networks,” in *Proc. IEEE Int. Conf. Intell. Sensors Sensor Netw. Inf. Process. (ISSNIP)*, 2009, pp. 319–354.

[47] Z. Qin, X. Zhang, K. Feng, Q. Zhang, and J. Huang, “An efficient identity-based key management scheme for wireless sensor networks using the bloom filter,” *Sensors*, vol. 14, no. 10, pp. 17937–17951, 2014.

[48] J. Liang, Z. Qin, S. Xiao, L. Ou, and X. Lin, “Efficient and secure decision tree classification for cloud-assisted online diagnosis services,” *IEEE Trans. Depend. Secure Comput.*, early access, Jun. 14, 2019, doi: 10.1109/TDSC.2019.2922958.

[49] M. A. Simplicio, Jr., M. V. M. Silva, R. C. A. Alves, and T. K. C. Shibata, “Lightweight and escrow-less authenticated key agreement for the Internet of Things,” *Comput. Commun.*, vol. 98, pp. 43–51, Jan. 2017.

[50] C. Li, X. Zhang, H. Wang, and D. Li, “An enhanced secure identity-based certificateless public key authentication scheme for vehicular sensor networks,” *Sensors*, vol. 18, no. 1, p. 194, 2018.

[51] M. E. S. Saeed, Q.-Y. Liu, G. Tian, B. Gao, and F. Li, “AKAIoTs: Authenticated key agreement for Internet of Things,” *Wireless Netw.*, vol. 25, no. 6, pp. 3081–3101, 2019.

[52] S. Chen, M. Ma, and Z. Luo, “An authentication scheme with identity-based cryptography for M2M security in cyber-physical systems,” *Security Commun. Netw.*, vol. 9, no. 10, pp. 1146–1157, 2016.

[53] J. Su, D. Cao, B. Zhao, X. Wang, and I. You, “EPASS: An expressive attribute-based signature scheme with privacy and an unforgeability guarantee for the Internet of Things,” *Future Gener. Comput. Syst.*, vol. 33, pp. 11–18, Apr. 2014.

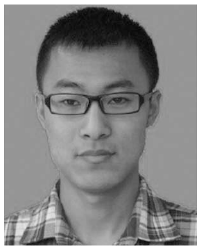
[54] L. Ou, Z. Qin, S. Liao, Y. Hong, and X. Jia, “Releasing correlated trajectories: Towards high utility and optimal differential privacy,” *IEEE Trans. Depend. Secure Comput.*, early access, Jul. 10, 2018, doi: 10.1109/TDSC.2018.2853105.

[55] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig, “Securing communication in 6LoWPAN with compressed IPSEC,” in *Proc. IEEE Int. Conf. Distrib. Comput. Sensor Syst. Workshops (DCOSS)*, 2011, pp. 1–8.

[56] S. Raza, S. Duquennoy, J. Höglund, U. Roedig, and T. Voigt, “Secure communication for the Internet of Things—A comparison of link-layer security and IPSEC for 6LoWPAN,” *Security Commun. Netw.*, vol. 7, no. 12, pp. 2654–2668, 2014.

[57] T. Kivinen, “Minimal Internet key exchange version 2 (IKEv2) initiator implementation,” IETF, RFC 7815, Mar. 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc7815.txt

[58] W. Jung, S. Hong, M. Ha, Y.-J. Kim, and D. Kim, “SSL-based lightweight security of IP-based wireless sensor networks,” in *Proc. Int. Conf. Adv. Inf. Netw. Appl. Workshops*, 2009, pp. 1112–1117.



Pengkun Li is currently pursuing the Ph.D. degree with the College of Computer, National University of Defense Technology, Changsha, China.

His research interests include Internet architecture and network security.

Jinshu Su (Senior Member, IEEE) received the B.S. degree in mathematics from Nankai University, Tianjin, China, in 1985, and the M.S. and Ph.D. degrees in computer science from the National University of Defense Technology, Changsha, China, in 1988 and 2000, respectively.

He is a Professor with the School of Computer Science, National University of Defense Technology, where he currently leads the Distributed Computing and High Performance Router Laboratory and the Computer Networks and Information Security Laboratory. He also leads the High Performance Computer Networks Laboratory, which is a Key Laboratory of Hunan, China. His research interests include Internet architecture and network security.

Xiaofeng Wang received the Ph.D. degree from the National University of Defense Technology, Changsha, China, in 2009.

His current research interests include trusted network, network security, and distributed intelligent data processing.