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

Pengkun Li, Jinshu Su, and Xiaofeng Wang

IEEE INTERNET OF THINGS JOURNAL, 2020

Chorom Hamm

crhamm@mmlab.snu.ac.kr

Nov 7, 2022

Outline

- Introduction
- Prerequisites: TLS 1.3 and Identity-based Cryptography
- System Design and Details
- Performance Evaluation
- Conclusion

Introduction: iTLS

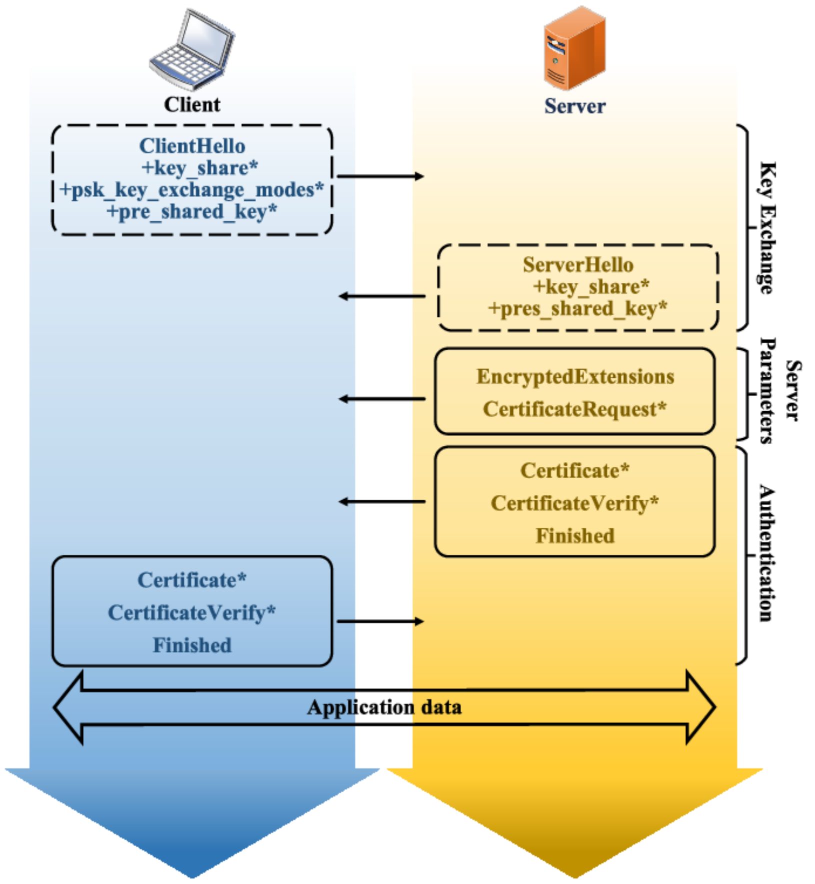
- End-to-end communication is crucial requirement in various IoT service domains *e.g., factory automation, medicine of healthcare, and smart city/home*
  - Secure communication is essential due to the sensitive data in IoT networks
  - Protocol should be lightweight because of resource-constrained IoT devices
- iTLS is the first lightweight secure transport protocol for IP-based IoT based on the identity-based key agreement protocol into TLS
  - It can deliver the protected data in the first handshake flight using the identity-based 0-RTT cryptographic handshake
  - It introduces the ephemeral secret ticket mechanism to provide perfect forward secrecy
  - It provides implicit mutual authentication without certificates

Standardized TLS Protocols

- *TLS with symmetric preshared key (PSK)*
  - \+ It consumes a small number of computational resources and bandwidth
  - − Key management including PSK generation and scalability issues exist
  - − The PSK established out of band is vulnerable to attacks due to the IoT devices' uncontrolled deployment environment and restricted security features
  - − Forward secrecy cannot be provided to data encrypted by the PSK with 0-RTT in TLS 1.3
- *TLS with public-key certificates*
  - \+ It can solve many challenges of PSK-based TLS
  - − There are overhead including long cert chain processing and revocation list checking

Prerequisite 1. TLS 1.3

- The handshake protocol
  - Negotiate cryptographic algorithms and parameters
  - Establish shared keying material
  - Authenticates the communicating parties
- Record protocol
  - Carry the handshake messages and application-layer data to be transmitted
  - Divide traffic up into a series of records
- TLS 1.3 reduces latency due to handshake and enhances security



Prerequisite 1. TLS Extensions

- *key_share*
  - To use (ED)DHE key establishment
  - ClientHello and ServerHello contain the client and server's Diffie-Hellman key shares
- *pre_shared_key*
  - To support PSK key establishment
  - It includes a set of PSK labels in ClientHello and the PSK identity in ServerHello
  - Both extensions can be contained when using (EC)DHE and PSK together
- *early_data*
  - To support 0-RTT model that sends the application data on the first flight
  - PSK is used to encrypt and decrypt data and authenticate each other during handshake

Prerequisite 2. Identity-based Cryptography

- Key generation
  - Public key: user's unique identifier
  - Private key: generated by private-key generator (PKG) using secret knowledge only possessed by the PKG
- The procedure of Identity-based authenticated key agreement (IBAKA)
  - $Setup(k) \rightarrow$ system public param SPP, master secret key msk
  - $KeyExtract(SPP, msk, ID_i) \rightarrow$ secret key sk_i
  - $KeyAgreement(SPP, sk_i, es_i, ID_p, EK_p) \rightarrow$ secret shk
- iTLS uses the pairing-based IBAKA algorithm because of high computational performance and the security properties

System Design and Key Concepts

- Identity-based Cryptography (IBC)
  - Establish an inherent binding between public key and entity presenting the public key
- Identity-based Authenticated Key Agreement (IBAKA)
  - Authenticate communication parties by establishing shared key without certificates
- Identity-based Dynamic Early Key (IDEK)
  - Generate the IDEK and encrypt data using IDEK for the first flight of 0-RTT model
- NewSessionTicket
  - Associate the ticket with an ephemeral server key to provide forward secrecy and replay protection

System Detail – Initialization

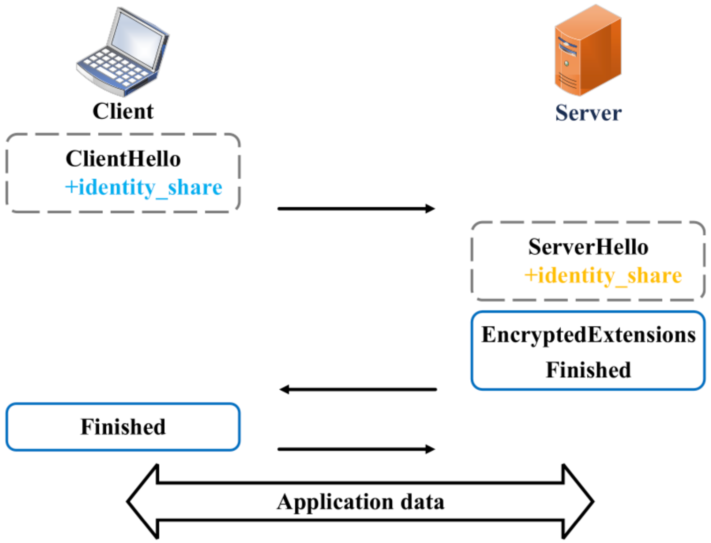
- Private Key Generator (PKG) initializes cryptographic system params and generate private keys
  - It stores the master secret key s and publishes the system parameters $SPP =< G, G_T, e, P, P_{pub}, H >$ by *Setup* algorithm
  - It adopts the *KeyExtract* algorithm to generate the corresponding private key

$$sk_i = sH(ID_i), \qquad PK_i = H(ID_i)$$

- The entities can fetch and update the public system parameters and private key based on the URI of the PKG
- The PKG protects the master secret using a security device like the trusted platform module (TPM) and update it periodically

1-RTT Handshake Protocol

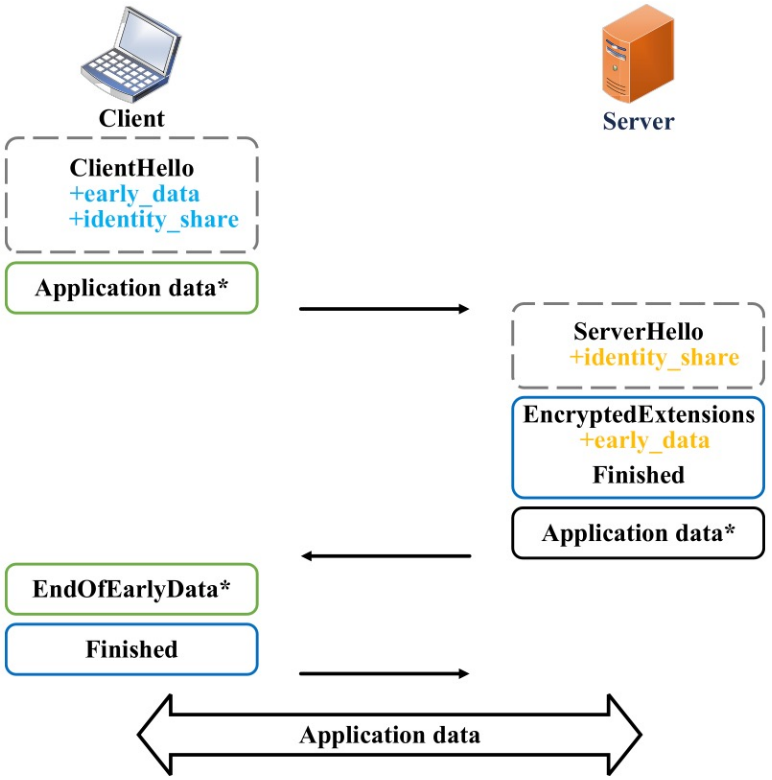
- *identity_share* extension exchanges identities and cryptographic params
- The *shared secret* is calculated with private key through *KeyAgreement* algorithm
- *Handshake secret* is extracted from *shared secret* through the HMAC-based key derivation function (HKDF)
- *EncryptedExtensions* and *Finished* messages are encrypted with *handshake traffic key*




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

0-RTT Handshake Protocol

- The identity-based dynamic early key (IDEK) is generated through IBAKA to protect early data
- Server extracts the client's identity from *identity_share* and compute IDEK to decrypt data
- An *EndOfEarlyData* message should be sent before transmitting the *Finished* message
- Rest of the handshake is the same as iTLS 1-RTT handshake



Forward Secrecy of 0-RTT Model

- The early data is not the forward secret and vulnerable to replay attack
- Server sends a *NewSessionTicket* message and stores the ephemeral secret and the identity of client in a database
- Client caches ticket and associated ephemeral server key with server's identity
- The ticket is included in the *early_data* extension within the *ClientHello* message
- The server can reject duplicate tickets to mitigate the replay attack

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

iTLS Security Analysis [1/2]

- *End-to-end security*
  - iTLS has the same record layer as the TLS 1.3 protocol
  - Only endpoints encrypt/decrypt data based on shared key established during handshake
- *Mutual authentication*
  - Client and server can authenticate each other by computing the shared secret using the Finished message
- *Perfect forward secrecy and Uniqueness of the session keys*
  - The shared session key should be computed by the long-term private key and randomly selected endpoints' ephemeral secrets
  - Session keys are unique and independent of private keys in each connection

iTLS Security Analysis [2/2]

- *Key compromise impersonation resistance*
  - Authentication cannot be broken with just one private key in a mutually authenticated connection
- *PKG escrow*
  - Session key cannot be recovered even though the PKG knows communication parties' private keys and master secret
- *Resistance to replay attack*
  - It guarantees that the ticket associated with the ephemeral public key used for the early secret establishment is accepted at most once

Performance Evaluation

- Implemetation (https://github.com/PengkunLi-nudt/iTLS)
  - WolfSSL library: a lightweight opensource TLS/SSL implementation
  - PBC library: a free portable C library for pairing-based cryptosystems
  - Comparing performance with TLS 1.3 using PSK, RSA certificate, ECC certificate as a baseline
- Evaluation metrics
  - Network traffic overhead: bytes of the records generated during the full handshake
  - Full handshake latency: time for one handshake process on the server side
  - 0-RTT model connection latency: the number of operations including iTLS's 0–RTT, TLS's 0–RTT with PSK, and TLS's 1-RTT handshake operations completed in 1 minute

Network Traffic Overhead

- Measurement based on two security level: 112-B and 128-B
- Large certificate messages in the case of TLS 1.3 with RSA and ECC makes communication overhead and energy consumption
- More overhead than the PSK-based TLS, but only due to “identity_share”

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

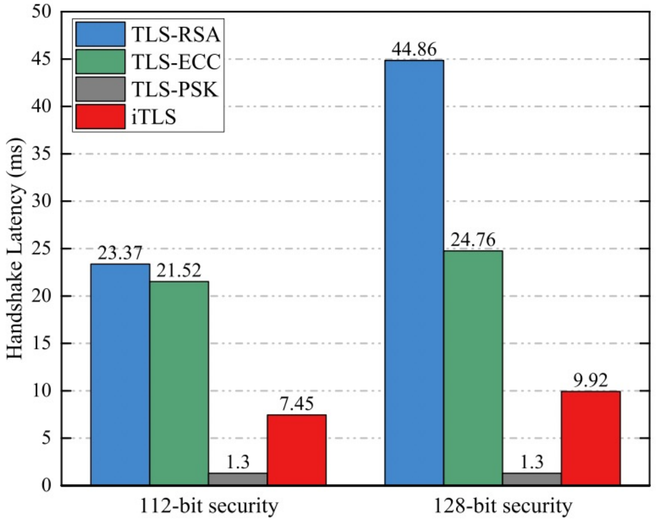
	TLS 1.3			iTLS
	RSA Certificate	ECC Certificate	PSK	
Client	2630	1482	237	501
Server	2603	1454	147	423
Total	5233	2936	384	924

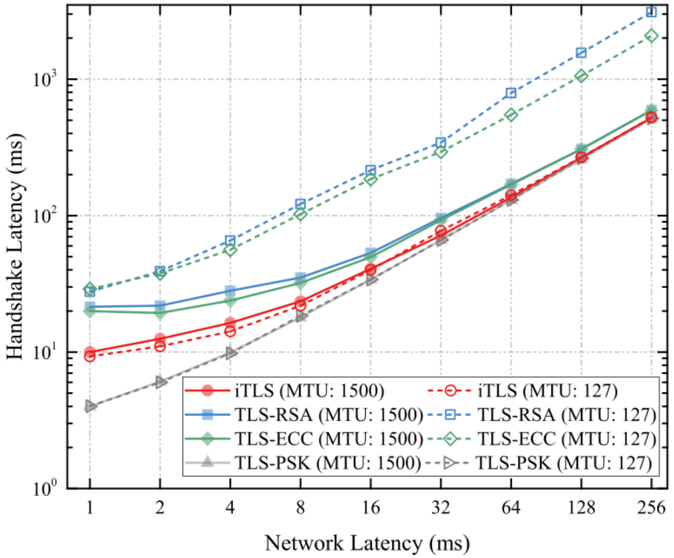
TABLE II
TRAFFIC OVERHEAD COMPARISON AT 128-B SECURITY (BYTES)

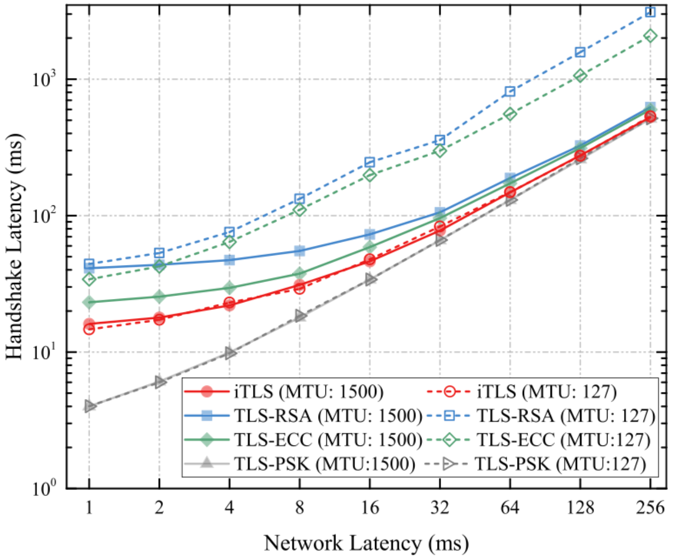
	TLS 1.3			iTLS
	RSA Certificate	ECC Certificate	PSK	
Client	3397	1529	237	627
Server	3370	1501	147	549
Total	6767	3030	384	1176

Full Handshake Latency

- Measurement on a network with zero latency, no packet loss, and unlimited bandwidth
- iTLS shows better performance due to no need of exchanging and processing Certificate and CertificateVerify messages

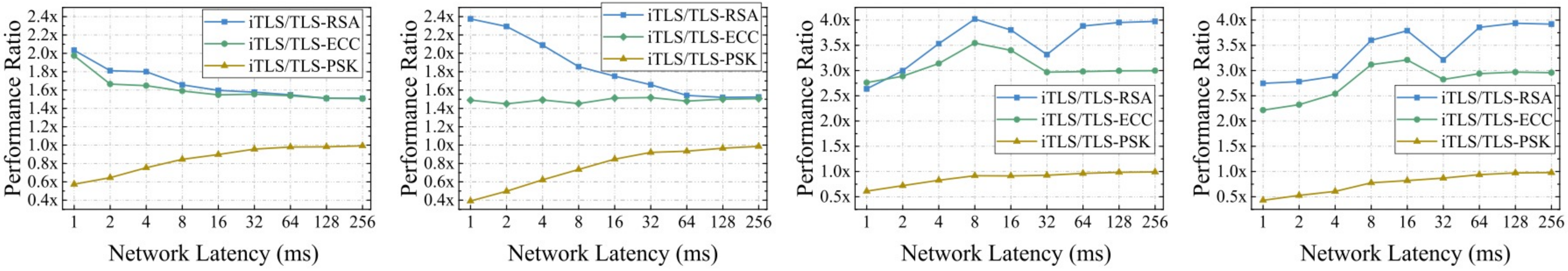






0-RTT Connection Latency

- Measurement of a TCP handshake, handshake for secure channel establishment, and an application data exchange
- The connection performance ratios approach to the number of the round trips at high latencies, showing better performance in iTLS
- IDEK calculation time should be added on iTLS, but negligible at high latencies



Conclusion

- The lightweight secure transport protocol for end-to-end communication is essential in many Internet-of-Things (IoT) application scenarios
  - Heavy overhead and security issues of Transport-layer security (TLS) and datagram TLS
- iTLS is the first lightweight secure transport protocol on the low-power and lossy IoT network environment
  - It delivers data in the first flight using IDEK with perfect forward secrecy
  - It provides implicit mutual authentication without certificates
  - It is fully compatible with TLS 1.3 using extensions like identity_share and early_data
- iTLS reduces at least 61.2% network traffic overhead and 60% latency

Thank you