TLS1.3へのIDベース認証鍵交換の適用と実装評価

木下　魁[1]　永井　彰[2]　鈴木　幸太郎[1]

概要：本稿では，最新の TLS プロトコルである TLS1.3 に対して，ID ベース認証鍵交換を適用する方式を提案する．提案方式による TLS1.3 は，ID ベース暗号のメリットである証明書の送受信が不要という特徴から，通信量が削減され，TLS ハンドシェイクを高速に行うことができ，特に IoT 分野に適している．実環境を想定した実装評価のために，提案方式に対して，富田らの ID ベース認証鍵交換を適用し，オープンソースソフトウェアの wolfSSL を元に実装後，インターネットを介する環境で通信量および処理時間を計測した．性能比較のために，通信量と暗号計算速度で優れている，X25519 鍵交換と Ed25519 署名を使用した PKI ベースの TLS1.3 に対して同様の計測を行った．結果，提案方式に富田らの ID ベース認証鍵交換を適用した TLS1.3 は，PKI ベースの TLS1.3 に比べ，通信量が約 34%に，相互認証に要する時間が約 58%に削減された．

キーワード：TLS1.3，ID ベース認証鍵交換，IoT

Application of ID-based Authenticated Key Exchange to TLS1.3 and Its Implementation

Kai Kinoshita[1]　Akira Nagai[2]　Koutarou Suzuki[1]

Abstract: In this paper, we propose a method that applies ID-based authentication key exchange to the latest TLS protocol, TLS1.3. The proposed method, TLS1.3, is characterized by the fact that it does not require sending and receiving certificates, which is a merit of ID-based encryption, and can reduce the amount of communication and perform TLS handshake at high speed, and is particularly suitable for the IoT field. For implementation evaluation assuming a real environment, ID-based authentication key exchange of Tomita et al. Is applied to the proposed method, and after implementation based on open source software wolfSSL, the amount of traffic and time required for mutual authentication are measured did. For performance comparison, we performed measurements similar to the proposed method for PKI-based TLS1.3 using X25519 key exchange and Ed25519 signature, which are excellent in traffic and encryption processing speed. As a result, compared with PKI-based TLS1.3, the proposed method reduced the amount of communication to about 34% and the time required for mutual authentication to about 58%.

Keywords: TLS1.3, IBAKE, IoT

1. はじめに

2018 年 8 月に RFC として TLS1.3 が正式にリリースされた．TLS1.3 は，暗号処理や通信の面でリソースの限られた IoT 機器での性能を意識し策定が進められ，前バージョンの TLS1.2 で安全性と速度にトレードオフの関係があったものを改善し，安全性を維持しつつ通信回数を削減することにより，通信遅延の影響が小さく，IoT 機器での

[1] 豊橋技術科学大学 情報セキュリティ研究室, 〒 441-8580 愛知県豊橋市天伯町雲雀ヶ丘 1-1
Information Security Lab., Toyohashi University of Technology, 1-1, Hibarigaoka, Tenpakucho, Toyohashi-shi, Aichi 441-8580, Japan
[2] 日本電信電話株式会社 NTT セキュアプラットフォーム研究所, 〒 180-8585 東京都武蔵野市緑町 3-9-11
NTT Secure Platform Laboratories, NTT Corporation, 3-9-11, Midoricho, Musashino-shi, Tokyo 180-8585, Japan

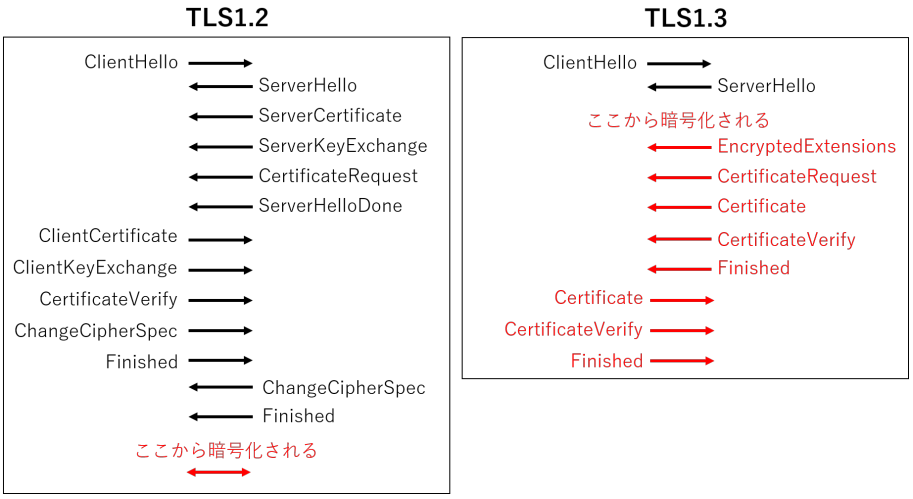

図 1 TLS1.2 と TLS1.3 の相互認証のフルハンドシェイクの違い

利用が期待されている．しかし，IoT 機器の中でも，特に LPWA(Low Power wide Area) などの環境においては通信性能にさらに制限がかかるため，通信遅延が想定され，処理時間が問題となり得る．これまでに，TLS1.2 に ID ベース暗号や ID ベース認証鍵交換を適用することにより，通信量が削減でき，処理時間を改善できることが酒見らの先行研究 [7,8] から示されている．しかし，図 1 を見ると分かるように，TLS1.3 は TLS1.2 と暗号化処理が開始されるタイミングが異なるため，酒見らの手法を TLS1.3 に適用することができるとは限らない。そのため，本稿では，TLS1.3 に対して ID ベース認証鍵交換を適用する方法を検討し，結果，新しい適用方法を提案する．また，実際の利用環境を想定した実装評価を行い，インターネット環境下では，TLS1.3 の最も速い暗号スイート（TLS1.3-X25519-Ed25519）よりも処理時間が短く，実用的な方法であることを確認した．本稿の構成を以下に示す．2 節で，TLS1.3 への ID ベース認証鍵交換の適用方法を説明する．3 節にて, 提案方式にの実装方法を説明し，4 節にて，実装の評価を行い，5 節にて，本稿をまとめる．

2. TLS1.3 への ID ベース認証鍵交換の適用

本節では，まず，TLS1.3 の TLS1.2 との仕様の違いに触れ，酒見らの提案方式を TLS1.3 にそのまま適用することが難しいことを示す．

酒見らの提案方式は, 1 ラウンドの鍵交換は ServerKeyExchange メッセージと ClientKeyExchange メッセージにより行い，1.5 ラウンド以上のラウンドを必要とする ID ベース暗号を適用する場合は ClientHello の TLS 拡張機能を使用して，ClientHello で ID を送信して，1.5 ラウンドの ID ベース認証鍵交換を実現していた．1.5 ラウンドを必要とする理由には，一時公開鍵の生成に通信相手の ID が必要であり，ID が未知である場合は，ID を一時公開鍵を生成する前に，どちらか一方が ID を送信する必要があるためである．相手の ID が既知である場合は，1 ラウンドで行うことができる．TLS1.2 で 1.5 ラウンドの ID ベース認証鍵交換が実現可能であった理由は，図 1 に示すとおり，TLS1.2 がハンドシェイクが暗号化されるまでに 2 ラウンドのやりとりが許されていたためである．しかし，TLS1.3 では，鍵交換がハンドシェイクの始めの ClientHello と ServerHello の 1 ラウンドにより鍵交換が行われてから，その後のハンドシェイクが共有鍵により暗号化される．そのため，その共有鍵に ID ベース認証鍵交換の共有鍵を使うためには，1 ラウンドの ID ベース認証鍵交換を適用するか，あるいは，TLS ハンドシェイクが開始される前に通信相手の ID が既知であるなどの前提の元で 1 ラウンドで認証鍵交換が完了する ID ベース認証鍵交換を適用する必要がある．次に，ClientHello と ServerHello の鍵交換は TLS1.3 で標準の (EC)DHE で行い，認証のために 1.5 ラウンド以上の ID ベース認証鍵交換を適用することを検討する．(EC)DHE により鍵交換を行うため，その後のハンドシェイクの暗号化と復号を行うことができるため，ClientHello と ServerHello に加えて，それ以降のハンドシェイクでも認証に必要な情報を送受信が可能となる．それ以降のメッセージには, EncryptedExtensions, CertificateRequest, Certificate, CertificateVerify，Finished があり，EncryptedExtensions と CertificateRequest，Certificate には拡張機能が付いており，認証に必要な情報をそれらのメッセージで送受信することが可能である．しかし，TLS1.3 では，アプリケーションデータの送受信が開始される前に，サーバがクライアントに対して自身が正当な通信相手であることを示すためには，Finished メッセージまでに限られる．しかし，図 2 を見て分かる通り，サーバの Finished を送信するまでに，サーバがクライアントから受信するメッセージは ClientHello のみである．相手の ID が未知である場合，クライアントは ClientHello で ID ベース認証鍵交換の一時公開鍵を送信することはできない．つまり，サーバは

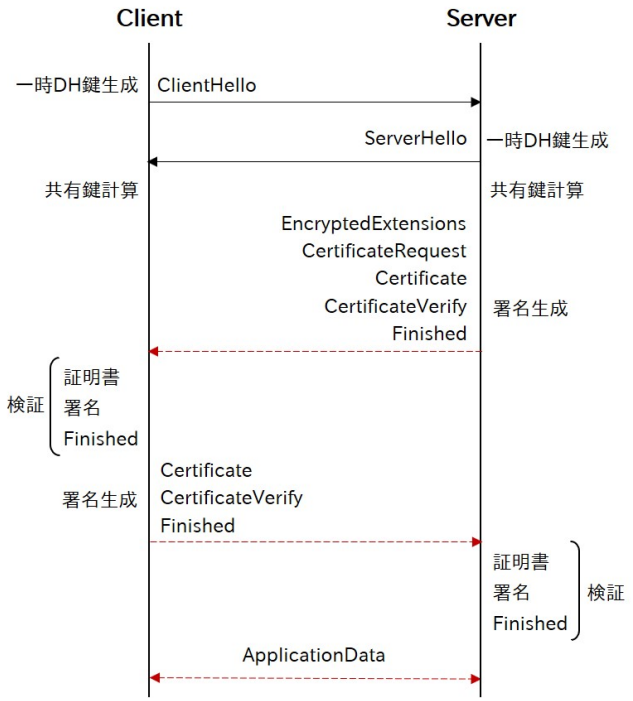

図 2 PKI ベースの TLS1.3 の相互認証のハンドシェイク

Finished メッセージを送信するまでに，ID ベース認証鍵交換の共有鍵を生成することが不可能であるため，クライアントに自信が正当なサーバであることを示すことができないため，相互認証を完了することができない．従って，1.5 ラウンド以上のラウンドを必要とする ID ベース認証鍵交換を TLS1.3 に適用する場合は，通信相手の ID が既知であるなどの前提の元で 1 ラウンドで鍵交換が完了する手法である必要がある．もしくは，通信を行う前に，通信相手の ID を確認する機構が必要となる．1 ラウンドの ID ベース認証鍵交換方式には，FSU [1–3] や岩井らの方式 [9] などがあり，1.5 ラウンドの ID ベース認証鍵交換方式には，富田らの方式 [4] や，modified-MB 方式 [8] が知られている．この中で最も計算効率がよいのが，富田らの方式であるため，実装評価においては富田らの方式を用いて評価することで，ID ベース認証鍵交換の TLS 適用における可能性を検証する．

次に，ID ベース認証鍵交換の TLS1.3 への適用方法を示す．

2.1 ID ベース認証鍵交換の適用方法

本節では，TLS1.3 への ID ベース認証鍵交換の適用方法について説明する．TLS1.3 に ID ベース認証鍵交換を適用するにあたって重要となる，一時公開鍵の送付方法と相互認証の手順について設計を行う。

2.1.1 一時公開鍵の送付方法

まず，一時公開鍵の送付方法について説明する．TLS1.3 における一時 DH 鍵は ClientHello と ServerHello 内の Extension(拡張領域) に格納され，メッセージと共に送信される．ClientHello と Extension を以下に示す．一時公開鍵を送信する場合は，extension_type には key_share であり，extension_data には，以下に示す，KeyShareClientHello の値が格納される．KeyShareClientHello には複数の KeyShareEntry が降順のクライアントの優先順位で格納される．KeyShareEntry を以下に示す．KeyShareEntry の group には鍵交換の識別子が，key_exchange には一時公開鍵が格納される．

```
struct {
    ProtocolVersion legacy_version =
                          0x0303; /* TLS v1.2 */
    Random random;
    opaque legacy_session_id<0..32>;
    CipherSuite cipher_suites<2..2^16-2>;
    opaque legacy_compression_methods<1..2^8-1>;
    Extension extensions<8..2^16-1>;
} ClientHello;

struct {
    ExtensionType extension_type;
    opaque extension_data<0..2^16-1>;
} Extension;

struct {
    KeyShareEntry client_shares<0..2^16-1>;
} KeyShareClientHello;

struct {
    NamedGroup group;
    opaque key_exchange<1..2^16-1>;
} KeyShareEntry;
```

secp256r1 を使用した ECDHE で非点圧縮の場合の KeyShareClientHello には，以下の client_share が格納される．group には鍵交換の識別子が，key_exchange には，secp256r1 上の楕円点の x 座標と y 座標を値として持つ UncompressedPointRepresentation を格納する．

```
KeyShareEntry client_share = {
  group = secp256r1;
  key_exchange = struct {
        uint8 legacy_form = 4;
        opaque X[coordinate_length];
        opaque Y[coordinate_length];
    } UncompressedPointRepresentation;
}
```

一時公開鍵を格納する key_exchange は最大 65535 バイトの領域が TLS1.3 の仕様により想定されており，ID ベース認証鍵交換の一時公開鍵の格納するにも十分な大きさである．つまり，TLS1.3 の仕様にある Extension:key_share を使用し，ID ベース認証鍵交換の一時公開鍵の送受信が可能である．従って，我々の提案方式では，ID ベース認証鍵交換の一時公開鍵の送付のために，特別な拡張は施さず，key_share を使用することとした．例として，FSU を適用した場合の，client_share の構造を示す．group には，FSU の識別子をセットし，key_exchange には，ID と楕円曲線上の 2 つの有理点が格納される．

```
KeyShareEntry client_share = {
  group = IBAKE_FSU;
  key_exchange = struct {
        opaque ID[ID_length];
        opaque X1[point_length];
        opaque X2[point_length];
    } IBAKE_FSU_EPK_Representation;
}
```

次に，相互認証の手順について説明する．

2.1.2 相互認証手順

次に，提案方式の相互認証手順を示す．図 3 に提案方式の相互認証のハンドシェイク図を示す．

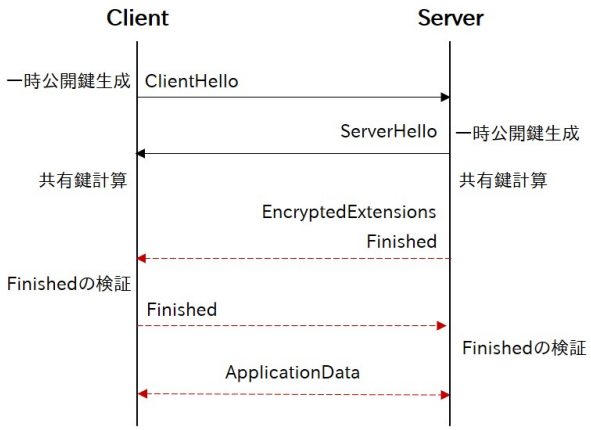

図 3　提案方式の TLS1.3 のハンドシェイク図

まず，クライアントとサーバは 2.1.1 節の一時公開鍵の送付方法に従って，一時公開鍵を生成し ClientHello と ServerHello を送受信する．クライアントとサーバはそれぞれ ServerHello と ClientHello の受信後に，共有鍵を生成し，ClientHello と ServerHello と共に鍵導出関数に入力し，以後のハンドシェイクを暗号化するための {sender}_handshake_traffic_secret を生成する．次に，サーバは EncryptedExtensions を生成し，暗号化したものをクライアントに送信する．次に，PKI ベースの相互認証では，クライアント認証を要求するために CertificateRequest を，証明書と署名の送信のために Certificate と CertificateVerify をクライアントに送信するが，提案方式では鍵共有に認証機能 (KGC より発行された，ID に対する正しい静的秘密鍵を所有している者だけが正当な共有鍵を生成できる) が付随しており，証明書と署名による認証が不要であるため，それらのメッセージは送信しない．Finished は TLS1.3 のハンドシェイクの正当性の検証のため送信する．クライアントはサーバから受信した EncryptedExtensions と Finished を復号し，処理を行う．EncryptedExtensions の復号に成功した場合，正当なサーバにより暗号化されていることになり，ID ベース認証鍵交換によるクライアントのサーバに対する認証に成功したこととなる．復号に失敗した場合，ハンドシェイクを終了する．成功した場合，続いて TLS1.3 のハンドシェイクの正当性の検証のため Finished の検証を行る．ハンドシェイクが正当であることが確認できれば，TLS1.3 のクライアントのサーバに対する認証が完了し，Finished メッセージを生成し，サーバに送信する．サーバはクライアントの Finished を受信し，復号化を行う．復号に成功した場合，正当なクライアントにより暗号化されていることとなり，ID ベース認証鍵交換によるサーバのクライアントに対する認証に成功したこととなる．続いて，Finished の検証を行い，ハンドシェイクの正当性が確認できれば，TLS1.3 のサーバのクライアントに対する認証が完了し，また，提案方式における相互認証が完了し ApplicationData の暗号化通信に移行する．

以上のようにして，提案方式は証明書を利用することなく TLS1.3 における相互認証を実現できる．

次に，実装方法について説明する．

3. ID ベース認証鍵交換を適用した TLS1.3 の実装方法

本節では，TLS1.3-IBAKE の実装方法について説明する．実装は汎用 PC 上で行い，TLS ライブラリ wolfSSL [6] に組み込むことにより実装を行う．wolfSSL を選択した理由は，TLS1.3 に対応しており，IoT 機器などの組み込み機器上でも動作させることが可能であることから，本稿の提案方式の将来的な適用先として想定している環境でも動作させることができるためである．OpenSSL も TLS1.3 には対応しているが，IoT 機器では動作させることができない．ここで，wolfSSL の暗号エンジンである wolfCrypt は，ID ベース暗号に必要となるペアリング計算には対応していないため，何らかの方法でペアリング計算ライブラリを用意する必要がある．本稿では，NTT セキュアプラットフォーム研究所より提供されたペアリングベース暗号計算ライブラリ (以後 NTT ライブラリと呼ぶ) を使用し，wolfSSL に TLS1.3-IBAKE を実装した．NTT ライブラリのペアリングのパラメータや各種演算方法については，[10] と同じであり，現在 128 ビット安全性を有する 462 ビットの BN 曲線上でのペアリングと楕円スカラー倍算を使用する．BN 曲線については，[5] を参照されたい．使用した PC 環境は，OS は Winsows10，統合開発環境として Visual Studio 2017 を用いた．図 4 に今回実装したソフトウェア構成図を示す．

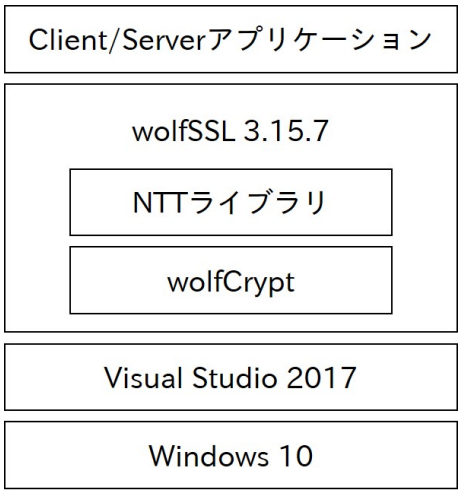

図 4　ソフトウェア構成図

3.1 一時公開鍵の生成と送付

IBAKE の一時公開鍵の生成に対する実装について説明する．wolfSSL の (EC)DHE の一時公開鍵の生成は TLSX_KeyShare_GenKey() 関数内で行われる．TLSX_KeyShare_GenKey() 関数は，サーバ側では，ハンドシェイクを行う wolfSSL_accept_TLSv13() 関数から TLSX_KeyShare_GenKey() 関数までは以下に示す木構造となっている．本稿での提案方式では，IBAKE の一時公開鍵を Extension:keyshare で送信するため，我々もこの関数内で IBAKE の一時公開鍵を生成することとした．

```
wolfSSL_accept_TLSv13()
  |
  +--TLSX_KeyShare_DeriveSecret()
  |    |
  |    +--TLSX_KeyShare_Process()
  |
  +--ProcessReply()
       |
       +--DoTls13HandShakeMsg()
            |
            +--DoTls13HandShakeMsgType()
                 |
                 +--DoTls13ClientHello()

DoTls13ClientHello()
  |
  +--MatchSuite()
       |
       +--VerifyServerSuite()
            |
            +--TLSX_KeyShare_Establish()
                 |
                 +--TLSX_KeyShare_GenKey()
```

TLSX_KeyShare_GenKey() 関数に IBAKE の一時公開鍵の生成を行えるように修正を行ったものを以下に示す．(EC)DHE の一時公開鍵は，Extension:key_share の key_exchange に対応する変数は，kse ->pubKey であり，そのため IBAKE の一時公開鍵もその変数に代入することで，一時公開鍵の送付の実装に関して，IBAKE のために特別な実装を行う必要はなく，TLS1.3 の仕様を変更する必要もない．これは，wolfSSL と互換性を維持させることが可能であることを意味しており，wolfSSL のバージョンが更新されても，ID ベース認証鍵交換を追加するだけで利用可能なため，実用性に優れている．特別な実装が不要であるということは，実用上重要なことであるため，我々は重要視している．

```
static int TLSX_KeyShare_GenKey(WOLFSSL *ssl,
  KeyShareEntry *kse)
{
    if (kse->group == IBAKE)             //Add
        return GenIBAKEKey(ssl,kse);     //Add
    if ((kse->group & NAMED_DH_MASK) ==
                          NAMED_DH_MASK)
        return TLSX_KeyShare_GenDhKey(ssl, kse);
    if (kse->group == WOLFSSL_ECC_X25519)
        return TLSX_KeyShare_GenX25519Key(ssl,
                                          kse);
    return TLSX_KeyShare_GenEccKey(ssl, kse);
}
```

3.2 一時公開鍵のパースと共有鍵の生成

一時公開鍵のパースと共有鍵の生成の実装について説明する．wolfSSL の (EC)DHE の一時公開鍵のパースと共有鍵の生成は TLSX_KeyShare_Process() 関数内で行われる．TLSX_KeyShare_Process() 関数は，サーバ側では，上に示す通り，wolfSSL_accept_TLSv13() 関数内の TLSX_KeyShare_DeriveSecret() 関数で実行される．本稿の提案方式では，一時公開鍵の送付と同様に IBAKE の一時公開鍵を Extension:keyshare で送信するため，パースも (EC)DHE と同じ箇所で行えば良く，我々もこの関数内で IBAKE の一時公開鍵のパースと共有鍵の生成を行うこととした．

TLSX_KeyShare_Process() 関数に IBAKE の一時公開鍵のパースと共有鍵の生成を行えるように修正を行ったものを以下に示す．(EC)DHE の共有鍵は，ssl ->arrays ->preMasterSecret に格納され，その後のハンドシェイクの暗号化に使用する sender_handshake_traffic_secret を生成する際の鍵導出関数の入力となる．我々の設計でも，IBAKE の共有鍵は同様に処理するため，IBAKE の共有鍵を ssl ->arrays ->preMasterSecret に代入することとした．

```
static int TLSX_KeyShare_Process(WOLFSSL* ssl,
                 KeyShareEntry* keyShareEntry)
{
    if (keyShareEntry->group == IBAKE)  // Add
        return TLSX_KeyShare_ProcessIBAKE(
                   ssl, keyShareEntry); // Add
    else if (keyShareEntry->group &
                              NAMED_DH_MASK)
        return TLSX_KeyShare_ProcessDh(
                   ssl, keyShareEntry);
    else if (keyShareEntry->group ==
                     WOLFSSL_ECC_X25519)
        return TLSX_KeyShare_ProcessX25519(
                   ssl, keyShareEntry);
    else
        return TLSX_KeyShare_ProcessEcc(
                   ssl, keyShareEntry);
}
```

3.3 証明書による認証処理のスキップ

次にサーバ側で実行する wolfSSL_accept_TLSv13() 関数で，SendTls13CertificateRequest() 関数，SendTls13Certificate() 関数, SendTls13CertificateVerify() 関数の実行部分をスキップするように変更した．クライアント側は CertificateRequest が届いた場合に限り Certificate と CertificateVerify を送信するため，wolfSSL_connect_TLSv13() 関数に対しては，一時公開鍵の送付と同様に実装上において修正を加える必要はない．

4. 評価

本節では， X25519 鍵交換と Ed25519 署名を使用した TLS1.3(以降, TLS1.3-X25519-Ed25519 と呼ぶ) と TLS1.3-IBAKE の通信量と処理時間を計測し比較する．本稿での X25519 鍵交換は Curve25519 という楕円曲線を使用した ECDHE であり，Ed25519 は Curve25519 を使用した DSA である．TLS1.3-X25519-Ed25519 は，PKI ベースの TLS1.3 の中で暗号計算速度と通信量に関して高性能の手法である．IBAKE には，富田らの手法 [4] を採用した．富

田らの手法は，id-eCK モデルにおいて安全な ID ベース認証交換方式であり，一回のハンドシェイクにつきペアリングの計算回数が一回で済むという特徴がある．しかし，一時公開鍵の生成には通信相手の ID を必要とするため，今回の評価の際は TLS ハンドシェイクを開始する前に事前にお互いの ID が既知であるものとして，実装評価を行った．また，参考として，secp256r1 という楕円曲線を使用した ECDHE である P256 鍵交換と RSA 署名を使用した TLS1.3(以後，TLS1.3-P256-RSA と呼ぶ) の計測も行う．表 1 に計測に使用したマシンの環境を示す．計測は，ローカル環境に加えて，実用面での評価を行うために地理的に離れた場所に位置するコンピュータとのインターネットを介した環境においても実施する．

表 1 計測に使用したマシンの環境

	PC1
CPU	Intel(R) Core(TM) i5-8250UU CPU 1.8GHz
コア数	4
RAM	8GByte
OS	Windows 10 (64 ビット)
地理的位置	愛知県
	PC2
CPU	Intel(R) Xeon(R) CPU E5-2650 v4 2.2GHz
コア数	4
RAM	4GByte
OS	Windows 10 (64 ビット)
地理的位置	東京都

4.1 通信量

通信量の計測には Wireshark と呼ばれるパケット解析ツールを使用し，各メッセージのパケットサイズを観測した．ここで，各メッセージを受信した際に送信される TCP の ACK メッセージは通信量として評価には入れない．表 2 に 3 つの手法の各メッセージのパケットサイズとそれらの合計を示す．通信量の合計を比較すると，TLS1.3-IBAKE は TLS1.3-X25519-Ed25519 に対して，約 34%の通信量となった．

4.2 処理時間

処理時間の計測はサーバで行い， TLS ハンドシェイクが開始される前の TCP3 ウェイハンドシェイクの直後から，クライアントから送信された Finished メッセージを受信し検証が成功するまでの間の実時間を計測する．実時間の計測には，Windows API により提供されている GetSystemTimePreciseAsFileTime 関数を使用する． GetSystemTimePreciseAsFileTime 関数は，1 マイクロ秒未満の分解能で実時間を取得する高精度の時刻計測関数である．表 3 に PC1 のローカル環境における計測結果を示す．インターネット機器を介さないため通信遅延は発生せず，暗号計算時間が支配的となる結果が示されている．計測の際には，クライアントとサーバでお互いの干渉を最小限にするために，別々のプロセスで実行している．TLS1.3-IBAKE は TLS1.3-X25519-Ed25519 と比べると約 8 倍の時間を要することが分かる．

表 4 に，PC1 をクライアント, PC2 をサーバとしインターネットを介した環境における処理時間の計測結果を示す．上に示す通り通信遅延の発生しないローカル環境における計測では，TLS1.3-IBAKE は TLS1.3-X25519-Ed25519 に対して，約 8 倍の処理時間を要していたが，インターネットを介する場合，TLS1.3-IBAKE は TLS1.3-X25519-Ed25519 と比べると，約 58%の処理時間となった．

5. まとめと今後の課題

本稿では，最新の TLS プロトコルである TLS1.3 に対して，ID ベース認証鍵交換を適用する方式を提案した．TLS1.3 は TLS1.2 から仕様が大きく変更されたため，先行研究である酒見らの ID ベース暗号や ID ベース認証鍵交換の TLS 適用手法をそのまま TLS1.3 に適用できないため，本稿により新たに方式を提案した．提案方式による TLS1.3 は，ID ベース暗号のメリットである証明書の送受信が不要という特徴から，通信量が削減され，TLS ハンドシェイクを高速に行うことができ，特に IoT 分野に適している．実環境を想定した実装評価のために，提案方式に対して富田らの ID ベース認証鍵交換を適用し，オープンソースソフトウェアの wolfSSL を元に実装後，通信量および処理時間を計測した．性能比較のために，通信量と暗号処理速度で優れている，X25519 鍵交換と Ed25519 署名を使用した PKI ベースの TLS1.3 に対して提案方式と同様の計測を行った．結果，提案方式に富田らの ID ベース認証鍵交換を適用した TLS1.3 は，PKI ベースの TLS1.3 に比べ，通信量が約 34%に，相互認証に要する時間が約 58%に削減された． 今回の実装評価では，楕円曲線上の点を一時公開鍵の一部としている手法に関しては，点の x 座標と y 座標のどちらも送信しているが，計算コストが必要となる代わりに x 座標だけを送信して，受信した側が y 座標を計算する方式も存在する．前述のような通信性能が処理時間の大部分を占めることが予想される環境においては，x 座標のみを送信する方式の方が処理時間の短縮に繋がることが期待できる．特に ZETA 環境では効果が期待できるため実装評価を行う予定である．

参考文献

[1] A. Fujioka, K. Suzuki, and B. Ustaoglu : “Ephemeral key leakage resilient and efficient ID-AKEs that can share identities, private and master keys,” Pairing 2010, pp.187-205, 2010.

[2] A. Fujioka, F. Hoshino, T. Kobayashi, K. Suzuki, B. Ustaoglu, and K. Yoneyama : “id-eCK Secure ID-Based

表 2　各メッセージのパケットサイズとその合計 (Byte)

	TLS1.3-IBAKE	TLS1.3-X25519-Ed25519	TLS1.3-P256-RSA(参考)
ClientHello	280	193	226
ServerHello	236	149	182
EncryptedExtensions	82	82	82
CertificateRequest	-	97	97
Certificate$^{(s)}$	-	687	1275
CertificateVerify$^{(s)}$	-	148	340
Finished$^{(s)}$	112	112	112
Certificate$^{(c)}$	-	693	1319
CertificateVerify$^{(c)}$	-	148	340
Finished$^{(c)}$	112	112	112
合計	822	2421	4085

$^{(s)}$ サーバ側の送信メッセージ

$^{(c)}$ クライアント側の送信メッセージ

表 3　PC1 のローカル環境における処理時間 (msec)

	処理時間
TLS1.3-IBAKE	11.34
TLS1.3-X25519-Ed25519	1.41
TLS1.3-P256-RSA(参考)	21.60

表 4　PC1-PC2 間の処理時間 (msec)

	処理時間
TLS1.3-IBAKE	94.70
TLS1.3-X25519-Ed25519	164.63
TLS1.3-P256-RSA(参考)	166.32

Authenticated Key Exchange on Symmetric and Asymmetric Pairing,”IEICE Trans. Fundamentals, Vol.E96-A No.6, pp.1139-1155, July 2013.

[3] ISO/IEC 11770-3:2014 Information technology - Security techniques - Key management - Part 3: Mechanisms using asymmetric techniques.

[4] J.Tomida, A. Fujioka, A. Nagai, and K, Suzuki : Strongly Secure Identiry-Based Key Exchange with Single Pairing Operation， ESORICS(2019).

[5] Paulo S. L. M. Barreto and Michael Naehrig: ”Pairing-Friendly Elliptic Curves of Prime Order,” In B. Preneel and S. Tavares, editors, *Selected Areas in Cryptography-SAC 2005.* volume 3897 of *Lecture Notes in Computer Science*, pages 319-331.Springer, 2006.

[6] wolfSSL: available from ⟨hhttps://www.wolfssl.com/⟩

[7] 酒見, 武仲, 金岡, :“ID ベース暗号による IoT 向け相互認証方式の提案,” SCIS2015, 2C2-1, IEICE, 2015.

[8] 酒見, 森川, 武仲, 落合, 金岡 :“ID ベース認証機能付鍵交換の TLS 適用とその実装評価,” SCIS2016, 3C3-3, IEICE, 2016.

[9] 岩井, 川口, 割木, 佐々木, 藤岡, 鈴木, 永井 :“非対称 Pairing を利用した ID ベース認証鍵交換: IoT 機器への適用,” SCIS2019, 3D3-3, IEICE, 2019.

[10] 永井 :“Cortex-M 上での BN 曲線におけるペアリングおよび群演算の実装と実用性の評価,” SCIS2019, 4B2-4, IEICE, 2019.