Computer Security Symposium 2018
22 - 25 October 2018

IoT 機器への適用に向けたTLS1.3の性能評価

小松 大河[1,a)]　松原 豊[1]　高田 広章[1]

概要：様々な IoT 機器の普及とともに，そのセキュリティ上の脅威や脆弱性が多数報告されている．IoT 機器による通信では，機器制御のための命令や，プライバシー情報など，機密性と完全性の保証が必要なデータがネットワーク上に流れるため，そのセキュリティが重要視される．一方で，IoT 機器は処理性能や動作環境に制限のある組込みシステムであるため，セキュリティ対策による通信時間や消費電力などのコスト増加を極力抑えることが求められる．本論文では，最新の TLS1.3 に準拠した，組込みシステム向け SSL/TLS ライブラリを対象に通信処理時間を測定し，現行の TLS1.2 と比較する．さらに，IoT 機器での利用を想定した場合の高速化手法について検討する．

キーワード：IoT 機器，TLS1.3，組込みセキュリティ

Performance Evaluation of TLS1.3 for IoT Devices

Taiga Komatsu[1,a)]　Yutaka Matsubara[1]　Hiroaki Takada[1]

Abstract: Along with the spread of various IoT devices, many security threats and vulnerabilities of IoT devices have been reported. In communication among IoT devices, data that require guarantee of confidentiality and integrity such as personal information and Instruction for device control , flows on the network, so it is necessary to enhance security. Meanwhile, since IoT devices are embedded systems that is limited in processing performance and operating environment, it is necessary to minimize cost increase such as communication time caused by security measure. In this paper, we measure communication processing time using the SSL/TLS library for embedded systems supporting TLS1.3 and compare it with TLS1.2. Furthermore, we consider the acceleration of TLS1.3 assuming use in IoT devices.

Keywords: IoT device, TLS1.3, Embedded system security

1. はじめに

近年，IoT 機器が急速に普及し，ネットワークに接続される組込みシステムが増加している．これに伴い，これらのシステムに対するセキュリティ上の脅威や脆弱性が多数報告され，脆弱なシステムを狙った攻撃も増加している．IoT 機器は医療機器や自動車，交通制御システム，インフラなど，人々の生命，生活に直接関わることに利用されるものが増えている [1]．このため，IoT 機器にセキュリティを確保することが必要不可欠となっている．IoT 機器のセキュリティには，一般的な情報セキュリティの問題に加え，「分散サービス拒否（DDoS）攻撃の踏み台にされるなど，機器の利用者自身には被害がないため，攻撃されていることに気づきにくい」，「機器のライフサイクルが長く，開発時の想定から製品の使用環境が変化する可能性がある」，「機器の動作が，直接現実の世界に影響を与えるため，攻撃によるシステムの誤動作が，社会の混乱に繋がる」，といった IoT 機器特有の脅威が存在する．

ネットワークに繋がるシステムを，これらの脅威から守り，セキュアな通信を行うための主な要件として，通信内容を秘匿する機密性（confidentiality），通信相手のなりすましを防ぐ真正性（authenticity），データの改ざんを防

[1] 名古屋大学 大学院 情報学研究科
Graduate School of Informatics, Nagoya University
[a)] taiga416@ertl.jp

© 2018 Information Processing Society of Japan

ぐ完全性（integrity）の 3 つが挙げられる．複数の要素技術を組み合わせて，これら 3 つの要件を満たす手段として，Secure Socket Layer / Transport Layer Security（以下 TLS）がある．TLS は，セキュアでない従来の TCP 接続上にセキュリティ層を実現し，HTTP や FTP など，様々なアプリケーションのデータを安全にやり取りするためのプロトコルであり，Google や Yahoo! が全ての通信を TLS 化することを推奨する [2], [3] など，汎用システムの世界では広く普及している．このため，クラウドや外部のサーバと通信する IoT 機器にも TLS を導入する動きが見られる．2018 年 3 月末には，最新の TLS1.3 が IETF に承認され，今後の普及が見込まれている．

一方で，組込みシステムは，価格やリアルタイム性などの面で様々な制約があるため，セキュリティ機能を高めた上で，セキュリティ確保のための実行オーバヘッドや，メモリ使用量，消費電力などを極力抑えることが求められる．TLS の実装による処理性能は，使用環境や通信で用いる暗号スイートによって変化する．TLS で使用される暗号暗号アルゴリズム単体について性能を評価したもの [4], [5], [6] や，TLS1.2 以前のバージョンについて評価した論文 [7], [8] が存在するが，TLS1.3 に対応したクライアントソフトウェアはまだ少なく，TLS1.3 について評価した論文はまだない．TLS1.3 のネットワーク遅延における性能評価した記事 [9] も存在するが，評価環境が明確でなく，ハンドシェイクの各処理に対する詳細な評価もされていない．また，TLS の処理オーバヘッドの削減方法として，共通鍵として用いられる AES 暗号や，公開鍵暗号である RSA 暗号などの使用される暗号化処理をハードウェア化するという方法が知られている [10] が，TLS の処理全体としてハードウェア化がどれほどの改善が見込めるのかについても評価はされていない．制限の厳しい機器を含めたあらゆる IoT 機器のセキュリティを低コスト，低オーバヘッドで実装するためには，実装における処理オーバヘッドの内容を詳細に分析する必要がある．

これらのことから，本論文では，TLS1.3 を IoT 機器に実装することを想定し，安全な通信を行うために必要な処理にかかる時間の構成を詳細に分析する．分析結果を基に，IoT 機器の使用環境を考慮した，処理オーバヘッドの削減方法を検討する．

具体的には，IoT 機器を模した組込みボードに TLS1.3 を用いた HTTPS 通信を行う簡易アプリケーションを用い，通信した際のクライアント側で行う処理にかかる時間を測定し，現行バージョンである TLS1.2 との比較や使用する暗号アルゴリズムを変更した場合の処理時間の比較，処理時間内訳の分析を行う．性能評価の結果と，IoT 機器を使用する際の環境特性を考慮し，その前提を基に高速化の可能性について検討する．

表 1 GR-PEACH の仕様

CPU	ARM Cortex-A9 400MHz RAM 10MB
Flash ROM	8MByte
Ethernet	100Mbps

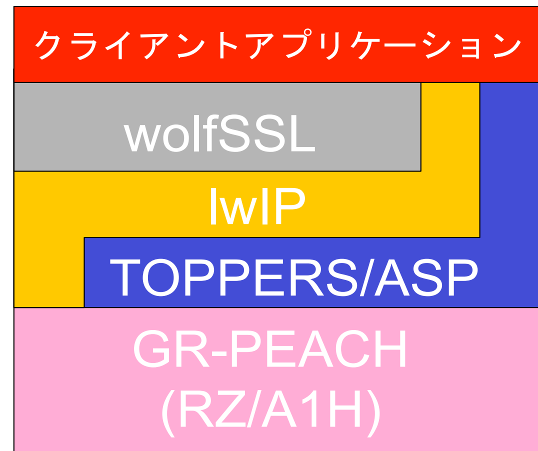

図 1 IoT 機器モデルの階層図

2. TLS1.3 の性能評価

2.1 評価環境

本節では，性能評価を行った環境について記す．IoT 機器のモデルには，組込みボード GR-PEACH，リアルタイム OS として，TOPPERS/ASP カーネル Release 1.9.2，組込みシステム向け TCP/IP スタックである lwIP のバージョン 1.4.0 を使用する．組込みシステム向け TLS ライブラリとして，オープンソースで提供されている wolfSSL のバージョン 3.14.0 を使用する．ソフトウェア環境は，GitHub 上のもの*1*2を使用している．GR-PEACH の仕様を**表 1** に，IoT 機器のモデルの階層図を**図 1** に示す．処理時間測定には，マイクロ秒の精度を持つ，TOPPERS/ASP カーネルの性能評価用システム時刻参照機能を使用する．測定区間の処理をそれぞれ 1000 回実行し，処理時間分布を取得する．

評価に用いたアプリケーションの動作は以下の通りである．最初に，クライアントが，接続先の内部/外部サーバに対して TCP 接続をした後，TLS ハンドシェイクを行い安全な通信路を確立する．通信路の確立後，クライアントがサーバへ HTTP GET リクエストを暗号化して送信，サーバからの応答メッセージを受信し，復号する．

ネットワーク接続環境を**図 2** に示す．GR-PEACH とネットワーク接続用 MacBook を Ethernet ケーブルで接続する．MacBook のインターネット共有機能を用いて外部の Web サーバと通信を行う．また，MacBook 内に TLS1.3 に対応した内部サーバを作成し，通信を行う．内部サーバ

*1 https://github.com/ncesnagoya/asp-gr_peach_gcc-mbed
*2 https://github.com/wolfSSL/wolfssl

© 2018 Information Processing Society of Japan

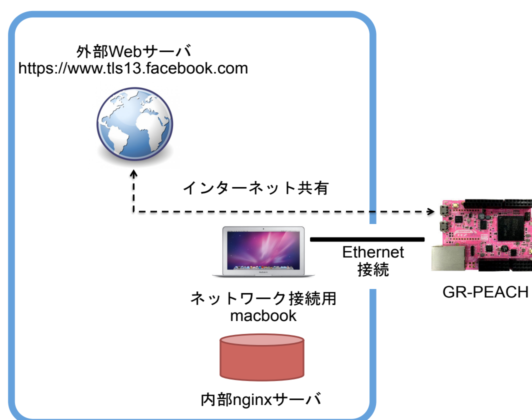

図 2　ネットワーク接続環境

として，TLS ライブラリに wolfSSL-3.14.0 を用いた nginx（バージョン 1.13.10）を，外部サーバとして，TLS1.3 に対応したサーバである https://www.tls13.facebook.com を使用する．nginx はそのままでは wolfSSL ライブラリを適用できないため，パッチを適用している．適用したパッチファイルと同じものが GitHub に公開されている[*3]．

2.2　測定内容

図 3 に TLS1.3 のフルハンドシェイクの流れを示す．TLS1.3 の性能評価として，このハンドシェイクと，アプリケーションデータの処理について，以下の 4 つの内容を測定する．

(1) TLS1.2 と TLS1.3 のハンドシェイク処理時間の比較
(2) 暗号アルゴリズムの違いによるハンドシェイク処理時間の比較
(3) 各ハンドシェイクメッセージのハンドシェイク処理時間内訳
(4) 共通鍵暗号アルゴリズムの違いによるアプリケーションデータの送受信時間の比較

まず最初に，TLS1.3 のハンドシェイク全体の処理時間を測定する．TLS1.3 では，ハンドシェイク手順に変更が加えられているため，比較対象として TLS1.2 のハンドシェイク処理時間を測定する．TLS1.3 は，暗号スイートに TLS_AES_128_GCM_SHA256 を，鍵交換アルゴリズムに一時的楕円曲線 Diffie-Hellman 鍵共有（ECDHE）を，サーバ認証アルゴリズムに RSA 暗号を使用する．TLS1.2 は暗号スイートに TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 を使用する．接続先サーバは，内部の nginx サーバの場合と外部の https://www.tls13.facebook.com の場合の 2 つを測定する．

TLS では，使用する暗号アルゴリズムの違いにより，処理時間性能に変化が生じる．このため，TLS1.3 で用いられる暗号アルゴリズムを変化させた際の処理時間を比較する．

[*3] https://github.com/wolfSSL/wolfssl-nginx

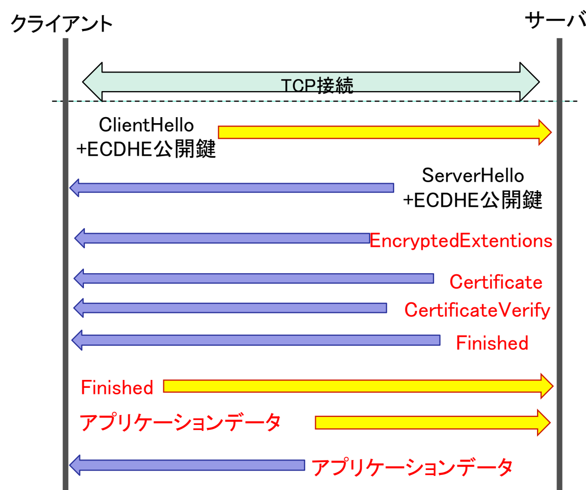

図 3　TLS1.3 のフルハンドシェイク

暗号スイートの比較として，TLS_AES_128_GCM_SHA256 と TLS_CHACHA20_POLY1305_SHA256 の 2 種類のハンドシェイク全体の処理時間を測定，比較する．どちらも鍵交換には ECDHE，サーバ認証には RSA を使用する．また，ECDHE 鍵共有で使用する楕円曲線の種類を変更した場合の，ハンドシェイク処理時間を測定，比較する．暗号スイートには，TLS_AES_128_GCM_SHA256 を用い，楕円曲線に，Curve25519，secp256r1 の 2 つを使用する．接続先サーバは，内部の nginx サーバである．

TLS1.3 のハンドシェイクにおける各メッセージに対するクライアント側の処理にかかる時間を測定する．暗号スイートに TLS_AES_128_GCM_SHA256 と TLS_CHACHA20_POLY1305_SHA256 を，ECDHE 鍵共有の楕円曲線に Curve25519，secp256r1 を用い，全 4 種類の組み合わせについて，ハンドシェイクにおける各メッセージの処理時間を測定し，平均値での処理時間内訳を求める．接続先サーバは，内部の nginx サーバである．

ハンドシェイクを行い通信路を確立した後の，アプリケーションデータの送受信にかかる時間を測定する．共通鍵暗号化方式として，AES-128-GCM と ChaCha20-Poly1305 を使用した場合の処理時間を比較する．wolfSSL で提供されているメッセージ送受信関数である wolfSSL_write 関数と wolfSSL_read 関数にかかる処理時間を測定する．接続先サーバは，内部の nginx サーバである．

3.　測定結果

3.1　TLS1.2 と TLS1.3 のハンドシェイク処理時間比較

TLS1.2 と TLS1.3 を用いて通信した場合のハンドシェイク全体の処理時間を比較した結果を**図 4**，**図 5** に示す．図 4 は内部の nginx サーバとの通信結果，図 5 は外部の https://www.tls13.facebook.com との通信結果である．図 4 より，内部での通信は，TLS1.2 が TLS1.3 に比べ 5 ミリ秒ほど処理時間が短いことがわかる．図 5 より，外部と

© 2018 Information Processing Society of Japan

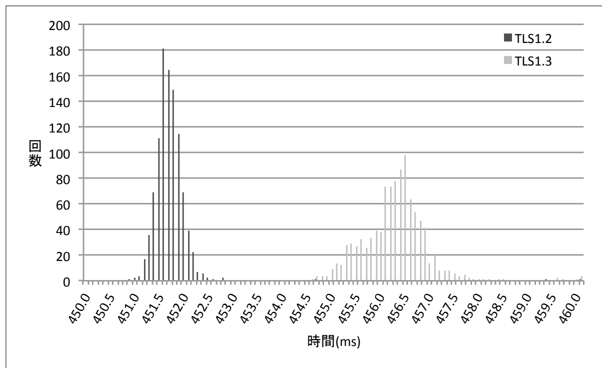

図 4　内部サーバと接続した際の
ハンドシェイク処理時間比較

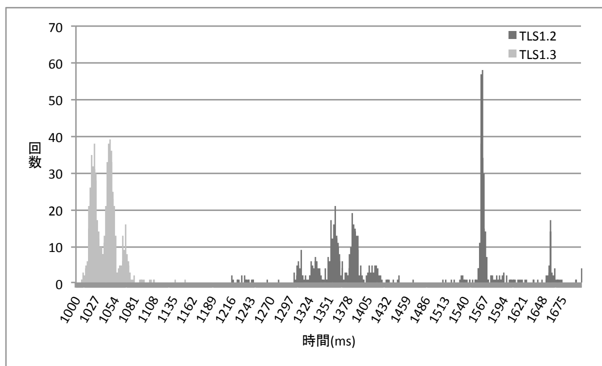

図 5　外部サーバと接続した際の
ハンドシェイク処理時間比較

の通信では，TLS1.2 と TLS1.3 のどちらも内部との通信に比べ処理時間が長くなり，実行ごとのばらつきも大きくなっている．TLS1.2 は TLS1.3 に比べ，ばらつきが大きく，処理時間が長いことがわかる．

3.2　暗号アルゴリズムの違いによるハンドシェイク処理時間の比較

図 6 に暗号スイートに TLS_AES_128_GCM_SHA256 と TLS_CHACHA20_POLY1305_SHA256 を使用した場合のハンドシェイク全体の処理時間を比較した結果を示す．2 つの暗号方式では，ChaCha20-Poly1305 を使用した方が 5 ミリ秒ほど処理時間が短いということがわかる．

図 7 に ECDHE 鍵交換で用いる楕円曲線に，secp256r1 を使用した場合と，Curve25519 を使用した場合のハンドシェイク処理時間を比較した結果を示す．楕円曲線の種類によって処理時間に大きな差があり，Curve25519 を使用すると，secp256r1 に比べ，処理時間が約 3 分の 1 になっていることがわかる．

3.3　TLS1.3 ハンドシェイクの処理時間内訳

図 8～**図 11** と**表 2** に TLS1.3 の各ハンドシェイクメッセージに対するクライアント側の処理時間の内訳を示す．使用する暗号アルゴリズムにより，メッセージ処理時間に

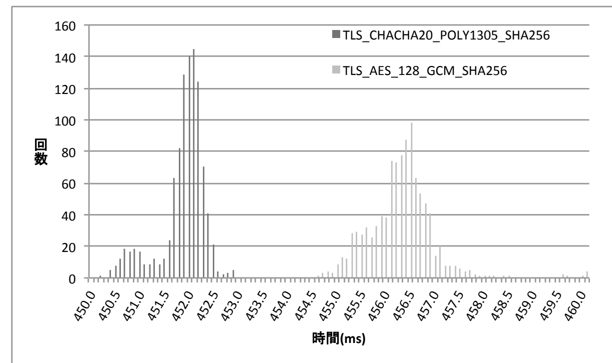

図 6　暗号スイートの違いによる
ハンドシェイク処理時間比較

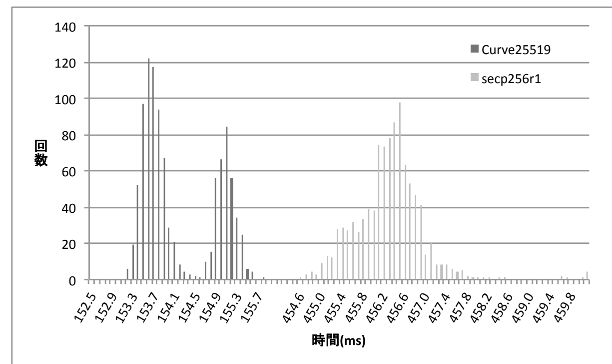

図 7　楕円曲線の違いによるハンドシェイク処理時間比較

差があるが，主に，ClientHello，ServerHello，Certificate，CertificateVerify メッセージの処理時間が大きな負荷になっていることがわかる．楕円曲線の種類で比較すると，Curve25519 を使用することにより，ClientHello，ServerHello メッセージの処理時間が大きく削減されていることがわかる．

暗号スイートで比較すると，TLS_CHACHA20_POLY1305_SHA256 を使用した場合に Certificate メッセージの処理時間が，5 ミリ秒ほど短くなることがわかる．CertificateVerify メッセージは，どの場合においても，処理時間にそれほど差がないことがわかる．

3.4　アプリケーションデータの送受信時間

図 12，**図 13** にハンドシェイク後のアプリケーションデータの送受信にかかる処理時間を測定した結果を示す．

送信処理にかかる時間は非常に短く，どちらの共通鍵暗号方式でも差はない．受信処理では，どちらの場合も送信よりも処理時間がかかっており，ChaCHa20-Poly1305 の方が約 1 ミリ秒処理時間が短いことがわかる．

4.　考察

4.1　TLS1.2 と TLS1.3 のハンドシェイク処理時間比較

図 4 より，内部サーバとの通信では，TLS1.2 のハンド

© 2018 Information Processing Society of Japan

表 2　各ハンドシェイクメッセージ平均処理時間

	TLS_AES_128_GCM_SHA256 secp256r1 （ms）	TLS_AES_128_GCM_SHA256 Curve25519 （ms）	TLS_CHACHA20_POLY1305_SHA256 secp256r1 （ms）	TLS_CHACHA20_POLY1305_SHA256 Curve25519 （ms）
ClientHello	172.69	23.15	172.50	22.77
ServerHello	176.55	25.81	176.15	25.05
EncryptedExtension	1.55	1.14	1.33	1.33
Certificate	69.40	67.81	64.86	62.82
CertificateVerify	30.16	31.10	29.60	31.27
Finished 受信	5.06	3.66	5.04	3.65
Finished 送信	2.02	2.00	1.97	1.92
合計	457.43	154.67	451.45	148.81

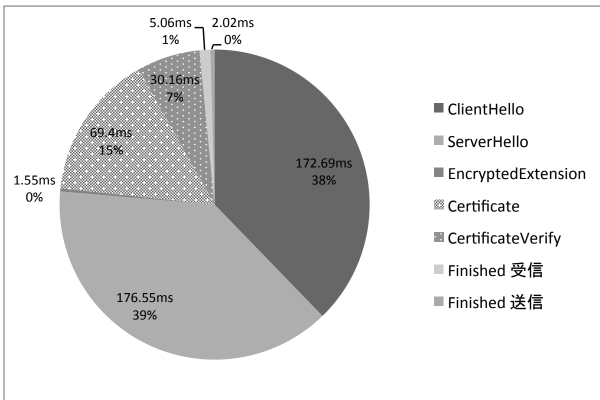

図 8　TLS_AES_128_GCM_SHA256，secp256r1 の ハンドシェイク処理時間内訳

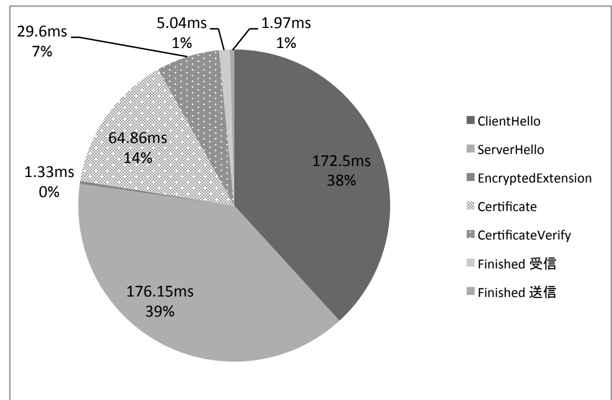

図 10　TLS_CHACHA20_POLY1305_SHA256，secp256r1 の ハンドシェイク処理時間内訳

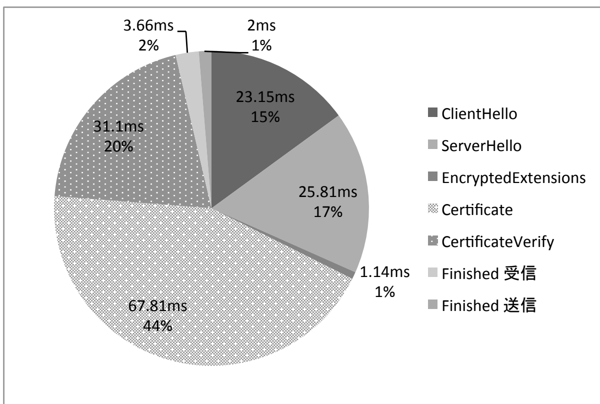

図 9　TLS_AES_128_GCM_SHA256，Curve25519 の ハンドシェイク処理時間内訳

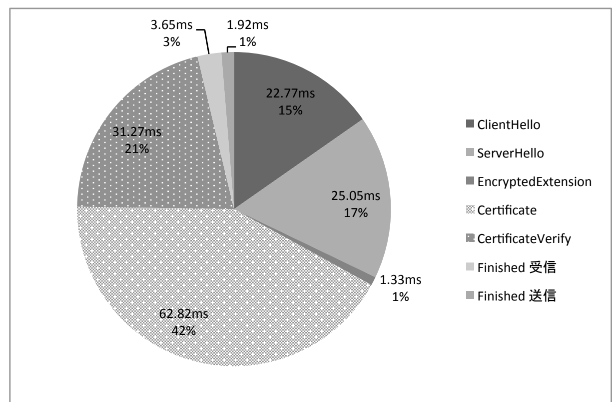

図 11　TLS_CHACHA20_POLY1305_SHA256，Curve25519 の ハンドシェイク処理時間内訳

シェイクが，TLS1.3 に比べ，処理時間が短いということがわかる．TLS1.2 と TLS1.3 では，ハンドシェイク手順にいくつか違いがあるが，ハンドシェイク全体で見ると，行う処理自体は，バージョン間で違いはない．しかし，TLS1.3 では，ServerHello 以降のメッセージは全て共通鍵により暗号化される．したがって，メッセージの暗号化/復号処理の分，TLS1.3 の処理時間が長くなったと考えられる．

一方で，外部サーバと通信を行う場合には，TLS1.3 の方が処理時間が短く，実行ごとのばらつきも小さい．これは，TLS1.3 のハンドシェイクの手順が変更され，メッセージの往復回数が 2 回から 1 回に減らされたことにより，ネットワーク状態の影響を受けにくくなったことが原因であると考えられる．

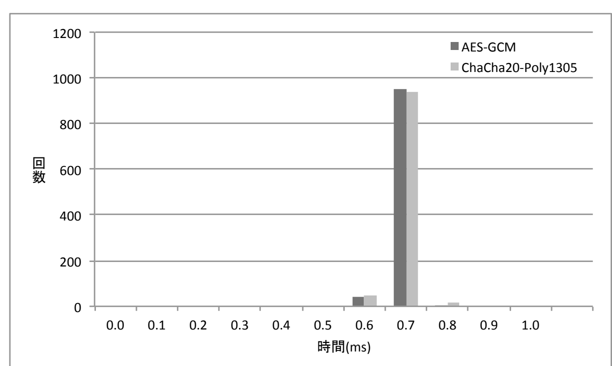

図 12　アプリケーションデータ送信処理時間

© 2018 Information Processing Society of Japan

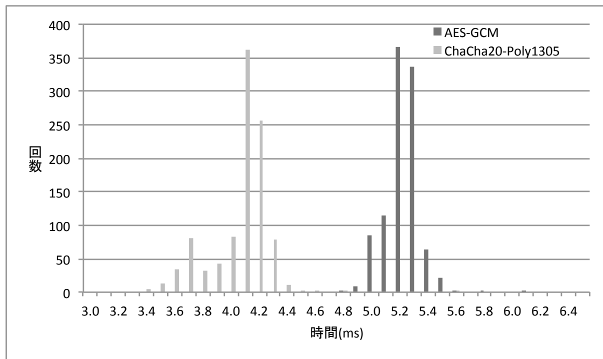

図 13 アプリケーションデータ受信処理時間

これらの結果から，外部サーバの場合は TLS1.3 を使用した方が，処理オーバヘッドを小さくすることができる．内部サーバと通信を行う場合は少々処理時間は長くなるが，強度の低い暗号アルゴリズムが廃止されたことや暗号化されるメッセージが増えてよりセキュアになったことなどその他の変更点を考慮すると，TLS1.3 を使用する恩恵は大きいと考えられる．

4.2 暗号アルゴリズムの違いによるハンドシェイク処理時間の比較

図 6 より，共通鍵暗号には，ChaCha20-Poly1305 を使用した場合に，わずかではあるが，ハンドシェイク処理時間が短いことがわかる．AES 暗号化方式は，Silicon Labs 社の EFM32 Pearl Gecko の CRYPTO や，参考文献 [10] など，ハードウェアによる高速化処理が知られている．これらを使用できる環境であれば，AES 暗号の方が高速に処理ができると考えられる．

図 7 より，楕円曲線には，Curve25519 を利用した場合に処理時間が大幅に短くなることがわかる．楕円曲線は，ECDHE 鍵交換処理に用いられる．鍵交換は，前方秘匿性を得るために，ハンドシェイクが行われるたびに新たに行うべきである．このため，より処理時間が短い Curve25519 を使用することは，パフォーマンスの大きな向上につながると考えられる．

4.3 TLS1.3 ハンドシェイクの処理時間内訳

図 8 と**図 9** の結果から，楕円曲線の種類により，ClientHello，ServerHello メッセージの処理時間が大きく変動することがわかる．TLS1.3 では，Hello メッセージの拡張フィールドで，鍵交換パラメータを送信する．このため，Hello メッセージ処理関数内の，このパラメータを生成する処理に時間がかかっていると考えられる．実際に，ソースコードを見ると，ClientHello 送信処理では，鍵交換パラメータ生成のための演算処理を行っていることがわかり，この処理時間を測定したところ，ClientHello メッセージ処理のほとんどの時間がこの処理であることがわかった．Curve25519 を使用することにより，この処理時間を大幅に削減できる．しかしその場合でも，ClientHello，ServerHello 合わせてハンドシェイク全体の 32%の処理時間を占めているため，ClientHello 送信処理における，鍵交換パラメータ生成のための演算処理を，ハードウェアサポートなどで高速化することが効果的であると考えられる．

4.4 アプリケーションデータの送受信時間

安全な通信路を確立した後の，アプリケーションデータの送受信にかかる時間は，共通鍵暗号の種類によって差があるが，ハンドシェイクの処理負荷が大きい部分に比べ負荷が小さいということがわかる．ただし，今回の通信で使用したメッセージサイズは，クライアント送信データが 40 バイト，受信データが 846 バイトとなっている．このメッセージサイズが大きくなるにつれ，処理時間も長くなると考えられる．そのため，アプリケーションデータの送受信回数により，共通鍵暗号を高速化することの効果は変動すると考えられる．

5. IoT 機器の特性を考慮した高速化の検討

これまでの結果と，IoT 機器の利用方法の特性を考慮して，高速化の手法の検討と，それによる処理性能の改善度合いを予想する．

5.1 IoT 機器を使用する際の前提

IoT 機器は，汎用コンピュータと異なり，用途が限定されるため，いくつか高速化の助けになると考えられる前提が存在する．その前提の例を以下に示す．

(1) 接続先のサーバが固定，もしくはいくつかに制限される
(2) 接続先がわかっていることから，PSK（Pre-Shared Key:事前共有鍵）を，製造時・出荷前にあらかじめ機器に組み込むことが可能であり，OTA（Over The Air）を用いて，変更することも可能となる
(3) サーバと通信するタイミング，通信周期が，ある程度予測することが可能である
(4) 通信で使用する暗号スイート，楕円曲線は Hello メッセージを見ればわかってしまうため，それ自体に脆弱性が発見されていなければ固定しても問題ない

5.2 前提から考えられる高速化手法

以上の前提から，高速化の手法を検討する．

前提 1，2 により，PSK を使用した TLS ハンドシェイクを行うことが可能になる．**図 14** に TLS1.3 の PSK によるハンドシェイクの流れを示す．PSK を用いたハンドシェイクでは，PSK を持っていることで，クライアントとサーバ間の相互認証を行うため，フルハンドシェイクから，Certificate，CertificateVerify メッセージを省略するこ

© 2018 Information Processing Society of Japan

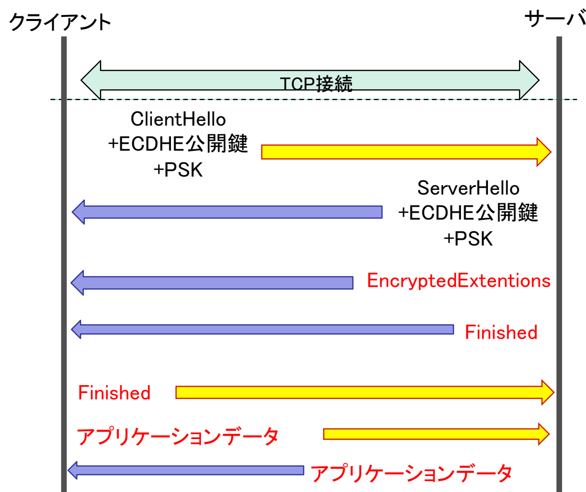

図 14　TLS1.3 の PSK ハンドシェイク

とが可能になる．

前提 3 により，ハンドシェイクに必要なデータをあらかじめ生成しておくことが可能になる．具体的には，ClientHello メッセージに必要な乱数や DH 鍵交換パラメータを事前にを生成しておくことにより，ClientHello メッセージかかる処理時間を削減できると考えられる．特に，評価結果より ECDHE 鍵交換パラメータの生成は大きく時間がかかっていることがわかっているため，この処理をあらかじめ行っておくことは処理時間の大きな削減につながると考えられる．

前提 1，4 より，使用する暗号アルゴリズムを高速なものに固定することができる．楕円曲線は，Curve25519 を使用した場合に，secp256r1 と比較して，公開鍵生成処理が，およそ 7.5 倍高速である．また，共通鍵暗号では，AES 暗号よりも，ChaCha20 暗号の方がわずかではであるが高速に処理ができる．しかし，AES 暗号化にはハードウェアサポートによる高速化が知られている．これらを使用することができる場合には，AES の方が高速に処理できると考えられる．

以上を踏まえ，前提を満たす状況における処理時間の削減効果を，簡易的な机上計算によって見積もる．使用するハードウェア，ソフトウェア共に性能評価と同じものを使用する場合を考える．まず，暗号スイートは TLS_CHACHA20_POLY1305_SHA256 を使用し，ECDHE 鍵交換での楕円曲線は Curve25519 を用いる．PSK を使用するため，証明書用公開鍵アルゴリズムは考慮する必要はない．ClientHello メッセージに必要なデータは全てハンドシェイクを開始する前に，あらかじめ生成しておくとする．これらを仮定した場合に，ハンドシェイクにかかる処理時間を計算する．ClientHello にかかる処理はほぼ全て事前に行うため，このメッセージの処理では完成したメッセージを送信するだけで良くなる．このため，このメッセージにかかる処理はほぼ考慮しなくて良いと考え

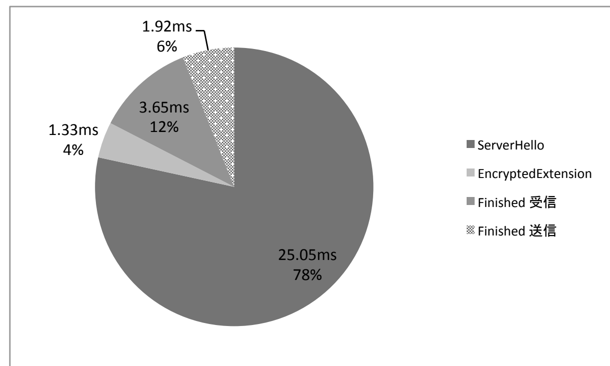

図 15　前提を考慮したハンドシェイク処理時間内訳の試算

られる．ServerHello，EncryptedExtensions メッセージに対する処理は省略不可能である．PSK を使用しているので，Certificate，CertificateVerify メッセージは不要である．このため，これらのメッセージの処理時間はかからない．Finished メッセージの送受信処理は省略不可能である．

これらを踏まえ，共通鍵に ChaCha20-Poly1305，楕円曲線に Curve25519 を使用している第 3.3 節の図 11 の処理時間のうち，ClientHello，Certificate，CertificateVerify メッセージ処理時間を 0 と仮定する．その結果，**図 15** のようなグラフになる．

グラフより，全体の処理時間は，31.95 ミリ秒となる．これは図 11 の 148.81 ミリ秒と比較すると，およそ 5 分の 1 まで処理時間を削減できる可能性があることが明らかになった．

6.　まとめ

本論文では，IoT 機器への TLS1.3 の適用を想定し，性能評価を行った．評価の結果と，IoT 機器の使用環境の前提を考慮し，前提を満たす状況における処理時間の削減効果を見積もった．性能評価の結果，TLS1.2 との比較は，外部サーバとの通信では，ネットワーク遅延の影響を受けにくく，処理時間が大幅に短縮されることが判明した．内部サーバとの通信の場合は，TLS1.3 の方が若干処理時間が長かったが，その増加分はわずかであり，セキュリティの向上や 0-RTT の導入など，他の改良点を考えると，許容範囲であると言えるだろう．

暗号スイートによる比較では，わずかではあるが，ChaCha20-Poly1305 が処理時間が短いことが判明した．AES にはハードウェアによる高速化があるが，価格の制約が厳しい IoT 機器に導入することは容易ではないため，ソフトウェア実装で高速な ChaCha20-Poly1305 を使用した方がオーバヘッドを削減できる．楕円曲線では，大幅に処理時間が短い Curve25519 を利用するべきだと思われる．

TLS1.3 の機能と，IoT 機器の使用環境を考慮した前提を組み合わせることで，ハンドシェイク処理時間を大幅に削減できると予想される．

© 2018 Information Processing Society of Japan

今後の課題として，TLS の処理オーバヘッドのさらなる削減が考えられる．前提を考慮した見積もりでは，ServerHello 処理の割合が大きい．ECDHE 鍵交換の演算処理をハードウェア化することでこの処理時間をさらに削減できると考えられる．共通鍵暗号は，今回の測定に用いたアプリケーションデータに対する高速化は効果が小さいが，データ量やデータの送受信回数が増えると，その効果も大きくなると考えられる．これらのハードウェアサポートを実装した上で，性能評価を行う必要がある．また，アプリケーションデータがクライアントから送信される場合には，ハンドシェイク開始からサーバの応答メッセージを受け取るまでに 2 往復必要である．さらに，TLS1.3 は TCP コネクション上で動作するため，TCP ハンドシェイクを含めると，データは 3 往復しなければならない．外部サーバとの通信ではネットワークの遅延が処理時間に大きく影響し，パケットの往復回数が増えるにつれ，その影響は大きくなる．この問題の解決策として，Google 社が 2013 年に開発した，「QUIC」と呼ばれる新しい Web プロトコルがある．QUIC は TCP ではなく，UDP ベースで TCP，TLS の機能の一部を実現する．UDP ベースで通信を行うため，TCP ハンドシェイクを行う必要がなくなる．つまり，1 往復分の初期オーバヘッドが削減される．QUIC は，2016 年秋から IETF で標準化が進められている．IETF で標準化される QUIC では，TLS1.3 を使用するため，0-RTT 接続を行うことが可能である．これを利用すると，ネットワークの状況にまったく依存せずに通信を行うことができる．現在使用策定中のため，普及には時間がかかると考えられるが，使用が定まれば，今後様々なアプリケーションでの利用が広がり，IoT 機器の通信でも利用されると考えられる．

今後も爆発的に，用途や利用数が拡大することが予想される IoT 機器において，低コストでセキュリティを実装することが必要になると考えられる．

参考文献

[1] IPA, 「米国における IoT（モノのインターネット）に関する取り組みの現状」 <https://www.ipa.go.jp/files/000047543.pdf >

[2] 「Google ウェブマスター向け公式ブログ : HTTPS をランキングシグナルに使用します」 <https://webmaster-ja.googleblog.com/2014/08/https-as-ranking-signal.html >

[3] 「Yahoo! JAPAN サービスは常時 SSL（AOSSL）に対応します．」<https://about.yahoo.co.jp/info/aossl/ >

[4] Diaa Salama Abd Elminaam, Hatem Mohamed Abdual Kader, and Mohiy Mohamed Hadhoud.: Evaluating The Performance of Symmetric Encryption Algorithms, *International Journal of Network Security*, Vol.10, No.3, pp. 213-219,（2010）.

[5] B. Padmavathi and S. Ranjitha Kumari.: A Survey on Performance Analysis of DES, AES and RSA Algorithm along with LSB Substitution Technique, *International Journal of Science and Research* , vol. 2, no. 4, pp. 170-174,（2013）.

[6] 「wolfSSL と wolfCrypt のベンチマーク」 <https://www.wolfssl.jp/wolfsite/benchmark/>

[7] Vipul Gupta, Sumit Gupta, Sheueling Chang, and Douglas Stebila.: Performance Analysis of Elliptic Curve Cryptography for SSL, *Proceedings of the 1st ACM Workshop on Wireless Security*, pp. 87-94,（2002）.

[8] Xiaodong Lin, Johnny W. Wong, and Weidong Kou.: Performance Analysis of Secure Web Server Based on SSL, *Lecture Notes in Computer Science, Springer-Verlag Heidelberg, Volume 1975/2000* pp. 249-261,（2000）

[9] 「wolfSSL、TLS1.3 初の商用リリースを発表」 <https://prtimes.jp/main/html/rd/p/000000002.000035612.html >

[10] Mohamed Khalil-Hani, Vishnu P. Nambiar and M. N. Marsono.: Hardware Acceleration of OpenSSL cryptographic functions for high-performance Internet Security, *2010 International Conference on Intelligent Systems, Modelling and Simulation*, pp.374-379,（2010）.

© 2018 Information Processing Society of Japan