オフラインでも利用可能な信頼できる時刻源の TPM と機密実行環境を用いた実装

宮崎想也[1]　新城靖[1]

概要：機密実行環境 (trusted execution environment, TEE) とは，CPU の働きにより OS や管理者からもデータを秘匿してプログラムを実行可能な領域である．その中で動作するアプリケーションで信頼できる時刻を必要とするものが多く存在するが，機密実行環境は信頼できる時計を持たず，OS が提供する時刻は改ざんされうる．従来の方法は，インターネット上の時刻サービスや複数のノードを必要とするため，オフライン環境や個人の PC では利用しにくい．この論文では，Trusted Platform Module (TPM) が持つ稼働カウンタと機密実行環境を用いて信頼できる時刻源を提供する方法を提案している．提案手法では，ネットワーク上の信頼できる時刻源を用いてキャリブレーションし，その後はオフラインで現在時刻とその誤差の上限を提供する．本論文では，提案手法に基づき信頼できる時刻源を Intel Software Guard Extensions (SGX) のエンクレーブ上で動作する Roughtime サーバとして実装する方法を示している．実験により，オフラインで 24 時間稼働させた後でも，誤差の上限を 1 分以内に保って信頼できる時刻を提供できることを示している．

1. はじめに

機密計算 (confidential computing) は，CPU が提供する機密実行環境 (trusted execution environment, TEE) の中でプログラムを実行し，OS や管理者からもデータを秘匿する技術である．機密実行環境を提供する CPU の機能には，Intel Software Guard Extensions (SGX) [1] や Trust Domain Extensions (TDX) [2]，AMD Secure Encrypted Virtualization Secure Nested Paging (SEV-SNP) [3]，ARM TrustZone [4] がある．

機密実行環境で動作するアプリケーションには，信頼できる時刻を必要とするものが多く存在する．たとえば，リースや著作権保護では利用期間の判定に，X.509 証明書の検証では証明書の有効期限の確認に時刻を必要とする．これらのアプリケーションでは，攻撃者が時刻を改ざんできると，リース期間外に不正にデータを利用されたり，有効期限が切れた証明書が有効と判定されることがある．

機密実行環境内のアプリケーションに信頼できる時刻源を提供するのは容易ではない [5]．機密実行環境の外にある OS やハイパーバイザが提供する時刻は容易に改ざん可能である．機密実行環境に信頼できる時刻源を提供する従来の方法には，インターネット上の時刻サービスを使うもの [6] と分散システムを使うもの [7], [8] があり，それぞれ問題がある．まず，インターネット上の時刻サービスを使う方法には 3 つの問題がある．1 つ目に，そのサービス自体を信頼できるとは限らない問題がある．2 つ目に，問い合わせのたびにサーバへ利用状況が伝わるため，プライバシーの観点で問題がある．3 つ目に，攻撃によりネットワークを遮断されたオフライン環境では時刻が得られない問題がある．また，複数のノードからなる分散システムで信頼できる時刻源を実装する方法は，ノードの用意にコストがかかり，個人では使いにくいという問題がある．

本研究では，Trusted Platform Module (TPM) 2.0 [9] が持つ稼働時間カウンタ (uptime timer) と機密実行環境を利用して，信頼できる時刻源を提供する手法を提案する．この稼働カウンタは TPM の起動からの経過時間を表すタイマであり，値を自由に書き換えることはできない．また，TPM の内部に保持された鍵による署名を付けて読みだせるため，改ざんを検知できる．

提案手法では，信頼できる時刻源を Intel SGX のエンクレーブ上で動作する Roughtime サーバとして実装する．このサーバは，起動時にネットワークを利用してリモートアテステーション (remote attestation) とキャリブレーションを行い，その後はオフラインで時刻を提供する．提案手法は単一のノードで動作するので，個人の

[1] 筑波大学

ⓒ2026 Information Processing Society of Japan

PC でも容易に導入することができる．また，時刻を得るたびにネットワークへ問い合わせることもないので，利用状況が外部に漏れることはない．

提案手法により得た時刻のアプリケーションとして本論文では，タイムスタンプサービスおよび X.509 証明書の検証を実現できることを示す．実験により，オフラインで 24 時間稼働させた後でも，誤差の上限を 1 分以内に保って信頼できる時刻を提供できることを示す．

2. TPM 2.0

TPM は，Trusted Computing Group (TCG) によって仕様が定められた，セキュリティに関する機能を提供するハードウェアモジュールである [9]．TPM の実現方法としてはマザーボード上に物理的に実装される Discrete TPM (dTPM) と，CPU のファームウェアとして実装される Firmware TPM (fTPM) が存在する．TPM の機能には，鍵の生成や管理，乱数の生成などがある．TPM は，生成した鍵や小さいデータを保管するための不揮発性メモリや，2.2 節で述べるクロックを持つ．

TPM のコマンドの一部には，その実行結果に TPM 内部の鍵で署名を付けて返すものがある．TPM との通信は，通常 `/dev/tpmrm0` などのデバイスファイルを介し，この通信は OS が仲介する．OS を信頼しない機密実行環境は，この署名を検証することで，OS を経由して受け取った値が改ざんされていないことを確認できる．

TPM 2.0 は，TPM の仕様の一つである．以前のバージョンである TPM 1.2 と比較して，対応する暗号アルゴリズムが増え，クロックを署名付きで読みだす `TPM2_GetTime` などのコマンドが定義された．

2.1 TPM のアテステーション

TPM は，Endorsement Key (EK) と呼ばれる鍵を持ち，TPM の製造業者は EK の公開鍵に対する証明書（EK 証明書）を発行する．この証明書は TPM 内部の不揮発性メモリから，または製造業者がネットワーク経由で入手可能である．EK 証明書を製造業者のルート証明書まで検証することで，その EK が正当な TPM のものであることを確認できる．EK は，署名に用いることはできないため，Attestation Identity Key (AIK) と呼ばれる署名鍵を生成し，これを用いて署名を行う．

AIK と EK が同じ TPM に存在することは，`TPM2_MakeCredential` と `TPM2_ActivateCredential` を用いて確認できる．そのために，検証者は `TPM2_MakeCredential` で EK の公開鍵と AIK の公開鍵などのハッシュ値を用いてランダムな値を暗号化して，TPM に送信する．次に `TPM2_ActivateCredential` コマンドが実行されたとき TPM は，その EK と指定された AIK の両方を内部に持つ場合に限り，その暗号化された値を復号できる．復号された値が元の値と一致すれば，検証者は AIK が EK と同じ TPM の内部にあると確認できる．

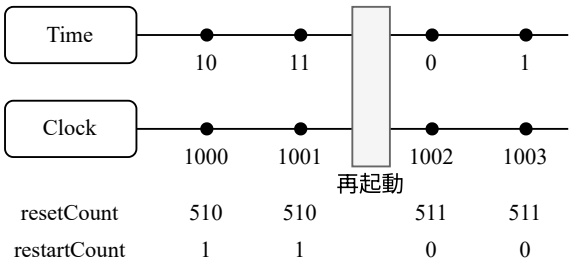

図 1 TPM の Clock と Time の挙動

2.2 TPM のクロック

TPM が持つ時刻は，Time と Clock の 2 種類が存在する．それらの挙動を図 1 に示す．Time は TPM が通電状態のとき増加する値であり，揮発性メモリに保持され，TPM の電源が切れると 0 にリセットされる．Time の値はコマンドによって変更することができない．Clock は Time に基づいて増加し，その値は一定間隔（最長 70 分間隔）で不揮発性メモリに書き込まれ，永続化される．Clock の値は，`TPM2_ClockSet` コマンドによって進めることができる．これらのクロックの進み方は，TPM が用いる発振器のクロック周波数に依存し，1 ミリ秒ごとに増加する．本研究では，Time を TPM の**稼働カウンタ**と表記する．

Time は PC が再起動されるたびに 0 に戻るため，Time の値が連続していることを確認するためには，それを引き起こす電源断を検知する必要がある．そのために，TPM は resetCount，および restartCount と呼ばれる値を不揮発性メモリに保持し，Time や Clock とともに提供される．resetCount は PC の起動時に `TPM2_Startup` が実行されるたびに増加する．TPM はこの `TPM2_Startup` が実行されるまで，他のコマンドを受け付けない．restartCount は TPM がスリープや休止状態などから復帰する際に増加し，resetCount が増加するときに 0 にリセットされる．

稼働カウンタの値は，`TPM2_GetTime` コマンドによって署名付きで取得できる．また，稼働カウンタの進行速度は，`TPM2_ClockRateAdjust` コマンドによって最大 32.5 % 程度調整することができる [10]．

2.3 TPM の稼働カウンタの精度

本研究では，予備実験として，具体的な TPM の稼働カウンタの精度を調査した．実験に用いた機器のハードウェア情報を**表 1** に示す．以下では，それぞれの機器を

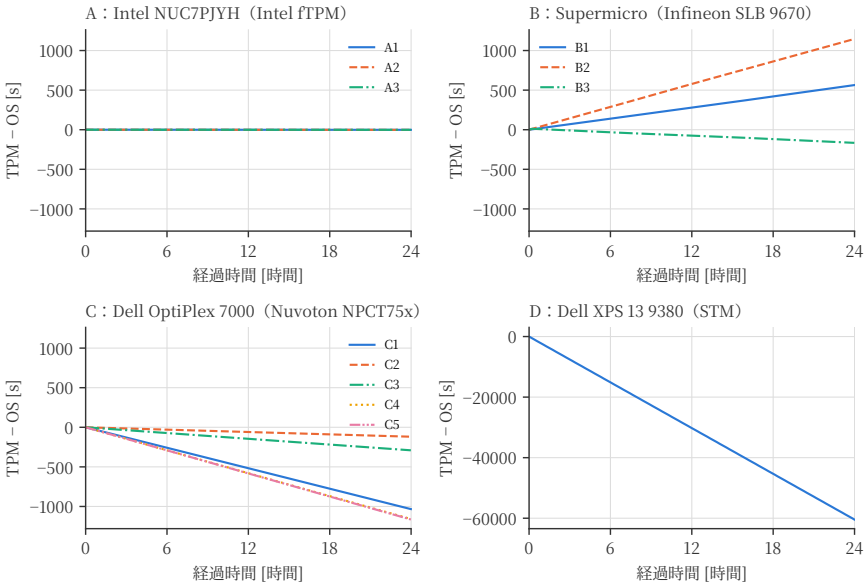

図 2　TPM のクロックの精度

表 1　実験に用いた機器のハードウェア情報

機器	CPU	メモリ	TPM ベンダ	TPM モデル	ファームウェア
A: Intel NUC	Intel Pentium Silver J5005 CPU	32 GB	Intel	—	403.0
B: Supermicro	Intel Xeon Silver 4514Y	64 GB	Infineon	SLB 9670	7.86
C: Dell OptiPlex 7000	12th Gen Intel Core i7-12700	32 GB	Nuvoton Technology	NPCT75x	7.2
D: Dell XPS 13 9380	Intel Core i7-1065G7	16 GB	STM	—	74.8

A から D で表す．

実験では，1 分おきに OS の時刻と TPM の稼働カウンタの値を取得し，その 2 つの時間の進み方を比較する．測定は 24 時間行った．その結果を図 2 に示す．横軸は NTP で同期された OS の時計で測定した測定開始からの経過時間，縦軸は，測定開始からの TPM の稼働カウンタと OS の経過時間の差である．

A の TPM は，1 日に 1.5 から 1.8 秒遅れる．B は個体差が大きく，1 日に 2 分程度遅れるものから 20 分程度遅れるものまであった．C も同様に個体差があり 3 分程度遅れるものから 20 分程度進むものまであった．特に D の TPM の時間の進み方のずれは大きく，稼働カウンタは OS の時刻の 1 分間に約 21 秒しか進まなかった．

この結果より，TPM の稼働カウンタの進み方には個体差があるため，機器ごとにキャリブレーションを行う必要があることが確認された．また，稼働カウンタの増加速度は，ほとんど一定であることが確認された．

2.4　コマンド監査機能

TPM 2.0 には，コマンドの実行を監査するための command audit と呼ばれる機能が備わっている．監査対象は，`TPM2_SetCommandCodeAuditStatus` によって設定することができ，`TPM2_GetCommandAuditDigest` で監査情報を取得できる．監査情報には，監査対象となるコマンド `commandDigest` や監査開始してからのコマンドの実行回数を表す `auditCounter` などが格納される．また，`TPM2_GetCommandAuditDigest` の実行結果は署名付きで取得できる．

2.5　本研究で利用するコマンド

本研究で利用する TPM のコマンドを表 2 に示す．`TPM2_CreatePrimary`，`TPM2_Create`，および `TPM2_ActivateCredential` は，初期化時に AIK を作成し，AIK が正当な TPM の内部にあることを確認するために用いる．`TPM2_GetTime` は，時刻の算出に用いる Time の値と，resetCount および restartCount を取得するために用いる．`TPM2_SetCommandCodeAuditStatus` と `TPM2_GetCommandAuditDigest` は，`TPM2_ClockRateAdjust` の実行を検知するために用いる．

3.　前提と脅威モデル

本研究で実装するプログラムは，以下のハードウェアやソフトウェアが利用できることを前提とする．

- ハードウェアによって保護された機密実行環境．この機密実行環境は，シーリングと呼ばれる環境固有の鍵導出機能を持ち，リモートアテステーションをサポートする．

ⓒ2026 Information Processing Society of Japan

表 2 使用する TPM のコマンド

コマンド	署名の有無	説明
TPM2_CreatePrimary	無	EK を作成する
TPM2_Create	無	AIK を作成する
TPM2_ActivateCredential	無	AIK が EK と同じ TPM にあることを確認する
TPM2_GetTime	有	TPM の持つ時計の値を署名付きで取得する
TPM2_SetCommandCodeAuditStatus	無	コマンド監査の対象を設定する
TPM2_GetCommandAuditDigest	有	コマンド監査情報を署名付きで取得する

- TPM 2.0 の仕様を満たす TPM．TPM には，2 章で述べたコマンド監査関連のコマンドと TPM2_GetTime が使えることを求める．
- LAN やインターネットなどの信頼できない通信路．
- OS のファイル等の信頼できない永続記憶
- キャリブレーション時の信頼できる外部の時刻源
- リモートアテステーションを行うサーバ

本研究では，通信路，永続記憶 (OS のファイル)，CPU の Time Stamp Counter (TSC) を信頼せず，SGX エンクレーブが外部とやり取りする通信路はすべて信頼しない．この通信路には，TPM やアプリケーションとの通信路に加え，リモートアテステーション用のサーバやキャリブレーション時の信頼できる時刻サーバとのインターネット越しの通信路も含む．

本研究では，以下の攻撃を想定する．

- 攻撃者は，通信路を流れるメッセージに対して遅延を挿入して時刻を操作する．
  - 普段は遅延させずに，必要な時だけ遅延させる．
  - 普段から遅延させておき，必要なときだけ遅延をやめる．
- 攻撃者は OS のファイルを変更・削除する．
- 攻撃者は CPU の TSC を書き換える．
- TPM2_ClockRateAdjust コマンドで TPM のクロックの進み方を速めたり遅めたりする．

これらの対処方法は，アプリケーションに依存する．本研究では，攻撃者は利益がない攻撃をしないと仮定する．たとえば，タイムスタンプのサービスでは，ある時刻でその書類が存在していたことを示したい．よって，攻撃者にとって時刻を遅らせる攻撃には意味がないので，攻撃者はそういう攻撃を行わない．

また，DoS 攻撃は想定しない．攻撃者は不正な時刻を得たいので，サービスが受けられなくなるような攻撃はしない．PC のスリープも本研究では扱わない．

4. 提案手法

本研究では，信頼できる時刻源を SGX エンクレーブ上で動作する Roughtime サーバ [11] として実装する．Roughtime は，署名によって時刻の真正性を保証する時刻同期プロトコルである．クライアントが nonce を含む要求を UDP で送信すると，サーバは現在時刻とその誤差を表す radius，およびクライアントから受け取った nonce に署名して返す．Roughtime を選んだ理由は，API として簡潔であること，および，誤差を一緒に提供することができることである．

本研究では，次のようなアプリケーションを想定する．

- タイムスタンプ・サービス
- X.509 証明書の有効期限の確認
- 著作物のリース

いずれのアプリケーションも，それほど高い精度は求められていない．提案手法では，時刻だけではなく誤差も一緒に提供するため，それぞれのアプリケーションがそれを加味して利用する．たとえば，リース期間の終了時刻が必要なアプリケーションは，誤差の上限の分だけ早い時刻を用いることができる．

4.1 構成

本研究で実装する信頼できる時刻源の構成を図 3 に示す．Roughtime サーバは，SGX エンクレーブで動作し，そのホストに接続された TPM にアクセスする．図 3 で鍵登録サーバは，Roughtime サーバの署名で利用する鍵を登録するサーバである．SGX エンクレーブ上で動作する Roughtime サーバは，鍵登録サーバとリモートアテステーションを行い，公開鍵を登録する．本研究では，起動時のキャリブレーションのためにインターネット上の信頼できる時刻源としても Roughtime のサーバを用いる．

提案手法では，オンライン時にキャリブレーションを行い，その結果をシールして信頼できない永続記憶に保存する．サービス提供時には，キャリブレーションの結果と TPM の稼働カウンタから現在時刻とその誤差の上限を算出し，それぞれ Roughtime の時刻と radius として返す．

Roughtime のサーバは，ビジーウェイトせず，OCALL で UDP のメッセージを待つ．同じ PC または，LAN 内のクライアントから要求を受けとると，TPM にアクセスし稼働カウンタを取得する．次に，キャリブレーション結果を用いて稼働カウンタから現在時刻と radius を算出し，署名を付与してクライアントに返す．クライア

ⓒ2026 Information Processing Society of Japan

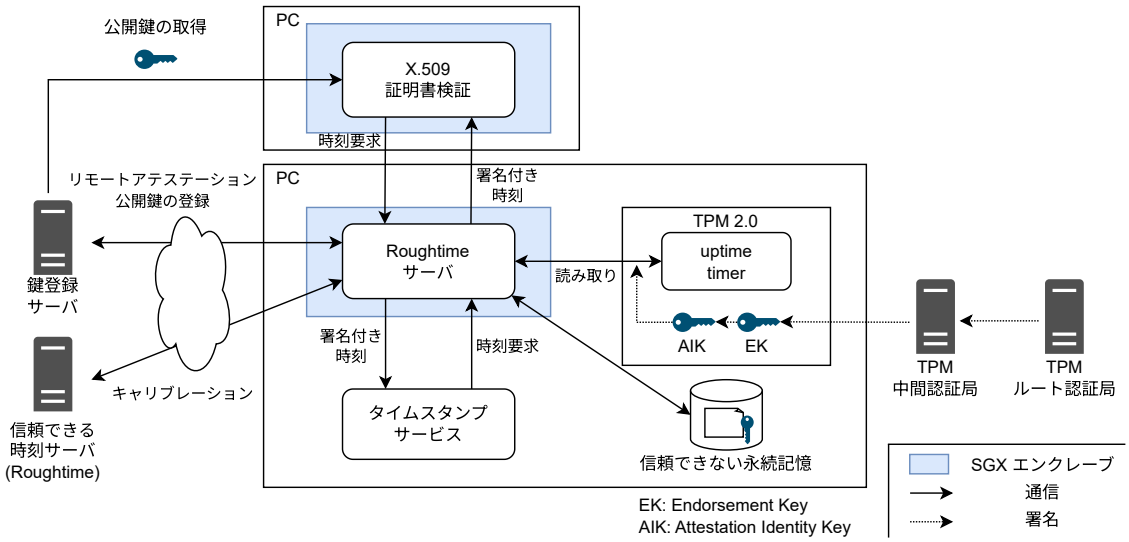

図 3　信頼できる時刻源の構成

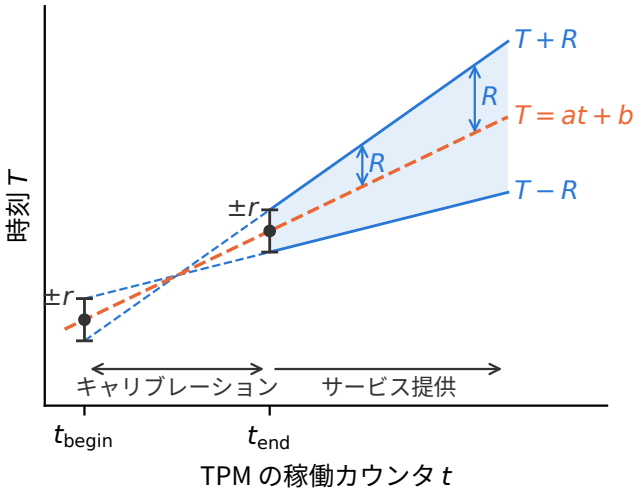

図 4　提案手法で提供する時刻と誤差

ントは，鍵登録サーバから取得した公開鍵を用いて署名を検証する．クライアントの例として，TLS における X.509 証明書検証を行うアプリケーションやタイムスタンプサービスなどが挙げられる．

4.2　提供する時刻と誤差

2.3 節で示したように，TPM の稼働カウンタの進み方は TPM によって異なる．そこで提案手法では，以下のように信頼できる時刻サーバから得た時刻と稼働カウンタの値を対応付ける．TPM の稼働カウンタの進み方が一定だと仮定すると，補正後の時刻 T は，稼働カウンタの値 t を使って次の式で表すことができる．

$$T = at + b \tag{1}$$

ただし，キャリブレーションでは TPM へのアクセスや通信に時間がかかるため，対応付けた時刻には誤差が生じ，その誤差は補正後の時刻では時間とともに広がる．

提案手法では図 4 に示すように，提供する時刻 T とともに，真の時刻が「$T \pm R$」の範囲に入る**誤差の上限** R を返す．

このために，キャリブレーション時に次の値を観測する．

- インターネット上の信頼できる時刻サーバから得られた時刻と誤差
- TPM の稼働カウンタ

これらを使ったキャリブレーションについては，4.4 節で述べる．

4.3　初期化

初期化として，オンライン時に次のことを行う．

(1) PC でエンクレーブを起動する．
(2) 鍵登録サーバに依頼して PC 上のエンクレーブのリモートアテステーションを行ってもらい，自身の健全性を確認する．
(3) TPM に Attestation Identity Key (AIK) を生成する．
(4) AIK を検証する．鍵登録サーバに依頼するか，自ら Privacy CA に問い合わせるか，コードに埋め込まれた製造業者の公開鍵を使って検証する．
(5) インターネット上の Roughtime サーバの時刻を参照し，キャリブレーションを行う．キャリブレーションについては 4.4 節で述べる．
(6) 問題がなければ，Roughtime 応答メッセージを署名するための鍵ペアを生成し，鍵登録サーバに公開鍵を送信する．
(7) 信頼できない永続記憶に次の情報をシールして保存する．
  - AIK の公開鍵
  - キャリブレーションの結果得られた TPM の稼働カウンタのオフセットと進み方の補正係数

ⓒ2026 Information Processing Society of Japan

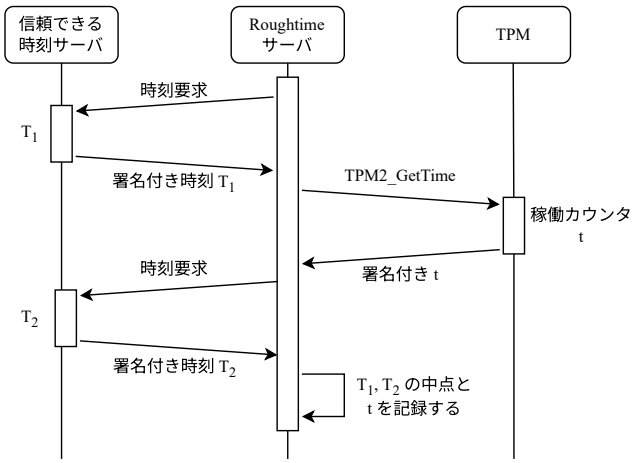

図 5 キャリブレーション時の処理

- TPM の resetCount
- Roughtime の応答メッセージに署名するための鍵ペア

TPM には 2.3 節で示したように，稼働カウンタの進み方が不適切なものが存在する．そのような TPM については，次の 2 つの方法で除外する．1 つ目に，リモートアテステーション時に TPM の機種を確認する．TPM のモデル情報などを鍵登録サーバに送信し，ホワイトリスト形式で確認する．登録されていない TPM を用いている場合は，リモートアテステーション時にエラーを返す．2 つ目に，キャリブレーション時に時刻の進み方を検証する．この時，異常な進み方が観測された場合，不適切な TPM と判断し，サービスの提供を行わない．

4.4 キャリブレーション

キャリブレーション時のメッセージのやりとりを図 5 に示す．キャリブレーションの開始時と終了時に，それぞれ次の操作を行う．

(1) 信頼できる時刻サーバから現在の時刻 T_1 とその radius r_1 を取得する．
(2) `TPM2_GetTime` を実行し，稼働カウンタの値 t を取得する．
(3) 信頼できる時刻サーバからもう一度，現在の時刻 T_2 とその radius r_2 を取得する．

観測された値を使って，次の値を求める．

- 信頼できる時刻サーバから取得した時刻の中点

$$T' = \frac{T_1 + T_2}{2} \tag{2}$$

- 稼働カウンタの読み取りにかかる時間

$$h = \frac{T_2 - T_1}{2} \tag{3}$$

- TPM の稼働カウンタと信頼できる時刻サーバから取得した時刻の対応付けで生じる誤差の上限

$$r = \max(r_1, r_2) + h \tag{4}$$

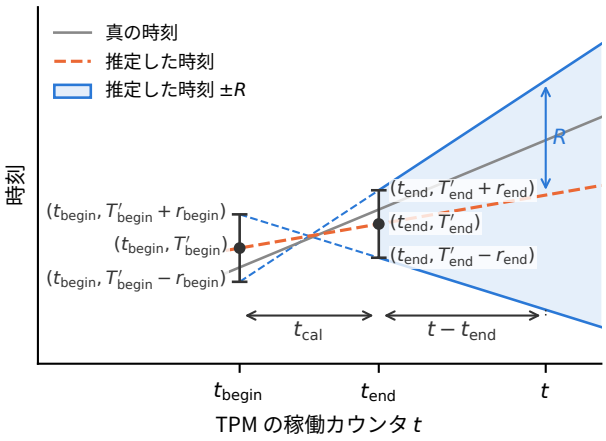

図 6 推定した時刻と誤差の上限

このような観測と測定をキャリブレーションの開始時と終了時の 2 回行う．その結果，次の値が得られる．

- 開始時の値: $T'_{\text{begin}}, t_{\text{begin}}, r_{\text{begin}}$
- 終了時の値: $T'_{\text{end}}, t_{\text{end}}, r_{\text{end}}$

なお，$t_{\text{end}} - t_{\text{begin}}$ を**キャリブレーション間隔**と呼び t_{cal} とする．図 6 に，これらの値を示す．

時刻の進み方が一定だと仮定し，最初と最後の 2 点 $(t_{\text{begin}}, T'_{\text{begin}})$，$(t_{\text{end}}, T'_{\text{end}})$ を通る直線として，式 1 の係数 a，b を以下の式で求める．

$$a = \frac{T'_{\text{end}} - T'_{\text{begin}}}{t_{\text{end}} - t_{\text{begin}}}, \quad b = T'_{\text{end}} - a t_{\text{end}} \tag{5}$$

求めた a および b を TPM の稼働カウンタの補正係数をとして保存する．

サービス提供時，稼働カウンタの値が t のとき，提供する時刻 T は式 1 で，誤差の上限 R は次の式で表す．

$$R = r_{\text{end}} + \frac{r_{\text{begin}} + r_{\text{end}}}{t_{\text{cal}}}(t - t_{\text{end}}) \tag{6}$$

第 1 項は，最後の測定点の誤差である．第 2 項は，傾き a の誤差によるずれである．真の時刻の直線は $T'_{\text{begin}} \pm r_{\text{begin}}$ と $T'_{\text{end}} \pm r_{\text{end}}$ の中を通るので，傾きの誤差は最大 $(r_{\text{begin}} + r_{\text{end}})/t_{\text{cal}}$ であり，これが $t - t_{\text{end}}$ に比例して積み重なる．図 6 中の青い直線は $T \pm R$ で表される．

4.5 サービス提供

サービスの起動は次の手順で行う．

(1) 信頼できない永続記憶からシールされたデータを読み出し，アンシールしてキャリブレーション結果を得る．
(2) TPM とセッションを確立する．
(3) TPM の resetCount を得る．(1) で読み出した値とずれていたらエラーにしてサービスを提供しない．
(4) サーバとしてサービスを開始し，クライアントから

ⓒ2026 Information Processing Society of Japan

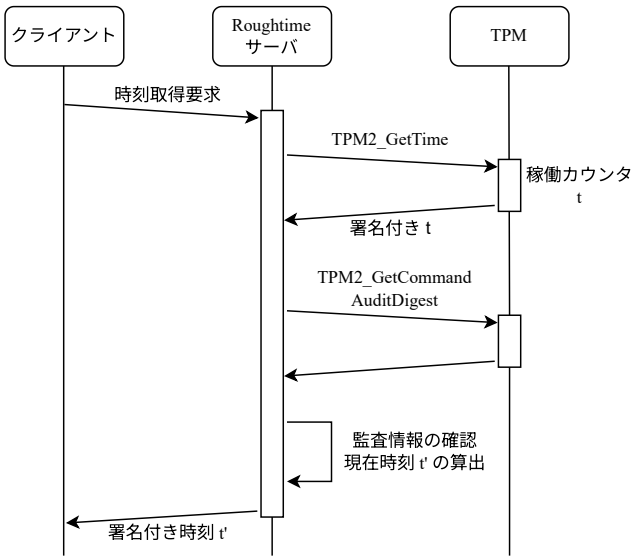

図 7 サービス提供時の処理の流れ

要求を受け付ける．

クライアントから要求を受け取った時のサービス提供時の処理の流れを図 7 に示す．

(1) `TPM2_GetTime` を実行し TPM の稼働カウンタの値 t と resetCount を取得する．

(2) resetCount の値が永続記憶からアンシールした resetCount と一致していなければ，エラーを返す．

(3) `TPM2_GetCommandAuditDigest` で TPM のコマンド監査情報を取得し，期待値と一致することを確認する．一致しない場合，時刻の進み方が変更されたと判断しエラーを返す．

(4) (1) で取得した値から現在時刻とその誤差を算出し，クライアントに補正後の時刻と誤差の上限を返す．

(4) では，式 1 の時刻 T と式 6 の R を算出し，T を Roughtime の時刻として，R を radius として返す．

このサーバは，サービス提供中はインターネットと通信しない．オンラインであれば，再キャリブレーションしてもよい．

5. 時刻を操作する攻撃の分析

攻撃者は，TPM のコマンド実行前後およびネットワーク通信に遅延を挿入することで，時刻を不正に操作することができる．また，TPM のコマンドを用いて時刻の進み方を操作することができる．本章では，これらの攻撃による影響と対策を述べる．

5.1 キャリブレーション時の遅延の挿入

キャリブレーションでは，4.4 節で述べたようにインターネット上の信頼できる時刻サーバから取得した二つの値 T_1, T_2 の中点 $T' = (T_1 + T_2)/2$ に TPM の稼働カウンタ t を対応付ける．このとき，攻撃者は図 8 の 1 から 6 の通信に遅延を挿入できる．この中で攻撃として意味があるのは，2 から 5 のタイミングでの遅延の挿入である．1 へ遅延を挿入しても t, T_1, T_2 がすべて挿入された遅延の分だけずれるだけであり，t が中点 T' からずれることはない．6 は t, T_1, T_2 がすべて決まった後であるため，このタイミングで遅延を挿入しても t と T' の対応付けに影響が出ない．

2 から 5 への遅延は，次の 2 つの攻撃に分けられる．挿入する遅延を d，1 回の測定のずれを $e = T' - \tau$ とする．τ は，TPM の稼働カウンタを読み取った瞬間における信頼できる時刻サーバの時刻である．

- 攻撃 A: 2 か 3 に遅延を挿入すると，τ と T_2 を決める時点が d だけ遅れ，$e = -d/2$ となる．
- 攻撃 B: 4 か 5 に遅延を挿入すると，T_2 を決める時点だけが d だけ遅れ，$e = +d/2$ となる．

攻撃 A も攻撃 B も，遅延が大きければ $T_2 - T_1$ から検出できることがある．たとえば，TPM から稼働カウンタの値を読むのに，表 1 の A では約 130 ミリ秒，B では約 80 ミリ秒かかる．Cloudflare の Roughtime サーバから時刻を取得するには，約 4.2 ミリ秒かかる．これらの合計と比較して，$T_2 - T_1$ が著しく大きい場合は攻撃があったとみなす．

キャリブレーションの開始時と終了時の 2 回の測定で生じたずれを $e_{\text{begin}}, e_{\text{end}}$ とし，その差を $\Delta = e_{\text{end}} - e_{\text{begin}}$ とする．また，キャリブレーション間隔を t_{cal} とする．稼働カウンタの値が t のとき，時刻は $D = \frac{\Delta}{t_{\text{cal}}}(t - t_{\text{end}})$ ずれる．1 回目に攻撃 B, 2 回目に攻撃 A を行うと $\Delta = -d$ となり，時刻は遅れていく．たとえば $\Delta = -100$ ミリ秒，$t_{\text{cal}} = 1$ 分なら，時刻は 1 日に 144 秒遅れる．

表 3 にキャリブレーション時の遅延によって生じる最大の誤差を示す．$t_{\text{cal}} = 1$ 分のときは，加えられた遅延 $d = 10$ ミリ秒でも 1 日で 14.4 秒ずれる．一方，$t_{\text{cal}} = 1$ 日のときは，加えられた遅延 $d = 1$ 秒でもずれは 1 秒にとどまる．このように，キャリブレーション間隔 t_{cal} を長くすることで，キャリブレーション時の遅延の挿入の影響を小さくすることができる．

表 3 キャリブレーション時の遅延によって生じうる最大の誤差

	キャリブレーション間隔 t_{cal}		
遅延 d	1 分	1 時間	1 日
10 ms	14.4 s	0.24 s	0.01 s
100 ms	144 s	2.4 s	0.1 s
1 s	1440 s	24 s	1 s

5.2 サービス提供時の遅延の挿入

攻撃者は図 9 中の 1 から 6 の通信に遅延を挿入できる．1 と 2 への遅延は `TPM2_GetTime` のコマンドへの遅延と，3 から 6 への遅延はその応答への遅延と同じ効果を持つ．ここでは，前者と後者をそれぞれ攻撃 C，D と

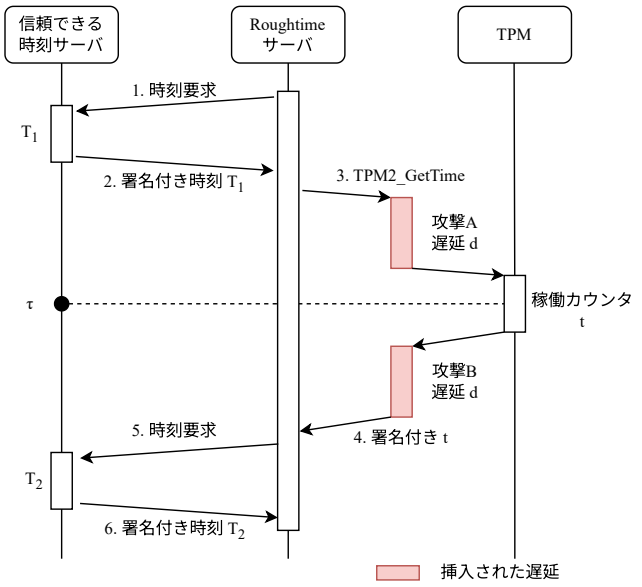

図 8 キャリブレーションへの遅延の挿入

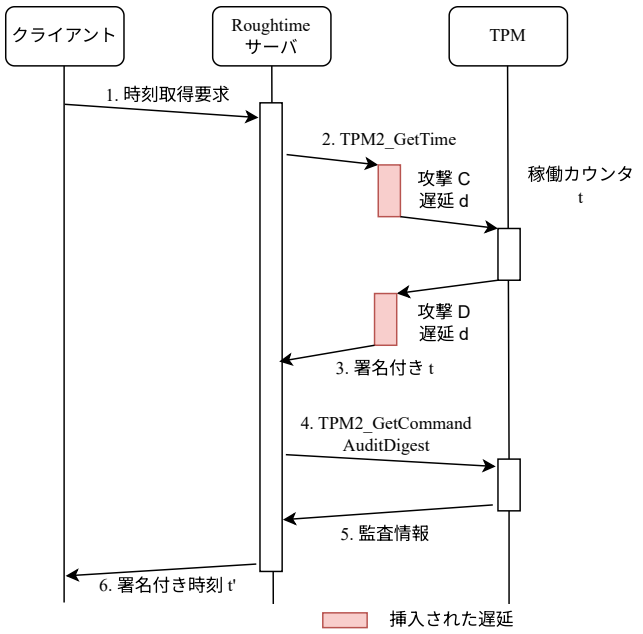

図 9 サービス提供時の遅延の挿入

してその影響を述べる．

攻撃 C では，挿入された遅延の分だけ，TPM が稼働カウンタを読む時点がそのまま遅れる．そのため，得られる時刻も遅延の後の時刻となり，古い時刻を読ませることはできない．したがって，攻撃 C によって過去の時刻を取得することができないため，問題とならない．

攻撃 D は問題になる可能性がある．返ってくる時刻は古い `TPM2_GetTime` の実行結果から求められたものであるため，時刻も古いものとなる．たとえば，12:00.00 にリクエストを送信し，10 分の遅延を加えると 12:10.00 に 12:00.00 の時刻を取得できる．しかし，リクエスト送信前の時刻は取得できない．そのため，タイムスタンプサービスのような用途では，この攻撃は意味のないものとなる．

攻撃 D は有効期限の検証などでは問題となるが，エンクレーブ内部で `TPM2_GetTime` の実行時間を測定することで検知できる．その方法として，エンクレーブ内のスレッドがスピンループでカウンタを増やし続け，`TPM2_GetTime` の要求から応答までのカウンタの増分を調べる方法が考えられる．攻撃者はカウンタを増加させるスレッドを停止することでこの方法を回避することができる．スレッドの停止には割り込みが必要であり，その際にはエンクレーブ脱出 (asynchronous exit, AEX) が生じる．このエンクレーブ脱出は，AEX-Notify [12] や State Save Area (SSA) を利用する手法 [13] によって検知できる．

エンクレーブ脱出は，OS のタイマ割り込みなどによっても生じるため，そのままでは攻撃によるものと区別できない．そのため，カウンタスレッドを実行するために一時的に 1 つの CPU コアを独占し，さらに，その CPU に対するタイマ割り込みを止めることで対処する．それでも `TPM2_GetTime` の実行中にエンクレーブ脱出が起きた場合は，やり直す．

5.3 TPM のクロックに対する攻撃

攻撃者は，`TPM2_ClockRateAdjust` によって稼働カウンタの進み方を変化させることができる．この攻撃は，次の手順で検知する．

4.3 節で述べた初期化の時に `TPM2_SetCommandCodeAuditStatus` で `TPM2_ClockRateAdjust` を監査対象にする．続いて `TPM2_GetCommandAuditDigest` で監査情報を取得し，含まれる署名を AIK で検証したうえで，auditCounter の値を期待値として記録する．以降は，TPM の稼働カウンタを読むたびに監査情報を取得し，同じように署名を検証する．auditCounter が期待値と異なる場合，`TPM2_ClockRateAdjust` が呼ばれたと判断し，時刻を提供しない．

5.4 アプリケーションに対する攻撃

タイムスタンプサービスと証明書の有効期限の検証の 2 つのアプリケーションに対する攻撃を例に，攻撃方法とその対処方法を述べる．

タイムスタンプサービスでは，攻撃者は文書を実際より前に作成したように見せかけるため，現在時刻よりも前の時刻を取得しようとする．サービス提供時の遅延では要求より前の時刻は得られない．キャリブレーション時の遅延によるずれは最大でも radius R に収まる．タイムスタンプの時刻として Roughtime サーバから返ってきた時刻 T に R を加えた時刻を用いることで対処する．

有効期限の検証では，図 9 中の 3 から 6 に遅延を挿

入することで，古い時刻を取得させ，期限切れの証明書を利用する攻撃が考えらえる．これは，5.2 節で述べた TPM2_GetTime の実行時間の測定により対処する．

6. 実装

提案手法に基づき SGX による機密実行環境で動作する Roughtime サーバを Automata SGX SDK を用いて Rust 言語で実装した．Automata SGX SDK は，Rust の標準ライブラリをエンクレーブ内で利用できるようにする Teaclave SGX SDK を基にしており，エンクレーブと非エンクレーブ側のアプリケーションを cargo で一括してビルドできる．

TPM の操作は libtss2-esys [14] が提供する API の Rust のラッパーである tss-esapi crate [15] を利用した．TPM とのやり取りは OCALL を介して非エンクレーブ側のホストアプリケーションが行う．5.2 節で述べた AEX の検出機能は実装中である．

6.1 AIK の登録

エンクレーブで動作している Roughtime サーバへの AIK の登録は次の手順で行う．

(1) エンクレーブの外で TPM2_CreatePrimary を実行して EK を取得し，その子として TPM2_Create で AIK を作成して，それらを引数として Roughtime サーバに与える．
(2) Roughtime サーバは，ランダムな値 c を生成し，EK と AIK を用いて TPM2_MakeCredential 相当の処理を行う．
(3) エンクレーブの外で TPM2_ActivateCredential を実行してチャレンジを復号し，その結果を エンクレーブに返す．
(4) エンクレーブで動作している Roughtime サーバは，返された値が c と一致すれば，AIK が EK と同じ TPM にあると判断し，AIK をシールして保存する．

6.2 Roughenough の移植

本研究では，Roughtime サーバの Rust 実装である roughenough [16] をエンクレーブ内に移植して用いた．元の roughenough は，時刻源として OS の提供する時刻を用いている．本研究では，それを TPM の稼働カウンタから算出した時刻 T に置き換え，応答の radius に誤差の上限 R を入れるように変更した．また，応答に署名に用いる鍵は初期化時に鍵登録サーバに登録したものを用いる．

6.3 タイムスタンプサービスの実装

実装した Roughtime サーバを利用するアプリケーションとして，本研究では RFC 3161 に基づくタイムスタンプサービス [17] を実装した．本研究では当初，RFC 3161 形式のトークンをエンクレーブ内で直接発行することを検討したが，そうはしなかった．その理由は，サービスをエンクレーブで動作させるためにリモートアテステーションで X.509 証明書の発行等の複雑な仕組みが必要になるためである．

そこで本研究では，オフライン時には Roughtime の nonce に文書のハッシュ値を入れ，Roughtime サーバに要求を送り，受け取った応答をファイルに保存するだけにした．そしてオンラインになった後にネットワーク上の変換サービスで RFC 3161 形式に変換することにした．この変換サービスの実装では，x509-tsp [18] を利用した．

6.4 X.509 証明書の検証

実装した Roughtime サーバを利用するアプリケーションとして，本研究では TLS 証明書の検証を実現した．具体的には Rust 言語で TLS を用いるためのクレート rustls を使用した．rustls では，証明書の検証を行う verify_server_cert() の引数に証明書検証に用いる時刻を与える必要がある．この時刻として，実装した Roughtime サーバから取得した時刻を用いるようにした．

7. 評価

7.1 実験に用いた Roughtime サーバ

実験では，キャリブレーション時の信頼できる時刻サーバとして Cloudflare の Roughtime サーバ（roughtime.cloudflare.com）を用いた．このサーバの radius は実験の間，常に 1 秒であった．このサーバとの ping による RTT は 3.2 ミリ秒であり，Roughtime の要求を送信してから応答を受信するまでの時間は約 4.2 ミリ秒であった．また，このサーバが返す時刻と NTP で同期したホストの時刻の差は，表 1 の A では 約 4 ミリ秒，B では 約 3 ミリ秒であった．

7.2 実装した時刻源の精度

実装した時刻源の精度を表 1 の TPM とホスト A および B で評価した．キャリブレーション間隔を 1 分，10 分，および 1 時間とし，それぞれ再キャリブレーションせずに 24 時間，1 分おきに時刻を要求した．返された時刻と NTP により同期された OS の時刻の差を誤差として測定した．

測定結果を図 10 と表 4 に示す．キャリブレーション間隔 t_{cal} を長くするほど OS の時刻との差は小さくなり，A ではキャリブレーション間隔を 1 時間としたとき，その差は 9 ミリ秒程度に収まった．ただし，アプリケー

ⓒ2026 Information Processing Society of Japan

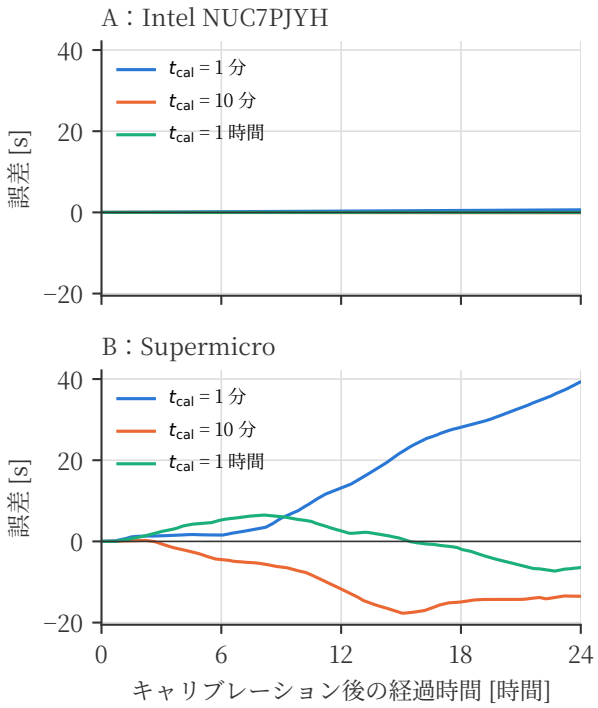

図 10 実装した時刻源の精度

表 4 オフラインで 24 時間時刻を提供したときの誤差と誤差の上限 R

機器	t_{cal}	差	24 時間後の差	24 時間後の R
A	1 分	0.63 s	0.63 s	3028 s
	10 分	94 ms	−92 ms	308 s
	1 時間	9.1 ms	−2.9 ms	52 s
B	1 分	39 s	39 s	2972 s
	10 分	18 s	−14 s	299 s
	1 時間	7.4 s	−6.5 s	51 s

ションが利用する誤差の上限 R は，52 秒であり，実際の誤差よりもかなり大きいものとなった．B は TPM の進み方の揺らぎが大きく，キャリブレーション間隔を長くしても秒単位の誤差が残った．アプリケーションが利用する誤差の上限 R は，51 秒であった．

8. 関連研究

T3E [6] は，TPM を用いた Intel SGX 向けの時刻源である．T3E はインターネット接続を前提とし，NTPSec サーバのような信頼できる時刻サーバと定期的に同期する．T3E はオフライン環境での利用を想定していない．また，T3E は提供する時刻の誤差の上限を示さず，TPM のクロックレートを操作する攻撃を想定していない．提案手法では，Roughtime プロトコルにより，誤差の上限を示し，また，クロックレートを操作する攻撃を検出できる．

Trusted Clock [19] は，x86 の動作モードであるシステムマネジメントモードを利用し，システムマネジメント割り込みを用いて RTC や HPET などのハードウェアクロックにアクセスする．しかし，この手法を利用するためにはファームウェアレベルの変更が必要であり，導入が容易ではない．提案手法は，TPM を利用できるマシンであれば容易に導入可能である．

TimeSeal [20] は，攻撃者によって変更できない信頼できるタイマーと，そのタイマーを補完するためのカウントスレッドを用いて時刻を提供する．TimeSeal は信頼できるタイマーとして `sgx_get_trusted_time` API を利用しているが，この API は 2020 年に廃止されている．

Triad [7] および TriHaRd [8] は，複数の TEE によって分散システムを稼働し，信頼できる時刻を提供する．Triad は，各 TEE が TSC を監視し，OS によってエンクレーブの実行が中断された場合にほかの TEE から時刻を取得し補正する．しかし，Triad には，遠隔の時刻源を用いて時刻同期を行う際に，遅延を挿入することで時刻を操作でき，その時刻がほかの TEE にも伝播するという課題が存在した．TriHaRd は，Triad のプロトコルを改良し，TEE 間の通信を時刻の一致の確認だけに用いて時計の調整には用いないことで，1 つの TEE の時刻のずれがほかの TEE に伝播することを防いだ．これらの手法は複数のノードを必要とする．提案手法は，単一のノードで利用できる点が異なる．

9. おわりに

本論文では，TPM 2.0 で利用可能な稼働カウンタを用いたオフライン環境向けの信頼できる時刻源の実装手法を提案した．提案手法は，Roughtime プロトコルに基づくサーバを Intel SGX エンクレーブで実行する．このサーバは，オンライン時に信頼できる時刻サーバを用いてキャリブレーションを行い，その後はオフラインで動作する．このサーバは，現在時刻とともに誤差の上限を返す．このサーバは，単一の PC で動作し，時刻が必要な時のみ CPU を消費するという特徴もある．その応用例として，この論文ではタイムスタンプサービスおよび X.509 証明書の検証を示した．

今後の課題は，AEX の検出機能を実装し，TPM 操作時間を精密に測定することである．

謝辞 この研究は部分的に，JSPS 科研費 21K19756 と 25K03104 の助成を受けた．

参考文献

[1] Intel: Intel Software Guard Extensions (SGX), `https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html`. アクセス日: 2026-10-13.
[2] Intel: Intel Trust Domain Extensions (TDX), `https://www.intel.com/content/www/us/`

ⓒ2026 Information Processing Society of Japan

en/products/docs/accelerator-engines/trust-domain-extensions.html. アクセス日: 2026-10-13.

[3] AMD: AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. アクセス日: 2026-10-13.

[4] ARM: Learn the architecture - TrustZone for AArch64, https://developer.arm.com/documentation/102418/0101/. アクセス日: 2026-10-13.

[5] Alder, F., Scopelliti, G., Van Bulck, J. and Mühlberg, J. T.: About Time: On the Challenges of Temporal Guarantees in Untrusted Environments, *Proceedings of the 6th Workshop on System Software for Trusted Execution*, SysTEX '23, New York, NY, USA, Association for Computing Machinery, pp. 27–33 (2023).

[6] Hamidy, G. M., Philippaerts, P. and Joosen, W.: T3E: A Practical Solution to Trusted Time in Secure Enclaves, *Network and System Security*, Cham, Springer Nature Switzerland, pp. 305–326 (2023).

[7] Fernandez, G., Brito, A. and Fetzer, C.: Triad: Trusted Timestamps in Untrusted Environments, *2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 169–176 (2023).

[8] Bettinger, M., Ben Mokhtar, S., Felber, P., Rivière, E., Schiavoni, V. and Simonet-Boulogne, A.: TriHaRd: Higher Resilience for TEE Trusted Time, *IEEE INFOCOM 2026 - IEEE Conference on Computer Communications*, IEEE, pp. 1–10 (2026).

[9] Trusted Computing Group: Trusted Platform Module 2.0 Library Specification, https://trustedcomputinggroup.org/resource/tpm-library-specification/. アクセス日: 2026-10-13.

[10] Trusted Computing Group: *Trusted Platform Module 2.0 Library, Part 1: Architecture*, Trusted Computing Group, version 185 edition (2026). https://trustedcomputinggroup.org/wp-content/uploads/Trusted-Platform-Module-2.0-Library-Part-1-Architecture_Version-185_pub.pdf.

[11] IETF: RFC 10049 - Roughtime: A Protocol for Rough Time Synchronization, https://datatracker.ietf.org/doc/rfc10049/. アクセス日: 2026-10-14.

[12] Constable, S., Bulck, J. V., Cheng, X., Xiao, Y., Xing, C., Alexandrovich, I., Kim, T., Piessens, F., Vij, M. and Silberstein, M.: AEX-Notify: Thwarting Precise Single-Stepping Attacks through Interrupt Awareness for Intel SGX Enclaves, *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA, USENIX Association, pp. 4051–4068 (2023).

[13] Trach, B., Faqeh, R., Oleksenko, O., Ozga, W., Bhatotia, P. and Fetzer, C.: T-Lease: a trusted lease primitive for distributed systems, *Proceedings of the 11th ACM Symposium on Cloud Computing*, SoCC '20, New York, NY, USA, Association for Computing Machinery, pp. 387–400 (2020).

[14] tpm2-software community: tpm2-tss: OSS Implementation of the TCG TPM2 Software Stack (TSS2), https://github.com/tpm2-software/tpm2-tss. アクセス日: 2026-10-14.

[15] Parsec Project: tss-esapi: Idiomatic Rust Wrapper for the TSS 2.0 Enhanced System API, https://github.com/parallaxsecond/rust-tss-esapi. アクセス日: 2026-10-14.

[16] int08h: roughenough: Rugged and performant Roughtime (RFC 10049) server and client implementation, https://github.com/int08h/roughenough. アクセス日: 2026-10-28.

[17] IETF: RFC3161 - Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP), https://datatracker.ietf.org/doc/html/rfc3161. アクセス日: 2026-10-28.

[18] crates.io: x509-tsp, https://crates.io/crates/x509-tsp/0.1.0. アクセス日: 2026-10-28.

[19] Liang, H. and Li, M.: Bring the Missing Jigsaw Back: TrustedClock for SGX Enclaves, *Proceedings of the 11th European Workshop on Systems Security*, EuroSec'18, New York, NY, USA, Association for Computing Machinery, pp. 1–6 (2018).

[20] Anwar, F. M., Garcia, L., Han, X. and Srivastava, M.: Securing Time in Untrusted Operating Systems with TimeSeal, *2019 IEEE Real-Time Systems Symposium (RTSS)*, pp. 80–92 (2019).

ⓒ2026 Information Processing Society of Japan