情報処理学会論文誌 Vol.53 No.9 2117–2129 (Sep. 2012)

TPMを用いたオフライン型タイムスタンプ

掛井 将平[1] 脇田 知彦[2] 毛利 公美[1] 白石 善明[2,a)] 野口 亮司[3]

受付日 2011年11月30日, 採録日 2012年6月1日

概要：電子署名に基づくタイムスタンプ・プロトコル（PKI TSP）がRFC3161で標準化されている．PKI TSPでは，信頼できる第三者機関であるTSA（Time Stamping Authority）が時刻認証を行う．このような時刻認証がTSAに依存するモデルでは，クライアントが時刻認証要求を出したときに通信ができないと時刻認証ができない．そこで本論文では，クライアントがTSAと通信できないときにも時刻認証ができるオフライン型タイムスタンプ方式を提案する．提案方式では，PKI TSPで定義されているTSAが時刻認証の権限をセキュリティチップTPM（Trusted Platform Module）を用いて認証した端末に委譲し，その端末がTPMの機能を用いて時刻認証を行う．認証された時刻の安全性について評価し，提案方式はTPMを用いることで時刻の改ざん・偽造が検知できることを示している．

キーワード：時刻認証，タイムスタンプ，オフライン型，TPM

Offline TimeStamp Using TPM

Shohei Kakei[1] Tomohiko Wakita[2] Masami Mohri[1]
Yoshiaki Shiraishi[2,a)] Ryoji Noguchi[3]

Received: November 30, 2011, Accepted: June 1, 2012

Abstract: Public Key Infrastructure Time-Stamp Protocol (PKI TSP) is standardized in RFC3161. In the PKI TSP, a TSA (Time Stamping Authority), which is a trusted third party, authenticates the time-stamp. In this model, the time-stamp authentication depends on the TSA, and a client must be in online status. This paper proposes an offline time-stamp scheme. The proposed scheme authenticates the time-stamp, even when a client cannot communicate with the TSA. The TSA delegates the authority of time-stamp authentication to a client which has been authenticated by a certification authority (CA) using a TPM (Trusted Platform Module). Then a client authenticates the time-stamp using the TPM. The paper also shows that the proposed scheme using the TPM can detect the falsification and forgery of the time.

Keywords: time-stamp authentication, time-stamp, offline-type, TPM

1. はじめに

電子データを証拠として扱う場合，「いつ」「誰が」「何を」の3要素が注目される[1]．この3要素のうち，「誰が」「何を」については電子署名により証明され，「いつ」は時刻認証により証明される．

電子データの時刻認証技術として，タイムスタンプ技術[1], [2]がある．これは，電子データに対して，信頼できる時刻が記されたタイムスタンプトークンを付与することで，時刻認証を行う技術である．タイムスタンプ技術には以下の要件が求められる．

［電子データの存在証明］ タイムスタンプトークンを付与した電子データがある時刻以前に存在していた，あるいは他の電子データとの順序関係を示すことができる．

［電子データの完全性証明］ タイムスタンプトークンを付与した時点から電子データが改ざんされた場合，これを検出できる．改ざんされていなければ，完全性が示される．

[1] 岐阜大学
Gifu University, Gifu 501–1193, Japan
[2] 名古屋工業大学
Nagoya Institute of Technology, Nagoya, Aichi 466–8555, Japan
[3] 株式会社豊通シスコム
Toyotsu Syscom Corp., Nagoya, Aichi 450–0002, Japan
[a)] zenmei@nitech.ac.jp

RFC3161 [3] では，電子署名に基づくタイムスタンプ・プロトコル（PKI TSP）が標準化されており，商用のタイムスタンプサービス [4], [5], [6], [7] で広く普及している．PKI TSP では，クライアントが信頼できる第三者機関（TTP）である TSA（Time Stamping Authority）にタイムスタンプトークンを要求し，発行してもらう．つまり，クライアントは TSA と通信することで時刻認証のサービスを受ける．

このモデルでは，端末がネットワークに接続していないオフライン状態の場合や災害などで TSA と通信ができない場合，時刻認証サービスを受けることができない．また，第三者が通信路で TSA との通信を観測することで，クライアントが時刻認証を行った事実を知られることになる．

本論文では，クライアントが TSA と通信できないときに時刻認証ができ，第三者に時刻認証の事実を知られないオフライン型タイムスタンプ方式を提案する．提案方式では，PKI TSP で定義されている TSA が時刻認証の権限を端末のクライアントに委譲し，そのクライアントがローカルで時刻認証を行う．タイムスタンプトークンに含まれる時刻情報の改ざん・偽造を防ぐためにセキュリティチップ TPM（Trusted Platform Module）を用いる．また，TPM を持たない第三者が正規のクライアントになりすますことを防ぐために，認証局（CA）が TPM に格納された鍵の公開鍵証明書を発行し，公開鍵証明書により TPM 搭載端末の認証を行う．

以下，2 章では RFC3161 準拠のタイムスタンプシステムと提案方式のコンセプトについて述べる．そして，提案方式のモデルを 3 章で示し，4 章でオフライン型タイムスタンプ方式を提案し，5 章でその実装について述べる．6 章で提案方式の安全性について述べ，7 章で提案方式の精度について述べる．最後に 8 章でまとめる．

2. RFC3161 準拠のタイムスタンプシステムと提案方式のコンセプト

2.1 RFC3161 準拠のタイムスタンプシステム

RFC3161 準拠のタイムスタンプシステムを図 **1** に示す．このシステムは，時刻認証要求を行うクライアントと時刻認証を行う TSA（TTP）の二者間モデルである．TSA は，内部時計を信頼できる時刻源（協定世界時や日本標準時など）と同期しており，この内部時計を用いて時刻認証を行う．

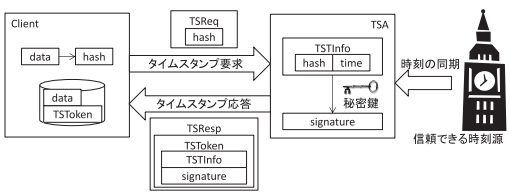


図 1 RFC3161 準拠のタイムスタンプシステム
Fig. 1 RFC3161-compliant TimeStamp System.

時刻認証の流れは次のようになる．Client は時刻認証対象の電子データのハッシュ値を計算して，これをタイムスタンプ要求 TSReq として TSA に送付する．TSA は，Client から受け取ったハッシュ値と信頼できる時刻からなる TSTInfo を生成し，TSTInfo の署名値を計算する．そして，TSTInfo と署名値からなるタイムスタンプトークン TSToken を生成し，これをタイムスタンプ応答 TSResp として，Client に送付する．Client は時刻認証対象の電子データと TSToken を一緒に保管する．

このシステムは構成がシンプルで，商用のタイムスタンプサービスにおいて広く普及している．しかし，TSA と通信できない場合に Client は時刻認証を受けることができない．また，TSA との通信を観測する第三者は時刻認証の事実を知ることになる．

2.2 提案方式のコンセプト

端末内で時刻認証ができれば，ネットワーク接続とは関係なく Client は時刻認証を受けることができ，第三者に時刻認証事実を知られることはなくなる．そこで，提案方式では，端末内にオフラインで時刻認証を行う TSA を置き，そして，PKI TSP の TSA から端末内の TSA に時刻認証権限の委譲を行うことで，オフラインでの時刻認証を実現する．認証局（CA）により発行された公開鍵証明書によって端末認証を行うことで，端末内で動作する TSA を把握し，不正なタイムスタンプトークンの発行を防ぐ．この端末認証とともに，認証された端末の利用者が不正に時刻認証をできないように，耐タンパ性のあるセキュリティチップ用いて，それを信頼点とした端末認証と時刻認証を行う．

3. 提案方式のモデル

図 **2** に提案方式のモデルを示す．本モデルは 5 つのエンティティにより構成される．

［認証局（CA）］ セキュリティチップに格納された鍵の公開鍵証明書を発行し，端末の正当性を保証する．CA は TTP とする．

［絶対時刻認証者（ATC: Absolute Time Certifier）］ PKI TSP で定義されている TSA と同一である．ATC は

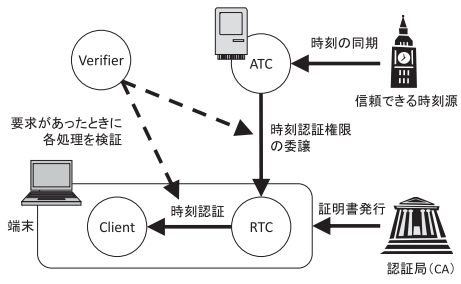


図 2 提案方式のモデル
Fig. 2 Model of the proposed scheme.

CA により認証された端末の内部で時刻認証を行う後述する RTC にのみ時刻認証権限を委譲する．ATC の内部時計は信頼できる時刻源（協定世界時や日本標準時などの絶対時刻）と同期しており，信頼できる絶対時刻を計測できる．ATC は TTP とする．

［相対時刻認証者(RTC: Relative Time Certifier)］ CA により認証されている端末内で時刻認証を行う TSA である．ATC から委譲された権限により，セキュリティチップを用いて時刻認証を行う．RTC の内部時計はセキュリティチップにより起点と時間間隔が保証された相対時刻（ある時点からの経過時間）を計測できる．RTC は不正をする可能性がある．

［時刻認証要求者（Client)］ RTC と同一の端末内にあり，RTC に対して時刻認証を要求する．

［検証者（Verifier)］ 時刻認証権限の委譲や時刻認証が適切に行われたかどうかを要求に応じて検証する．

提案方式は，ATC から時刻認証権限を委譲される RTC を端末内に配置することで，ATC と通信できないときにも時刻認証を受けることができる．また，RTC が Client と同一端末内にあることで，通信を観測する第三者に時刻認証の事実を知られることはなくなる．

4. オフライン型タイムスタンプ

本章では，時刻認証権限委譲処理，時刻認証処理，検証処理からなるオフライン型タイムスタンプ方式を提案する．提案方式では，時刻認証要求者の端末に搭載されたセキュリティチップとして TPM（Trusted Platform Module）[8], [9], [10] を利用する．

TPM とは，TCG (Trusted Computing Group) が策定した仕様に基づき，マザーボードに実装されているセキュリティチップである．耐タンパ性を持っており，TPM 内で実行される処理や TPM 内で保管されているデータは安全性が保証される．TPM には鍵保管機能があり，端末認証や署名計算の鍵を安全に保管できる．また，時間の間隔を保証する TickCounter 機能が備わっており，提案方式に適したハードウェアである．TickCounter 機能に対して，時間の経過とともに増加するカウンタ値 TCV (Tick Counter Value) と，TickCounter が起動してからリセットされるまでを 1 つのセッションとし，異なる TCV に対してセッションが同一であるかを確認するための識別子 TSN（Tick Session Nonce）などの値を TPM_CURRENT_TICKS と呼ばれる構造体で取得できる．以下では，TPM_CURRENT_TICKS を相対時刻を表す Ticks と呼ぶ．

ATC/RTC が持つ TSToken の生成/検証用の鍵ペアを**表 1** に示す．RTC は TPM の署名鍵である AIK（Attestation Identity Key）と SK（Signing Key）を利用する．AIK は TPM 内に格納された情報に対してのみ署名できる鍵で，SK は任意のデータに対して署名できる鍵である．

表 1 ATC/RTC の鍵ペア
Table 1 Key pair of ATC/RTC.

	署名鍵	検証鍵
ATC	K	K_verify
RTC（TPM）	AIK	AIK_verify
	SK	SK_verify

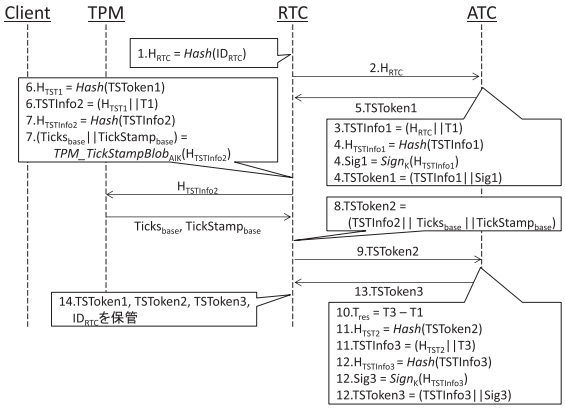


図 3 時刻認証権限委譲処理
Fig. 3 Time authentication delegation process.

以下で使用している記号 "‖" はデータの単純な連結を表す．たとえば，データ D1 とデータ D2 を連結したデータは D1‖D2 と表現する．

4.1 時刻認証権限委譲処理

RTC は時間間隔が保証された相対時刻を計測できるが絶対時刻を計測できない．そこで，時刻認証権限委譲処理では RTC が時刻認証処理で利用する絶対時刻を相対時刻から算出できるように，RTC の相対時刻を ATC の絶対時刻と同期する．ある時点での相対時刻に対する絶対時刻が分かれば，その時点からの経過時間から絶対時刻を算出して知ることができる．

ATC が RTC の相対時刻に対して時刻認証を行うことで，相対時刻と絶対時刻の同期を行う．RTC は権限の委譲を受けて，時刻認証のための絶対時刻を算出する．

図 3 に示した時刻認証権限委譲処理は次のようになる．

1. RTC は自身の識別情報 $\mathrm{ID_{RTC}}$ のハッシュ値 $\mathrm{H_{RTC}}$ を計算する．

$$\mathrm{H_{RTC}} = \mathit{Hash}(\mathrm{ID_{RTC}})$$

2. RTC は $\mathrm{H_{RTC}}$ を ATC に送付する．
3. ATC は RTC から受信した $\mathrm{H_{RTC}}$ と絶対時刻 T1 からなる TSTInfo1 を生成する．

$$\mathrm{TSTInfo1} = (\mathrm{H_{RTC}} \| \mathrm{T1})$$

4. ATC は TSTInfo1 のハッシュ値 $\mathrm{H_{TSTInfo1}}$ を計算し，署名鍵 K で $\mathrm{H_{TSTInfo1}}$ の署名値 Sig1 を計算し，TSTInfo1

と Sig1 からなる TSToken1 を生成する．

$$H_{TSTInfo1} = Hash(TSTInfo1)$$
$$Sig1 = Sign_K(H_{TSTInfo1})$$
$$TSToken1 = (TSTInfo1 \| Sig1)$$

5. ATC は TSToken1 を RTC に送付する．
6. RTC は ATC から受信した TSToken1 から TSTInfo1 に含まれている絶対時刻 T1 を取り出し，TSToken1 のハッシュ値 H_{TST1} を計算し，H_{TST1} と T1 からなる TSTInfo2 を生成する．

$$H_{TST1} = Hash(TSToken1)$$
$$TSTInfo2 = (H_{TST1} \| T1)$$

7. RTC は TPM に，署名鍵 AIK で TSTInfo2 と TPM の相対時刻 $Ticks_{base}$ を関連付けた時刻付き署名値 $TickStamp_{base}$ の計算を要求する．

$$H_{TSTInfo2} = Hash(TSTInfo2)$$
$$(Ticks_{base} \| TickStamp_{base}) = TPM\_TickStampBlob_{AIK}(H_{TSTInfo2})$$

$TPM\_TickStampBlob_{AIK}(H_{TSTInfo2})$ では，以下の処理が実行される．まず，$TickStamp_{base}$ が次のように計算される．入力された $H_{TSTInfo2}$ と TPM 内部で取得される相対時刻 $Ticks_{base}$ から TPM_SIGN_INFO を作成する．TPM_SIGN_INFO のハッシュ値 $H_{TPM\_SIGN\_INFO}$ を計算して，その時刻付き署名値 $TickStamp_{base}$ を計算する．

$$TPM\_SIGN\_INFO = (H_{TSTInfo2} \| Ticks_{base})$$
$$H_{TPM\_SIGN\_INFO} = Hash(TPM\_SIGN\_INFO)$$
$$TickStamp_{base} = Sign_{AIK}(H_{TPM\_SIGN\_INFO})$$

そして，相対時刻 $Ticks_{base}$ と $TickStamp_{base}$ を結合して出力する．

8. RTC は TSTInfo2 と $Ticks_{base}$ と $TickStamp_{base}$ を含む TSToken2 を生成する．

$$TSToken2 = (TSTInfo2 \| Ticks_{base} \| TickStamp_{base})$$

9. RTC は TSToken2 を ATC に送付する．
10. ATC は RTC から TSToken2 を受信し，現在の絶対時刻 T3 と TSToken1 を生成したときの絶対時刻 T1 から，RTC の応答時間 T_{res} を調べる．

$$T_{res} = T3 - T1$$

11. T_{res} が許容応答時間 T_{allow} 未満なら，ATC は TSToken2 のハッシュ値 H_{TST2} を計算して，H_{TST2} と T3 からなる TSTInfo3 を生成する．

$$H_{TST2} = Hash(TSToken2)$$
$$TSTInfo3 = (H_{TST2} \| T3)$$

もしも許容応答時間を超えていたら処理を中断し，RTC にエラーと通知する．

12. ATC は TSTInfo3 のハッシュ値 $H_{TSTInfo3}$ を計算し，署名鍵 K で $H_{TSTInfo3}$ の署名値 Sig3 を計算し，TSTInfo3 と Sig3 からなる TSToken3 を生成する．

$$H_{TSTInfo3} = Hash(TSTInfo3)$$
$$Sig3 = Sign_K(H_{TSTInfo3})$$
$$TSToken3 = (TSTInfo3 \| Sig3)$$

13. ATC は TSToken3 を RTC に送付する．
14. RTC は TSToken1, TSToken2, TSToken3, ID_{RTC} を一緒に保管する．

STEP7 の *TPM_TickStampBlob* [10] は，TPM 内部の相対時刻を用いて時刻付き署名値を計算する TPM 固有のコマンドである．ここでは，$H_{TSTInfo2}$ に対して TPM の相対時刻 $Ticks_{base}$ と相対時刻付き署名値 $TickStamp_{base}$ が出力される．$TickStamp_{base}$ は TPM に安全に格納された署名鍵 AIK で計算されるので改ざん・偽造はできない．ここで用いられる署名鍵 AIK に公開鍵証明書を発行 [11] することで端末認証ができる．

STEP11 では，RTC の応答時間のチェックを行う．TSToken2 の発行に時間がかかると STEP12 で行う相対時刻と絶対時刻のずれが大きくなり，適切に時刻の同期ができなくなる．そこで，許容応答時間を超過した場合，STEP12 以降の処理は行わず，RTC にエラーを返す．

STEP12 では，ATC が署名をして TSToken2 と T3 の関連付けをしている．これにより，RTC の相対時刻 $Ticks_{base}$ が ATC の絶対時刻 T3 と関連付けられる．$Ticks_{base}$ と T3 から絶対時刻を計算し，次に説明する時刻認証処理を行う．

4.2 時刻認証処理

RTC は時刻認証権限委譲処理で同期した時刻からの経過時間から現在の絶対時刻を算出し，その絶対時刻を用いて Client に対して時刻認証を行う．

図 4 に示した時刻認証処理は次のようになる．

1. 現在の $TSN_{current}$ と $Ticks_{base}$ に含まれている TSN_{base} が一致するか調べる．

$$TSN_{current} \stackrel{?}{=} TSN_{base}$$

一致すれば STEP2 を，そうでなければ時刻認証権限委譲処理を実行する．

2. Client は時刻認証対象の電子データ D のハッシュ値 H_D を計算する．

$$H_D = Hash(D)$$

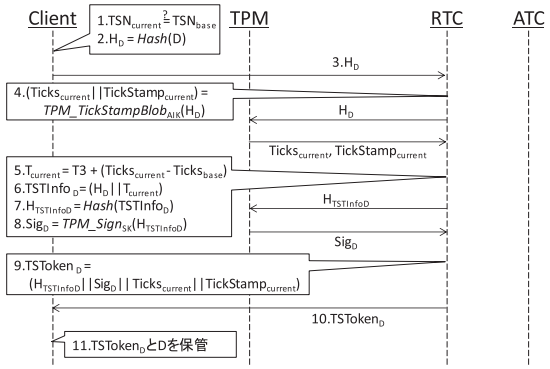


図 4 時刻認証処理
Fig. 4 Time authentication process.

3. Client は $\mathrm{H_D}$ を RTC に送付する．
4. RTC は Client から $\mathrm{H_D}$ を受信して，TPM に署名鍵 AIK で $\mathrm{H_D}$ と TPM 内部で取得される相対時刻 $\mathrm{Ticks_{current}}$ が関連付いた時刻付き署名値 $\mathrm{TickStamp_{current}}$ の計算を要求する．

$$(\mathrm{Ticks_{current}} \| \mathrm{TickStamp_{current}}) = \mathit{TPM\_TickStampBlob}_{\mathrm{AIK}}(\mathrm{H_D})$$

$\mathit{TPM\_TickStampBlob}_{\mathrm{AIK}}(\mathrm{H_D})$ では，以下の処理が実行される．まず，$\mathrm{TickStamp_{current}}$ が次のように計算される．入力された $\mathrm{H_D}$ と TPM 内部で取得される相対時刻 $\mathrm{Ticks_{current}}$ から TPM_SIGN_INFO を作成する．TPM_SIGN_INFO のハッシュ値 $\mathrm{H_{TPM\_SIGN\_INFO}}$ を計算して，その時刻付き署名値 $\mathrm{TickStamp_{current}}$ を計算する．

$$\mathrm{TPM\_SIGN\_INFO} = (\mathrm{H_D} \| \mathrm{Ticks_{current}})$$
$$\mathrm{H_{TPM\_SIGN\_INFO}} = \mathit{Hash}(\mathrm{TPM\_SIGN\_INFO})$$
$$\mathrm{TickStamp_{current}} = \mathit{Sign}_{\mathrm{AIK}}(\mathrm{H_{TPM\_SIGN\_INFO}})$$

そして，相対時刻 $\mathrm{Ticks_{current}}$ と $\mathrm{TickStamp_{current}}$ を結合して出力する．
5. RTC は権限委譲時の絶対時刻 T3 からの経過時間を $\mathrm{Ticks_{current}}$ と $\mathrm{Ticks_{base}}$ から計算して，現在の絶対時刻 $\mathrm{T_{current}}$ を求める．

$$\mathrm{T_{current}} = \mathrm{T3} + (\mathrm{Ticks_{current}} - \mathrm{Ticks_{base}})$$

6. RTC は $\mathrm{H_D}$ と $\mathrm{T_{current}}$ を含む $\mathrm{TSTInfo_D}$ を生成する．

$$\mathrm{TSTInfo_D} = (\mathrm{H_D} \| \mathrm{T_{current}})$$

7. RTC は $\mathrm{TSTInfo_D}$ のハッシュ値 $\mathrm{H_{TSTInfoD}}$ を計算する．

$$\mathrm{H_{TSTInfoD}} = \mathit{Hash}(\mathrm{TSTInfo_D})$$

8. RTC は TPM に，署名鍵 SK で $\mathrm{H_{TSTInfoD}}$ に対する署名値 $\mathrm{Sig_D}$ の計算を要求する．

$$\mathrm{Sig_D} = \mathit{TPM\_Sign}_{\mathrm{SK}}(\mathrm{H_{TSTInfoD}})$$

9. RTC は $\mathrm{TSTInfo_D}, \mathrm{Sig_D}, \mathrm{Ticks_{current}}, \mathrm{TickStamp_{current}}$ を含む $\mathrm{TSToken_D}$ を生成する．

$$\mathrm{TSToken_D} = (\mathrm{TSTInfo_D} \| \mathrm{Sig_D} \| \mathrm{Ticks_{current}} \| \mathrm{TickStamp_{current}})$$

10. RTC は $\mathrm{TSToken_D}$ を Client に送付する．
11. Client は $\mathrm{TSToken_D}$ と時刻認証対象データ D を一緒に保管する．

STEP5 では，現在の絶対時刻を求めている．RTC が持つ TPM の内部時計は時間間隔が保証されているが，現在の絶対時刻 $\mathrm{T_{current}}$ を知ることができない．そこで，$\mathrm{T_{current}}$ を知るために，権限委譲時に関連付けた $\mathrm{Ticks_{base}}$ と T3 を用いて現在の絶対時刻 $\mathrm{T_{current}}$ を求める．

4.3 検証処理

RTC の識別情報 $\mathrm{ID_{RTC}}$，時刻認証権限委譲処理で発行された TSToken1，TSToken2，TSToken3 と時刻認証処理で発行された $\mathrm{TSToken_D}$ から，各処理が適切に行われたか検証する．本処理は，Client の要求に応じて実行される．

宇根らが文献 [12], [13], [14] で示している TSToken の検証処理は 6 種類ある．提案方式では，宇根らが示した 6 種類の検証処理のうち，‘ハッシュ値の照合’，‘TSToken の一貫性確認’ を行っている．

ハッシュ値の照合とは，TSToken に含まれているハッシュ値とそのもととなるデータのハッシュ値が一致するか確認する処理で，TSToken の一貫性確認とは，TSToken に含まれている署名値で，TSToken の発行者と TSToken 発行後に TSToken が改ざんされていないことを確認する処理である．

Client は，$\mathrm{ID_{RTC}}$，電子データ D，TSToken1，TSToken2，TSToken3，$\mathrm{TSToken_D}$ を検証者に送付する．検証者は以下に示す処理を順番に実行する．

［RTC の確認］ 検証者は，TSToken1 に含まれているハッシュ値 $\mathrm{H_{RTC}}$ と $\mathrm{ID_{RTC}}$ のハッシュ値を比較することで，RTC の確認を行う．

$$\mathrm{H_{RTC}} \stackrel{?}{=} \mathit{Hash}(\mathrm{ID_{RTC}}) \quad (1)$$

［時刻認証権限委譲処理の検証］ 時刻認証権限委譲処理で発行された TSToken1，TSToken2，TSToken3 が改ざん・偽造をされていないか確かめるために，次の処理を行う．

$$\mathit{Verify}_{\mathrm{K\_vefiry}}(\mathrm{H_{TSTInfo1}}, \mathrm{Sig1}) \quad (2)$$
$$\mathrm{H_{TST1}} \stackrel{?}{=} \mathit{Hash}(\mathrm{TSToken1}) \quad (3)$$
$$\mathit{Verify}_{\mathrm{AIK\_verify}}(\mathit{Hash}(\mathrm{H_{TSTInfo2}} \| \mathrm{Ticks_{base}}), \mathrm{TickStamp_{base}}) \quad (4)$$
$$\mathrm{H_{TST2}} \stackrel{?}{=} \mathit{Hash}(\mathrm{TSToken2}) \quad (5)$$
$$\mathit{Verify}_{\mathrm{K\_verify}}(\mathrm{H_{TSTInfo3}}, \mathrm{Sig3}) \quad (6)$$

［時刻認証処理の検証］ 時刻認証処理で発行された $\mathrm{TSToken_D}$ が時刻認証対象データ D に対して作られており，署名の検証により改ざん・偽造の有無を確かめる．$\mathrm{TSToken_D}$ には，$\mathrm{TickStamp_{current}}$ と $\mathrm{Sig_D}$ の 2 種類の署名が含まれているので，次のようにそれぞれの署名を検証する．

$$\mathrm{H_D} \stackrel{?}{=} \mathit{Hash}(\mathrm{D}) \tag{7}$$

$$\mathit{Verify}_{\mathrm{AIK\_verify}}(\mathit{Hash}(\mathrm{H_D} \| \mathrm{Ticks_{current}}), \mathrm{TickStamp_{current}}) \tag{8}$$

$$\mathrm{T_{current}} \stackrel{?}{=} \mathrm{T3} + (\mathrm{Ticks_{current}} - \mathrm{Ticks_{base}}) \tag{9}$$

$$\mathit{Verify}_{\mathrm{SK\_verify}}(\mathrm{H_{TSTInfoD}}, \mathrm{Sig_D}) \tag{10}$$

5. 提案方式の実装

5.1 TPM による安全な時刻認証

TPM は，時刻認証権限委譲処理の STEP7，時刻認証処理の STEP4 と STEP8 で利用される．*TPM_Sign* と *TPM_TickStampBlob* は署名と時刻付き署名を得る次のようなコマンドであり，署名鍵を指定して実行する．署名鍵には，SK（SigningKey）と AIK（Attestation Identity Key）の 2 種類がある．SK は任意のデータに対して署名できる鍵である．AIK は TPM 内に格納された情報に対して署名できる鍵である．

［*TPM_Sign*］ 任意のデータに対する署名値を計算するコマンドである．このコマンドには SK が利用できる．しかし，AIK は利用できない．

［*TPM_TickStampBlob*］ 任意のハッシュ値に対して TPM 内で取得された現在の時刻情報 Ticks と関連付いた時刻付き署名値 TickStamp を計算するコマンドである．このコマンドには，SK と AIK が利用できる．

TPM_TickStampBlob で得られる TickStamp は TPM の相対時刻が付いた署名値なので，同じ署名鍵を用いて同じ相対時刻 Ticks とハッシュ値を *TPM_Sign* に入力して実行すれば同じデータを得ることができる．また，任意の時刻情報を入力できれば，過去や未来の時刻情報に対する TickStamp を計算できることになる．

提案方式では，TickStamp が現在の相対時刻 Ticks によるものであることを確認するために，*TPM_Sign* ではなく *TPM_TickStampBlob* から得たことを確認できなければならない．*TPM_TickStampBlob* で利用できて，*TPM_Sign* で利用できない署名鍵，すなわち，AIK を入力して実行することで，*TPM_TickStampBlob* から得られたことが確認できるようになっている．

5.2 性能評価

端末の OS は Windows7 Professional，CPU は Intel® Core™ i5 M560 2.67 GHz，メモリは 4.00 GB である．TPM は Infineon 製でバージョンは 1.2 である．開発言語として Java を用いた．タイムスタンプ関連の処理に IAIK TSP-2.1 [15]，TPM 関連の処理に IAIK jTSS-0.7 [16] を用いた．通信部分は HTTP で実装を行い，JSR311-1.1.1 [17] を用いて ATC を Web サーバとして実装した．

以上のような環境を構築し，提案方式の時刻認証権限委譲処理/時刻認証処理/検証処理にかかる時間を計測した．時刻認証権限委譲処理では 4.1 節の STEP1 から STEP13 を，時刻認証処理では 4.2 節の STEP1 から STEP10 を，検証処理では 4.3 節のすべての処理を，それぞれ 100 回行いその平均時間を測った．時刻認証対象のファイルとして 100 KB の PDF ファイルを使用した．

時刻認証権限委譲処理は約 805 ミリ秒，時刻認証処理は約 1,140 ミリ秒，検証処理は約 15 ミリ秒であった．上記の環境では 1 秒程度に 1 回の時刻認証ができることを確認した．

6. 提案方式の安全性評価

本章では，まず，時刻認証サービス利用事実の漏えいについて，次に，TPM 搭載端末の利用者での処理遅延による不正な時刻認証について，そして，時刻認証権限委譲処理で発行された TSToken1，TSToken2，TSToken3 と時刻認証処理で発行された $\mathrm{TSToken_D}$ の時刻認証対象の電子データのハッシュ値/時刻情報の偽造・改ざんについて評価を行う．

6.1 時刻認証サービス利用事実の漏えい

6.1.1 攻撃者の設定

［攻撃者］ 端末の利用者および ATC の運用者とは異なる第三者．

［攻撃の内容］ 端末と ATC の通信を観測する．

［攻撃の目的］ 時刻認証サービスの利用事実を知ること．

［攻撃における前提］ ATC と CA は信頼できる第三者機関であり攻撃に参加しない．

6.1.2 攻撃者の攻撃

提案方式では，ATC と通信を行うのは時刻認証権限委譲処理のときのみであり，時刻認証処理は端末内で実行される．つまり，攻撃者は端末と ATC の通信を観測しても，時刻認証権限委譲処理がいつ実行されたかを知ることはできるが，それ以降のどの時点で時刻認証をしたのかを知ることはできない．

しかし，時刻認証権限委譲処理が時刻認証処理と同じ頻度で実行されるような状況を作ることができれば，攻撃者は時刻認証の利用事実を推測できることになる．このような状況は，端末の TickCounter の TSN が変わったときと，ATC の公開鍵証明書の有効期限が切れたときに生じる．前者は TickCounter がリセットされたときに起こるが，端末の利用者ではない第三者が行うことは難しい．後者は ATC

に対して CA が短い有効期限の公開鍵証明書を発行することになるが，ATC と CA が信頼できる第三者機関であればこのようなことは起こらない．

以上のように，攻撃者は端末と ATC の通信を観測しても時刻認証サービスの利用事実を知ることができない．

6.2 遅延による不正な時刻認証

6.2.1 攻撃者の設定

［攻撃者］ TPM 搭載端末の利用者．

［攻撃の内容］ 時刻認証権限委譲処理で一時停止する．

［攻撃の目的］ 未来の時刻が含まれる $\mathrm{TSToken_D}$ を作ること．

［攻撃における前提］ ATC と CA は信頼できる第三者機関であり攻撃に参加しない．

6.2.2 攻撃者の攻撃

時刻認証権限委譲処理では，ATC が RTC から TSToken2 を受け取って，相対時刻と絶対時刻を同期する．攻撃者は，ATC への TSToken2 の送付（4.1 節の STEP9）を一時停止することで，攻撃者が処理に介入しない場合に同期されるはずの絶対時刻よりも未来の絶対時刻と相対時刻が同期されることになり，結果として時刻認証処理の STEP5 で未来の絶対時刻が算出される．

提案方式では，4.1 節の STEP10 で，ATC が RTC の応答時間 $\mathrm{T_{res}} = \mathrm{T3} - \mathrm{T1}$ を調べている．この時間が許容応答時間 $\mathrm{T_{allow}}$ を超えるとき，ATC は RTC にエラーと通知する．つまり，攻撃者は $\mathrm{T_{allow}}$ を大きくしなければ未来の絶対時刻を算出できない．

より未来の時刻で $\mathrm{TSToken_D}$ を作りたければ ATC が $\mathrm{T_{allow}}$ を大きくすることになるが，ATC が信頼できる第三者機関であればこのようなことは起こらない．つまり，攻撃者は未来の時刻が含まれる $\mathrm{TSToken_D}$ を作ることができない．

6.3 TSToken の偽造

6.3.1 攻撃者の設定

［攻撃者］ 2 種類の攻撃者を考える．TPM 非搭載端末の利用者（攻撃者 A）．TPM 搭載端末の利用者（攻撃者 B）．

［攻撃の内容］ TSToken の時刻認証対象データのハッシュ値や時刻情報の偽造を行う．元のデータ X に対して，偽造したデータを X’ で示す．

［攻撃の目的］ 攻撃を行ったことを，検証者に検知されないこと．

［攻撃における前提］ ハッシュ関数は第二原像探索が計算量的に困難であるとする．すなわち，1 つの TSToken に対して，異なる複数の時刻認証対象データを探索することが困難である．また，署名値は秘密鍵なしで計算できないとする．ATC と CA は信頼できる第三者機関であり攻撃に参加しない．

表 2 偽造攻撃のパターン
Table 2 Pattern of the forgery attack.

	攻撃者 A	攻撃者 B
TSToken1	A-1	B-1
TSToken2	A-2	B-2
TSToken3	A-3	B-3
$\mathrm{TSToken_D}$	A-4	B-4

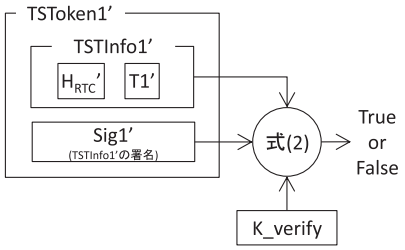


図 5 攻撃者 A による TSToken1’ の偽造の検知方法
Fig. 5 How to detect forgery TSToken1’ by an attacker-A.

6.3.2 攻撃者の攻撃

攻撃者 A と攻撃者 B の TSToken1，TSToken2，TSToken3，$\mathrm{TSToken_D}$ に対する偽造のパターンを**表 2** に示す．本項では，各パターンに対する偽造攻撃の方法とその検知の方法を示す．

6.3.2.1 攻撃者 A による TSToken1 の偽造（A-1）

TSToken1 には，4.1 節の Step4 に示すように，TSTInfo1 と署名値 Sig1 が含まれている．TSTInfo1 には，4.1 節の Step3 に示すように，$\mathrm{ID_{RTC}}$ のハッシュ値 $\mathrm{H_{RTC}}$ と絶対時刻 T1 が含まれている．攻撃者 A は，$\mathrm{H_{RTC}}$’，T1’ を偽造，すなわち，TSTInfo1’ を偽造し，それに合うように Sig1’ を作り，それらから TSToken1’ を作る．

このとき，攻撃者 A は TSTInfo1’ と整合的な Sig1’ を計算しようとする．しかし，Sig1 は ATC の署名鍵 K で計算されているので，ATC しか計算できない．つまり，**図 5** に示すように式 (2) で Sig1’ を ATC の検証鍵 K_verify で検証することで偽造を検知できる．

6.3.2.2 攻撃者 A による TSToken2 の偽造（A-2）

TSToken2 には，4.1 節の Step8 に示すように，TSTInfo2 と相対時刻 $\mathrm{Ticks_{base}}$ と署名値 $\mathrm{TickStamp_{base}}$ が含まれている．TSTInfo2 には，4.1 節の Step6 に示すように，TSToken1 のハッシュ値 $\mathrm{H_{TST1}}$ と絶対時刻 T1 が含まれている．攻撃者 A は，$\mathrm{H_{TST1}}$’，T1’ を偽造，すなわち，TSTInfo2’ を偽造，あわせて，$\mathrm{Ticks_{base}}$’ を偽造，それらに合うように $\mathrm{TickStamp_{base}}$’ を作り，それらから TSToken2’ を作る．

このとき，攻撃者 A は TSTInfo2’，$\mathrm{Ticks_{base}}$’ と整合的な $\mathrm{TickStamp_{base}}$’ を計算しようとする．しかし，$\mathrm{TickStamp_{base}}$ は TPM の署名鍵 AIK で計算されているので，TPM を持つ RTC しか計算できない．つまり，**図 6** に示すように，式 (4) で $\mathrm{TickStamp_{base}}$’ を RTC の検証鍵

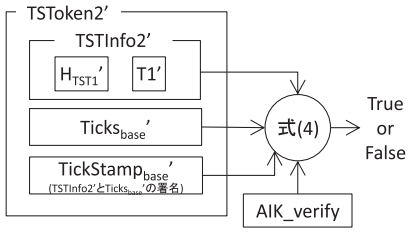


図 6 攻撃者 A による TSToken2' の偽造の検知方法
Fig. 6 How to detect forgery TSToken2' by an attacker-A.

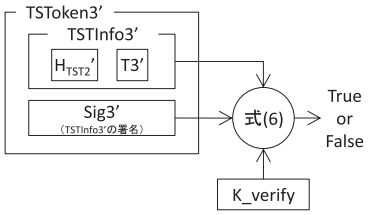


図 7 攻撃者 A による TSToken3' の偽造の検知方法
Fig. 7 How to detect forgery TSToken3' by an attacker-A.

AIK_verify で検証することで偽造を検知できる．

6.3.2.3 攻撃者 A による TSToken3 の偽造（A-3）

TSToken3 には，4.1 節の Step12 に示すように，TSTInfo3 と署名値 Sig3 が含まれている．TSTInfo3 には，4.1 節の Step11 に示すように，TSToken2 のハッシュ値 H_{TST2} と絶対時刻 T3 が含まれている．攻撃者 A は，H_{TST2}'，T3' を偽造，すなわち，TSTInfo3' を偽造し，それに合うように Sig3' を作り，それらから TSToken3' を作る．

このとき，攻撃者 A は TSTInfo3' と整合的な Sig3' を計算しようとする．しかし，Sig3 は ATC の署名鍵 K で計算されているので，ATC しか計算できない．つまり，**図 7** に示すように，式 (6) で Sig3' を ATC の検証鍵 K_verify で検証することで偽造を検知できる．

6.3.2.4 攻撃者 A による $TSToken_D$ の偽造（A-4）

$TSToken_D$ には，4.2 節の Step9 に示すように，$TSTInfo_D$ と署名値 Sig_D と相対時刻 $Ticks_{current}$ と署名値 $TickStamp_{current}$ が含まれている．$TSTInfo_D$ には，4.2 節の Step6 に示すように，時刻認証対象データ D のハッシュ値 H_D と絶対時刻 $T_{current}$ が含まれている．攻撃者 A は，H_D', $T_{current}$' を偽造，すなわち，$TSTInfo_D$' を偽造し，それに合うように Sig_D' を作る．また，$Ticks_{current}$' を偽造し，H_D' と $Ticks_{current}$' に合うように $TickStamp_{current}$' を作る．そして，これらから $TSToken_D$' を作る．

このとき，攻撃者 A は H_D'，$Ticks_{current}$' と整合的な $TickStamp_{current}$' を計算しようとする．しかし，$TickStamp_{current}$ は TPM の署名鍵 AIK で計算されているので，TPM を持つ RTC しか計算できない．つまり，**図 8** に示すように，式 (8) で $TickStamp_{current}$' を TPM の検証鍵 AIK_verify で検証することで偽造を検知できる．

また，攻撃者 A は $TSTInfo_D$' と整合的な Sig_D' を計算しようとするが，Sig_D は TPM の署名鍵 SK で計算されているので，TPM を持つ RTC しか計算できない．つまり，式 (10) で Sig_D' を TPM の署名鍵 SK_verify で検証することで偽造を検知できる．

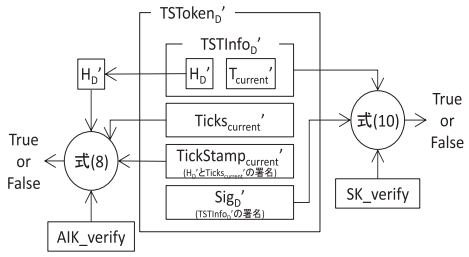


図 8 攻撃者 A による $TSToken_D$' の偽造検知の方法
Fig. 8 How to detect forgery $TSToken_D$' by an attacker-A.

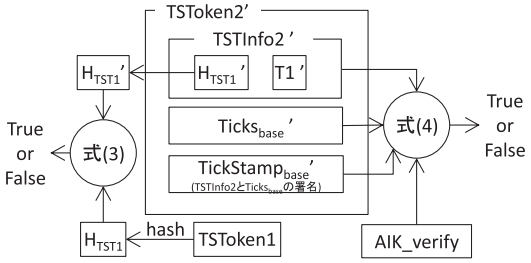


図 9 攻撃者 B による TSToken2' の偽造検知の方法
Fig. 9 How to detect forgery TSToken2' by an attacker-B.

6.3.2.5 攻撃者 B による TSToken1 の偽造（B-1）

TSToken1 は ATC が作るものであり，攻撃者 B が TPM を持つことはアドバンテージにならない．A-1 に示したように，式 (2) で攻撃者 B による TSToken1 の偽造を検知することができる．

6.3.2.6 攻撃者 B による TSToken2 の偽造（B-2）

TSToken2 には，4.1 節の Step8 に示すように，TSTInfo2，相対時刻 $Ticks_{base}$，署名値 $TickStamp_{base}$ が含まれている．TSTInfo2 には，4.1 節の Step6 に示すように，TSToken1 のハッシュ値 H_{TST1}，絶対時刻 T1 が含まれている．攻撃者 B は，H_{TST1}'，T1' を偽造，すなわち，TSTInfo2' を偽造，あわせて，$Ticks_{base}$' を偽造，それらに合うように $TickStamp_{base}$' を作り，それらから TSToken2' を作る．

攻撃者 B は TPM を持っているので，正しく TSToken2' を作ることができる．つまり，TSToken2' の作成者の確認だけでは TSToken2' の偽造を検知できない．そこで，H_{TST1}' と T1' と $Ticks_{base}$' が偽造されていないことを確認することで，正しい TSToken2 であることが分かる．

式 (1)，式 (2) で正しいと確認された TSToken1 に対して，**図 9** に示すように，まず，式 (3) で TSToken1 のハッシュ値と H_{TST1}' が一致するか確認する．T1' は，TSToken1 の T1 と同じであり，TSToken1 の T1 を確認することで，正しい T1 を知ることができる．次に，式 (4) で $Ticks_{base}$ 偽造されていないことを確認する．$Ticks_{base}$ を任意の相対時刻で偽造するには，それと整合がとれるような $TickStamp_{base}$' を計算しなければならない．この計算は TPM の署名鍵 SK と

TPM_Sign を利用するしかないが，提案方式では TPM の署名鍵 AIK と *TPM_TickStampBlob* を利用するので，任意の $\text{Ticks}_{\text{base}}$' と整合がとれた $\text{TickStamp}_{\text{base}}$' を計算できない．つまり，TPM の検証鍵 AIK_verify で $\text{TickStamp}_{\text{base}}$' を検証すれば，$\text{Ticks}_{\text{base}}$' が偽造されているか分かる．

ここで，$\text{TickStamp}_{\text{base}}$ は，$\text{H}_{\text{TSTInfo2}}$ と $\text{Ticks}_{\text{base}}$ からなる署名値であるので，攻撃者 B が $\text{H}_{\text{TSTInfo2}}$ を偽造して $\text{TickStamp}_{\text{base}}$ に合うように $\text{Ticks}_{\text{base}}$ を偽造する場合を考える．しかし，*TPM_TickStampBlob* では，$\text{H}_{\text{TSTInfo2}}$ と $\text{Ticks}_{\text{base}}$ から構造体 TPM_SIGN_INFO を作り，このハッシュ値に対して $\text{TickStamp}_{\text{base}}$ を計算する．

$$\text{TPM\_SIGN\_INFO} = (\text{H}_{\text{TSTInfo2}} \| \text{Ticks}_{\text{base}})$$

$$\text{H}_{\text{TPM\_SIGN\_INFO}} = \mathit{Hash}(\text{TPM\_SIGN\_INFO})$$

$$\text{TickStamp}_{\text{base}} = \mathit{Sign}_{\text{AIK}}(\text{H}_{\text{TPM\_SIGN\_INFO}})$$

ハッシュ値の第二原像探索が計算量的に困難であるという前提から，攻撃者 B は $\text{TickStamp}_{\text{base}}$ と整合的な $\text{H}_{\text{TSTInfo2}}$ と $\text{Ticks}_{\text{base}}$ の他の組を探索することはできない．

6.3.2.7 攻撃者 B による TSToken3 の偽造（B-3）

TSToken3 は ATC が作るものであり，攻撃者 B が TPM を持つことはアドバンテージにならない．A-3 に示したように，式 (6) を実行することで攻撃者 B による TSToken3 の偽造を検知することができる．

6.3.2.8 攻撃者 B による $\text{TSToken}_{\text{D}}$ の偽造（B-4）

$\text{TSToken}_{\text{D}}$ には，4.2 節の Step9 に示すように，$\text{TSTInfo}_{\text{D}}$ と署名値 Sig_{D} と相対時刻 $\text{Ticks}_{\text{current}}$ と署名値 $\text{TickStamp}_{\text{current}}$ が含まれている．$\text{TSTInfo}_{\text{D}}$ には，4.2 節の Step6 に示すように，時刻認証対象データ D のハッシュ値 H_{D} と絶対時刻 $\text{T}_{\text{current}}$ が含まれている．攻撃者 B は，H_{D}', $\text{T}_{\text{current}}$' を偽造，すなわち，$\text{TSTInfo}_{\text{D}}$' を偽造し，それに合うように Sig_{D}' を偽造する．また，$\text{Ticks}_{\text{current}}$' を偽造し，$\text{H}_{\text{D}}$' と $\text{Ticks}_{\text{current}}$' に合うように $\text{TickStamp}_{\text{current}}$' を作る．

攻撃者 B は TPM を持っているので，正しく $\text{TSToken}_{\text{D}}$' を作ることができる．つまり，$\text{TSToken}_{\text{D}}$' の作成者の確認だけでは $\text{TSToken}_{\text{D}}$' の偽造を検知できない．しかし，$\text{H}_{\text{D}}$', $\text{Ticks}_{\text{current}}$', $\text{T}_{\text{current}}$' が偽造されていないことを確認することで正しい $\text{TSToken}_{\text{D}}$ であることが分かる．

式 (1) から式 (6) で TSToken1，TSToken2，TSToken3 は正しいと確認されたうえで，**図 10** に示すように，まず，式 (7) で H_{D}' が時刻認証対象データ D のハッシュ値と一致するか確認する．次に，式 (8) で $\text{Ticks}_{\text{current}}$' が偽造されていないことを確認する．$\text{Ticks}_{\text{current}}$ を任意の相対時刻で偽造するには，それと整合がとれるような $\text{TickStamp}_{\text{current}}$' を計算しなければならない．この計算は TPM の署名鍵 SK と *TPM_Sign* を利用するしかないが，提案方式では TPM の署名鍵 AIK と *TPM_TickStampBlob* を利用するので，任意の $\text{Ticks}_{\text{current}}$' と整合がとれた $\text{TickStamp}_{\text{current}}$'

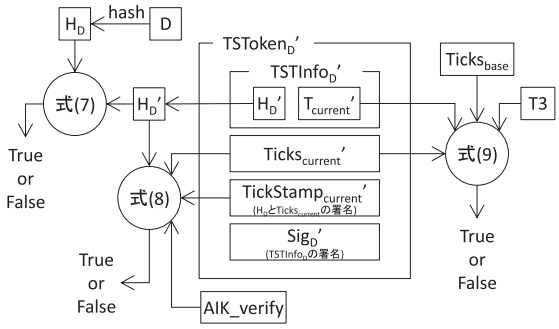


図 10 攻撃者 B による $\text{TSToken}_{\text{D}}$' の偽造検知の方法
Fig. 10 How to detect forgery $\text{TSToken}_{\text{D}}$' by an attacker-B.

を計算できない．つまり，TPM の検証鍵 AIK_verify で $\text{TickStamp}_{\text{current}}$' を検証すれば，$\text{Ticks}_{\text{current}}$' が偽造されているか分かる．最後に，式 (9) で $\text{T}_{\text{current}}$' が $\text{Ticks}_{\text{current}}$ と $\text{Ticks}_{\text{base}}$ と T3 から正しく計算されていることを確認する．

もう 1 つの署名 Sig_{D} に関して，攻撃者 B は TPM を持っているので，4.2 節の Step8 に示す，*TPM_Sign* を実行して Sig_{D} を計算できる．つまり，Sig_{D} は偽造検知に利用できない．

ここで，$\text{TickStamp}_{\text{current}}$ は，H_{D} と $\text{Ticks}_{\text{current}}$ からなる署名値であるので，攻撃者 B が H_{D} を改ざんして $\text{TickStamp}_{\text{current}}$ に合うように $\text{Ticks}_{\text{current}}$ を改ざんする場合を考える．しかし，*TPM_TickStampBlob* では，H_{D} と $\text{Ticks}_{\text{current}}$ から構造体 TPM_SIGN_INFO を作り，このハッシュ値に対して $\text{TickStamp}_{\text{current}}$ を計算する．

$$\text{TPM\_SIGN\_INFO} = (\text{H}_{\text{D}} \| \text{Ticks}_{\text{current}})$$

$$\text{H}_{\text{TPM\_SIGN\_INFO}} = \mathit{Hash}(\text{TPM\_SIGN\_INFO})$$

$$\text{TickStamp}_{\text{current}} = \mathit{Sign}_{\text{AIK}}(\text{H}_{\text{TPM\_SIGN\_INFO}})$$

ハッシュ値の第二原像探索が計算量的に困難であるという前提から，攻撃者 B は $\text{TickStamp}_{\text{current}}$ と整合的な H_{D} と $\text{Ticks}_{\text{current}}$ の他の組を探索することはできない．

6.3.3 TSToken の偽造のまとめ

以上の各パターンについて，攻撃の検知に関わる式を**表 3** にまとめた．

偽造攻撃では，攻撃者 A は ATC や TPM の署名鍵を持っていないことから，署名値の検証を行うだけで，偽造を検知できる．しかし，攻撃者 B による偽造攻撃では，TPM の署名鍵を利用できるので，署名値の検証だけでは攻撃を検知できない．そこで，署名値の計算に利用されたデータが正しいことを確認して攻撃を検知する．

表 3 に示したように，提案方式では 4.3 節の検証処理を実行することで，想定する偽造攻撃をすべて検知することができる．

表 3 偽造攻撃に対する安全性評価結果
Table 3 Security evaluation results for the forgery attack.

	攻撃者 A	攻撃者 B
TSToken1	(2)	(2)
TSToken2	(4)	(1)+(2)+(3)+(4)
TSToken3	(6)	(6)
$TSToken_D$	(8) (10)	(1)+(2)+(3)+(4)+(5)+(6)+(7)+(8)+(9)

表 4 改ざん攻撃のパターン
Table 4 Pattern of the tampering attack.

	攻撃者 A	攻撃者 B
TSToken1	A-5	B-5
TSToken2	A-6	B-6
TSToken3	A-7	B-7
$TSToken_D$	A-8	B-8

6.4 TSToken の改ざん

6.4.1 攻撃者の設定

［攻撃者］ 2 種類の攻撃者を考える．TPM 非搭載端末の利用者（攻撃者 A）．TPM 搭載端末の利用者（攻撃者 B）．

［攻撃の内容］ TSToken の時刻認証対象データのハッシュ値や時刻情報の改ざんを行う．元のデータ X に対して，改ざんしたデータを X” で示す．

［攻撃の目的］ 攻撃を行ったことを，検証者に検知されないこと．

［攻撃における前提］ ハッシュ関数は第二原像探索が計算量的に困難であるとする．すなわち，1 つの TSToken に対して，異なる複数の時刻認証対象データを探索することが困難である．また，署名値は秘密鍵なしで計算できないとする．ATC と CA は信頼できる第三者機関であり攻撃に参加しない．

6.4.2 攻撃者の攻撃

攻撃者 A と攻撃者 B の TSToken1，TSToken2，TSToken3，$TSToken_D$ に対する改ざんのパターンを**表 4** に示す．本項では，各パターンに対する改ざん攻撃の方法とその検知の方法を示す．本項で説明する改ざん攻撃は参考文献 [14] の攻撃と同じである．

6.4.2.1 攻撃者 A による TSToken1 の改ざん（A-5）

TSToken1 には，4.1 節の Step4 に示すように，TSTInfo1 と署名値 Sig1 が含まれている．TSTInfo1 には，4.1 節の Step3 に示すように，ID_{RTC} のハッシュ値 H_{RTC} と絶対時刻 T1 が含まれている．攻撃者 A は，H_{RTC}，T1 をそれぞれ H_{RTC}”，T1” に改ざんする．

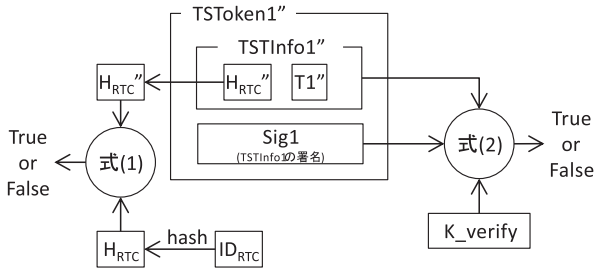


図 11 攻撃者 A による TSToken1” の改ざん検知の方法
Fig. 11 How to detect tampering TSToken1” by an attacker-A.

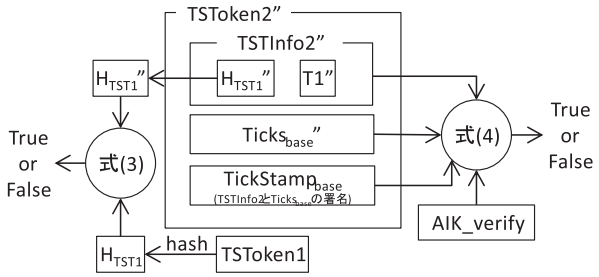


図 12 攻撃者 A による TSToken2” の改ざん検知の方法
Fig. 12 How to detect tampering TSToken2” by an attacker-A.

このとき，正しい H_{RTC}，T1 から TSToken1 が生成された後に改ざんされていないことを確認することで正しい TSToken1 であることが分かる．

図 11 に示すように，まず，式 (1) で H_{RTC}” が ID_{RTC} のハッシュ値と一致するか確認する．T1 は，ATC が付与するので，TSToken1 発行前に改ざんされることはない．次に，式 (2) で Sig1 の検証を行い，TSToken1 発行後に H_{RTC}”，T1” が改ざんされていないことを確認する．H_{RTC}”，T1” と改ざんする場合，それと整合がとれるように Sig1” を計算しなければならない．しかし，攻撃者 A は ATC の署名鍵 K を持っていないので Sig1” を計算できず，Sig1 を ATC の検証鍵 K_vefiry で検証することで，TSToken1 発行後に H_{RTC}”，T1” が改ざんされていないことを確認できる．

6.4.2.2 攻撃者 A による TSToken2 の改ざん（A-6）

TSToken2 には，4.1 節の Step8 に示すように，TSTInfo2 と相対時刻 $Ticks_{base}$ と署名値 $TickStamp_{base}$ が含まれている．TSTInfo2 には，4.1 節の Step6 に示すように，TSToken1 のハッシュ値 H_{TST1} と絶対時刻 T1 が含まれている．攻撃者 A は，H_{TST1}，T1，$Ticks_{base}$ をそれぞれ H_{TST1}”，T1”，$Ticks_{base}$” に改ざんする．

このとき，正しい H_{TST1} と T1 と $Ticks_{base}$ から TSToken2 が生成された後に改ざんされていないことを確認することで正しい TSToken2 であることが分かる．

式 (1)，式 (2) で TSToken1 は正しいと確認されたうえで，**図 12** に示すように，まず，式 (3) で TSToken1 のハッシュ値と H_{TST1}” が一致するか確認する．T1” は，TSToken1 の T1 と同じであり，TSToken1 の T1 を確認す

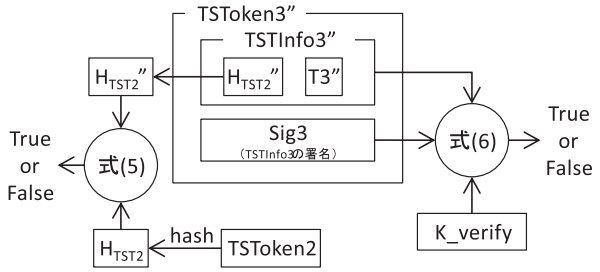


図 13 攻撃者 A による TSToken3” の改ざん検知の方法
Fig. 13 How to detect tampering TSToken3” by an attacker-A.

ることで正しい T1 を知ることができる．次に，式 (4) で $\text{Ticks}_{\text{base}}$” が TPM の署名鍵 AIK と *TPM_TickStampBlob* で取得されているか，$\text{TickStamp}_{\text{base}}$ を検証することで確認する．$\text{Ticks}_{\text{base}}$ を改ざんする場合，それと整合がとれるように $\text{TickStamp}_{\text{base}}$” を計算しなければならない．しかし，攻撃者 A は TPM の署名鍵 AIK を持っていないので $\text{TickStamp}_{\text{base}}$” を計算できず，$\text{TickStamp}_{\text{base}}$ を TPM の検証鍵 AIK_verify で検証することで，$\text{Ticks}_{\text{base}}$” の改ざんを検知できる．また，$\text{TickStamp}_{\text{base}}$ を検証することで，TSToken2 発行後に H_{TST1}”，T1”，$\text{Ticks}_{\text{base}}$” が改ざんされていないことを確認できる．

ここで，$\text{TickStamp}_{\text{base}}$ は，$\text{H}_{\text{TSTInfo2}}$，$\text{Ticks}_{\text{base}}$ からなる署名値であるので，攻撃者 A が $\text{H}_{\text{TSTInfo2}}$ を改ざんして $\text{TickStamp}_{\text{base}}$ に合うように $\text{Ticks}_{\text{base}}$ を改ざんする場合を考える．*TPM_TickStampBlob* では，$\text{H}_{\text{TSTInfo2}}$ と $\text{Ticks}_{\text{base}}$ から構造体 TPM_SIGN_INFO を作り，このハッシュ値に対して $\text{TickStamp}_{\text{base}}$ を計算する．

$$\text{TPM\_SIGN\_INFO} = (\text{H}_{\text{TSTInfo2}} \| \text{Ticks}_{\text{base}})$$

$$\text{H}_{\text{TPM\_SIGN\_INFO}} = \mathit{Hash}(\text{TPM\_SIGN\_INFO})$$

$$\text{TickStamp}_{\text{base}} = \mathit{Sign}_{\text{AIK}}(\text{H}_{\text{TPM\_SIGN\_INFO}})$$

ハッシュ値の第二原像探索が計算量的に困難であるという前提から，攻撃者 A は $\text{TickStamp}_{\text{base}}$ と整合的な $\text{H}_{\text{TSTInfo2}}$，$\text{Ticks}_{\text{base}}$ の他の組を探索することはできない．

6.4.2.3 攻撃者 A による TSToken3 の改ざん（A-7）

TSToken3 には，4.1 節の Step12 に示すように，TSTInfo3 と署名値 Sig3 が含まれている．TSTInfo3 には，4.1 節の Step11 に示すように，時刻認証対象データ TSToken2 のハッシュ値 H_{TST2} と絶対時刻 T3 が含まれている．攻撃者 A は，H_{TST2}，T3 をそれぞれ H_{TST2}”，T3” に改ざんする．

このとき，正しい H_{TST2}，T3 から TSToken3 が生成された後に改ざんされていないことを確認することで正しい TSToken3 であることが分かる．

式 (1) から式 (4) で TSToken1，TSToken2 は正しいと確認されたうえで，**図 13** に示すように，まず，式 (5) で TSToken2 のハッシュ値と H_{TST2}” が一致するか確認する．T3 は，ATC が付与するので，TSToken3 発行前に改ざんされることはない．次に，式 (6) で Sig3 の検証を行い，

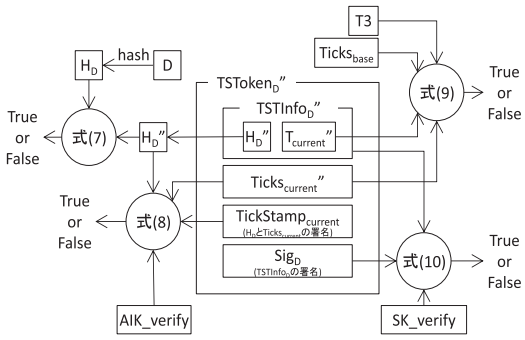


図 14 攻撃者 A による $\text{TSToken}_{\text{D}}$” の改ざん検知の方法
Fig. 14 How to detect tampering $\text{TSToken}_{\text{D}}$” by an attacker-A.

TSToken3 発行後に H_{TST2}”，T3” が改ざんされていないことを確認する．H_{TST2}”，T3” と改ざんする場合，それと整合がとれるように Sig3” を計算しなければならない．しかし，攻撃者 A は ATC の署名鍵 K を持っていないので Sig1” を計算できず，Sig3 を ATC の検証鍵 K_vefiry で検証することで，TSToken3 発行後に H_{TST2}”，T3” が改ざんされていないことを確認できる．

6.4.2.4 攻撃者 A による $\text{TSToken}_{\text{D}}$ の改ざん（A-8）

$\text{TSToken}_{\text{D}}$ には，4.2 節の Step9 に示すように，$\text{TSTInfo}_{\text{D}}$，署名値 Sig_{D}，相対時刻 $\text{Ticks}_{\text{current}}$，署名値 $\text{TickStamp}_{\text{current}}$ が含まれている．$\text{TSTInfo}_{\text{D}}$ には，4.2 節の Step6 に示すように，時刻認証対象データ D のハッシュ値 H_{D}，絶対時刻 $\text{T}_{\text{current}}$ が含まれている．攻撃者 A は H_{D}，$\text{Ticks}_{\text{current}}$，$\text{T}_{\text{current}}$ をそれぞれ H_{D}”，$\text{Ticks}_{\text{current}}$”，$\text{T}_{\text{current}}$” に改ざんする．

このとき，正しい H_{D}，$\text{Ticks}_{\text{current}}$，$\text{T}_{\text{current}}$ から $\text{TSToken}_{\text{D}}$ が生成された後に改ざんされていないことを確認することで正しい $\text{TSToken}_{\text{D}}$ であることが分かる．

式 (1) から式 (6) で TSToken1，TSToken2，TSToken3 は正しいと確認されたうえで，**図 14** に示すように，式 (7) で H_{D}” が時刻認証対象データ D のハッシュ値と一致するか確認する．次に，式 (8) で $\text{Ticks}_{\text{current}}$ が TPM の署名鍵 AIK と *TPM_TickStampBlob* で取得されているか確認する．$\text{Ticks}_{\text{current}}$ を改ざんする場合，それと整合がとれるように $\text{TickStamp}_{\text{current}}$” を計算しなければならない．しかし，攻撃者 A は TPM の署名鍵 AIK を持っていないので $\text{TickStamp}_{\text{current}}$” を計算できず，$\text{TickStamp}_{\text{current}}$ を TPM の検証鍵 AIK_verify で検証することで，$\text{Ticks}_{\text{current}}$” の改ざんを検知できる．そして，式 (9) で $\text{T}_{\text{current}}$” が $\text{Ticks}_{\text{current}}$ と $\text{Ticks}_{\text{base}}$ と T3 から正しく計算されていることを確認する．最後に，式 (10) で Sig_{D} の検証を行い，$\text{TSToken}_{\text{D}}$ 発行後に H_{D}” と $\text{Ticks}_{\text{current}}$” と $\text{T}_{\text{current}}$” が改ざんされていないことを確認する．

ここで，$\text{TickStamp}_{\text{current}}$ は，H_{D} と $\text{Ticks}_{\text{current}}$ からなる署名値であるので，攻撃者 A が H_{D} を改ざんして

表 5 改ざん攻撃に対する安全性評価結果
Table 5 Security evaluation results for the tampering attack.

	攻撃者 A	攻撃者 B
TSToken1	(1)+(2)	(1)+(2)
TSToken2	(1)+(2)+(3)+(4)	(1)+(2)+(3)+(4)
TSToken3	(1)+(2)+(3)+(4)+(5)+(6)	(1)+(2)+(3)+(4)+(5)+(6)
TSToken_D	(1)+(2)+(3)+(4)+(5)+(6)+(7)+(8)+(9)+(10)	(1)+(2)+(3)+(4)+(5)+(6)+(7)+(8)+(9)+(10)

$\text{TickStamp}_\text{current}$ に合うように $\text{Ticks}_\text{current}$ を改ざんする場合を考える．しかし，*TPM_TickStampBlob* では，H_D と $\text{Ticks}_\text{current}$ から構造体 TPM_SIGN_INFO を作り，このハッシュ値に対して $\text{TickStamp}_\text{current}$ を計算する．

$$\text{TPM\_SIGN\_INFO} = (\text{H}_\text{D} \| \text{Ticks}_\text{current})$$
$$\text{H}_\text{TPM\_SIGN\_INFO} = \mathit{Hash}(\text{TPM\_SIGN\_INFO})$$
$$\text{TickStamp}_\text{current} = \mathit{Sign}_\text{AIK}(\text{H}_\text{TPM\_SIGN\_INFO})$$

ハッシュ値の第二原像探索が計算量的に困難であるという前提から，攻撃者 A は $\text{TickStamp}_\text{current}$ と整合的な H_D と $\text{Ticks}_\text{current}$ の他の組を探索することはできない．

6.4.2.5 攻撃者 B による TSToken1 の改ざん（B-5）

TSToken1 は ATC が作るものであり，攻撃者 B が TPM を持つことはアドバンテージにならない．A-5 に示したように，式 (1)，式 (2) で攻撃者 A による TSToken1 の改ざんを検知することができる．

6.4.2.6 攻撃者 B による TSToken2 の改ざん（B-6）

攻撃者 B は TPM を持っているが，改ざんの対象となる TSToken2 の作成者の署名鍵 AIK を持っていないので，TPM を攻撃に利用できない．つまり，攻撃者 A と同じ状況なので，式 (1) から式 (4) で改ざんを検知できる．

6.4.2.7 攻撃者 B による TSToken3 の改ざん（B-7）

TSToken3 は ATC が作るものであり，攻撃者 B が TPM を持つことはアドバンテージにならない．A-7 に示したように，式 (1) から式 (6) で攻撃者 A による TSToken3 の改ざんを検知することができる．

6.4.2.8 攻撃者 B による TSToken_D の改ざん（B-8）

攻撃者 B は TPM を持っているが，改ざんの対象となる TSToken_D の作成者の署名鍵 AIK と署名鍵 SK を持っていないので，TPM を攻撃に利用できない．つまり，攻撃者 A と同じ状況なので，式 (1) から式 (10) で改ざんを検知できる．

6.4.3 TSToken の改ざんのまとめ

以上の各パターンについて，攻撃の検知に関わる式を**表 5** にまとめた．

改ざん攻撃では，他から発行された TSToken に対して攻撃を行うので，TPM を持っていることはアドバンテージにならない．つまり，攻撃者 A と攻撃者 B は同じ条件で攻撃を行うので，評価結果が同じになる．

表 5 に示したように，提案方式では 4.3 節の検証処理を実行することで，想定する改ざん攻撃をすべて検知することができる．

7. 提案方式の精度

提案方式では，時刻認証権限委譲処理の所要時間に応じて時刻認証処理で作成されるタイムスタンプの時刻に誤差が発生する．TSToken1 を生成した時刻 T1 から TSToken3 を生成した時刻 T3 までが時刻認証権限委譲処理の所要時間である．時刻認証処理で算出された T_current には，T3-T1 の誤差が含まれていることになる．この誤差の範囲内では，提案方式を利用した異なる端末の間での時刻の順序性を判断することができない．

T3-T1 は，4.1 節で述べたように許容応答時間 T_allow を上限としており，T_allow の範囲に誤差に収めることができる．5.2 節の実装環境では，T3-T1 は約 805 ミリ秒であった．このような値に ATC と端末間の 2 回の通信遅延時間を加えた値を運用時には T_allow に設定することとなる．

8. おわりに

本論文では，クライアントが TSA と通信できないときに時刻認証ができ，第三者に時刻認証の事実を知られない，TPM を用いたオフライン型タイムスタンプ方式を提案した．提案方式は，PKI TSP で定義された TSA が端末内の時刻認証クライアントに時刻認証権限を委譲することで外部と通信せずに時刻認証できる．提案方式は，RFC3161 準拠のタイムスタンプトークンフォーマットを利用しているので，既存のタイムスタンプサービスをもとに導入が容易な方式となっている．

不正な時刻認証を防ぐために，現在時刻と関連付いていることが保証された署名が作られなければならない．TPM で署名する場合，*TPM_Sign* コマンドと *TPM_TickStampBlob* コマンドがある．*TPM_Sign* では任意の時刻情報に対する署名が作れる．*TPM_TickStampBlob* は現在時刻と関連付いた署名を作れる．そこで，*TPM_Sign* には利用できず *TPM_TickStampBlob* に利用できる署名鍵 AIK を利用することで，*TPM_TickStampBlob* が実行されたことを確認する方式とした．提案方式の実装を行い，TPM 搭載端末で動作することを確認した．

提案方式で認証された時刻の安全性について評価したところ，TPM の機能を用いた端末認証と時刻認証により，不

正な端末の利用を防ぎ，任意の時刻による時刻認証ができないことを確認した．

提案方式で発行されたタイムスタンプの精度は許容応答時間と関係する．適切な許容応答時間を運用により明らかにすることは今後の課題としてあげられる．

参考文献

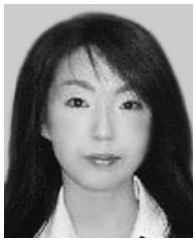
[1] 情報処理推進機構：タイムスタンプ・プロトコルに関する技術調査 (2004).
[2] 情報処理推進機構：タイムスタンプ技術に関する調査報告書 (2004).
[3] Adams, C., Cain, P., Pinkas, D. and Zuccherato, R.: Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP), RFC3161 (Aug. 2001).
[4] アマノビジネスソリューションズ株式会社：アマノタイムスタンプサービス 3161，入手先 ⟨http://www.e-timing.ne.jp/⟩（参照 2011-10-07）．
[5] PFU：PFU タイムスタンプサービス，入手先 ⟨http://www.pfu.fujitsu.com/tsa/⟩（参照 2011-10-07）．
[6] ドコモエンジニアリング北陸株式会社：e-DCM タイムスタンプサービス，入手先 ⟨http://dcmtsa.nttdocomo.co.jp/⟩（参照 2011-10-07）．
[7] セイコープレシジョン株式会社：SEIKO Cyber Time，入手先 ⟨http://www.seiko-cybertime.jp/⟩（参照 2011-10-07）．
[8] Trusted Computing Group: TPM Main Part1 Design Principles (Oct. 2003).
[9] Trusted Computing Group: TPM Main Part2 TPM Structures (Oct. 2003).
[10] Trusted Computing Group: TPM Main Part3 Commands (Oct. 2003).
[11] 篠田昭人，脇田知彦，福田洋治，毛利公美，白石善明，野口亮司：TPM に基づく端末認証のための公開鍵証明書発行支援，情報処理学会第 73 回全国大会講演論文集，6Y-8 (2011).
[12] 宇根正志，松本 勉：タイムスタンプの安全性と検証手続きとの関連性，2001 年暗号と情報セキュリティシンポジウム予稿集，pp.629–634 (Jan. 2001).
[13] 宇根正志，松本 勉：時刻・証拠付タイムスタンプの安全性と検証情報管理コスト，情報処理学会研究報告 CSEC, Vol.2001, No.15, pp.31–36 (Feb. 2001).
[14] 宇根正志，松本 勉：タイムスタンプ方式における 10 の安全性クラス，情報処理学会研究報告 CSEC, Vol.2001, No.75, pp.141–148 (July 2001).
[15] CRYPTO Toolkit TSP SIC, available from ⟨http://jce.iaik.tugraz.at/sic/Products/Public-Key-Infrastructure/TSP⟩ (accessed 2011-10-07).
[16] Trusted Computing for the Java(tm) Platform, IAIK, available from ⟨http://trustedjava.sourceforge.net/index.php?item=jtss/readme⟩ (accessed 2011-10-07).
[17] GlassFish JSR 311, Java.net, available from ⟨http://jsr311.java.net/⟩ (accessed 2011-10-07).

掛井 将平 （学生会員）

平成 23 年岐阜大学工学部応用情報学科卒業，同大学大学院博士前期課程入学，現在，在学中．時刻認証技術の研究に従事．

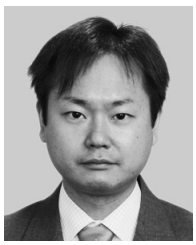
脇田 知彦

平成 22 年名古屋工業大学工学部情報工学科卒業，平成 24 年同大学大学院博士前期課程修了．同年（株）豊通シスコム．



毛利 公美

平成 5 年愛媛大学工学部情報工学科卒業．平成 7 年同大学大学院工学研究科情報工学専攻博士前期課程修了．平成 14 年博士（工学）（徳島大学）．平成 7 年香川短期大学助手．平成 10 年徳島大学工学部知能情報工学科助手，平成 15 年同講師．平成 19 年岐阜大学総合情報メディアセンター准教授．主に，コンピュータネットワーク，ネットワークセキュリティ，符号・暗号理論等の研究・教育に従事．



白石 善明 （正会員）

平成 7 年愛媛大学工学部情報工学科卒業．平成 9 年同大学大学院博士前期課程修了．平成 12 年徳島大学大学院博士後期課程修了．博士（工学）．平成 14 年近畿大学理工学部情報工学科講師．平成 18 年名古屋工業大学大学院情報工学専攻助教授，現在，准教授．情報セキュリティ，コンピュータネットワーク，教育支援，知識流通支援等の研究・教育に従事．平成 14 年電子情報通信学会オフィスシステム研究賞，平成 15 年暗号と情報セキュリティシンポジウム（SCIS）20 周年記念賞，平成 18 年 SCIS 論文賞．平成 19，20，23 年情報処理学会 DICOMO2007，2008，2011 優秀論文賞．

野口 亮司

（株）豊通シスコム取締役，アドバンス IT 事業部本部長．