5X-3

TPM を用いた二要素認証のための構成証明識別鍵による認証方式

篠田　昭人† 福田　洋治‡ 毛利　公美†† 白石　善明† 野口　亮司‡‡

†名古屋工業大学 ‡愛知教育大学 ††岐阜大学 ‡‡（株）豊通シスコム

1. はじめに

個人情報や機密情報を扱うなど強固なセキュリティが求められるサービスには，パスワードのみではなく複数の要素を使った認証（多要素認証）が推奨されている[1].

TPM （ Trusted Platform Module ） は TCG （ Trusted Computing Group）が仕様を策定する耐タンパー性を備えたセキュリティチップである．TPM は RSA 鍵を生成し，保護された領域で演算・保管ができる機能を持ち，別の端末に移して使うことができない[2].

AIK（Attestation Identity Key）は TPM で生成される RSA 鍵の一つである．AIK は端末の構成を証明する際の署名鍵として使われる[3]．AIK による署名は AIK を生成した TPM でのみ生成できる．TPM が別端末に移して使えないことから，AIK を用いて端末固有の署名を生成できることになる．

本研究では，TPM を搭載する端末を利用者の所有物として利用者認証の要素に使うことを考えている．

我々は TPM を二要素認証に使うために，AIK を利用者と紐づける AIK の公開鍵証明書を発行するフレームワークを提案している[4]．フレームワークに沿って発行される AIK の公開鍵証明書を使うことで，AIK の署名から利用者を特定でき，TPM を搭載する端末を利用者認証の要素の一つとすることができるようになる．

文献[5]には公開鍵ペアを用いたリモート認証の方式としてチャレンジアンドレスポンス方式が示されている．この方式では，被認証クライアントが，認証サーバから受け取った乱数に署名し，認証サーバが公開鍵証明書を使い署名を検証することで認証する．しかしながら，TCG の定める仕様により，AIK は署名可能なデータフォーマットが決まっており，そのままでは RSA 鍵を用いた認証に使うことができない．

本稿では，まず AIK を利用者認証に用いるための認証方式を提案する．次に，提案する認証方式を実装し，認証にかかる時間を計測する．

2. AIK を用いた認証方式の検討

2.1 公開鍵署名を用いたリモート認証

公開鍵署名を用いた認証の方式に，チャレンジアンドレスポンス方式がある．文献[5]では，IC カードに格納された秘密鍵の署名を利用した典型的なチャレンジアンドレスポンス方式の例が示されている．手順は図 1 のようになる．

Step 1. （クライアント）サーバに認証を要求する
Step 2. （サーバ）認証要求に対して，乱数 NONCE を生成してクライアントに送る
Step 3. （クライアント）サーバから受け取った NONCE を元に IC カードに対して署名を要求する
Step 4. （IC カード）NONCE に対してカード上で署名を施し，署名結果をクライアントに返す
Step 5. （クライアント）受け取った署名結果をサーバに返す
Step 6. （サーバ）NONCE に対する署名を，公開鍵証明書を使って検証する

An Authentication Scheme by Attestation Identity Key for Two-Factor Authentication Using TPM
† Akihito SHINODA and Yoshiaki SHIRAISHI・Nagoya Institute of Technology
‡ Youji FUKUTA・Aichi University of Education
†† Masami MOHRI・Gifu University
‡‡ Ryoji NOGUCHI・Toyotsu Syscom Corp.

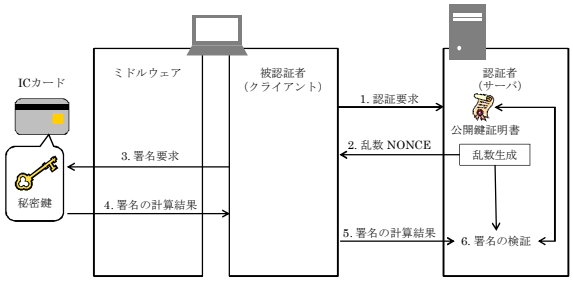

図 1　チャレンジアンドレスポンス方式によるリモート認証

	領域名	ビット列	ビット列の説明	提案方式での対応箇所
TPM_QUOTE_INFO	version	0x01 0x01 0x00 0x00 ・・・	固定値	cnonce
	fixed			
	digestValue	0xXX 0xXX ・・・ 0xXX	PCR値（端末の構成情報のハッシュ値）	
	externalData	0xXX 0xXX ・・・ 0xXX	任意の値	nonce

図 2　AIK の署名に使う構造体とその内容

この方式は，サーバが送信するチャレンジ（乱数）をもとにクライアントは秘密情報でレスポンス（署名）を作成し，サーバがレスポンスを検証することで認証する方式となっている．公開鍵署名を認証に使うには，クライアントは秘密鍵で乱数に署名する必要がある．

2.2 AIK が署名可能なデータフォーマット

前節で示した方式は，秘密鍵を公開することなく秘密情報を持っていることを証明でき認証に使える．しかし AIK が署名できるデータのフォーマットは定められているので，AIK を方式にそのまま適用することはできない．以下で詳しく説明する．

AIK の署名は TPM のコマンド TPM_Quote で生成できる．TPM_Quote は TPM 内に記録された端末の構成情報が，確かに TPM で生成されたものであることを証明するために使われる．端末の構成情報を含める構造体 TPM_QUOTE_INFO は図 2 のようなフォーマットになっている．version，fixed 領域は固定のビット列を格納する．digestValue 領域には，TPM 内に記録されている端末の構成情報を入れる．この値は TPM 内の PCR（Platform Configuration Register）というレジスタに保管されている値を使うため TPM 外のデータを指定することはできない．externalData はリプレイ攻撃対策と，完全性証明のために使われる[6]．コマンド実行ごとに異なる値を入れ署名を生成すれば，認証者はリプレイ攻撃やデータの改ざんを検知できる．

以上のように AIK は任意のデータに対する署名を生成することができない．そこで externalData 領域にチャレンジアンドレスポンス方式の乱数を格納して署名することにした．

2.3 認証サーバの処理の軽減

認証システムを大規模なサービスにおいても実用に耐えうるようにするには，サーバの処理はできるだけ簡素であることが望ましい．2.1 節のチャレンジアンドレスポンス方式をそのまま採用することを考えると，Step 6. でのレスポンス検証のために Step 2. でクライアントに送信した乱数をサーバが保持して

おく必要がある．その場合，クライアントごとに送信した乱数を記録するか，サーバがあるクライアントの認証処理をしているときには他のクライアントの認証要求を受け付けないといった排他制御をする必要があり，処理が煩雑になる．

次のようにすれば，乱数を保持せずにサーバがレスポンスを検証できるようになる．まず，サーバは乱数と，乱数をサーバのみが復号できる形で暗号化したものをチャレンジに含めてクライアントに送る．クライアントはチャレンジに含まれる乱数に対する署名を生成し，暗号化された乱数をそのままレスポンスに含めてサーバに返す．サーバはレスポンスに含まれる暗号化された乱数を復号し署名の検証に使うことで，乱数の保持の必要がなくなる． クライアントが乱数を改ざんしても，暗号化された乱数を復号したものを署名の検証に使うことで検知できる．

また，敵にチャレンジとレスポンスの組が流出しても解析するのに十分な時間を与えないように，チャレンジに有効期限を設ける．有効期限を設けることで，受動的攻撃による不正アクセスを試みる敵の対策ができる．

以上の検討を踏まえ，AIK を用いた認証方式を次章で示す．

3. AIK を用いた利用者認証方式の提案

AIK を用いた利用者認証方式を図 3 に示し以下に手順を説明する．サーバが認証者，クライアントが被認証者である．なお，記号‖はビット列の結合を表す．

Step 1. （クライアント）サーバに認証を要求する

Step 2. （サーバ）乱数 nonce と有効期限 limit を生成し，nonce ‖ limit をサーバの持つ対称鍵で暗号化した token を作成する

Step 3. （サーバ）Challenge={nonce, token}をクライアントに返す

Step 4. （クライアント）乱数 cnonce を生成し cnonce ‖ nonce に TPM の AIK で署名した signature を生成する

Step 5. （クライアント）Response={token, cnonce, signature}をサーバに送信する

Step 6. （サーバ）token を復号して nonce と limit を得る．現在時刻が limit を過ぎていないか検証し，有効期限内の場合に，signature を cnonce ‖ nonce と AIK の公開鍵証明書で検証する

Step 7. （サーバ）Step 6. の検証結果をクライアントに返す

図 2 に示すように，cnonce は PCR に保管されている端末の構成情報のハッシュ値を利用する．

この方式では，認証者が生成した乱数と有効期限をトークンとして被認証者に渡し，レスポンスにトークンを含めることで認証者が乱数を保持する必要がなくなる．トークンは認証者が持つ対称鍵で暗号化されているため，認証者以外が作ったトークンや書き換えられたトークンであれば正しく復号できず認証者は改ざんを検知できる．

4. 認証システムの実装と性能評価

3 章で提案した方式を，表 1 の環境で実装した．TPM を扱うためのソフトウェアスタック TSS（TCG Software Stack）には Java で実装されている IAIK の jTSS[7]を用いている．

性能評価のためにクライアントが認証要求を出して，サーバから認証結果を受け取るまでの時間を計測した．サーバはクライアントと別の端末に構築し LAN ケーブルで直接つないで実験した． 5 回試行して，結果は平均 3650.1ms であった．このうち約 3316.1ms は，3 章 Step 4. の AIK による署名生成に費やされている．端末に比べ TPM の演算能力が著しく低いため，クライアント側の処理時間は認証手続き全体の約 90.8%を占めた．一方のサーバ側の処理時間は Step 2. のチャレンジ生成と Step 6．のレスポンス検証でそれぞれ 4.3ms，7.9ms であった．

端末が認証手続きをするのはシステムにログインする際の 1 回のみのであるので 3650.1ms の処理時間は実用的な時間と考えられる．また，サーバは複数の認証要求を処理するが，この処理時間では，実運用に耐えうると考える．

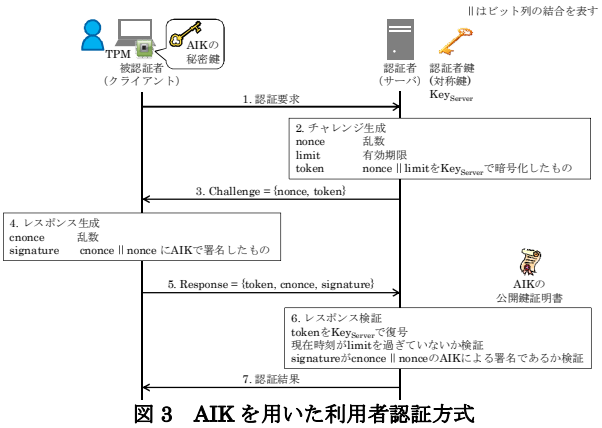

図 3　AIK を用いた利用者認証方式

表 1　実装環境

CPU	Intel Core i5 M560 2.67GHz	
Memory	4.00GB	
OS	Windows 7 Professional	
PC	hp probook 6550b (TPM搭載)	
TPM	ver.1.2 Infineon製	
サーバ	GlassFish Server3.1.1	
言語	Java 1.7.0_05	
ライブラリ	サーバ	Jersey1.3
	クライアント	Apache HttpClient 4.2
	TPM, 暗号処理	IAIK jTSS 0.7[7], IAIK TCcert 0.2.5, IAIK JCE 4.0, IAIK CMS 4.1

5. おわりに

本稿では，TPM を利用者認証の要素の一つとするために，TPM で生成され，端末の構成証明に使われる RSA 鍵の AIK を使った認証方式を提案した．提案した認証方式を実装し，性能評価した結果，認証にかかる時間は平均 3650.1ms となることより実利用できる範囲であると考える．また，認証サーバは乱数を記憶する必要がなく処理を簡素にできた．認証サーバの通信以外にかかる処理時間は合計で 12ms 程度であることより大規模な環境での利用を見込める．

参考文献

[1] 八木哲志，山田慈朗，上野磯生，高杉英利：企業向け多要素認証プラットフォームの設計，情報処理学会デジタルプラクティス，Vol.3，No.2，pp.147-154（2012）．

[2] 中村智久，東川淳紀：PC 搭載セキュリティチップ(TPM)の概要と最新動向，情報処理，Vol.47，No.5，pp.473-478（2006）．

[3] 上杉忠興，坪毅，宗藤誠治，吉濱佐知子：Trusted Network Connect：TPM の利用管理技術の動向，情報処理，Vol.48，No.11，pp.1232-1241（2007）．

[4] 篠田昭人，毛利公美，白石善明，野口亮司：TPM を用いた二要素認証のための AIK 証明書の発行フレームワークとその支援システム，信学技報，Vol.112, No. 315, ICSS2012-52, pp. 43-48（2012）．

[5] IPA：IC・ID カードの相互運用可能性の向上に係る基礎調査ニーズ編報告書（オンライン），入手先 <http://www.ipa.go.jp/security/fy18/reports/ICID>（2007）．

[6] Trusted Computing Group: TPM Main Specification Level 2 Version 1.2, Revision 116 (online), available from <http://www.trustedcomputinggroup.org/resources/tpm_main_specification> (accessed 2013-01-05).

[7] Institute for Applied Information Processing and Communications (IAIK): Trusted Computing for the Java™ Platform (online), available from <http://trustedjava.sourceforge.net/> (accessed 2013-01-05).