6Z-6

TPM を用いた Virtual Monotonic Counter の階層型接続による順序認証システムと Java によるライブラリ

掛井将平† 毛利公美† 白石善明‡ 野口亮司††

†岐阜大学 ‡名古屋工業大学 ††(株)豊通シスコム

1. はじめに

タイムスタンプ技術により，どのデータが先に存在していたかといった前後関係を証明できる．しかし，時刻情報だけでは連続したデータの並びを示す順序関係を証明できない．例えば，ログの解析では，どのような順番でどのような処理が実行されたかの順序関係が必要になるときがある．ログにタイムスタンプが付されていたとしても，解析対象のログが連続したものであることを確認する手段がない．つまり，ログの紛失や全て揃っているかの確認ができない．

順序関係を証明する単純な方法として時刻情報ではなく通し番号を付すことが考えられる．文献[1]では，Trusted Platform Module（TPM）に搭載された単調増加なカウンター値を出力する Monotonic Counter を仮想的に複数個に拡張した Virtual Monotonic Counter（VMC）が提案されている．TPM とは耐タンパ性を備えたセキュリティチップである．TPM から取り出すことのできない署名鍵 *AIK* により，TPM に由来していることが第三者に対して保証された署名値を生成できる[2].

VMC では，TPM のコマンドのログを記録する TransportSession 機能によりカウンター値を *AIK* で署名することで TPM から出力された値であることを保証している．VMC にはカウンター値を 1 増加させるインクリメントと読み出しを行うコマンド *IncAndSignClock* と *ReadAndSignClock* がある．カウンター値を過去の値に戻すようなことはできず，値の改ざん・偽造は *AIK* の署名の検証により検知できる．

この VMC を用いれば，信頼できる第三者が要求に応じて通し番号を付与していくような方式が考えられるが，集中型モデルの制約を受けることになる．もれや抜けのない順序認証をするには，必要なときに順序認証を要求できることが望ましい．

本稿では TPM を用いた Virtual Monotonic Counter の階層型接続による順序認証システムを提案する．下位の VMC がデータの順序関係を証明し，上位の VMC が異なる下位の VMC のカウンター値の前後関係を証明する“順序交差”という考え方をシステムに導入している．提案システムの実装には，TPM を利用するためのポリシーの設定や TPM で扱うデータ構造などの理解が必要である．システムの実装を支援するために TPM の利用部分を隠蔽した Java による順序認証ライブラリを開発した．

2. 順序認証と順序交差

2.1. 順序認証

図 1 に VMC の *IncAndSignClock* を用いた順序認証処理 *OrderStamping* を示す．VMC の識別子 *ctrID*，順序認証対象データ *D* のハッシュ値 H_D，カウンター値 *ctrVal*，時間情報 *Ticks*，*IncAndSignClock* の入出力のログ *log*，*log* の署名値 sig_{log} を含む *IncCert* が VMC ごとのデータの順序関係を保証する．

2.2. 順序交差

VMC のカウンター値は各 TPM において固有のものなので異なる端末間では単純にカウンター値を比較できない．下位端末のカウンター値 *LCV*（*Local Counter Value*）に上位端末のカウンター値 *GCV*（*Global Counter Value*）を対応付けられれば *GCV* を基準に異なる端末間で *LCV* を比較できるようになる．

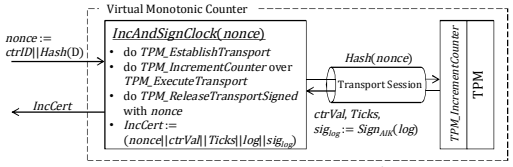

図 1 *IncAndSignClock* による順序認証

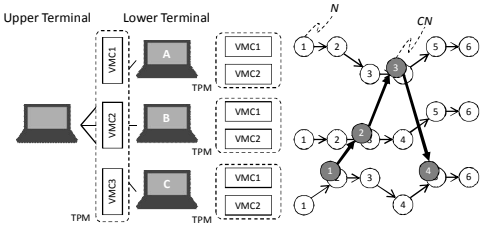

図 2 順序交差

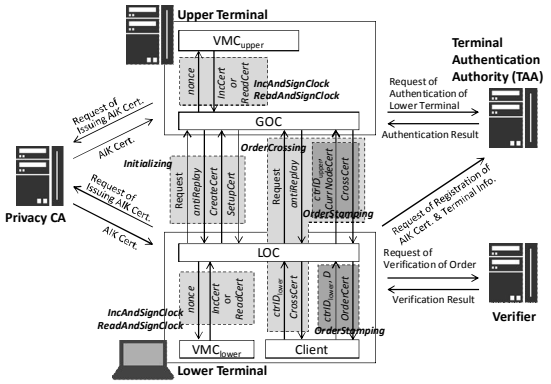

図 3 順序認証システムのモデル

順序交差の様子を図 2 に示す．下位端末がデータに *LCV* を割り当てた点を“ノード”と呼び *N* と表す．上位端末が *N* に *GCV* を割り当てた点を“クロスノード”と呼び，*CN* と表す．*CN* と *N* が重なった点が *GCV* と *LCV* が対応付いた点である．

3. 順序認証システム

3.1. 構成要素

図 3 に順序認証システムのモデルを示す．

[Privacy CA] Trusted Computing Group（TCG）[3]が定めたプロトコルに従い，*AIK* の公開鍵証明書 *AIKCert* を発行する信頼できる第三者機関である．*AIK* は *AIKCert* 発行後に利用できるようになっている．

[TAA（Terminal Authentication Authority）] 端末の識別情報と *AIKCert* を紐付けて保管する信頼できる第三者機関である．*AIK* による署名を検証することでどの端末が生成した署名か知ることができる．

[LOC（Local Order Certifier）] データの順序認証を行う．さらに，後述する GOC に対して VMC の生成順序交差を要求する．事前に Privacy CA から *AIKCert* を受け取り，TAA に登録しておく．

[Client] LOC に順序認証，順序交差を要求する．

[GOC（Global Order Certifier）] TAA により認証された LOC の

Order-Stamping System by Hierarchically Connecting Virtual Monotonic Counter Using TPM and its Java Library

† Shohei KAKEI and Masami MOHRI・Gifu University

‡ Yoshiaki SHIRAISHI・Nagoya Institute of Technology

†† Ryoji NOGUCHI・Toyotsu Syscom Corp.

要求に応じて順序交差を行い，異なる端末間でのデータの前後関係を保証する．TAA により TPM 搭載端末の LOC の順序交差要求のみ受け付ける．事前に Privacy CA から *AIKCert* を受け取っておく．

[Verifier] Client の要求に応じて異なる端末で順序認証されたデータの前後関係を比較する．

3.2. トランザクション

AIKCert の発行，TAA への端末情報と *AIKCert* を TAA に登録したうえで以下の処理を実行する．

[順序認証開始処理 *Initializing*] 上位の VMC と下位の VMC が生成される．下位の VMC 生成時のカウンター値 $ctrVal_{LOC}$ を保証する *CreateCert* に対して上位の VMC で順序交差することで順序関係の開始のノードを保証する *SetupCert* が作成される．

[順序認証処理 *OrderStamping*] Client が指定する VMC に対して LOC が *IncAndSignClock* を実行し，順序認証対象データ *D* の順序を表すカウンター値 $ctrVal_{LOC}$ を保証する証明書 *OrderCert* が出力される．

[順序交差処理 *OrderCrossing*] Client が指定する VMC のカウンター値 $ctrVal_{LOC}$ を保証する *CurrentNodeCert* に対して GOC が *IncAndSignClock* を実行し，$ctrVal_{LOC}$ と上位の VMC のカウンター値 $ctrVal_{GOC}$ を保証する証明書 *CrossCert* が出力される．

[前後関係比較処理 *Comparing*] 順序認証処理，順序交差処理で生成された証明書 *OrderCert*，*CrossCert* に含まれる *LCV*，*GCV* から異なる端末間でのノードの前後関係を比較する．単純にカウンター値の比較だけでは，前後関係を比較できない状況がある．そのときは，*OrderCert*，*CrossCert* に含まれている *Ticks* からノード間の時間差を計算することでノードの前後関係を比較できる．

4. 開発したライブラリ

4.1. ライブラリの機能

図 4 に開発したライブラリのクラス構成を示す．網掛け部分がライブラリに含まれるクラスである．TPM を Java で利用するために IAIK jTSS 0.7a[4]を，ASN.1 ベースの構造体の作成に IAIK JCE 4.0[4]を用いた．開発者はライブラリが提供するクラスを用いて *Initializing*，*OrderStamping*，*OrderCrossing*，*Comparing* を実装する．

以下に本ライブラリが提供する代表的なクラスを示す．

[TpmCounter クラス] TPM の Monotonic Counter のインクリメント／カウンター値の取得を行う．

[VMC クラス] TpmCounter クラスを利用して，VMC の生成／*IncAndSignClock* の実行を行う．

[OsData クラス] 順序認証システムの各処理で生成されるデータの親クラスである．データの種類を示すタグとデータがバイト配列として保持される．保持されたデータは子クラスを通して型変換される．例えば，OsNonce クラスは int 型の VMC の識別子 *ctrID* と byte[]型の順序認証対象データのハッシュ値 H_D を結合したバイト配列を OsData クラスとやりとりする．

4.2. ライブラリの利用方法

本ライブラリを利用するために，まず，BIOS から TPM を利用可能な状態にする．次に，tpm.msc を実行し，TPM 初期化ウィザードから TPM を有効にし，所有権の設定を行う．コマンド管理画面からコマンドの許可状態を確認でき，必要に応じて許可／禁止を設定できる．次に，Java で TPM を利用するためのライブラリ IAIK jTSS をダウンロードし，setup.exe を実行する．最後に，ダウンロードしたライブラリと本ライブラリにクラスパスを通す．

4.3. ライブラリの評価

Initializing，*OrderStamping*，*OrderCrossing*，*Comparing* の実装

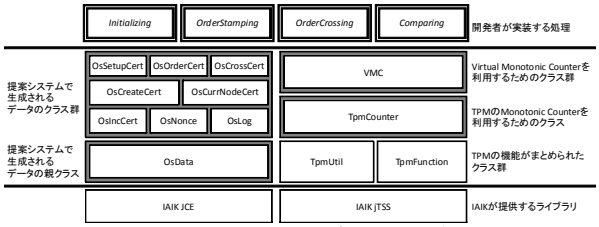

図 4 開発したライブラリの構成

表 1 ライブラリ利用／未利用におけるステップ数の比較[step]

	Client		LOC		GOC		Verifier	
	未利用	利用	未利用	利用	未利用	利用	未利用	利用
Initializing	–	–	651	34	690	47	–	–
OrderStamping	130	30	694	28	–	–	–	–
OrderCrossing	149	14	688	41	690	43	–	–
Comparing	–	–	–	–	–	–	181	1

において，本ライブラリを利用した場合と利用しない場合のソースコードのステップ数を表 1 に示す．VMC クラスの実装に必要なステップ数が 566 ステップであり，*IncAndSignClock* を実行する LOC，GOC において約 95%以上のステップ数が削減された．システム全体においては約 75%のステップ数が削減された．

5. 順序認証システムの試作

端末の OS は Windows7，TPM はバージョン 1.2 で開発言語に Java 1.7.0_06 を用いた．各主体をライブラリ JSR311 3.1.1[5]を用いて同一端末内で動作する Web サーバとして実装した．VMC は TPM の TransportSession 内で実行することで安全性が保証されている．VMC の動作確認を目的として，TransportSession 機能を利用せずにシステムを試作した．

試作したシステムで Initializing，OrderStamping，OrderCrossing，Comparing を 100 回実行しその平均処理時間を計算した．Initializing は 1714.44ms，OrderStamping は 859.54ms，OrderCrossing は 1634.67ms，Comparing は 9.05ms であることを確認した．

6. おわりに

本稿では，TPM を用いた Virtual Monotonic Counter の階層型接続による順序認証システムと Java によるライブラリを提案した．VMC を階層型に接続することで，単一のカウンターにアクセスすることなくそれぞれの端末内でデータの順序関係を保証できる．提案システムの実装を容易にするための TPM の利用部分を隠蔽したライブラリを開発した．

開発したライブラリを用いてシステムを試作したところステップ数がシステム全体で約 75%削減された．試作したシステムでは，順序認証は平均 859.54ms で処理できることを確認した．

参考文献

[1] L. Sarmenta, M. van Dijk, C. O'Donnell, J. Rhodes, and S. Devadas: Virtual Monotonic Counters and Count–Limited Objects using a TPM without a Trusted OS, ACM CCS-STC '06, 2006, pp. 27-42.

[2] Trusted Computing Group: TPM Main Specifications Level 2 Version 1.2, Revision 116 (Parts 1−3) (online), available from <http://www.trustedcomputinggroup.org/resources/tpm_main_specification> (accessed 2012-11-30)

[3] Trusted Computing Group (online), available from http://www.trustedcomputinggroup.org/ (accessed 2013-01-10)

[4] IAIK TU Graz: Trusted Computing Group (online), available from <http://trustedjava.sourceforge.net/index.php?item=jtss/readme> (accessed 2012-11-30)

[5] Java Comunity Process: The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 311 (online), available from <http://jcp.org/en/jsr/detail?id=311> (accessed 2012-11-30)