社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－DPS－123（6）
2005／6／2

フォレンジックコンピューティングのための効率的なログ署名手法の提案

小畑 直裕*　川口 信隆*　東 雄介*　重野 寛*　岡田 謙一*

本稿ではフォレンジックコンピューティングのための効率的なログ署名手法を提案する．信頼性の高いフォレンジックコンピューティングを行うには，解析対象となる電子的証拠，すなわちログの真正性を検証できる必要がある．このためには公的な機関が発行した電子署名が必要になる．しかし多数のロギングホストが電子署名を発行するサーバに接続することで署名サーバへのトラフィックが増大することが予想される．そこで本稿では署名サーバへのトラフィックを減少させる効率的なログ署名手法を提案する．本提案は分散 Merkle ツリーアルゴリズムを用いてホストが協調的にロギングを行うことで署名サーバへの送受信負荷を軽減する．

Efficient log authentication for Forensic Computing

Naohiro OBATA *　Nobutaka KAWAGUCHI*
Yusuke AZUMA*　Hiroshi SHIGENO*　Kenichi OKADA*

In this paper, an efficient log authentication scheme for Forensic Computing is proposed. To conduct reliable Forensic Computing, it is required that the logs as digital evidence be verified. To verify them, digital signatures issued by authorities are needed. However, when many logging hosts connect to the server that issues the signatures, the traffic of the server will increase. Therefore we propose an efficient log authentication scheme for Forensic Computing. Our scheme reduces the traffic of the Sign Server by using distributed Merkle Tree Algorithm among the logging hosts.

1　はじめに

近年，社会におけるデジタル情報の重要性は増す一方であり，特に組織や政府のシステムなどの大規模なシステムが攻撃された時の被害は計り知れない．このような攻撃に対する対抗策としてフォレンジックコンピューティングが注目されている．フォレンジックコンピューティングは「電子的証拠として使用できる，ネットワークやコンピュータに残されたデータをもとに，攻撃者や攻撃手法を解析，特定する技術」のことである．

フォレンジックコンピューティングにおいてログなどのデータを法廷で電子的証拠として使用するためには後からログの真正性を検証できる必要があり，一般的に検証には電子署名が用いられる．署名は信頼できる公的機関が運営する署名サーバに署名してもらう．今後フォレンジックコンピューティングの重要性から，多くのホストが署名サーバに署名を求めるようになると署名サーバへのトラフィックが増えることが予想される．この問題への対処法としては，署名サーバの台数を増やすことや署名サーバへのトラフィック容量を増やすことなどが考えられる．しかしフォレンジックコンピューティングの重要性が増すにしたがって，サーバだけでなく多数の PC やモバイル端末も署名サーバにログを送信することが考えられる．そのため署名サーバ，ロギングを行うクライアント双方での負荷軽減が求められる．

そこで本稿ではフォレンジックコンピューティングのための効率的なログ署名手法を提案する．本提案は分散 Merkle ツリーアルゴリズムを用いてホストが協調的にロギングを行うことで署名サーバのトラフィックを軽減させることができる．本手法では複数のホストで木構造を構築し，最上位のルートノードのみが署名サーバと直接通信を行うことにより署名サーバのトラフィックを大幅に減らすことを可能とした．またロギングホストのトラフィック増加量

* 慶應義塾大学 理工学部 情報工学科
Department of Instrumentation(Information), Faculty of Science and Technology, Keio University

もホスト数の増加に対してスケーラビリティを確保することができた．また提案手法の有効性を実証実験を通じて示した．

本稿では，まず 2 章で既存のロギングとタイムスタンプ技術について触れ，3 章で効率的ログ署名手法を提案する．4 章では本提案の有効性を実証実験を通じて示し，5 章を本稿のまとめとする．

2 ロギングとタイムスタンプ

本章ではログの完全性，真正性を保障するのに不可欠なロギング手法とタイムスタンプ手法を紹介する．

2.1 既存のロギング手法

1) Syslog-sign

Syslog-sign [1] は Syslog に対して，メッセージの完全性，リプレイ攻撃への耐性，メッセージの順序付け機能を追加したプロトコルである．クライアントホストはロギングサーバにログを送信し，保管してもらう．

Syslog-sign では通常のメッセージの他に，署名ブロックメッセージ，証明書ブロックメッセージがある．署名ブロックメッセージは過去に送信したメッセージのハッシュ値と，それらに対する署名を持っている．この署名を検証することで過去における他メッセージを検証することができる．証明書ブロックメッセージは署名に用いられた秘密鍵の検証に用いられる．

2) Forward Integrity

ロギングでは過去に生成されたログが改ざんや削除された場合，それを検知できる必要がある．Forward Integrity [2] は，Bellare と Yee により提案されたセキュリティの概念で，たとえホストがクラックされ，秘密鍵が露呈しても，クラック以前に生成されたログの改ざん，削除の検知を可能とする．

ログの改ざん検知に用いられる MAC(Message Authentication Code) は秘密鍵を利用して生成されるが，MAC 生成後に秘密鍵をホスト上から削除することで，攻撃者侵入の際の鍵の露呈を防ぐことができる．しかし，予め，ロギングに必要な全ての鍵を用意しておくという方法は難しい．このため鍵生成にハッシュ関数を用いる．ここで，K_i を，i 番目のメッセージの MAC を生成するために用いる鍵として，$H()$ をハッシュ関数とする．このとき，

$$K_1 = H(K_0), K_2 = H(K_1), ..., K_i = H(K_{i-1})$$

というように，K_1 ー K_i を再帰的に生成する．K_0 自体は後からログの検証を行う際に必要となるので，IC カードや他のホストといった安全な領域に保管する必要がある．次に利用する鍵 (K_{i+1}) を生成した直後に，生成元の鍵 (K_i) をメモリ上から削除することで，ホストに攻撃者が侵入してログを改ざんしたとしても，後からの検知が可能となる．

2.2 既存のタイムスタンプ手法

上記で紹介したロギング手法でもログの完全性を示すことはできる．しかし法廷での証拠として扱うためには，公的な機関による証明が必要である．一般的にデジタルデータの真正性や完全性を証明するために，公的機関にタイムスタンプを発行してもらう．タイムスタンプを発行する公的機関はタイムスタンプ生成機関 (TSA) と呼ばれる．本稿では TSA と署名サーバは同義とする．

TSA とクライアントホスト間の通信はタイムスタンププロトコルを用いて行われる．以下にシンプルプロトコルとリンキングプロトコルに関して述べる．

1) シンプルプロトコル

シンプルプロトコル [3] では図 1 のように，TSA がクライアントから受け取ったハッシュ値に対する署名とタイムスタンプを生成する．

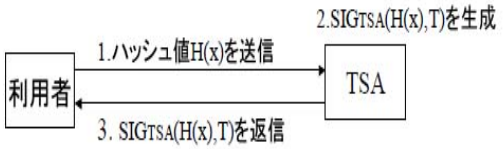

図 1: シンプルプロトコル

シンプルプロトコルは TSA が信頼できることが前提条件となる．そのため TSA が信頼できない場合，本プロトコルは利用できない．

2) リンキングプロトコル

リンキングプロトコル [4] は，TSA が複数の利用者のハッシュ値を相互に関連付けするリンク情報を生成し，それに署名を行うプロトコルである．例えば，n 番目のハッシュ値 H_n の署名は以下のようになる．

$$s = SIG_{TSA}(n, t_n, ID_n, H_n, L_n)$$

ここで t_n は H_n を受信した時刻情報，ID_n は署名を一意に識別するための ID を表している．
またリンク情報である L_n は以下のように定義される．

$$L_n = (t_{n-1}, ID_{n-1}, H_{n-1}, H(L-1))$$

各署名はリンク情報を用いて検証され，検証に必要な認証情報サイズは $O(n)$ となる．リンク情報は定

期的に新聞などのメディアで公表されるため，例えTSA が信頼できなくなっても，それ以前に生成された署名は信頼することができる．

またリンキングプロトコルをより効率化したツリープロトコル [5] が提案されている．リンキングプロトコルは，複数のハッシュ値でツリー構造を構築するために Merkle ツリーアルゴリズム [6] を用いる．検証に必要な認証情報サイズは $O(logn)$ となり，従来のリンキングプロトコルより減っている．本提案でも Merkle ツリーを用いてるため，ここで Merlkle ツリーに関して具体的に述べる．

Merkle ツリーは図 2 で示されるような 2 分木構造である．木構造の葉（リーフ）である $D_1 - D_8$ は署名されるデータを表す．まず各データ D_i のハッシュ H_i が計算され，次に近傍のハッシュ H_i, H_{i+1} を連結した $H_{i-i+1} = H(H_i, H_{i+1})$ が計算される．演算結果のハッシュ値は H_i, H_{i+1} の親になり，最終的に一つの親に集約されるまでこのプロセスを繰り返す．図 2 では H_{1-8} になる．最後にこのハッシュ値に署名を施す．

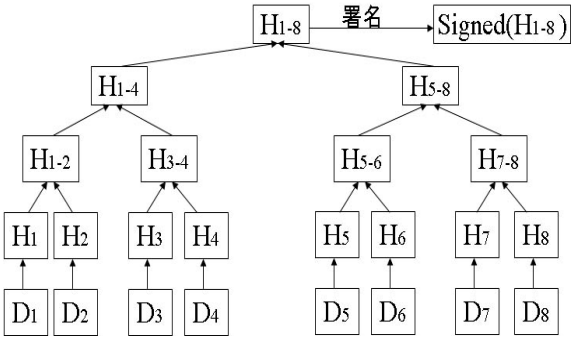

図 2: Merkle ツリー

D_i の署名を検証するのに必要な認証情報は，D_i-ルート間の兄弟ハッシュ値である．例えば D_1 の署名を検証するには H_2, H_{3-4}, H_{5-8} が必要である．署名検証者は認証情報をもとに木構造を再構築し，ルートハッシュ値を求めて署名の検証を行う．木構造のリーフの数が N 個の時，署名検証に必要な認証情報は $log(N)$ となり，署名回数は常に 1 回である．リンキングプロトコルではリーフは各データのハッシュ値となり，ルートハッシュ値が関連付けされてリンク情報を形成していく．

3 効率的なログ署名手法の提案

3.1 目的

多数のロギングノードが署名サーバに署名を求めるようになると署名サーバのトラフィックが増加する．そのためクライアントノード，署名サーバの双方で負荷分散が求められる．

そこで本章では分散 Merkle ツリーを用いた効率的なログ署名手法を提案する．分散 Merkle ツリーでは複数のノードで木構造を構築する．そして全てのノードのハッシュ値がルートノードに集められ、ルートノードを通じて署名サーバへ送信される．署名サーバと直接通信を行うのはルートノードのみであるため署名サーバのトラフィックを大幅に減らすことができる．またロギングノードのトラフィック増加量もノード数の増加に対してスケーラビリティを確保することができた．

3.2 モデル

図 3 に分散 Merkle ツリーのモデルを示す．
各ロギングノードはお互いに接続することでツリー

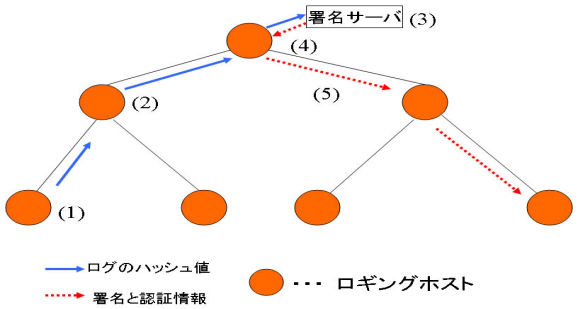

図 3: 分散 Merkle ツリー

構造を構築する．なお本稿ではノード間での分散 Merkle ツリー構築手法に関しては触れず，そのような手法があるものとして話を進める．本モデルでは以下の流れで署名や認証情報の送受信が行われる．

1) ログのハッシュ値の演算

各ロギングノードはログが生成されると同時にハッシュ値を演算する．そしてリーフに存在するロギングノードは一定間隔時間 (署名間隔時間) ごとに，時間内に生成されたハッシュ値から Merkle ツリーを構築しルートハッシュを求める．そして上位ノードへルートハッシュを送信する．

2) ハッシュ値の上位ノードへの送信

上位ノードでのハッシュ値の送受信は以下のようになる．

(1) 親ノードは署名間隔時間ごとに自身が直接接続

している子ノードからハッシュ値を受信する．
(2) 親ノードは，自身が署名間隔時間内に生成したハッシュ値と子ノードから受信したハッシュ値を連結し Merkle ツリーを構築する．ツリーのルートハッシュを上位の親ノードへと送信する．
(3) 子ノードから受信したハッシュ値や自身が生成したハッシュ値，ルートハッシュは後の使用のために一時的に保存される．

3) 署名サーバでの処理

最終的にツリーの最上位に位置するルートノードが署名サーバと通信を行う．ルートノードからハッシュ値を受け取った署名サーバはハッシュ値とタイムスタンプを連結させた値に対して署名を行う．

4) 署名サーバ（上位ノード）からの認証情報，署名の受信

上位ノードから各ロギングノードは署名と認証情報を受け取り、署名の検証をする必要がある．ここで署名と認証情報のことを，署名の検証に必要なデータという意味で署名検証データと呼ぶことにする．図 4 に署名サーバや上位ノードから下位ノードに対して送信される署名検証データのフォーマットを示す．

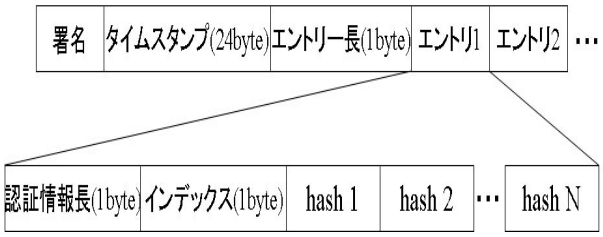

図 4: 署名検証データのフォーマット

各エントリには Merkle ツリーを再構築するために必要な認証情報が入っている．署名検証データを用いて，上位ノードへ送信したログのハッシュ値は以下のように検証される．
(1) 上位ノードに送信した後で一時的に保存しておいたルートハッシュとエントリ 1 を利用して Merkle ツリーを構築する．このとき，エントリ 1 のインデックスが自身のルートハッシュのツリー上での位置となる．
(2) エントリ 1 で構築したツリーのルートハッシュ値とエントリ 2 のデータをもとにさらにツリーを構築する．この手順を最後のエントリのルートハッシュが生成されるまで行う．
(3) ルートハッシュとタイムスタンプを連結した値と署名を比較することで検証を行う．

5) 下位ノードへの認証情報，署名の送信

下位ノードへ署名検証データを送信する時に，自身が以前に下位ノードから受け取ったハッシュ値をもとに生成したツリーのデータをエントリ 1 として署名検証データに挿入する．下位ノードごとに独立に署名検証データが生成され，下位ノードへ送信される．

4 評価と考察

本章では提案の評価と考察を理論値比較と実証実験を通じて行う．

4.1 理論値比較

ここで本提案手法をツリー接続と呼ぶことにする．そして全てのノードが直接署名サーバに接続する手法を直接接続と呼ぶことにする．ロギングノード数が N，ツリーの次数が d であるツリー接続と直接接続の理論値を表 1 に示す．

表 1: 理論値の比較

評価項目	ツリー接続	直接接続
署名遅延時間	$O(log_d N)$	$O(1)$
署名サーバ送信速度	$O(1)$	$O(Nlog_2 N)$
署名サーバ受信速度	$O(1)$	$O(N)$
ノード送信速度	中間ノード:$O(dlog_2 N)$	$O(1)$
	リーフノード:$O(1)$	
ノード受信速度	$O(log_2 N)$	$O(log_2 N)$

まず，署名遅延時間はノードがログを生成してからハッシュ値を署名サーバに送信して署名が戻ってくるまでの時間（署名サーバでの署名レートが同一．署名サーバでの演算時間等は除く）を意味する．署名遅延時間が長いということは，ハッシュ値を送信してから戻ってきた署名と認証情報を安全な領域に保存するまでの時間が長いということである．署名遅延時間が長くなるほど，攻撃を受けたときに失われるログの認証情報と署名数が多くなる．このため署名遅延時間は短いことが望ましい．以下，それぞれの理論値について説明する．

1) 署名遅延時間: 直接接続の場合，直接署名サーバと通信を行うため伝送遅延時間は常に 1hop 分であり，$O(1)$ となる．対してツリー接続の場合，葉に存在するノードはツリーの高さ分だけ余分に hop する必要があり，ツリーの高さは $log_d N$ と表されるため，$O(log_d N)$ が成り立つ．

2) 署名サーバ送信速度: 直接接続の場合，ツリー

は全てのロギングノードと通信を行わなければならない．また，署名サーバは一定時間内（署名間隔時間内）に送信されたN個のハッシュ値からツリー型のリンキング手法のようにMerkleツリーを構築することを想定する．この場合，1つのハッシュ値を検証するのに必要なハッシュ値の個数は$O(log_2N)$となる．このためN個のホストに送信する場合には送信速度は$O(Nlog_2N)$が成り立つ．対して，直接接続の場合は，ツリーのルートノードのみが通信を行うため，署名サーバは署名のみを1つのノードに送ればよいので送信速度は$O(1)$が成り立つ．

3) 署名サーバ受信速度: ツリー接続の場合はルートノードからハッシュ値を受け取るだけなので署名サーバの受信速度は$O(1)$となる．対して直接接続の場合は全てのホストからハッシュ値を受け取る必要があるため受信速度は$O(N)$となる．

4) ノード送信速度: 直接接続の場合はハッシュ値を署名サーバに送信するだけなので送信速度は$O(1)$となる．ツリー接続の場合は，リーフノードとそれ以外の中間ノードでは送受信量が大きく変わる．リーフノードは直接接続同様の送信しか行わないため$O(1)$が成り立つ．しかしそれ以外の中間ノードの場合，上位のノードから来た認証情報に新しいエントリを追加して下位に送信する必要がある．この時の認証情報量は$O(log_2N)$でありこれをdの下位ノードに送信するため送信速度は$O(dlog_2N)$が成り立つ．

5) ノード受信速度: これはツリー接続，直接接続でも同一の値$O(log_2N)$となる．直接接続の場合は署名と認証情報量，ツリー接続の場合は下位ノードから受信するハッシュ値と上位ノードから受信するハッシュ値が受信速度に影響する．

表1から直接接続に対してツリー接続は署名サーバ送受信速度を大幅に改善している一方，署名遅延時間，中間ノードの送信速度では直接接続よりも値が悪くなっていることが分かる．しかし，直接接続の署名サーバ送受信速度が$O(N)$なのに対してツリー接続の署名遅延時間，中間ノードの送信速度は$O(log_2N)$でありNに対してスケールする．この点からツリー接続は直接接続に比べてスケーラビリティに優れていると言える．

4.2 実測値の比較

署名サーバ，ノードの送受信速度の実測値を測定するため実証実験を行った．

1) 実装環境

実装に用いた環境，実験パラメータは以下の通り．

1. **ロギングノードと署名サーバ:** PC16台 (OS windows 2000 or windows XP, CPU 550MHz-3GHz，Memory 256MB-2GB)
2. **開発環境:** j2sdk1.4.2
3. **実験環境:** 100Mbps LAN リピータハブ&スイッチングハブ
4. **署名間隔時間:** ツリー接続，直接接続とも100msec
5. **電子署名:** 1024bits RSA with SHA-1
6. **ハッシュ関数:** SHA-1

15台のロギングノードと1台の署名サーバで構成され，ツリー接続では次数2のMerkleツリーを構築し，直接接続では全てのロギングノードが署名サーバに直接接続するものとする．署名サーバを含まない15台のツリー構造において，最上位のルートノードをレイヤー1とし，順に下位の階層に位置するノードをレイヤー2，レイヤー3とし，リーフノードをレイヤー4とする．

2) 実験結果

表2にツリー接続と直接接続の署名サーバ送受信速度，ツリー接続の中間ノードの送受信速度と直接接続のノードの送受信速度の測定結果を示す．測定値は一定時間通信を続けた時の送受信速度の平均値を示している．署名サーバの送受信速度においてツリー接続は直接接続に対して5%程度の送信速度，6.5%程度の受信速度となっている．またノードの送受信速度に関してだが，これはツリー接続の場合，ノードの送受信速度は属するレイヤーによって異なるので，ここでは平均的な送受信速度に関して述べ，後ほどレイヤーごとの送受信速度に関して言及する．

表2: 署名サーバと各ノードの送受信速度

評価項目	ツリー接続	直接接続
署名サーバ送信速度 [kbps]	16.19	347.76
署名サーバ受信速度 [kbps]	5.75	87.75
ノード送信速度 [kbps]	28.36	5.85
ノード受信速度 [kbps]	28.88	23.50

ツリー接続の中間ノードは署名検証データを下位ノードへ送信するため直接接続より送信量は大きくなる．またノード受信速度はツリー接続と直接接続であまり差はない．

表 3: レイヤーごとの送受信速度

	送信速度 [kbps]	受信速度 [kbps]
レイヤー 1	44.88	28.15
レイヤー 2	51.01	31.00
レイヤー 3	57.91	34.55
レイヤー 4	5.86	25.61

表 4: 平均署名遅延時間

	ツリー接続	直接接続
署名遅延時間 [msec]	335.02	199.41

表 5: レイヤーごとの平均署名遅延時間

	署名遅延時間 [msec]
レイヤー 1	103.25
レイヤー 2	207.89
レイヤー 3	320.35
レイヤー 4	432.56

表 3 にツリー接続の各レイヤーごとの送信速度と受信速度を示す．レイヤー 4 の送信負荷だけ低いのはリーフノードであるため下位ノードにデータを送信する必要がないためである．またレイヤー 4 の受信量が最も低いのは，レイヤー 4 のノードは下位ノードからハッシュ値を受け取る必要がないためである．

表 4 にツリー接続と直接接続のロギングノードの平均署名遅延時間の比較を示す．ツリー接続では直接接続に対して 2 倍強の遅延が発生していることが分かる．

最後に表 5 にツリー接続の各レイヤーごとの平均署名遅延時間を示す．署名サーバに近いレイヤーであるほど，署名遅延時間は短くなっていることが言える．

5 まとめ

本稿では分散 Merkle ツリーを用いた効率的なログ署名手法を提案した．理論値比較や実証実験を通じて，署名サーバの送受信トラフィックにおいての本提案の有効性を示すことができた．またロギングノードのトラフィック増加量もノード数の増加に対してスケーラビリティを確保することができた．

今後の課題としては，分散 Merkle ツリーを効率よく構築するための手法を検討する必要がある．また，本稿では悪意のあるノード，署名サーバが信頼できない場合やネットワークの不調による通信の中断などは考慮していない．このような問題への対応が今後の課題である．

参考文献

[1] J.Kelsey,J.Callas: ”Syslog-Sign Protocol DRAFT”, *Network Working Group*. June 2002.

[2] M.Bellare, B.S.Yee: ”Forward Integrity For Secure Audit Logs”, *1997 University of California, San Diego*.

[3] Internet X.509 Public Key Infrastructure Time-Stamp Protocol, RFC 3161.

[4] K.Matsuura, H.Imai: ”Digital Timestamps for Dispute Settlement in Electronic Commerce: Generation, Verification, and Renewal”, *Proceedings of 4th International Conference on Enterprise Information Systems (ICEIS 2002), Volume 2, ICEIS Press, pp.962-967*, April, 2002.

[5] Bayer, D., Haber, S.A. and Stornetta, W.S.: ”improving the efficiency and reliability of digital time-stamping.”, *In Capocelli, R., DeSantis, A., and Vaccaro, U., editors, Sequences’91: Method in Communication, Security, and Computer Science, pp.329-334, Berlin, New York, Tokyo. Springer-Verlag*, 1992.

[6] R.C.Merkle: ”A certified digital signature”, *Proc of Advances in Cryptology-Crypto ’89, Lecture notes on Comuter Science, Vol.0435, pp.218-238, Springer-Verlag*, 1989.