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

2003－CSEC－23 (2)
2003／12／19

フォレンジックコンピューティングのための安全で効率的なロギングアーキテクチャの提案

川口 信隆* 宮地 玲奈* 小畑 直裕* 重野 寛* 岡田 謙一*

フォレンジックコンピューティングでは，ログの運用に高い信頼性が要求される．本稿では，安全かつ効率的にログを運用するためのロギングアーキテクチャを提案する．このアーキテクチャではログの運用を生成・署名・保存・解析の４つに分け，各々の処理を別々の組織が担当する．そして各組織でログを安全性を確保しつつ効率的に処理するための暗号・MAC・署名方式について述べる．

Secure Efficient Logging Architecture for Forensic Computing

Nobutaka KAWAGUCHI* Reina MIYAJI*
Naohiro OBATA* Hiroshi SHIGENO* Kenichi OKADA*

In forensic computing, highly reliable operation of logs is required. In this paper, we propose a secure and efficient logging architecture to operate the logs. In the architecture, operation is divided into following sections, generation, signing, preservation and analysis. Discrete organizations take charge of each operation. Finally, we describe secure and efficient methods to encrypt, compute MAC, sign the logs in each organization.

1 はじめに

近年，国や企業の業務の電子化に伴い，ネットワークやコンピュータ資源の重要性は益々高まってきている．このため，クラッカーから攻撃を受けた場合には甚大な被害が生じることが予想される．

このような状況の中で，被害を受けた場合にネットワークやコンピュータに残されたログを元に被害規模や攻撃手法，攻撃元を解析し，刑事訴訟や損害賠償など法的な措置を取るための技術であるフォレンジックコンピューティングが注目されてきている．

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ためには信頼性の高いログが必要になる．このためログの生成・保管・解析といった処理には高度な技術が求められる．しかし１つの組織がその全ての処理を行うのは負担が大きく，また組織にとって都合の悪いログを改ざんするといった不正が発覚しづらいという問題がある．そこで，複数の組織が連携してログを運用するという形が考えられる．本稿では，ログをジェネレータ・署名サーバ・保管サーバ・アナライザの４者が連携して，ログの作成・署名・保管・解析を行うためのアーキテクチャを提案する．ログはジェネレータで作成され，署名サーバで署名を付加され，保管サーバで保管され，アナライザによって解析される．

又，ログはタイプによっては短時間に大量に生成されるため，オーバーヘッドは小さく，処理は高速に行われなければならない．そこでログに対して効率的に暗号化や署名を行うための方式を示す．

本論文は以下のように構成される．第２章ではフォレンジックコンピューティングについて，第３章ではロギングを行う上で重要な概念となる Forward Integrity について述べる．そして第４章でアーキテクチャを提案し第５章を本稿のまとめとする．

2 フォレンジックコンピューティングと関連技術

フォレンジックコンピューティング (Forensic Computing) は，クラッカーによりネットワークやコンピュータ資源が被害を受けた場合に，訴訟や損害賠償の際に証拠として利用するために，残されたログの集積・解析を行い被害規模や攻撃手法，攻撃者を特定するための技術の総称である．もともと Forensic とは法医学を意味し，フォレンジックコンピューティングはコンピュータの分野に法医学の概念を適用したものと言える．

* 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

フォレンジックコンピューティングによって解析されたログは法廷証拠として認められる程度の信頼性が求められる．このためログの保管や解析は適切に行われる必要がある [1].

フォレンジックコンピューティングとその関連技術は"ログを扱う技術"という視点から (1) ログの生成技術, (2) ログの保管技術, (3) ログの解析技術の 3 つに分類することができる．ログの生成技術には，クラッカーの攻撃場所を特定するためにルータやスイッチでパケットログを生成する IPTraceBack などがある．ログの保管技術としては IDA を用いたログの分散管理手法 [2] などがある．ログの解析ではニューラルネットワークやデータマイニングなどを用いた解析手法が提案されている．

これまでログの生成・保管・解析は同一の閉じた組織内で行われることが多かった．しかし組織の業務の電子化に伴いログの重要性が向上し，より大量で質の高いログが求められるようになるにつれ，従来の運用では限界が出てくると考えられる．

まず，ログを生成している組織内のネットワークにログを保存しておく方法ではネットワークの管理者権限が奪われた場合に改ざんや削除を受けるおそれがあることから安全性を保てなくなる恐れがある．このため，ログを信頼性の高いログの保管場所が必要となる．またログの解析はある程度の自動化が可能であるにしても最終的には高い能力を持った専門家による解析が必要であり，すべてを組織内で行うのは負担が大きい．さらに，組織内で全ての処理が行われると自身に都合の悪いログが削除される可能性がある．これは法的証拠としては明らかに不適格である．このため生成されたログは信頼できる第 3 者の手によって生成時刻と完全性が保証される必要がある．

このため，ログの処理を，生成・保管・解析といった独自の役割を持つ組織が連携して行うという形が考えられる．そのためのアーキテクチャを第 4 章で提案する．

3 Forward Integrity

ロギングでは，侵入を受けた場合に，以前に作成したログが改ざんされたり削除されるといった事態を極力少なくすることが重要になる．M.Bellare らが提案した Forward Integrity [3] は，「ホストがクラックされて鍵などの秘密情報が奪われ，以前に生成されたメッセージが改ざんされたり削除された場合，後からの検知が可能」というセキュリティ特性を指す．

メッセージの改ざんの検知には MAC(Message Authentication Code) を用いるのが一般的である．MAC は鍵を使って生成されるので，Forward Integrity を実現するためにはあるメッセージの MAC を生成するのに使われた鍵は，生成後に破棄し，端末がクラックされても鍵を奪われ MAC が偽造されないようにする必要がある．しかし予めメッセージの数だけ別々の鍵を用意しておくのは困難であるため，ある鍵から複数の鍵を生成でき，かつ生成した鍵から生成元の鍵を推定できないことが求められる．そこで鍵の生成方法として不可逆性ハッシュ関数を用いる．i 番目のメッセージの MAC の生成に用いられる鍵を K_i，ハッシュ関数を $Hash()$ とすると

$$K_1 = Hash(K_0), K_2 = Hash(K_1), \ldots, K_i = Hash(K_{i-1})$$

となり，最初の鍵 K_0 から連鎖的に K_i までの鍵を生成することができる．そして予め，K_0 を端末外に安全に保存しておき，次の鍵の生成後に前の鍵をメモリから消去することで，侵入を受けた場合でもクラッカーの侵入以前に生成された MAC の改ざんを防ぐことができる．

この Forward Integrity の性質は，安全でないホストでログを生成する際に活用できる．B.Schneir らは，Forward Integrity の性質を用いたロギングプロトコル [4] を提案している．この中では，ログの生成者であるジェネレータとログを保存するサーバの間で予め暗号鍵 K_0 を交換しておく．そしてジェネレータはハッシュ演算でログのエントリごとに鍵を変えながら暗号化，MAC の生成を行った後でローカルに保存し，決められた量だけログが溜まったらサーバへ送信する．このプロトコルはログとその MAC を作成直後にサーバへ送信するのが困難であったり，トラフィックの効率化のためにまとめて送信したい場合などに有効である．

4 提案

本章では，ログを複数の独自の役割を持った組織間で安全かつ効率的に処理するためのアーキテクチャを提案する．

4.1 構成要素と要求条件

本提案では以下の 4 つをアーキテクチャの構成要素として扱う．

1. **ジェネレータ:** ログを生成する.

2. **署名サーバ:** ジェネレータから受け取ったログに対して時刻情報とデジタル署名を付加する.

3. **保管サーバ:** ログを保管してアナライザに提供する.

4. **アナライザ:** 保管サーバからログを受け取り, 解析を行う.

ログは図1に示されるように, ジェネレータで生成され署名サーバに送信される. そして署名サーバで署名が付加され, 署名サーバから保管サーバへ送信される. 保管サーバでログは保管され, 後にアナライザにより解析されることになる.

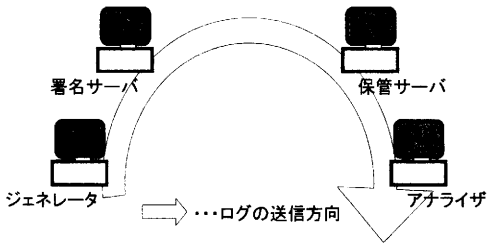

図 1: 構成要素

署名サーバは, ログがジェネレータによって後で改ざんされることを防ぐ. このため十分に信頼された存在である必要がある. また, 保管サーバはジェネレータに対し「ログが削除を受けずに保管される」ことを保証する. またアナライザはジェネレータと契約を結んだ専門家の他に, ログに対して閲覧権のある組織内のユーザや組織からサービスを受けている顧客, さらに外部の専門家や研究者など多岐に渡ることが考えられる.

これらの構成要素のもとでロギングアーキテクチャに求められる要件を以下に示す.

1. **エントリごとの復号・認証:** 保管されたログのうち解析の結果, 重要とみなされたエントリのみが抽出され長期間保存されることになる. 各エントリを復号・認証するためのオーバーヘッドはできる限り小さくする必要がある.

2. **Forward Integrity の保持:** ジェネレータ内にログが存在する間に改ざんがおきても検知できるように Forward Integrity を保持する必要がある.

3. **エントリのアクセスコントロール:** ログには様々なプライバシー情報が含まれるため, アナライザに応じたアクセスコントロールが必要となる. 例えば, パケットログのエントリでは, ヘッダパートには全てのアナライザがアクセスできるが, ペイロードパートへのアクセスは特別なアナライザに限定されるといったアクセスコントロールが考えられる.

4. **効率的な署名・暗号化:** ログはタイプによっては短時間に大量に生成されるため, 上記の処理は効率的に行われる必要がある.

以上の要件を満たす暗号, MAC, 署名方式について以下で説明していく.

4.2 定義

本提案で用いる記法を以下のように定義する.

1. $PKE_K(D)$: メッセージ D を公開鍵 K で暗号化した値.

2. $SE_K(D)$: メッセージ D を共有鍵 K で暗号化した値.

3. $Hash_i(D)$: メッセージ D に対して i 回ハッシュ演算を行った値.

4. $MAC_K(D)$: メッセージ D に対する共有鍵 K による MAC 値.

5. D_1, D_2: メッセージ D_1 とメッセージ D_2 を連結した値.

また, 一定数のエントリの集合をエントリセット, エントリを構成する部分をエントリパートと定義する.

4.3 ロギングの開始

ジェネレータ, 署名サーバ, 保管サーバの公開鍵をそれぞれ PK_{gene}, PK_{sign}, PK_{save} とする. またアナライザの数を n 人として各々の公開鍵を PK_{ali}(ただし $1 \leq i \leq n$) とする.

まずジェネレータはアナライザごとの暗号鍵 EK_{ali}, 各エントリの MAC を生成する鍵 EMK,

エントリセットのMACを生成する鍵SMKを生成する．

次にロギングの開始を要求するために以下のようなメッセージM_1を生成し署名サーバへ送信する．

$$\begin{aligned} X_1 &= ID_{gene}, START, \\ &\quad 保管サーバのアドレス, \\ &\quad PKE_{PK_{sign}}(SMK) \\ M_1 &= X_1, Sign_{SK_{gene}}(X_1) \end{aligned}$$

ID_{gene}はジェネレータの認識子を，$START$はロギングの開始要求を意味する．ジェネレータは署名サーバからロギングの許可を示すメッセージが帰ってきた場合，保管サーバに以下のようなM_2を送信する．

$$\begin{aligned} X_2 &= ID_{gene}, START, \\ &\quad PKE_{PK_{save}}(SMK), \\ &\quad PKE_{PK_{al1}}(EK_{al1}, EMK), \\ &\quad PKE_{PK_{al2}}(EK_{al2}, EMK), \\ &\quad \ldots, PKE_{PK_{aln}}(EK_{aln}, EMK) \\ M_2 &= X_2, Sign_{SK_{gene}}(X_2) \end{aligned}$$

M_2にはアナライザが使用する鍵が含まれるため，保管サーバはこれを長期間保存しアナライザに提供する必要がある．また必要に応じて保管サーバと署名サーバの間で相互認証が行われたりセッションが張られる．

4.4 ジェネレータでの処理

ロギングが開始されると，ジェネレータは生成したパケットログやシステムログなどのログに対して暗号化，MACの付加を行い，署名サーバに送信する．

4.4.1 暗号化

ジェネレータは表1に示されるようなエントリパートアクセスマップを保持し，アナライザごとに閲覧を許可するエントリパートを規定する．

$Part$はエントリパート，mはエントリパートの数，alはアナライザを意味し，Yはアクセスを許可すること，Nはアクセスを許可しないことを意味する．ジェネレータは，アナライザに応じてYであるエントリパートのみを暗号化する．

アクセスマップに即したエントリの暗号化としては，アナライザごとに暗号化を行う方式1と，エントリパートごとに暗号化を行う方式2が考えられる．以下でそれぞれの方式について述べ比較を行う．なお，アクセスマップの各セルがYである確率をP，エントリパート1つのサイズをL_{part}，エントリパートを暗号化する鍵のサイズをL_{key}とする．

表1: エントリパートアクセスマップ

アナライザ	$Part_1$	$Part_2$	...	$Part_m$
al_1	Y	N	...	Y
al_2	N	Y	...	N
...	...	...	...	...
al_n	Y	Y	...	Y

1. 方式1

   方式1ではアナライザごとにエントリの暗号化を行う．アナライザiが，エントリk(ロギングの開始からk番目のエントリ）の暗号に用いる鍵$KEY_{i,k}$は

   $$KEY_{i,k} = Hash_1(Hash_{k-1}(EK_{ali}), 0)$$

   となる．EK_{ali}に対して$k-1$回ハッシュ演算を行うのは，Forward Integrityを実現するためにエントリごとに鍵を換える必要があるからである．最初のエントリの暗号化ではこの処理は不要である．さらに0と連結して1回ハッシュ演算を行うのは，アナライザが第3者に対して安全に鍵を渡せるようにするためである．このハッシュ演算が行われない場合，第3者は渡された鍵に対しさらにハッシュ演算を行い以降のエントリを復号するための鍵を生成できてしまう．エントリkのうちアナライザiがアクセス可能なエントリパートの集合を$ES_{al_i,k}$とすると，暗号化されたエントリkである$Enc(Entry_k)$は

   $$\begin{aligned} Enc(Entry_k) &= SE_{KEY_{1,k}}(ES_{al_1,k}), \\ &\quad \ldots, SE_{KEY_{n,k}}(ES_{al_n,k}) \end{aligned}$$

   となる．$Enc(Entry_k)$のサイズは$L_{part}Pmn$，暗号演算回数はn回，ハッシュ演算回数は$2n$回となる．

2. 方式2

方式2ではエントリパートごとの暗号化を行う．まず，全てのエントリパートに対して，鍵 $PKEY_{j,k}(1 \leqq j \leqq m)$ を乱数から生成し，エントリパートごとに暗号化する．次に，アナライザ i，エントリパート $Part_{j,k}$ ごとに鍵

$$KEY_{i,j,k} = Hash_1(Hash_{k-1}(EK_{ali}), j)$$

を求める．そして，

$$Pad_{i,j,k} = KEY_{i,j,k} \oplus PKEY_{j,k}$$

となる $Pad_{i,j,k}$ を求めエントリに追加する．アナライザはエントリパートを復号するために $Pad_{i,j,k}$ と自身で計算した $KEY_{i,j,k}$ から鍵 $PKEY_{j,k}$ を求める．

$Part_{j,k}$ に付加される $Pad_{i,j,k}$ の集合を $PS_{j,k}$ とすると

$$Enc(Entry_k) = PS_{1,k}, SE_{PKEY_{1,k}}(Part_{1,k}) \ldots, PS_{m,k}, SE_{PKEY_{m,k}}(Part_{m,k})$$

となる．サイズは $L_{key}Pmn + L_{part}m$，暗号演算回数は m 回，ハッシュ演算回数は $2Pmn$ 回となる．

3. 比較

ログの暗号化ではオーバーヘッドを小さくすることが重要となる．方式1と方式2のどちらがより小さいサイズを実現するかは，P, n, L_{key}, L_{part} の値に依存する．例えば，$P = 0.5, n = 3, L_{part} = 4L_{key}$ であるとき，方式1：方式2＝12：11となり方式2が優位となる．また暗号演算回数については $n < m$ であるとき方式1が優位となる．このためオーバヘッドと暗号演算回数の両方から適した方式を選択することになる．

4.4.2 エントリセットの作成

エントリの暗号化後，ジェネレータはエントリのメタ情報 $Info$ と MAC を作成する．$Info$ にはエントリの順番やアクセスコントロール情報などの公開情報が含まれる．最終的に，k 番目のエントリ EM_k は

$$EX_k = Info, Enc(Entry_k)$$

とすると

$$EM_k = EX_k, MAC_{Hash_{k-1}(EMK)}(EX_k)$$

となる．エントリはローカルに保存される．保存されたエントリは一定の条件が満たされたときに，エントリセットとしてまとめられ署名サーバへ送信される．この条件としては (i) エントリ数が一定に達したとき (ii) 一定時間ごと (iii) ジェネレータの帯域が空いているときなどが考えられる．ここでは一定数に達したとき，エントリセットが作成されるとする．エントリセットに含まれるエントリの数を S とすると l 番目に作成されるエントリセット ESX_l は

$$ESX_l = ID_{gene}, LOG, EM_{(l-1)S+1}, \ldots, EM_{lS}$$

となる．LOG はエントリへの署名・保管要求を示す．最後に，ESX_l に ESK で MAC を付加した

$$ESM_l = ESX_l, MAC_{ESK}(ESX_l)$$

を生成し署名サーバに送信する．

4.5 署名サーバでの処理

署名サーバではジェネレータから受信したエントリに対して時刻情報と署名を付加する．エントリセット全体に署名する方法では，個々のエントリを単独で認証する際のオーバーヘッドが大きすぎるという問題がある．一方，署名演算は計算負荷が高いためエントリごとに署名するのは効率的でない．そこで，Wong らが提案した署名ツリーアルゴリズム [5] を使用する．署名ツリーアルゴリズムは，パケットの集合から署名ツリーを構築し複数のパケットを認証できる署名を1つ生成することで，リアルタイム性が求められる通信で高速な1パケットごとの認証を実現する．パケット4つを用いた署名ツリーの構築例を図2に示す．

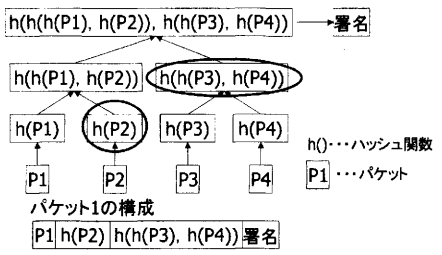

図 2: 署名ツリー

各パケットでハッシュ値を計算し，以降2つのハッシュ値を連結した値のハッシュ値をルートまで再帰的に計算することで．認証ツリーを構築する．そし

てツリーのルートのハッシュ値に対して署名をする．各パケットはこの署名と認証ツリーのルートまでに連結した相手のハッシュ値(図2の円で囲んだ箇所)のオーバーヘッドとして持つ．よってパケットの集合全体に署名する場合よりもオーバーヘッドは小さくなる．パケットの認証の際は，このハッシュ値を使って署名ツリーを再構築し，ツリーのルートを検証することで，パケットを認証する．

署名サーバはこのアルゴリズムを使って，エントリセット内のエントリから署名ツリーを構築する．そして署名ツリーのルートハッシュと現在の時刻情報を連結してハッシュ演算を行った値に対して署名を付ける．

署名後，署名サーバはESKでMACを付加して保管サーバへ送信する．時刻情報と署名を付加したESX_lを$Signed(ESX_l)$とおくと，保管サーバへ送信するメッセージSSX_lは

$$SSX_l = Signed(ESX_l), MAC_{ESK}(Signed(ESX_l))$$

となる．

4.6 保管サーバでの処理

保管サーバでは署名サーバから受信したSSX_lを保管する．またアナライザからの要求に応じてログを提供する．提供されるエントリは$\log S$個のハッシュ値と署名，時刻情報をオーバーヘッドとして持つ．

4.7 アナライザでの処理

アナライザはまず保管サーバからM_2を取得しエントリの復号や検証に用いる鍵を得る．次に，保管サーバに対して解析を行うエントリを要求する．保管サーバからエントリを取得後，認証を行う．認証ではEMKによるMACを用いてエントリが生成後署名サーバに送信されるまでの期間，署名サーバによる署名を用いて署名が付加されてからアナライザに取得されるまでの期間，改ざんが起きていないことを確認する．認証後，各エントリ・エントリパートごとの鍵を生成しエントリを復号化し解析を行う．

またアナライザが第3者に対しなんらかの理由でログを提示する場合には，エントリとそのエントリを復号化するために用いる鍵を渡す．鍵はエントリ，エントリパートごとに固有であるため，第3者はこの鍵を使って他のエントリを復号することはできない．

5 おわりに

近年，フォレンジックコンピューティングの重要性の高まりに伴い，より信頼性の高いログの運用が求められるようになってきている．ログはクラッカーによる改ざんや削除の危険にさらされる．また生成されたログが生成者自身により後で都合の良いように書き換えられる可能性がある．さらに，ログは短時間に大量に生成され且つ，様々な種類のプライバシー情報を含んでいるため効率的な暗号化や署名が必要となる．

そこで本稿ではジェネレータ，署名サーバ，保管サーバ，アナライザの4者がそれぞれの役割を持ちログを運用することでログの信頼性を確立するアーキテクチャを提案した．そして，ログに対するForward Integrityを用いたMACの付加，アナライザの権限に応じたエントリ・エントリパートごとの暗号化，署名ツリーによる高速な署名を行うための方式について示した．

今後はアーキテクチャや方式のさらなる研究，実装・実験を通した評価を行っていく予定である．

謝辞

この研究は，応用セキュリティフォーラムの支援を受けて行われた．

参考文献

[1] RFC3227: Guidelines for Evidence Collection and Archiving

[2] A.Aroma, D.Bruschi, E.Rosti: Adding Availability to Log Services of Untrusted Machines, 15th Annual Computer Security Applications Conference pp. 199–207 (1999).

[3] M.Bellare, B.S.Yee: Forward integrity for secure audit logs, Technical report, UC at San Diego, Dept. of Computer Science and Engineering,(1997).

[4] B.Schneir, J.Keley: Secure audit logs to support computer forensics, ACM Transactions on Information and System Security,vol.2 pp. 159–176 (1999).

[5] Chung Kei Wong, Simon S. Lam: Digital Signatures for Flows and Multicasts, IEEE/ACM Transactions on Networking,vol.7 pp. 502–513 (1999).