マルチメディア通信と分散処理　90－10
コンピュータセキュリティ　2－10
（1998．9．18）

2分ハッシュ木を用いた証明書廃止・更新システム

菊池 浩明　　安部 謙介　　中西 祥八郎

東海大学工学部

概要: 現在、公開鍵証明書の廃止方法は、X.509 で定められた CRL(Certificate Revocation List) が一般的である。しかし、CRL には発行間隔や肥大化等の問題がある為、改良案として OCSP や Delta-CRL、CRT(Certificate Revocation Tree)、Authenticated Directory 等が提案されている。そこで本稿では、実際に CRT の基本機能をもつシステムを試作し、それぞれの方法についての比較、検討を行う。また、Naor らが提案した方法よりも、効率のよい更新方式の提案する。

Certificate Revocation and Update Using Binary Hash Tree

Hiroaki Kikuchi　　Kensuke Abe　　Shohachiro Nakanishi

Tokai university, Faculty of Engineering

Abstract: A CRL(Certificate Revocation List) defined in X.509 is currently used for revocation. To corp with issue of CRL, that includes a high communication cost and low latency for update, OCSP, Delta-CRL, CRT(Certificate Revocation Tree) and Authenticated Directory have been proposed. In this paper, we implement experimental CRT system, and the expected reduction of communication cost in comparison with CRL. We also propose a new update method which is more efficient in communication than Naor's evaluate method.

1　はじめに

公開鍵証明書は、インターネットにおけるセキュリティ基盤を成す技術である。X.509 は、廃止された証明書のシリアル番号にタイムスタンプを束ねて CA が署名した、CRL(Certificate Revocation List) による廃止を定めている [X.509AM]。CRL は証明書と同様に公開情報であり、これが、分配するディレクトリやリポジトリでの取り扱いを容易にしている。

ところが、CRL には次の問題点が認識されている。

1. CRL の発行間隔
   通常 CRL の発行は、CA の負荷をおさえる為、複数の廃止証明書をためて、定期的に行われる。従って、ある証明書 (の秘密鍵) が盗まれたとしても、次の CRL の発行日までは不正利用が防止出来ない。かといって、むやみに間隔を狭めては、通信帯域を無駄に浪費し、せっかくの証明書のオフライン性が損なわれる。

2. CRL の肥大化
   CRL の大きさは、有効期限より前に廃止された証明書の数に線形であり、廃止証明書数は証明書の有効期限の長さと単一の CA が発行する証明書数に比例する。すなわち、その CA に属するユーザ数に比例すると考えてよい。従って規模が大きくなるほど CRL は肥大化し、通信帯域を圧迫する。しかも、CRL のほとんどは静的

な情報なので、同じ情報が何度もネットワークを流れることとなり、効率が悪い。

この2つの問題に対して、次の改良案が提出されている。

1. Online Certificate Status Protocol(OCSP)
   [OCSP] で提案されている、要求に応じてオンラインで証明書の状態を知らせるプロトコル。CRL のリポジトリが CA に代わって検証結果に署名を行う。CRL に含まれているかどうかのみを交換するため、通信コストも小さい[1]。

2. Delta-CRL(差分 CRL)
   [X.509AM] で提案。新しい CRL が発行されてもその大部分は前回と変わることはないので、変化した部分だけを変換する方式。

3. CRL Distribution Points
   [PKIX] で提案されている証明書の拡張子。CRL を獲得する URL を指定する。単一の CA が証明書ごとに異なる CRL 配布先を定めることが許されており、これにより、配布元の負荷を減らし、CRL のサイズを削減することが出来る。

4. Indirect CRL(代理 CRL)
   [X.509AM] の提案。2 や 3 とは逆の方向で、複数の CRL を束ねて単一の CRL として取り扱う方法である。CRL 配布専用のサーバを用意することで、各 CA の負担を減らすことを意図している。

これらはいずれも、既存の CRL のフレームワークに準じたアプローチであり、実用化が容易な反面、スケーラビリティの問題は、本質的には未解決のままである。OCSP の導入により、確かに通信コストは小さくなるが、リポジトリ自身の秘密鍵を管理する必要が生じ、トータルなセキュリティレベルを逆に低下させてしまうことだろう。

これに対して、CRL を超えた画期的な提案が Naor、Nissim によって成されている [NN98]。彼らのアプローチは、List に代わり、ハッシュの木構造を用いるもので、"Authenticated Dictionary(認証辞書)" と呼ばれている。ユーザ数 n について、CRL のサイズは線形なのに対し、ハッシュ木は $\log n$ のオーダに収まる。しかも、ディレクトリには、ハッシュ木という公開情報の他には秘密に管理する情報はない。従って、帯域とセキュリティの両面から通信コストを下げ、結果として、廃止情報の発行間隔を短くする。

ところで、類似の試みに Kocher による CRT (Certificate Revocation Tree) がある [CRT]。これに対して、Naor らは、CA とリポジトリ (ディレクトリ) の間の通信コストにも着目し、廃止情報の更新にもハッシュ木を活用する点に、新規性を主張している。しかしながら、実は、この更新のハッシュ木は冗長であり、より小さな通信コストで CA とディレクトリ間で廃止情報の同期が出来ることを、本稿で示す。CRT の効果を、より厳密に評価するする為、我々は CRT を操作する基本機能を試験実装した。この試作システムを用いて、CRT と CRL、及び、我々の提案方法を処理速度と通信コストについて比較した結果を報告する。

2 PKI モデル

本稿で考える PKI モデルは、CA、ディレクトリ (サーバ)、エンドユーザから成る (図 1)。更に、独自の秘密鍵を持ち検証サービスを行うディレクトリサーバを OCSP サーバと呼ぶ。

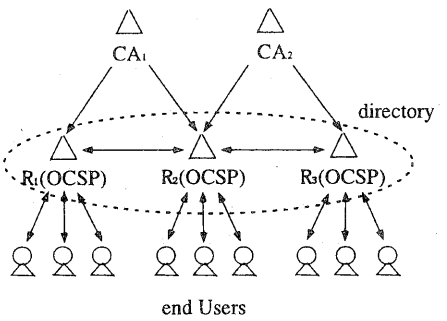

図 1: PKI モデル

ユーザ数 n に対して、CRL のサイズ s がどの様に変化するか考える。証明書の有効期間 (例えば 2 年) を単位時間とし、有効期限前に破棄される確率を廃止率と呼び、p で表す。単位時間内に発行される証明書の総数を m とおく。p が一定で、独立であると仮定すると、m 個中 k 個廃止される確率 $P(k)$ は、p の 2 項分布で与えられる。

$$P(k) = \left(\begin{array}{c} m \\ k \end{array} \right) p^k (1-p)^{m-k}$$

[1] オンラインで証明書の状態を知らせるサービスは、IETF 独自のものではなく、例えば、1997 年に鮫島らによって VA(Verification Authority) が提案されている [S97]。

よって、その期待値は、$E(k) = \sum_k^m kP(k) = mp = s$ で与えられ、同様にして有効な証明書の平均値も $m(1-p)$ となる。これがユーザ数 n に等しくなるので、$m = n/(1-p)$ を得る。結局、CRL のサイズ s は、n と p で定まり、

$$s = \frac{p}{1-p}n$$

となる。すなわち、s は n に比例する。

3 新しい廃止方法

3.1 CRT

Kocher は、2 分ハッシュ木を用いた CRT を提案している [CRT]。廃止したい証明書のシリアル番号 $X_1, \ldots, Xs$ を木構造で表し、ルートのハッシュ値について署名を行ったものを CRT という。例えば、図 2(a) の CRT を考えよう。ここで、$X_1, \ldots, X_4$ は

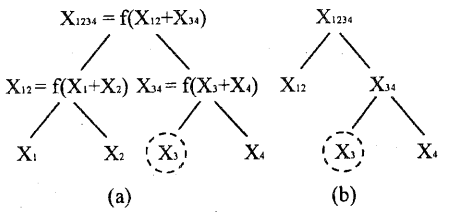

図 2: 木の例

$X_1 < X_2 < X_3 < X_4$ を満たすシリアル番号列であり、"$X_1 + X_2$" は文字列の連結を表す。CA は、X_{1234} についてのみ署名 $\sigma(X_{1234})$ を計算する。

ディレクトリは、エンドユーザからの検証要求に対して、ルートから該当する葉までの経路のみを返す。再署名の必要はない。例えば、図 2(a) の CRT において、X_3 が廃止されていることを証明するには、同図 (b) の部分木だけで十分である。X_i はソートされているので、ある X が廃止されていないことを示すには、$X_i < X < X_j$ となる隣接した X_i と X_j を含む部分木を示せばよい。ディレクトリは秘密鍵を管理する必要がなく、ディレクトリ-エンドユーザ間の通信コストも $\log n$ のオーダに下がる。

3.2 Authenticated Dictionary

Kocher の CRT は、複数の CA の廃止情報を収集する仕様が含まれていて、多少複雑である。これに対して、Naor らは、単一の CA に簡略化し、更に CA とディレクトリの間での廃止情報の同期にも 2 分ハッシュ木を用いる Authenticated dictionary(認証辞書) を提案している [NN98]。また、ハッシュ関数として利用できるための必要条件を明らかにしている。

Naor らのアイディアは、CA が新廃止証明書 X_5 を加える時、木を全て送るのではなく、やはり、ルートからの X_5 へ到る部分木だけを送る、というものである。例えば、X_5 が $X_3 < X_5 < X_4$ で、X_3 の右に付くのならば、図 3(a) の部分木と新しく計算した署名 $\sigma(X_{12354})$ を送る。言わば、差分だけを交換して通信コストを減らす試みであり、差分 CRL に似ている。

3.3 提案方式

Naor らの方法の背景には、木の更新が一意でないことが関わっている。例えば、前節の $X_3 < X_5 < X_4$ の例では、図 3 に示す (a)、(b)2 通りの木が生じる可能性がある。しかし、この問題は、次のルール 1 を導

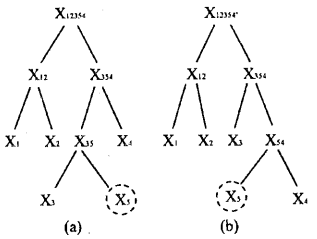

図 3: 木の更新問題

入することで一意にすることが出来る。

(ルール 1)

- 新廃止証明書 X について、$X_i < X < X_j$ を満たす隣接した証明書を X_i と X_j とする。X_i と X_j のルートからの深さを d_i, d_j とする。
- $d_i < d_j$ の時、X_i の右に X を追加する (図 3 の (a))。$d_i > d_j$ の時は、X_j の左に加える (同図の (b))。
- $d_i = d_j$ の時、X_i(小さい方) の右に追加する。

このルールの導入により、木は一意に更新されるので、CA は、新廃止証明書 X と新署名 $\sigma(X_{root})$ だけをディレクトリに配分すればよい。単位時間に廃止される証明書の数は一定なので、更新にかかる通信コストも定数となる。

3.4 バランス木

証明書はランダムに廃止されていくので、CRTは深さがまちまちの木に育つ。これをランダム木と呼ぶ。それに対して、どの葉の深さも一定な木をバランス木と呼ぶ。葉の数が2のべき乗でないときも、平均深さを最小化にする木をバランス木とする。一般に、バランス木の方が計算量が小さくなり望ましい。そこで、ルール1の代わりに、CAとディレクトリ間で常にバランス木を作るようにルール2を定める。

(ルール2)

- 新廃止証明書を加えて、ソートした廃止証明書列を $X_1^1, \ldots, X_s^1$ とする。
- $i = 1, \ldots, \lfloor s/2 \rfloor$ について、$X_i^j = f(X_{2i-1}^{j-1})$ を求める。s が奇数の時は、$X_{s/2}^j = f(X_s^{j-1})$ とする。
- 以上を、$j = 1, \ldots, \lfloor \log(s) \rfloor$ まで繰り返し、最後の X_1^j を X_{root} とする。

このルールの元では、生成される木は一意に決まる。検証にかかる処理時間の平均も、ルール1より小さくなると考えられる。必ずしも最適なバランス木ではないが、一意に決まれば十分である。

4 CRTの実装

4.1 CRTの形式

本実装では、木構造の表現にS式を用いた。ただし、2分木なので、リストの要素は常に2である。ハッシュ値は128bit長のHex表記、証明書のシリアル番号は10進表記、あるいは'[ ]'で閉じることで区別する。CRTの用途からすると、有効期限の表示が必要だが、今回は省略した。例えば、図4の木は次の様に書く。

L_1 = (((3 12) (18 26)) ((31 60) 78))

4.2 基本操作コマンド

1. hash L
   CRT L のハッシュ値を求める。要素の連結には、各要素のMD5の結果をHex表記にして、'+'でつなぐ。全ての葉が証明書である木についても、一部が既にハッシュされた部分木についても、同様に適用出来、結果は変わらない。
   hash L_1 = 006f4df87c409cbc260437fd77b4ccf6

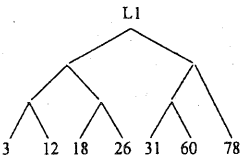

図4: 木L1

2. cut X L
   CRT L から、L の葉 X を含む部分木を求める。部分木の深さは、L における X の深さに一致する。

   L_2 = cut 18 L_1
   = ((h(3 12) (18 h(26))) h(31 60 78))
   hash L_2 = 006f4df87c409cbc260437fd77b4ccf6
   hash L_1 = hash L_2 に注意。
   また、h(3 12) は h(h(3)+h(12)) を示す。

3. add X L
   CRT L に新規証明書を加えた木を生成する。追加する位置は、ルール1に従う。

   L_3 = add 44 L_2
   = (((3 12) (18 26)) (((31 44) 60) 78))
   hash L_3 = 8fd44a7a3d0c9ab70b5856344a4f45ac

4. exist X L
   CRT L の中に X が存在する時、cut X L を返す。存在しない時、すなわちその証明書が有効ならば、$X_i < X < X_j$ となる隣接した X_i と X_j の両方を含む部分木を返す。いずれの場合も、ハッシュ値は変化しない。

   L_4 = exist 44 L_3
   = (h(3 12 18 26) (((h(31) 44) h(60) h(78)))
   hash L_4 = 6b079cee55b033d98da6cd9ed8e336c5
   L_5 = exist 22 L_3
   = ((h(3 12) (18 26)) h(31 44 60 78))
   hash L_5 = 18fba2dbe60992736d15965005d8f99d

4.3 処理フロー

初期状態

- CA: 証明書DB、廃止証明書、CRT L、ハッシュ値 H = hash L、CA の秘密鍵を持つ。
- ディレクトリ D: 証明書 DB、CRT L、$\sigma(H)$、CA の公開鍵を持つ。
- エンドユーザ U: 自分の証明書、CA の公開鍵を持つ。

検証

1: $U \rightarrow D$: 有効性を確かめたい証明書 Xを送る。
2: $D \rightarrow U$: 部分木 L_X = exist X L を求め、署名$\sigma(L)$と共に送る。
3: U: $x \in L_X$かどうかを見て、Xの有効性を知る。$\sigma(H)$ と hash L_X = Hを調べ、改ざんがないことを確かめる。

更新

1: $U \rightarrow CA$: 廃止される証明書 Xが、安全な方法で送られる。CA は $L' = addXL$、$H' = hashL'$、再署名 $\sigma(H')$ を計算する。
2: $CA \rightarrow D$: $\sigma(H')$ と X を送る。D は、CA と同様に、L'、H'を更新し、$\sigma(H')$ を確証する。

4.4 処理時間

ここでは、Sun Ultra(S-7/300U)167MHz、Solaris2.5.1 上で処理時間を計測する。実装システムは、Perl5 と C による MD5 から成る。対する CRL の処理には、SSLeay0.9b を用いた。

まず、hash の処理時間を、図 5 に示す。測定は、OS のクロックを用いて 10 回行った平均値を取った。グラフは一次関数で最小二乗法を適用して求めている。比較の為に、CRL の処理時間 (ハッシュのみ) を示す。共に廃止証明書数 s に対して線形だが、CRT の方が処理の時間が多い。実際には、両方とも署名処理を 1 回づつ求められるが、これは s に依らない。

更新には、ルール 1 を用いたランダム木と、ルール 2 によるバランス木がある。各々の結果を図の上でプロットしているが、実はハッシュを取る処理量に違いはない。ハッシュの計算は、いかなる木も常に $s-1$ 回行われるからである。

次に、cut の処理時間を図 6 に示す。cut を求める為に、全てのハッシュを計算する必要があり、hash と同様に s に対して線形に増加する。それゆえ、ランダム木とバランス木の間で、顕著な差は見られない。

図 7 は、add の処理時間を表している。リストの挿入なので、それほど重い処理ではない。CA とディレクトリの処理としては、この後にハッシュを求める操作が来る。

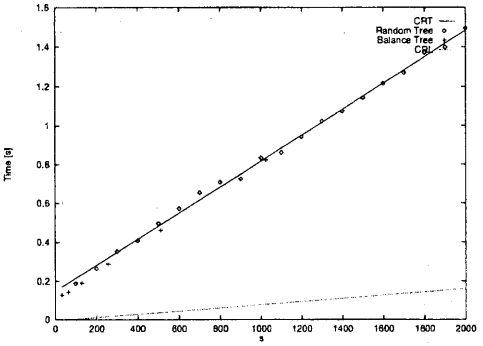

図 5: hash の処理時間

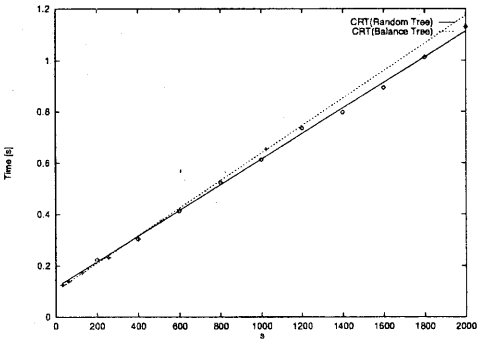

図 6: cut の処理時間

図 8 に示された exist の処理時間は、ディレクトリサーバの負荷を表している。CRT の場合は、廃止されている時と有効な時で処理が異なるので、各々について求めている。また、その各々について、バランス木とランダム木の違いもプロットしている。いずれの場合も、s に線形に増加していく。

一方、OCSP サーバの負荷は、ほとんど署名によって生じるものであり、s に依存しないと考えてよい。SSLeay で 1024bit RSA を使ってサイズ 1 の CRL を作った時の、処理時間 0.185[s] をもって、OCSP1 個当たりの処理とする。図の上で見ると、$s > 100$ あたりから、CRT の処理時間が OCSP のそれを上回っている。OCSP CRT サーバは、署名をする必要がないので、処理も OCSP より軽くなることが予想されていたが、今回の試験実装においてはその効果はほとんどなかった、と言える。

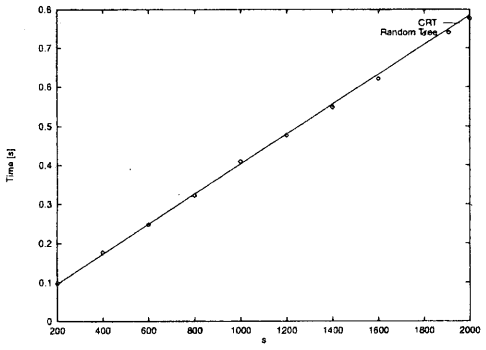

図 7: add の処理時間

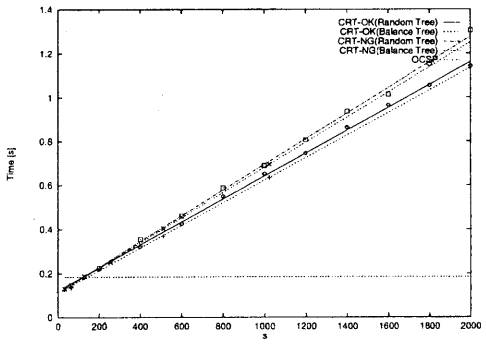

図 8: exist の処理時間

4.5 通信コスト

図 9 は、CRL、CRT の s に対するサイズの変化を表している。この単位で、エンドユーザとディレクトリサーバ間の通信が行われるので、その閉通信路における通信コストの推移と解釈してもよい。s に対して、CRL は線形に、CRT は $\log n$ のオーダで増加する (グラフでは、x 軸について対数をとっている)。また、CRT にはランダム木とバランス木があったが、それらによっても大きな差が生じている。CA とディレクトリ間の通信コストも、これらの値に支配される。CRT が CRL に対して有利なのは明白である。更に、3 節で提案した廃止証明書と署名だけによる更新方法を用いると、通信量は s に依存しなくなり、より効果的である。

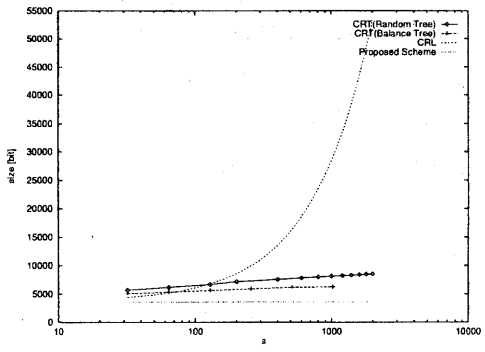

図 9: 通信コスト

5 結論

証明書廃止方法のいくつかについて検討した。その中で CRT システムを試験実装し、処理量と通信コストの面から評価を行った。CRT は、CRL と OCSP を用いたシステムに比べて、通信コストを劇的に下げるが、処理量は廃止証明書数に線形に増加し、OCSP サーバよりも大きくなることがわかった。また、新しい更新方式を提案し、それが、Naor らが提案した方法よりも効率のよいことを示した。

参考文献

[OCSP] X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP, 1998

[X.509AM] Amendment 1 to ITU-T Recommendation X.509—ISO/IEC 9594-8: 1995, *Information Technology - Open Systems Interconnection - The Directory: Authentication Framework*

[PKIX] Internet Public Key Infrastructure, Part I: X.509 Certificate and CRL Profile, 1996

[NN98] Mni Naor and Kobbi Nissim, Certificate Revocation and Certificate Update, in proc. of Seventh USENIX Security Symposium, pp.217-228, 1998

[CRT] P. Kocher, A Quick Introduction to Certificate Revocation Trees (CRTs), `http://www.valicert.com/company/crt.html`

[S97] 鮫島, X.509 認証フレームワークの問題点と解決案, SCIS'97-8B, 1997

[CRS] S. Micali, Efficient Certificate Revocation, Technical Memo MIT/LCS/TM-542b, 1996

[SSLeay] E. Yang, SSLeay, `http://www.ssleay.org`