マルチメディア通信と分散処理 97−23
コンピュータセキュリティ 8−23
(2000. 3. 22)

ハッシュ木を用いたオンライン証明書状態検証サーバ

安部 謙介　　菊池 浩明　　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式の 1 つにハッシュ木を用いた CRT(Certficate Revocation Tree) がある。我々は [KA98] において、S 式を用いた CRT の試験実装を行い、CRL に対して通信コスト面での優位性を示した。本稿では、CRT を 2 分検索木で表現したオンライン証明書状態検証サーバの実装について述べるとともに、[KA98] との比較検討を行う。その結果から、処理コストにおいても CRT が CRL に比べ高速であることを示す。また、本実装に実際の CRL を適用した場合、パフォーマンスがどの程度改善されるかについても報告する。

Online Certificate Status Verification Server Using Binary Search Hash Tree

Kensuke Abe　　Hiroaki Kikuchi　　Shohachiro Nakanishi

Tokai university, Faculty of Engineering

Abstract: CRT(Certificate Revocation Tree) is a method using hash tree for public-key certificate revocation. In [KA98], we have implemented an experimental CRT system using the S-expression, and shown that its communication cost is smaller than that of CRL. In this paper, we implement an online certificate status verification server using CRT expressed in binary search tree, and examine the system performance in comparison with [KA98]. Based on experimental data, we show that the latency of CRT is smaller than that of CRL. We also estimate the performance of the system to which an actual revocation data derived from a CRL is applied.

1 はじめに

CRL(Certificate Revocation List) は、PKI における証明書廃棄方法のスタンダードである [X.509AM]。しかしながら、発行間隔が広く即時的な利用に不向きな点と、通信コストが大きい点が問題であり、これに対して、証明書廃止木 (CRT:Certificate Revocation Tree)[CRT]、認証辞書 (AD:Authenticated Dictionary) [NN98]、差分リスト [KA98] が提案されている。

我々はこれまでに、差分リスト法を試験実装し、測定結果を基にして、差分リストによる計算コストと通信コストの削減量を示した [KA98, IWSEC]。さらに、[SCIS99, ISW] では、2 分木であった CRT を k 分木に拡張し、計算コストと通信コストの両面から、[KA98] との比較を行った。[KA98] の試験実装では、CRT の通信コストの優位性を示すことが主題である為、S 式による CRT の表現方法を用いていた。それゆえ、部分木を求める際、木全体の走査を行う必要性があった。また、キャッシング等の最適化を行っていなかった為、各操作の度に、木全体のハッシュ値を再計算する必要があった。結局、処理時間は廃止証明書数 n に線形に増加し、木構造の特長を生かせていなかった。

そこで我々は、JAVA を用いて、動的に木構造を

メモリ上に構成するオンライン証明書状態検証サーバの実装を行った。本稿では、システムの実装とそのパフォーマンスを報告する。本システムでは、S式の代わりに2分検索木を主記憶上に有機的に展開して失効証明書データを表現し、さらに各ノードについてハッシュ値をキャッシングして処理を効率化した。実測データより、処理コスト面でも、CRTはCRLと比較して遥かに優位であることが示された。さらに、実際のCRLに基づいた失効データを本実装のCRT形式に変換し、パフォーマンスがどの程度改善されるかについても報告する。

2 基本定義

2.1 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。ディレクトリサーバは、独自の秘密鍵を持ち検証サービスを行う。

2.2 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$X_1, \ldots, X_n$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。ここで、シリアル番号は常に木のリーフで表現されることに注意せよ(図1(b)参照)。

3 2分検索木

本節では、本実装で用いる2分検索木(Binary Search Tree: 以下BS木と略す)について詳しく説明する。[KA98]で用いた2分木の場合、シリアル番号は全てリーフにあった。しかし、BS木の場合は、ノードにもシリアル番号が入る為、木のハッシュ計算方法を再考する必要が生じる。

3.1 データ構造

BS木は、以下の様な特徴を持つ。

- n個のノードから成るデータ構造である(リーフ: 子を1つも持たないノード)。
- 各ノードは、key、$hash$、$left$、$right$の属性を持つ。ノードxにおける$key[x]$は、失効証明書のシリアル番号、$hash[x]$はハッシュ値である。
- ノードxの右部分木に属する任意のノードy、左部分木に属する任意のノードzについて、$key[y] \leq key[x] \leq key[z]$が成り立つ。

図1(a)にBS木の例を示す。ここで、X_i: シリアル番号であり、$X_1 < X_2 < X_3 < X_4$が成り立つ。

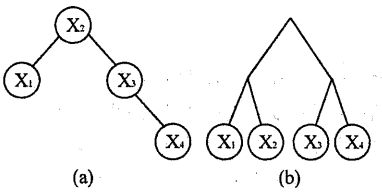

図1: ハッシュ木の例((a)BS木、(b)[KA98]の2分木)

3.2 ハッシュ値の定義

ノードxのハッシュ値$hash[x]$を次の様に定める。

1. xがNILの時、$hash[x] = NIL$とする。
2. $hash[x] = f(hash[left[x]]||key[x]||hash[right[x]])$ 但し、fは一方向性ハッシュ関数、'||'は文字列の連結とする(fの値域は文字列とする)。
3. 木Tのハッシュ値は、$hash[T] = hash[root[T]]$と定める。但し、$root[T]$はTの根となるノードである。

(例) 図1の(a)の木T_aと(b)の木T_bについて、
$h(T_a) = f(f(X_1)||X_2||f(X_3||f(X_4)))$
$h(T_b) = f(f(X_1||X_2)||f(X_3||X_4))$

最後に、部分木の例を図2に示す。ここで、●: 検証要求値、○: シリアル番号、×: ハッシュ値である。BS木(同図(a))と2分木の場合(同図(b))との違いを認識されたい。

3.3 2分検索木の更新

Kocharらのリーフに失効証明書を表すCRTの場合、CRTの更新の際に2通りの木が生じる可能性があった。一方、本システムの2分検索木の場合、ノードへの追加の順序について更新は一意に行われる。よって、2分検索木の場合も、[KA98]と同様に、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。

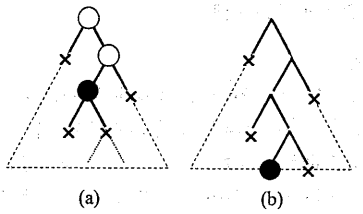

図 2: 部分木の例 ((a)BS 木、(b)[KA98] の 2 分木))

3.4 ノードに証明書を割り当てるデータ構造の通信コストについて

[KA98] や Kochar らの CRT では、失効証明書はリーフのみに許されていた。一方、本稿の BS 木では、全てのノードにも失効証明書が挿入される。この 2 つのデータ構造について、検証要求に対する通信コストの差を考察する。

まず、証明書数 n に対する木の高さについては、リーフで表現する時の高さ h_L よりも、全てのノードで表現する時の高さ h_N の方が、明らかに低い。完全バランス木の仮定の下では、各々、$h_L = \log(n)$、$h_N = \log(n+1) - 1 < h_L$ で与えられる。

次に、部分木のサイズを見積もる。リーフで表現する場合は、ルートから該当するリーフノードまでの h_L 個のノード各々について、対応する h_L 個のハッシュ値が必要である。ハッシュ値の大きさを共に固定長と仮定すると、合計 $l_L = h_L + 1$ 個分のデータ長になる。一方、全ノードで表現する場合 (BS 木) には、それに加えて、h_N 個のシリアル番号が必要となる為、$l_N = 2h_N + 1$ 個分のデータ長となる。結局、高さとノード当たりのデータ長にトレードオフがあり、図 3に示す通信コストが得られる。実質的には、殆どの場合で $l_N > l_L$ であることが分かる。

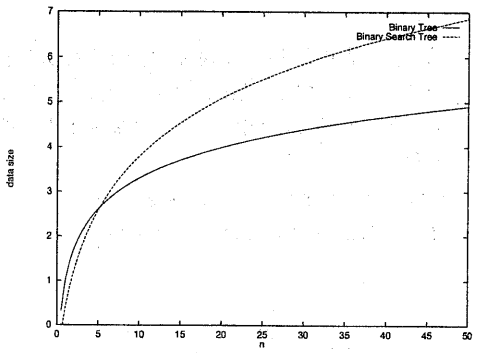

図 3: リーフのみで表現する 2 分木と、全ノードで表現する BS 木における、部分木のデータ長の比較

表 1: システム緒言

プラットフォーム	Sun Ultra S-7/300U 167MHz Solaris 2.5
開発言語	JAVA Development Kit Ver.1.1
ハッシュ関数	MD5(coded in JAVA)
通信プロトコル	独自ソケット通信方式

4 証明書状態検証サーバの実装

4.1 システム構成

PKI モデルにおけるディレクトリとエンドユーザ間の CRT 検証時の通信を、CRT サーバとクライアントで実現した。CA とディレクトリ間は、CRT サーバ上でのデータベースの更新処理に対応する。システム緒言を表 1、各初期状態を次に示す。

- CA: 証明書データベース (DB)、廃止証明書、CRT T、ハッシュ値 H = hash $[T]$、CA の秘密鍵を持つ。
- ディレクトリ D: 証明書データベース (DB)、CRT T、H についての CA の署名 $\sigma_{CA}(H)$、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。

4.2 CRT のデータベース形式

CRT の入出力データには、以下の形式を用いた。データ中におけるシリアル番号は、ロードした時に同一の木を再現できる順序でソートされる。

1. 全体木

   $x_1, \ldots, x_n$ の n 個のノードから成る CRT 全体は、次の形式で表現される。

   $key[x_{i_1}]{:}hash[x_{i_1}], \ldots, key[x_{i_n}]{:}hash[x_{i_n}]$

   ここで、key は失効証明書のシリアル番号を ACSII 表記したもの、$hash$ も同様である。$i_1, \ldots, i_n$ は、$1, \ldots, n$ の並び換えであり、n 個のノードを挿入する順序を決定する。シンボル':' と',' はデリミタである。また、$:hash[\,]$ は省略可とする。

   (例) 図 4の CRT の場合

   50,25,12,37,30,33,43,75,87,93,97

2. 部分木

部分木の表現方法も、基本的に全体木の場合と同様である。但し、検証の際に重要な値 (検証値、又は両端値) はそのシリアル番号を'[ ]' で囲むことで区別する。

(例) 図 4の CRT の場合 (検証値: 12)
50, 25, [12], 37:hash[37], 75:hash[75]

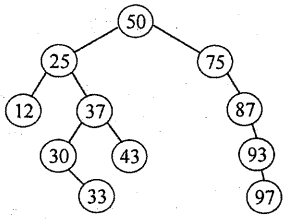

図 4: サンプル木: T

4.3 基本操作コマンド

4.3.1 CRT サーバ側

1. hash T
CRT Tのハッシュ値を求める。その際、各ノードにサブツリーのハッシュ値をキャッシュする。

2. add x T
CRT Tに新しいシリアル番号 $key[x]$ を挿入する。その際、必要なノードのみハッシュの更新を行う。

3. exist y T
ユーザからの検証要求 yに対して、$key[x] = y$ となるノード x を含む部分木を求める。存在しない場合、即ちその証明書が有効ならば、$y_i < y < y_j$となるノード y_iと y_jの両方を含む部分木を返す。その際、部分木に含まれるハッシュ値は、各ノードにキャッシュされた値を利用する。従って、ハッシュ再計算は一切生じない。

4. save *filename*
CRT の木全体を 4.2 節-1 の形式でファイルに保存する (CRT の保存)。

5. load *filename*
CRT の木全体をファイルからメモリへロードする (CRT の復元)。

4.3.2 クライアント側

1. verify y
CRT サーバに検証したい証明書のシリアル番号 yを送る。exist(4.3.1 節) の手続きに従って計算された部分木、及びルートのハッシュ値 $hash[T]$ を、CRT サーバから受信する。データ形式は 4.2 節-2 に従う。

2. hash T'
サーバから受信した部分木T'のハッシュ値 $hash[T']$ を計算する。その値と受信したルートのハッシュ値を比較し、不正が無いことを確かめる。

4.4 処理フロー

検証処理を次に示す。

1. $U \rightarrow D$: 有効性を確かめたい証明書のシリアル番号 yを送る。

2. $D \rightarrow U$: 部分木 T_y = exist y Tを求め、署名 $\sigma_{CA}(T)$ と共に送る。

3. U: $y \in T_y$かどうかを見て、yの有効性を知る。$\sigma(H)$ と hash $[T_y] = H$を調べ、改ざんがないことを確かめる。

更新処理を次に示す。

1. $U \rightarrow CA$: 廃止される証明書のシリアル番号 y が、安全な方法で送られる。CA は T' = add y T、H' = hash $[T']$、再署名$\sigma_{CA}(H')$ を計算する。

2. $CA \rightarrow D$: $\sigma_{CA}(H')$ と y を送る。D は、CA と同様に、T'、H'を更新し、$\sigma_{CA}(H')$ を確証する。

5 評価

5.1 処理時間

本実装と [KA98] で求めた S 式形式による CRT (Perl5 と C による MD5 から成る) との処理時間を比較する。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 100 回行った平均値を取っている (S 式は 10 回行った平均値)。図の直線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。3 つとも、廃止証

明書数 n に対して線形だが、CRT の方が処理時間が多い。また、[KA98] と本システムを比較すると、やや本システムの方が速いが大きな差は見られない。実際のハッシュの計算は、[KA98] の場合 $n-1$ 回、本システムの場合 n 回必要となる。

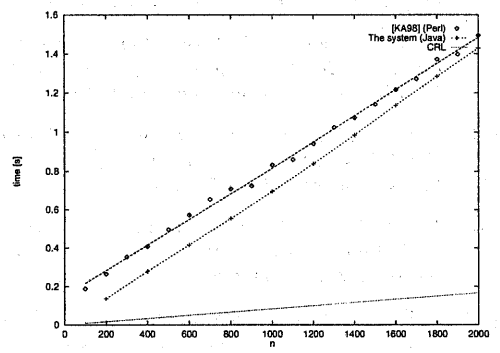

図 5: hash の処理時間

次に add、exist の処理時間を図 6に示す。まず、add の処理時間では、[KA98] は $O(n)$ で、本システムは $O(\log n)$ で増加している (グラフでは、x 軸について対数をとっていることに注意せよ)。[KA98] では、木にハッシュ値のキャッシングを行っていないので、毎回全体のハッシュを計算する必要がある。よって、hash と同様な特性が得られる。それに対し、本システムでは木のノードにハッシュ値のキャッシング行っている為、挿入する際に通ったノードのみを新たにハッシュするだけで良い。

また、注目すべきなのは、本システムの処理時間が CRL のそれを上回っていることである。処理時間の面でも、CRT は CRL よりも優位であると言える。

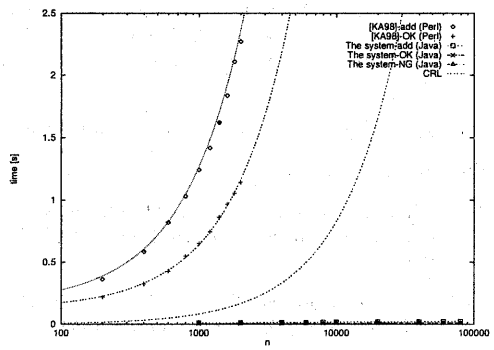

図 6: add、exist の処理時間

次に、exist の処理時間では、[KA98] の場合は $O(n)$ で、本システムの場合は $O(\log n)$ で増加する。[KA98] の場合、部分木を求める際に全てのハッシュを再計算する必要があるが、本システムでは、各ノードにハッシュ値がキャッシュされている為、再計算する必要はない。木の深さに比例する検索コストしかかからず、これが $O(\log n)$ の理由である。また、[KA98] の場合では、証明書が廃止されている場合と有効な場合で処理時間に差が見られたが、本システムの場合は、違いを認識することは出来なかった。

5.2 通信コスト

図 7は、CRL、CRT の n に対する部分木のサイズの変化を示している。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できる。n に対して、CRL は線形に、CRT は $O(\log n)$ で増加する。また、BS 木の方が S 式と比較して全体的に部分木のサイズが大きくなる (3.4 節参照)。

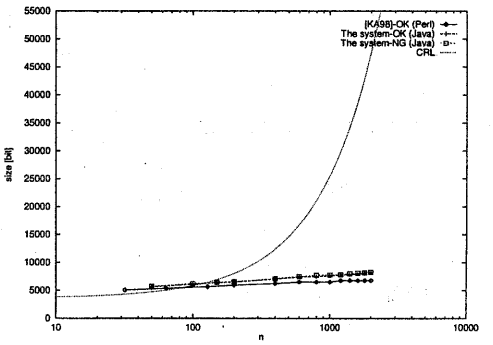

図 7: 通信コスト (部分木のデータ長)

5.3 実際の失効データによる評価

本実装に、実際の CRL データを入力した場合、CRT がどの程度不平衡になるかを調べる。データでは、[VeriSign] から入手した RSA 社の CRL から求めた。詳細を表 2に示す。シリアル番号を失効日順にソートしてから、本 CRT に挿入した。

図 8に、CRT の平均深さを求めた結果を示す。実測値、完全にバランスした時の理論上の最小値とも、$O(\log n)$ で増加しているが、実測値の平均深さの方が 1.26 倍大きくなっている。実際の CRT は不平衡にな

表 2: CRL 詳細

CRL 名	RSASecureServer.crl
期間	1997.02.14 - 1999.09.27
無効証明書数	20336
シリアル番号のサイズ	128 [bit]

ることが分かった。

ランダムな順番に挿入される場合、一般的な BS 木はほぼ平衡な形となる。しかし、ソートされた順番で挿入される場合は、$O(n)$(線形の深さ) に近くなる。証明書は番号順に発行されるので、失効される証明書の順番も少なからずソートされていると予想される。通信コスト面を考慮すると、今後何らかの対策が必要である。

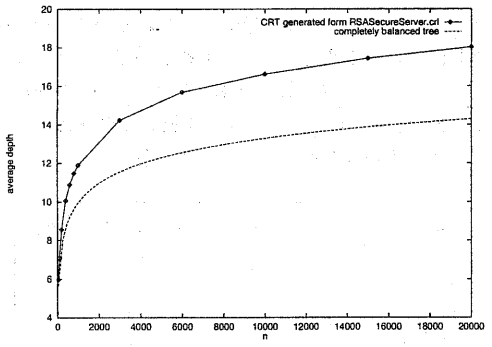

図 8: CRT の平均深さ

6 結論

JAVA を用いて、オンライン証明書状態検証サーバを実装した。CRT のデータ構造に 2 分検索木を用いることにより、ハッシュ値のキャッシング等を実現した。その結果、[KA98] に比べて検証時の処理時間を劇的に高速化し、本来の CRT のパフォーマンスを示した。また、実際の CRL データの基づいて、木の平衡について議論した。

参考文献

[X.509AM] ITU-T Recommendation X.509—ISO/IEC 9594-8: 1995

[PKIX] R. Housley, W. Ford, W. Polk, D.Solo, Internet X.509 Public Key Infrastructure Certificate and CRL Profile, Internet RFC 2459, 1999

[OCSP] M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams, X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP, Internet RFC 2560, 1999

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

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

[RFC2510] C. Adams, S. Farrell, Internet X.509 Public Key Infrastructure Certificate Management Protocols, Internet RFC 2510, 1999

[RFC2559] S. Boeyen, T. Howes, P. Richard, Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2, Internet RFC 2559, 1999

[RFC2585] R. Housley, P. Hoffman, Internet X.509 Public Key Infrastructure Operational Protocols, Internet RFC 2585, 1999

[RFC2587] S. Boeyen, T. Howes, P. Richard, Internet X.509 Public Key Infrastructure LDAPv2 Schema, Internet RFC 2587, 1999

[KA98] 菊池, 安部, 中西, 2 分ハッシュ木を用いた証明書廃止・更新システム, 情報研報 Vol.98, No.84,pp.51-56,1998

[SCIS99] 菊池, 安部, 中西, k分ハッシュ木による証明書廃止木 (CRT) の更新方式の提案と評価, 1999 年暗号と情報セキュリティシンポジウム (SCIS'99), Vol.2, pp.621-626, 1999

[IWSEC] H. Kikuchi, K. Abe, S. Nakanishi, Performance Evaluation of Public-key Certificate Revocation System with Balanced Hash Tree, proc. of International Workshop on Security (IWSEC'99), 1999

[ISW] H. Kikuchi, K. Abe, S. Nakanishi, Performance Evaluation of Certificate Revocation Using k-Valued Hash Tree, International Information Security Workshop (ISW'99), Springer Lecture Notes in Computer Science 1729, pp.103-117, 1999

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

[VeriSign] VeriSign, `http://www.verisign.com`

[ALG] Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, Introduction to Algorithms, 近代科学社, 1995

[JAVA] Robert Lafore[著], 岩谷 宏 [訳], Java で学ぶアルゴリズムとデータ構造, SOFTBANK, 1999