マルチメディア通信と分散処理　106－6
コンピュータセキュリティ　16－6
（２００２．２．１４）

HTTPS における証明書管理方式

高須紀樹†　若山公威†　村瀬晋二‡　鈴木春洋‡　岩田彰†

†名古屋工業大学　電気情報工学科
‡株式会社シーティーアイ

HTTP に SSL/TLS を適用した HTTPS を利用すれば，通信相手の認証，データの改竄防止，データの暗号化といった機能が得られ，通信の安全性が保障される．しかし HTTPS を利用する場合には，利用者側に PKI 技術の正しい理解が求められる．本論文では，その複雑さゆえに利用者側で起こりうる問題点を述べ、イントラネット内のユーザが外部の Web サイトに HTTPS で通信する場合にターゲットを絞った対策方法をを提案する. さらにこの方式を実装, 評価したのち, その利用形態について考察する. この方式の適用により，ユーザにかかる負担を軽減し，署名検証作業の効率化を図ることができると考えられる.

Method of certificate management for HTTPS

Toshiki TAKASU†， Kimitake WAKAYAMA†， Shinji MURASE‡，
Shunyo SUZUKI‡, Akira IWATA†

†Nagoya Institute of Technology
‡CTI Co.,Ltd.

If HTTPS which applied SSL/TLS to HTTP is used, communicative safety is secured by obtaining the function of authentication of the peer, prevention of message alteration, and encryption message. However, a user side is asked for the right understanding of PKI technology when using HTTPS. In this paper, we refer the problem that the complexity may generate in a user side, and make a proposal in case users in intranet access outer website. Furthermore, after implement and evaluate in this system, we consider the use form. By appling a proposal method, we think that the burden to a user is mitigated and signature verification work can be done efficiently.

1. はじめに

近年，ショッピングや株取引に代表されるオンラインビジネスが盛んになってきており，インターネットの利用が企業活動の中心となりつつある。

そこではいかに安全性を確保するかが重要な課題であり，システムの信頼性に直接関わるため，オンラインビジネスにとって暗号通信，通信相手認証といったセキュリティ技術の利用は今や欠かすことができない．この分野は「電子署名および認証業務に関する法律」（電子署名法）が 2001 年 4 月に施行されるなど，法整備も進んでおり関心が高まっている.

Web 上で安全性を確保する方法としては，SSL(Secure Sockets Layer)[1]/TLS(Transport Layer Security)[2]を用いた HTTPS がある．これは HTTP での暗号通信方式として最も有力なもので，多くの Web サイトで利用されている.

SSL ではサーバ認証のみではなく，クライアント認証の機能も用意されている．したがってこの機能を利用すれば，不特定多数の人からのアクセスを許可したくない会員制サイトなどで，より厳密なアクセス制御が行えると考えられるが，現状ではサーバ認証を行った上でユーザ ID とパスワードを入力する方法が一般的である.

厳密な認証が可能となる方法があるにもかかわらず，それがあまり使用されていない理由として，運用方法が複雑であり，ユーザへの負担が大きいことが挙げられる. この問題は SSL のみに関わらず電子証明書を利用する PKI 技術全般について言えることである．我々はこれまでにもこの問題に関していくつかの改善策の提案とその実装を行ってきた．[3][4][5]

本稿では，HTTPS 通信の際の証明書管理と検証の複雑さについて言及し，その改善方法を提案する.

2. SSL

2.1. 概要

SSL は Netscape Communications 社により提唱された，通信の暗号化，相手認証，データ完全性の保障を実現する技術である．

SSL プロトコルはセキュアなデータ通信を確立するまでのハンドシェイク，そして機密性を持ったデータ通信を行うフェーズの 2 つの部分からなる．ハンドシェイク内では，サーバ／クライアント間で暗号通信するときに使用する鍵の共有や通信相手の認証が行われ，これには電子証明書を利用した公開暗号方式が用いられる．そして実際のデータ通信には共通鍵暗号方式が用いられている．

2.1.1. セッションとコネクション

SSL プロトコルでは，通信する 2 つのプロセス間に，セッション，コネクションと呼ばれる 2 つの接続状態を持つ．セッションではセッション ID，相手の証明書，マスタシークレットなどの情報を保持し，コネクションでは暗号通信の際に使用される鍵データなどの情報を保持している．また 1 つのセッション内に複数のコネクションを保持することが可能で，一度の暗号方式の取り決めで複数回のデータ交換をすることができる．

2.2. ハンドシェイク

SSL では以上のような機構を設けることにより，ハンドシェイク･プロトコルにおいて新規セッション確立の場合と，既存セッション再開の場合を用意して通信の効率化を図っている．セッションの管理はセッション ID と呼ばれる識別子を使って行う．クライアントからは，直前まで使われていたセッション ID を送信することにより，セッションの再開要求を出すことができる．セッションを再開する場合は，新規セッション開始よりも少ないメッセージ交換でアプリケーションデータのやり取りを開始することができる．

以下に相互認証(サーバ認証+クライアント認証)を新規に開始する場合の SSL 処理の概要を示す．

1. クライアントからサーバへ Client Hello メッセージを送信．このメッセージはクライアントが生成した乱数 *Rc* などを含んでいる．
2. サーバは Server Hello，Server Certificate メッセージなどを送信．Server Hello メッセージにはサーバが生成した乱数 *Rs* やセッション ID などが含まれている．
3. クライアントはサーバ証明書の検証を行い．ユーザ証明書，プレマスタシークレットと署名データをそれぞれ Client Certificate，Client Key Exchange, Certificate Verify メッセージでサーバに送信．
4. 両者で共有された *Rc*，*Rs*，プレマスタシークレットからマスタシークレットを生成．さらに *Pc*，*Rs*，マスタシークレットから暗号化のための共有鍵を生成．
5. 残りの通信処理を行いセッション確立．
6. 暗号通信の開始．

既存セッションを再開するときには，Client Hello, Server Hello メッセージを交換し，その後，数個のメッセージ交換をした上で暗号通信が開始される．このときの Client Hello メッセージには，前回の通信で受け取ったセッション ID を含める．

3. HTTPS 利用上の問題点

HTTPS を使った Web サイトにアクセスする場合に，利用者側で起こりうるいくつかの問題点を本章で説明する．

3.1. サーバ認証時にかかるユーザへの負担

標準で組み込まれていない CA 証明書を使って認証を行う場合は，あらかじめそれらの証明書をブラウザに組み込んでおく必要がある．イントラネット内で CA を構築し運用している場合，この作業は必須となる．

また，組織のポリシに従って信頼できない CA の証明書をブラウザから削除しておく必要がある．

さらに，厳密な認証を行うために，CRL(Certificate Revocation List)などを定期的に取得し，サーバ証明書の失効状態を検証する必要がある．もしこれを行わない場合は，不正なサイトであっても気づかないために不利益を被る可能性がある．

3.2. クライアント認証におけるユーザへの負担

クライアント認証には，ユーザ証明書とそれに対応した秘密鍵が必要であるため，それらをユーザ自身が保持しておく必要がある．もしユーザ証明書の有効期限が切れた場合には，CA に再発行を要求して新しい証明書を入手する必要がある．

3.3. 失効情報の通信量増加

サーバ証明書の失効状態を検証するために CRL を利用する場合, CRL を更新間隔に合わせて定期的に入手する必要がある．CRL は各端末で必要になるため，イントラネット内の通信量が増加する可能性がある．

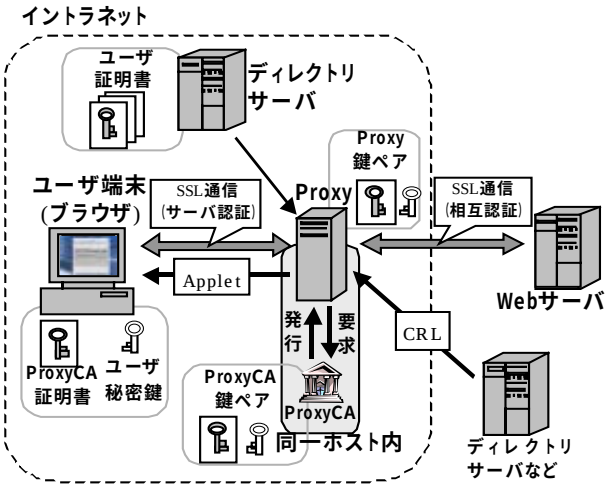

図 1　システムの全体構成

4.　問題点に対する解決策

前章の問題点を解決するために, Proxy を利用することにより, ユーザが自分の証明書の管理やサーバ証明書の検証について気にすることなく, HTTPS 通信が行える方式を提案する. なお本方式では, ユーザが Proxy を信頼できるイントラネットのような環境を想定している.

4.1.　システム構成

システムの全体像は図 1 のようになる.

この方式では, イントラネット内メンバの各ユーザ証明書はディレクトリサーバに一括して保管しておく. また, その証明書の対となるユーザの秘密鍵は, ユーザ端末に ProxyCA の証明書とともに保管しておく. この ProxyCA 証明書は有効期限が切れたら更新する必要がある.

もし Web サーバからユーザ証明書の提示を求められた場合は, Proxy がディレクトリサーバから証明書を取得し Web サーバに送信する. このように, ブラウザによるユーザ証明書の保管をやめることで, ユーザ証明書のインポートなどの作業からユーザが解放される. また, サーバ証明書の確認は Proxy に於いて行うようにする. 必要であればサーバ証明書発行元 CA から CRL を取得して証明書の失効状態の検証を行う. これにより, サーバ証明書の検証時にかかるユーザの手間を省くことができる.

さらに CRL は Proxy だけが取得すれば良いので, 各ユーザ端末が個別に CRL を取得するときと比べ, イントラネット内のトラヒックが削減される.

4.2.　実装上で発生する問題点とその解決法

前章で示した方式を実装しようとした場合, Proxy はサーバ証明書の検証やユーザ証明書の提示を行うために, SSL 通信の内容について知る必要が生じる. このため, ブラウザ-Proxy (以下 B-P), Proxy-Web サーバ (以下 P-W) 間でそれぞれ別のセッションを確立する必要がある. しかし, そうなれば当然, 各セッション間で共有する共有鍵が別のものになることにより, Proxy では暗号通信の都度, 一方から送られてきたデータを一度復号化し, 他方へ暗号化しなおして送信するという処理が生じてしまう. このことは Proxy での負荷が増大することに繋がる. また, Proxy がブラウザに送るサーバ証明書は Proxy の証明書であるため, セッションの確立時にブラウザが警告メッセージを表示してしまう.

そこでこれらの問題を考慮し, 提案方式では従来の SSL ハンドシェイクを展開する形で, 証明書の管理および更新作業からユーザを解放しつつ, Proxy での負荷をなるべく軽減させることを目指す. このため, アプリケーションデータをやり取りする際には極力, 通信内容への干渉を避けたい. また, 警告メッセージが表示されてしまうことについても対策を行う. このことを実現させるためにはセッション ID, 共有鍵, 署名データ, そして Proxy 証明書について考慮する必要がある.

4.2.1.　セッション ID

Proxy で証明書関連の処理を行う場合, B-P 間, P-W 間でそれぞれ別のセッションが確立する. しかし, もし B-P 間, P-W 間で同じセッション ID を共有することができれば, 仮想的に一つのセッションで B-W 間が結ばれているとみなすことができる.

そこで新規セッションを確立させるときに, 一方の Client Hello あるいは Server Hello メッセージに含まれるセッション ID を抜き出し, 他方のそれとして利用する.

4.2.2.　共有鍵

B-P 間, P-W 間ではセッションが異なるため, 当然共有鍵も別のものとなる. しかし, もし B-P 間, P-W 間で同じ共有鍵を持つことができれば, Proxy で復号化／暗号化の処理が必要なくなる. そこで, 鍵生成に必要なマスタシークレットを共有することを考える. マスタシークレットは, Server Hello, Client Hello メッセージに含まれる乱数と, Client Key Exchange メッセージに含まれるプレマスタシークレットを基に生成される. したがってこれら 3 つの値をブラウザ, Proxy, Web サーバで共有させてやればよい.

4.2.3.　署名データ

SSL クライアント認証にはユーザ証明書の他に, ユーザ秘密鍵を使って生成する署名データが必要となる. し

かし秘密鍵はユーザ端末に保管されているため，Proxy では署名データを生成することはできない．そこでブラウザに Applet を送り，その Applet を介して署名データを取得することでこの問題の解決を図る．アプレットには書名をつけた上で Java2 のセキュリティ機構を利用して安全策を講じておく．

4.2.4. Proxy 証明書

通常 Web サーバから送信されてくるサーバ証明書に記述されている，Subject 内の CN には，Web サーバの DN が入っている．ブラウザではこの箇所を URL と比較し，証明書のチェックを行っている．本方式では，ブラウザに送られてくるサーバ証明書は Proxy 証明書であるため，Web サーバの DN と Subject 内の CN が一致しない．したがってセッションの確立時に，ブラウザで意図しているサーバ証明書でない旨の警告メッセージが表示されてしまう．そこで Proxy と同一ホストにイントラネット内 CA の下位 CA(ProxyCA)を構築し，ProxyCA にて Subject 内 CN を接続先 Web サーバ DN と一致させるように動的に証明書を作成する．これにより警告メッセージの発生を回避する．

4.2.5. 提案方式のハンドシェイク

図 2 は B-W 間でデータ交換を開始するまでに，提案方式で使用するハンドシェイクを表している．B-W 間のセッションが確立するまでに少なくとも 4 回のコネクションを B-P 間に確立させる．このセッションが確立されれば，マスタシークレットの共有により，B-W 間で共通の鍵を持つことができる．この結果，アプリケーションデータを一度復号化してから暗号化するという処理が必要無くなる．さらにセッション ID の共有により，既存セッション再開時のハンドシェイクでは，一方からのメッセージ内容をそのまま他方へ送信することが可能になる．

以下では相互認証時の B-W 間で，データ通信が可能となるまでの手順を示す．

1. ブラウザが送信した Client Hello メッセージを受け取った Proxy は，そのメッセージに含まれる乱数 *Rc1* を抜き出し，Web サーバに対して *Rc1* を利用して自らが Client Hello メッセージを作成して送信する．
2. Web サーバから Server Hello メッセージを受信する．Proxy は *SessionID* と乱数 *Rs1* を抜き出し，その値を利用して作成した Server Hello メッセージをブラウザに送信する．Web サーバから受信したサーバ証明書の Subject 部を利用して Proxy 証明書を ProxyCA で発行し，ブラウザにサーバ証明書を送る．

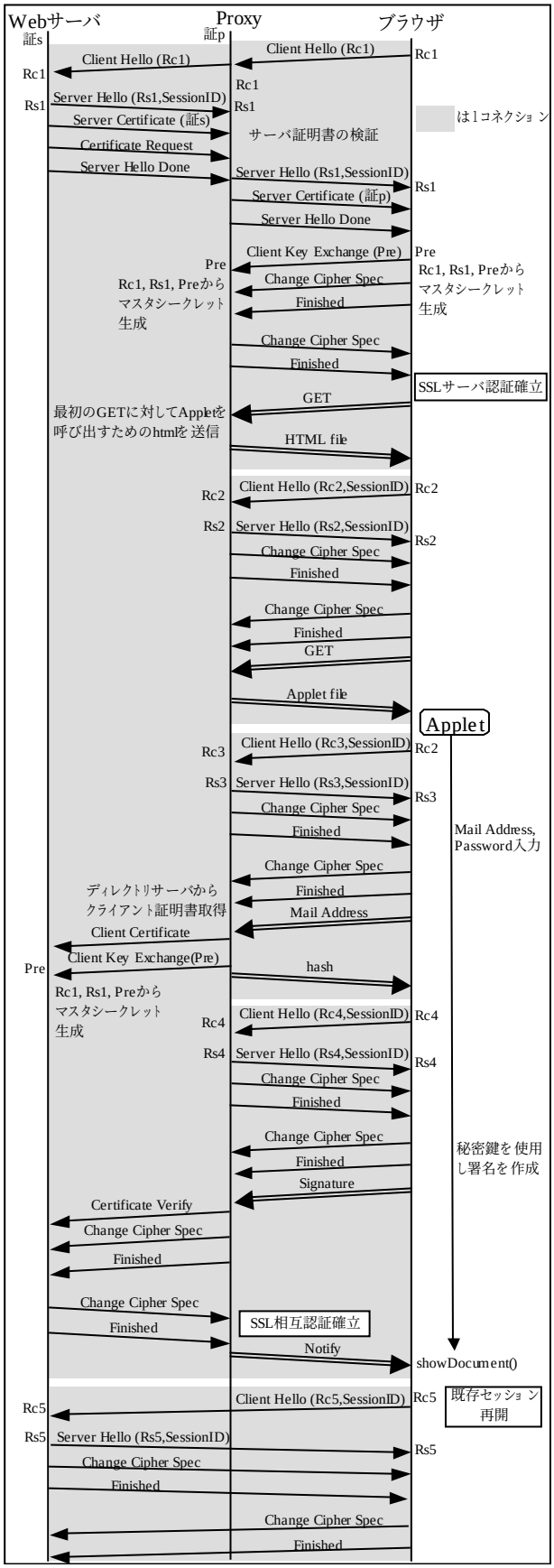

図 2　提案方式のハンドシェイク

3. Clieht Key Exchange メッセージによってブラウザからプレマスタシークレット *Pre* を受信した Proxy では *Rc1*, *Rs1*, *Pre* を使い，マスタシークレットを生成する．さらに *Rc1*, *Rs1*, マスタシークレットから共有鍵を生成する．*Pre* は後で Web サーバに送信するために記憶しておく. 最後に Finished メッセージを交換することにより，まずは B-P 間で SSL サーバ認証が確立する.
4. ブラウザから閲覧したいページの URL を受信するが，そのデータの代わりにあらかじめ用意しておいた HTML ファイルをクライアントへ送信する．この HTML にはアプレット表示のためのタグが含まれているため，引き続きブラウザは Proxy からアプレットをダウンロードする.
5. ユーザはアプレット上でメールアドレスとパスワードを入力する. メールアドレスは Proxy へ送信され，ディレクトリサーバからユーザ証明書を取得するときに使用する．パスワードは秘密鍵にアクセスするために使用するので，ローカルから外に出ることは無い.
6. ユーザ証明書と *Pre* を Web サーバに送信する. これにより Web サーバでは *Rc1*, *Rs1*, *Pre* からマスタシークレットを得ることができる．その後 Proxy は署名に必要なハッシュデータをアプレットに送信する.
7. ローカルに保存された秘密鍵でハッシュデータに署名を行う．署名データを受信した Proxy はこの値を Change Cipher Spec メッセージにし Web サーバへ送信する.
8. 全ての処理が滞りなく終了し，P-W 間で Finished メッセージが交換されると相互認証が確立する.
9. 正常終了したことを受け，アプレットは java.applet.AppletContext クラスの showDocument()メソッドを実行し，本来閲覧したかったページの URL をもう一度要求するようブラウザに促して終了する.
10. B-W 間でデータの送受が開始される.

以上により，ユーザ端末でユーザ証明書を保管していなくても，相互認証を行う Web サーバとのデータ交換が可能となる．ただし，本方式では上記の 9.で showDocument()メソッドを使用しているため，ブラウザからの最初の HTTP メソッドが GET 以外の場合には対応していない.

5. 実装と評価

5.1. 実装

本方式では，電子署名付きアプレットからローカル資源への詳細なアクセス制限を設定するため, Web ブラウザには Java2 Plug-in が必要となる．また，電子書名付きアプレットからユーザ秘密鍵を読み込めるように，あらかじめポリシーファイルを設定しておく．ユーザ端末には署名検証のため，ProxyCA の証明書が必要となるが，それ以外の CA 証明書，ユーザ証明書，CRL などは不要となる.

Proxy の実装には暗号ライブラリ AiCrypto1.8.3, iPlanet Directory SDK for C 5.0[6]を用いた．ブラウザには JRE1.3.1[7] を組み込んだ Netscape Communicator4.78 を用いて動作の検証を行った.

5.2. 評価

本方式では B-W 間でデータ通信を行うまでに，数多くの処理を必要とする．そこで,

SSL 相互認証を行う Web サーバにアクセスするときに，本方式を使用した場合と，通常の方式でユーザが受ける速度差がどの程度あるのかを測定した.

評価環境として，Web サーバには PC/AT 互換機(PentiumII 333MHz，メモリ 64MB)，Proxy には Sun Ultra Enterprise2(UltraSPARC 200MHz，メモリ 256MB)，ユーザ端末には PC/AT 互換機(PentiumIII 866MHz，メモリ 256MB)を用いた.

Proxy 鍵ペアとユーザ鍵ペアは, 1024bit の RSA 鍵とした．クライアント認証時の最初の Web 画面として，合計サイズ 3,946byte の HTML ファイルと画像ファイルを用意した．それぞれ 10 回計測を行い平均を求めたものを表 1 に示す.

提案方式(キャッシュなし)とは，メモリ上に JavaVM がキャッシュされていない状態である．提案方式（キャッシュあり）とは，一度アプレットを起動しメモリ上に JavaVM をキャッシュさせた状態を示す. 最初のページが表示されるまでとは，アプレットに変更を加えて，メールアドレスとパスワード入力を省略して測定したものである．リンク先が表示されるまでとは，最初のペ

表 1 従来方式と提案方式の処理時間

	最初のページが表示されるまで	アプレットが表示されるまで	リンク先が表示されるまで
通常方式	1.37[s]	-	0.147[s]
提案方式（キャッシュなし）	10.70[s]	8.50[s]	0.189[s]
提案方式（キャッシュあり）	4.00[s]	2.48[s]	-

ージ内にあるリンクをクリックしてから, 次のページが表示されるまでの時間を測定したものである.

キャッシュの有無で比較した場合, JavaVM の起動時間に多くの時間がかかっていることが分かる. キャッシュありでも通常方式と比べ時間がかかっているのは, 通信回数が多いことと, アプレットの表示に時間がかかっているためと思われる.

リンク先が表示されるまでで分かるように, この方式では最初のページ表示には時間がかかるが, 2 回目以降のデータ通信では通常方式とほぼ変わらないため, ユーザ絵の影響は少ないと考えられる.

6. 考察

本方式では, イントラネット内のユーザ秘密鍵と ProxyCA 証明書をユーザ自身が安全に管理しておく必要がある. また, ProxyCA 証明書は有効期限が切れたら更新する必要がある. Proxy の管理者は不正なことを行わないと仮定している.

以上の前提が必要となるものの, 提案方式を用いることにより, 次のような利点が生まれる.

- ユーザは自身の証明書をブラウザに組み込む必要が無い. またユーザ証明書の有効期限のみを延長する場合にも, 管理者がディレクトリサーバに更新した証明書を追加するだけで済み, ユーザがすべき作業はない.
- ユーザ自身が CRL を取得してサーバ証明書を検証する手間が無くなる. また Proxy で証明書受け入れポリシーを管理することが可能となるので, 組織のポリシーに合致しないサーバ証明書を受け入れてしまうようなユーザの不注意を避けることができる. さらに Proxy のみが CRL を取得すればよいので, 全体の通信負荷が軽減される.
- Proxy で通信データの復号化および暗号化をする必要が無いため, Proxy の負荷がそれほど大きくはならない.

一方, 以下のような問題も生じてしまう.

- ユーザ端末では, 署名付きアプレットが秘密鍵にアクセス可能な設定となっている. このため悪意のある署名付きアプレットが秘密鍵にアクセスできてしまう. また Proxy では B-W 間で使用する共有鍵を生成できるため, 意図すれば暗号通信を解読することができる.
- B-W 間でデータ通信を行う前に独自の通信処理を行うため時間がかかる.

7. 本方式の応用例

本方式は次のような環境での利用が有用と考えられる.

- 組織のポリシに従ってアクセス制御を行いたい場合
- 証明書の検証パスに CA 同士の相互認証があるような場合

Proxy で証明書の検証を行うため, もし組織のポリシにより, 信頼できない CA から発行されたサーバ証明書を受け取った場合, Proxy でフィルタリングが可能となる. これにより, ユーザの不注意で不正な Web サイトを信頼してしまう可能性が無くなる.

現状のブラウザでは証明書検証パスに CA 同士の相互認証が含まれている場合には, 署名の検証が行えない. 本方式では Proxy で検証することにより柔軟な対応が可能となる.

8. おわりに

証明書の管理や検証の煩雑さが原因で, HTTPS を活用しようとすると様々な問題が発生する. 本稿ではこれらの対策として, Proxy を利用した証明書管理のアプローチを示し, 実装を行った. そして従来の方法と比較し本方式の利点と欠点について考察した. 今後はこの欠点の対策を検討していきたい.

参考文献

[1] Transport Layer Security Working Group, “The SSL Protocol Version 3.0”,http://home.netscape.com/eng/ssl3/draft302.txt

[2] Dierks, T. and Allen, “The TLS Protocol Ver.1.0”, http://www.ietf.org/rfc/rfc2246.txt

[3] 高須, 若山, 村瀬, 鈴木, 岩田：「証明書と秘密鍵の集中管理のための SSL Proxy サーバ開発」,第 60 回情報処理学会全国大会, 2000

[4] 若山, 中山, 村瀬, 鈴木, 岩田:「S/MIME でのユーザによる証明書管理軽減方式の実装」, 第 101 回マルチメディア通信と分散処理研究会, 第 12 回コンピュータセキュリティ研究会,2001

[5] 高須, 若山, 村瀬, 鈴木, 岩田：「HTTPS における証明書管理削減方法」,マルチメディア, 分散, 協調とモバイル(DICOMO 2001)シンポジウム論文集, 2001

[6] Sun Microsystems, Inc:http://www.iplanet.com/downloads/developer/2091.html

[7] Sun microsystems, Inc:http://java.sun.com/