HTTPSでの証明書管理機構の実装と評価

若山公威[†] 高須紀樹[†,☆] 村瀬晋二[††]
鈴木春洋[††] 岩田彰[†]

SSL/TLS上でHTTP通信を行うHTTPSを用いることにより，通信相手認証，メッセージの改竄防止，メッセージの暗号化を行うことが可能となる．しかし，実際にHTTPSを運用する際には，証明書管理と検証の複雑さが原因でセキュリティ上の問題が発生する場合がある．本論文では，まず，この問題について言及する．そして，イントラネットやエクストラネット内のユーザが外部のHTTPSプロトコルを利用したWebサイトにアクセスする場合にターゲットを絞り，対策方法の提案，実装と評価を行った．提案方式を用いることにより，ユーザが自身証明書の管理をする必要がなくなり，サーバ証明書の検証ミスを防ぐことができる．

Implementation and Evaluation of Certificate Management Mechanism for HTTPS

Kimitake Wakayama,[†] Toshiki Takasu,[†,☆] Shinji Murase,[††]
Shunyo Suzuki[††] and Akira Iwata[†]

By using HTTPS, which performs HTTP communication on SSL/TLS, it becomes possible to authenticate the peer, prevent message alteration, and encrypt messages. However, in case HTTPS is actually employed, the problem on security may occur owing to the complexity of certificate management and verification. In this paper, we refer to this problem first. After that, we make a proposal and implement and evaluate it in case users in intranet or extranet access websites. By using a proposal method, it becomes unnecessary for users to manage their own certificates, and they can prevent the verification mistake of server certificates.

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さかんになってきており，企業間での電子取引も始まりつつある．このような流れは今後さらに加速し，社会経済で重要な位置を占めていくと考えられる．そこではいかに安全性を確保するかが重要な課題となり，通信相手認証と暗号通信などのセキュリティ技術の利用は今や欠かすことができない．これらの技術のための公開鍵配布基盤として利用できるPKI（Public Key Infrastructure）については，2001年4月に「電子署名および認証業務に関する法律」（電子署名法）が施行されるなど，法律面での進展が進んでおり関心が高まっている．

HTTPでの暗号通信方式としては，SSL（Secure Sockets Layer）[1]/TLS（Transport Layer Security）[2]を用いたHTTPS[3]が業界標準の地位を獲得しており，多くのWebサイトで利用されている．SSLはNetscape Communications社により提唱された，通信の暗号化，相手認証，データの完全性を実現する技術である．これはOSI参照モデルでいうところのセッション層で提供されるため，HTTP，FTPなど多くの通信アプリケーションデータを，それ自体に変更を加えることなく暗号化することができる．SSLはIETF（The Internet Engineering Task Force）での標準化作業が行われ，一部内容を変えてTLSとして公開された．本論文の方式はTLSにも適用できるが，以下では簡略化のためSSLのみを明記する．

ショッピングサイトでは，サーバ認証上でクレジットカード番号の入力を行っている．このサーバ認証でさえ，実際にはすべての場合において安全に運用されているとはいい難い．また，SSLではクライアント認

† 名古屋工業大学
Nagoya Institute of Technology
†† 株式会社シーティーアイ
CTI Co., Ltd.
☆ 現在，新日鉄ソリューションズ株式会社
Presently with NS Solutions Corporation

証機能も用意されており，サーバ側で通信相手（クライアント）を X.509 厳密認証することができる．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のサイトなどで，厳密なアクセス管理ができると考えられる．しかし，現状では，SSL が使われる Web サイトの多くは，サーバ認証により暗号通信路を確立し，その上でパスワードによってユーザ認証を行っている．

このように SSL のすべての機能が使われているわけではなく，かつ，セキュリティ上の問題が発生する利用がされている理由として，運用方法が複雑であり，ユーザへの負担が大きいことがあげられる．我々はこれまで，PKI 技術をさらに普及させるため，PKI 技術を用いる場合にユーザの負担を軽減するための運用方法の検討とシステム開発を行ってきた[4)~6)]．

本論文では，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，改善方法を提案する．そして，イントラネットやエクストラネット内のユーザが外部の HTTPS プロトコルを利用した Web サイトにアクセスする場合にターゲットを絞り，対策方法の提案と実装を行った．以下では，まず 2 章で SSL の概要を述べ，3 章で SSL 運用上でのユーザによる証明書の扱いと，ユーザの不注意によって引き起こされるセキュリティの問題について述べる．そして，4 章で 3 章の問題に対する対策方式を提案し，5 章でシステムの実装と評価を行う．さらに，6 章で本システムについて考察を行い，7 章で他の方式との比較を行う．

2. SSL 概 要

2.1 セッションとコネクション

SSL プロトコルは，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．セッションでは相手の証明書，暗号アルゴリズムの種類，マスターシークレットなどの情報を保持し，コネクションでは実際の暗号通信用鍵データなどの情報を保持する．1 つのセッション中に複数のコネクションを持たせることが可能となり，一度の暗号方式の取り決めで複数回のデータの交換をすることができる．

2.2 ハンドシェーク

SSL では以上のことを実現するため，ハンドシェーク・プロトコルにおいて新規セッション確立の場合と，既存セッションの再開の場合を用意している．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われている．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のやりとりでアプリケーションデータのやりとりを開始することができる．

以下に，暗号方式として RSA を利用した典型的な SSL ハンドシェークシーケンス手順を示す．

(1) クライアントからサーバへ Client Hello メッセージを送る．このメッセージには，クライアントがサポートする暗号アルゴリズムリスト，クライアントが生成した乱数などが含まれる．
(2) サーバからクライアントへ，Server Hello, Server Certificate メッセージを送る．Server Hello メッセージには，利用する暗号アルゴリズム，サーバが生成した乱数，セッション ID などが含まれる．Server Certificate メッセージにはサーバ証明書が含まれている．
(3) クライアント認証を行う場合は，サーバから Certificate Request メッセージを送る．このメッセージには，サーバが受け付けるクライアント証明書発行 CA（Certification Authority）の DN（Distinguished Name）リストなどが含まれる．
(4) サーバからクライアントへ Server Hello Done メッセージを送り，サーバからのメッセージ送信が終了したことを告げる．
(5) クライアントでサーバ証明書の検証を行う．
(6) クライアント認証の場合は，クライアントからサーバへ Client Certificate メッセージを送る．このメッセージには，クライアント証明書が含まれている．
(7) クライアントでプレマスターシークレットを作成し，サーバ公開鍵で暗号化して，サーバへ Client Key Exchange メッセージに含めて送る．
(8) クライアントとサーバとも，クライアント乱数，サーバ乱数とプレマスターシークレットから，マスターシークレットを生成する．
(9) クライアントからサーバへ，Certificate Verify メッセージを送る．このメッセージには，クライアントの秘密鍵による署名データが含まれる．
(10) クライアントからサーバへ，Change Cipher Spec，Finished メッセージを送り，ハンドシェークプロトコルが正常に終了したことを伝える．Finished メッセージ以降は，合意したアルゴリズムによって暗号化されている．用いる暗号鍵，MAC（Message Authentication Code）シークレットは，サーバ乱数，クライアント乱数と

マスターシークレットから作成される．

(11) 同様に，サーバからクライアントへ，Change Cipher Spec，Finished メッセージを送る．

(12) 暗号通信路上で，アプリケーションデータのやりとりを行う．

既存セッションの再開時には，クライアントが再開したいセッションのセッション ID を Client Hello メッセージに入れて，サーバへ送る．サーバでこのセッション ID がキャッシュされていれば，このセッションを再開する．

3. SSL 運用上の問題点

SSL を用いることにより経路上の暗号化や認証を行うことができるが，実際に運用する際にいくつかの問題が生じる．本章でこの問題点を説明する．ここでは，特に HTTPS を取り上げて説明するが，SSL 全般についていえることである．

3.1 サーバ認証時のユーザへの負担

サーバ認証を利用するには，ブラウザへ CA 証明書をインポートしておく必要がある．複数の CA を信用する場合は，そのすべての証明書をインポートする必要がある．特に，イントラネット内などで独自の CA を立ち上げて利用する場合には必須の作業となる．また，現在広く利用されている Internet Explorer や Netscape のブラウザには，初期状態で数十もの CA 証明書があらかじめ組み込まれており，標準ではすべてが信頼できる CA として扱われてしまうため，組織のポリシーにより信頼できない CA の証明書は削除しておく必要がある．信頼できる CA 証明書については，証明書が改竄されていないか確認するため，フィンガープリントを調べる必要がある．さらに，複数のブラウザを用いていて CA 証明書を共有できない場合は，ユーザの負担が増える[7]．

たとえ証明書が信頼できる機関から発行されており，有効期間内としても，その証明書は失効されている可能性がある．あるいは，誤発行の場合もある．広く知られた例としては，2001 年 1 月の米国 VeriSign 社による米国 Microsoft 社名証明書の誤発行があげられる．もしそのような証明書を持つサイトにアクセスすると，ブラウザはそのサイトを正当であると見なしてしまい，不利益を被る可能性がある．これを防ぐため，CRL（Certificate Revocation List）などの証明書失効情報を用いてサーバ証明書が失効されていないかどうかの検証が必要である．しかも，CRL は一度取得すればそれで終わりではなく，定期的に入手する必要がある．証明書や CRL の入手と検証がすべて自動で行えればよいが，現状のブラウザでは自動化されていないものが多く，ユーザへの負担となっている．この面倒さが原因で，相手証明書の失効状態検証を行わなかったり，受け取った署名の検証を行わなかったりした場合は，通信相手の成りすましにより不利益を被る可能性がある．

このように，サーバ証明書検証作業はユーザへの負担が重い．また，検証作業がユーザにのみ任されているため，ユーザが検証を怠る，あるいは不注意により，セキュリティ上の問題が発生することもある．

3.2 失効情報の通信量増加

証明書の失効状態を検証するために CRL を入手する場合は，通常，証明書の CRL Distribution Points フィールドに書かれている場所から取得することになる．証明書を大量に発行する CA の場合，失効情報は大きくなる傾向となる．ダイアルアップ接続のような回線容量が小さい接続を行っている場合，CRL 入手のための時間が長くなるため，その間は作業に支障が出ることになる．この結果，ユーザが CRL を入手しての検証を行うことを嫌うようになる．

3.3 クライアント 認証時のユーザへの負担

クライアント認証時には，ユーザの秘密鍵が必要となる．さらに，ユーザの証明書をサーバへ送信するため，あらかじめ端末でユーザ自身の証明書を保持している必要がある．この証明書の有効期限が切れた場合や，組織編成の変更や異動により証明書内の記載項目に変更が生じた場合には，CA から再発行された証明書を入手することになる．ユーザ証明書も，CA 証明書と同じく，複数のブラウザで証明書を共有できない場合が多い．IETF の PKIX WG により標準化作業が行われている CMP（Certificate Management Protocol）[8]のような自動化するためのプロトコルがあるが，現在広く使われているブラウザではこれらの証明書管理プロトコルが実装されておらず更新作業が自動化されていないため，ユーザが作業を行う必要がある．ユーザは概してこのような作業に煩わしさを感じている．

4. HTTPS での解決法

4.1 システム構成

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 通信を行える方式を提案する．なお本方式では，イントラネット内ユーザが外部のサーバにアクセスする場合を想定しており，ユーザが Proxy を信頼できることを前

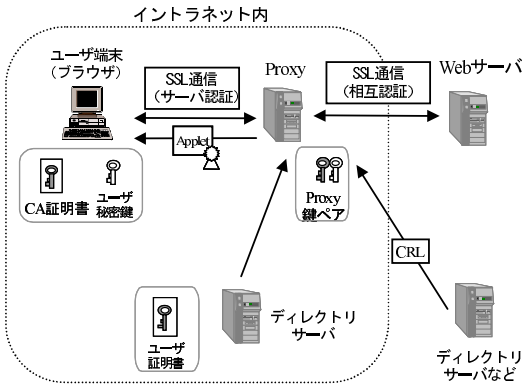

図 1 システムの全体構成
Fig. 1 Structure of system.

提としている．

システムの全体像は**図 1** のようになる．前提としてイントラネット内メンバのユーザ証明書が保管されているディレクトリサーバが設置されているものとする．ユーザ端末には，パスフレーズを用いて暗号化したユーザの秘密鍵を保管しておき，パスフレーズはユーザのみが記憶しておく．このユーザ秘密鍵は，ブラウザが本来備えているクライアント認証機能に利用するものではなく，署名付きアプレットが Web サーバとのクライアント認証に利用するものである．ユーザは自身の秘密鍵のほかに，CA 証明書のみを保持している．この CA 証明書は有効期限が切れたら更新する必要がある．

LDAP（Lightweight Directory Access Protocol）[9]対応ディレクトリサーバは証明書のリポジトリとして注目されており，すでに主にイントラネットなどで利用されている．ただし現状では，単に証明書を組織内で公開するための保管場所として使われているのみである．証明書をディレクトリサーバから取得した後は，各ユーザのブラウザで証明書を保管しており，証明書更新時にユーザの手を煩わすことになる．本研究では，ディレクトリサーバでのみユーザ本人の証明書を管理することを試みる．この結果，ブラウザへのユーザ証明書インポートなどの作業からユーザが開放される．

また，Proxy において，システム管理者があらかじめ信頼できると設定した CA から発行されたサーバ証明書であるかの確認を行う．必要であればサーバ証明書発行元 CA などから CRL を取得してきて証明書の失効状態を検証する．これにより，ユーザによるサーバ証明書検証の手間と検証ミスによるトラブルを避けることができる．

さらに，ユーザの端末ではなく，Proxy にて CRL を取得して保管することにより，各ユーザ端末が個別に CRL を取得するよりもトラフィック削減につながる．

4.2 HTTPS における Proxy での処理の問題点と解決方法

前述のことを実現する場合，ブラウザに本来備わっている HTTPS クライアント認証の機能を用いずに，独自の方式を実装する必要がある．本研究では，ブラウザの改良が不要なことと，ユーザの手間をかけずにバージョンアップ作業が行えることを重視して，Java アプレットを用いて認証機能を実現させる．ブラウザから HTTPS クライアント認証通信を開始して最初の Web ページを要求するときに，Proxy からブラウザへアプレットタグが含まれた HTML ファイルを HTTPS 上で送信する．このためには，Proxy で通信内容を解読できる必要がある．

HTTP では，Proxy において通信内容を解析して通過を制限することができるものの，HTTPS の場合は通信内容が暗号化されているため，Proxy では解読することができない．このため，解読するには，ブラウザ–Proxy，Proxy–Web サーバ間でそれぞれ別々のセッションが確立することになる．そうなれば当然両者で共有する鍵は別のものとなり，Proxy では一方から送られてきたデータを復号化した後に別の鍵で暗号化し，他方に送信するという処理がそれぞれのコネクションで必要となる．これは Proxy での負荷が増大することにつながる．かつ，Web サーバはユーザ端末ではなく Proxy をクライアント認証することになってしまう．

そこで提案方式では，SSL ハンドシェークを展開することにより，クライアント認証に必要なユーザ証明書をユーザ自身が管理しなくてもよく，Proxy を介してもなおブラウザ–Web サーバ間で鍵の共有を行い，ユーザ端末の情報を用いて Proxy–Web サーバ間にクライアント認証を実現させる．このことを実現するためには，共有鍵の共有と，署名データの生成方法について考慮しなければならない．

4.2.1 共 有 鍵

共有鍵を生成するには，マスターシークレットが必要となる．このマスターシークレットは，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ーシークレットを基に生成される．これらの値をブラウザと Web サーバのみでなく，Proxy でも共有することを考える．

Server Hello と Client Hello には平文で乱数が含ま

れているため，Proxy で取得可能である．

しかし，プレマスターシークレットは Web サーバの公開鍵で暗号化されているので，Proxy では復号できない．そこで，Server Certificate メッセージにおいて，Proxy からブラウザへは Proxy の証明書を送ることにする．Proxy の証明書をブラウザへ送信するため，サーバ認証 SSL 確立時に，ブラウザで意図しているサーバでない旨の警告が表示される場合がある．この対策として，Proxy をイントラネット内 CA の下位 CA として稼働するようにし，Proxy にて Subject を接続先 Web サーバ DN として動的に証明書を作成することとする．

4.2.2 署名データ

クライアント認証では，ユーザ証明書と署名データが必要となる．この署名データはユーザ秘密鍵がないと作成できない．そこでブラウザにアプレットを送り，そのアプレットが署名データのみを Proxy に返すことによってこの問題の解決を図る．アプレットには署名を付けたうえで Java2 のセキュリティ機構を利用して安全策を講じておく必要がある．

4.3 ハンドシェーク

図 2 に，提案方式で使用するハンドシェークを表す．ブラウザ–Web サーバ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ブラウザ–Proxy 間に確立させる．このセッションが確立されれば，ブラウザ–Web サーバ間で共通の鍵を持つことができるため，セッション確立後は Proxy で暗号データをいったん復号し暗号化する必要がなくなる．

以下では，クライアント認証時のブラウザ–Web サーバ間でデータの送受が可能となるまでの概要を示す．

(1) まずクライアントは Proxy に乱数 Rc_1 などを送る．Proxy では Rc_1 を記録し，そのまま Web サーバへ送る．

(2) Web サーバから Proxy へ乱数 Rs_1，Web サーバ証明書などを送る．Proxy でディレクトリサーバなどから CRL を取得して自らが保管している CA 証明書と合わせて Web サーバ証明書の検証を行う．そして，Server Hello メッセージ内の乱数 Rs_1 の値を記録し，そのままの状態でクライアントへ送る．Server Certificate メッセージからはサーバ証明書 Cs を抜き出し，Proxy の証明書 Cp を組み入れてクライアントへ送信する．

(3) クライアントでは，Proxy から送られてきた証明書の検証をする．証明書が問題なければ，プレマスターシークレット (Pre) を生成し Proxy 公開鍵で暗号化して Proxy へ送信する．クライアントと Proxy では，Rc_1，Rs_1，Pre からマスターシークレットを生成し，Rc_1，Rs_1，マスターシークレットから最初のコネクション共有鍵を生成する．Proxy から Finished メッセージを送り，Proxy とクライアント間でサーバ認証が確立する．

(4) クライアントから閲覧したい Web ページの URL を送る．Proxy では，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，引き続きクライアントは 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フレーズを入力する．メールアドレスは Proxy へ送り，Proxy にてディレクトリサーバから証明書を取得し，Web サーバへ送信する．パスフレーズは，アプレットがユーザ端末内の暗号化された秘密鍵にアクセスするときに復号するために利用する．

(6) Proxy から，署名に必要なハッシュ値をクライアントへ送信する．クライアントでは，このデータにユーザ秘密鍵を用いて署名し Proxy へ送る．Proxy では，この値を Certificate Verify メッセージに含めて Web サーバへ送信する．

(7) アプレットから java.applet.AppletContext クラスの showDocument() メソッドを実行することにより，ユーザが本来閲覧したい Web ページを表示する．

(8) すべての処理が滞りなく終了すれば，ブラウザ–Web サーバ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の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，本方式は，上記の (7) で showDocument() メソッドを利用しているため，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

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御を設定するため，Web ブラウザには Java2 Plug-in が必要となる．Proxy の署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．アプレットの署名を検証するためにクライアントにはイントラネット内

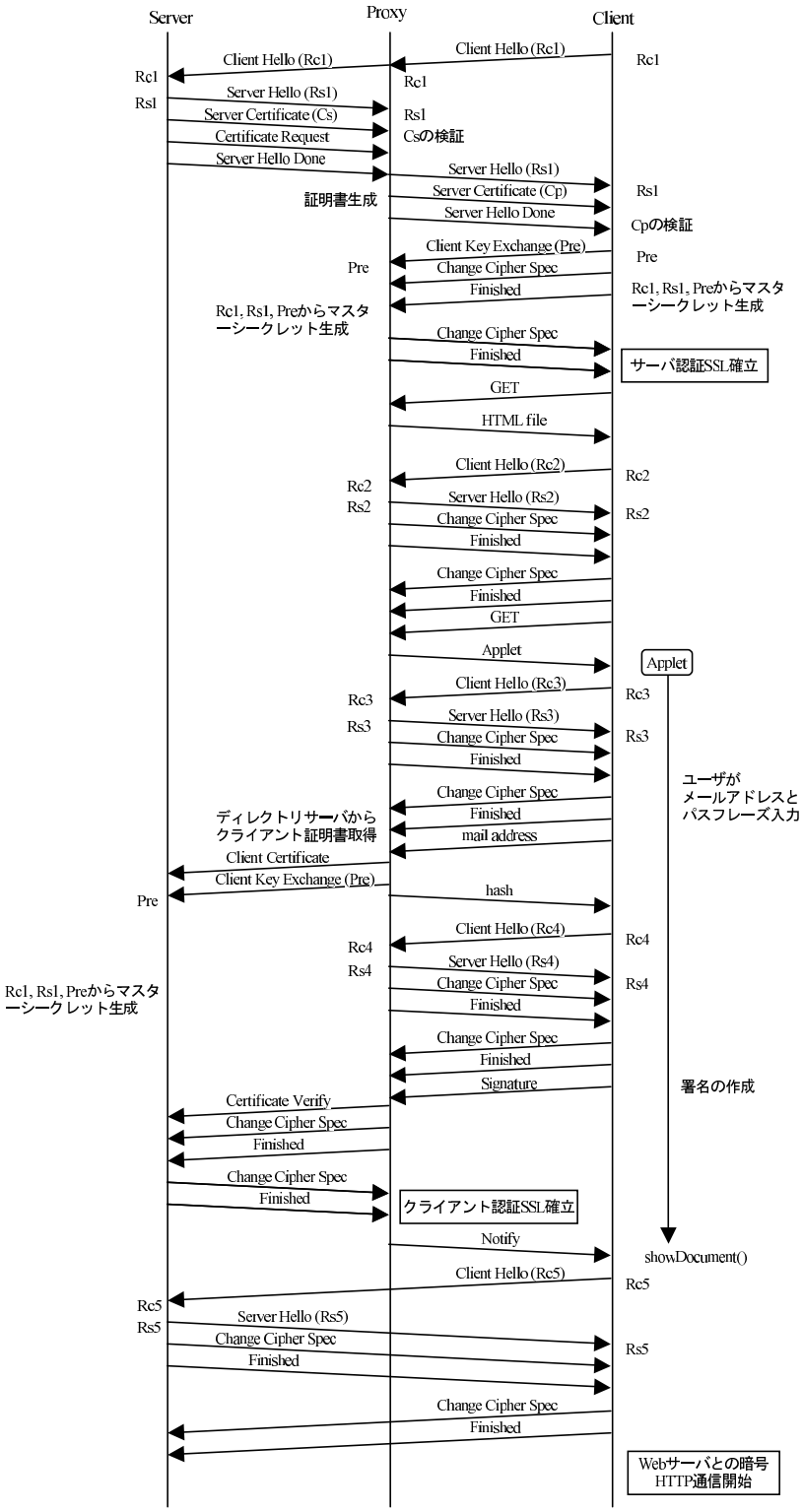

図 2 提案方式のハンドシェーク
Fig. 2 Handshake of proposed method.

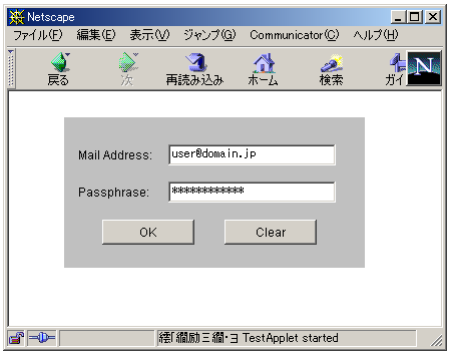

図 3 メールアドレスとパスフレーズ入力画面
Fig. 3 Input window of mail address and passphrase.

CA 証明書が必要となる．Proxy を上述した CA の下位 CA としたことにより，Proxy が作成する証明書の検証も可能となっているため，ユーザ端末には，これ以外の CA 証明書，本人証明書は不要となる．

Proxy の実装には，暗号ライブラリ AiCrypto 1.8.3[10]，Java2 SDK 1.3.1[11]を用いた．ディレクトリサーバとしては，OpenLDAP 2.0.11[12]を用いた．ブラウザとして Netscape Communicator 4.78 を用いて，動作の検証を行った．

クライアント認証の場合，**図 3** のようにアプレット画面にメールアドレスとパスフレーズを入力する．このメールアドレスをもとに，ディレクトリサーバからユーザ証明書を取得する．パスフレーズは，ユーザ端末に保存している秘密鍵にアクセスするためのものである．これ以外は，通常の Web ブラウジングと同様に行う．

5.2 評　　価

本方式を利用した場合，通常の Web ブラウジングと比べて速度がどの程度異なるかを調べた．

評価環境として，Web サーバは PC/AT 互換機（PentiumII 333 MHz，メモリ 64 MB）を，Proxy は Sun Ultra Enterprise2（UltraSPARC 200 MHz，メモリ 256 MB），ディレクトリサーバとして Sun Ultra10（UltraSPARC-IIi 333 MHz，メモリ 256 MB），クライアント端末として PC/AT 互換機（PentiumIII 866 MHz，メモリ 256 MB）を用いた．

Proxy 鍵ペアとユーザ鍵ペアは，RSA 暗号方式の 512 ビット鍵とした．測定用に，アプレットでのパスフレーズの入力を省略した．クライアント認証時の最初の Web 画面として，合計サイズが 3,946 byte の HTML ファイルと画像ファイルを用意し，この画面が表示されるまでの時間を計測した．

まず，ディレクトリサーバに 1 ユーザのエントリのみを作成して実験した．10 回計測を行って平均を求めたものを**表 1** に示す．通常方式は，Proxy を介さない HTTPS 通信である．提案方式（キャッシュなし）は，Java VM とアプレットがメモリ上にまったく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）は，1 度アプレットを起動して Java VM とアプレットをメモリ上にキャッシュしておいた場合を示す．この結果から，アプレットのダウンロードと Java VM の起動に時間がかかっていることが分かる．メモリ上にキャッシュにしている場合でも，通常の方式と比べると大幅に時間がかかっている．これは，通常方式よりもハンドシェークのやりとりが多いことと，アプレットの表示と RSA による署名処理に時間がかかるためと考えられる．

表 1 従来方式と提案方式の処理時間
Table 1 Performance comparison between existing and proposed method.

通常方式	提案方式（キャッシュなし）	提案方式（キャッシュあり）
0.63 sec	10.69 sec	4.00 sec

表 2 ディレクトリサーバ内ユーザエントリ数変更にともなう処理時間
Table 2 Performance comparison with changing number of directory entries.

ユーザエントリ数	100	1,000
処理時間	11.33 sec	19.21 sec

次に，実際のイントラネット環境での利用を考えて，ディレクトリサーバのユーザエントリ数を増加させて実験を行った．Java VM キャッシュなしでの結果を**表 2** に示す．エントリ数を増加させることにより，合計の処理時間が大幅に増加しており，提案システムの実行時間にはディレクトリサーバでの検索時間が大きく影響していることが分かる．今回ディレクトリサーバとして用いた OpenLDAP のカスタマイズや高速なサーバへ変更することにより，時間の短縮は可能と考えられるが，大規模な環境で利用するには Proxy でのキャッシュなどの検討が必要である．ただ，クライアント認証時の証明書のやりとりは 1 セッションのうち最初のページへのアクセス 1 回のみであり，それ以降の暗号通信時にはこの処理が不要なため，ユーザへの影響は比較的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本人秘密鍵と CA 証明書を，ユーザ本人が安全に保持している必要があり，この CA 証明書は有効期限が切れたら更新する必

要がある．また，Proxyの管理者は不正なことを行わないと仮定している．

以上の前提が必要であるが，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へ組み込む必要はない．また，異動などにより証明書記載事項が変更された場合や，公開鍵と秘密鍵のペアを変更しないでユーザの本人証明書の有効期限のみを延長する場合は，管理者がディレクトリサーバのユーザエントリに更新した証明書を追加するだけで済み，ユーザの作業はまったく必要ない．
- ユーザ自身がCRLなどを取ってきて，サーバ証明書を検証する手間がなくなる．あるいは，ユーザがサーバ証明書を検証して，さらにProxyでも検証することが可能である．こうすることにより，ユーザの不注意により組織のポリシーに合致しないサーバ証明書を受け入れることを避けることができる．また，Proxyのみが CRLを取得するため，通信量を削減することが可能となる．
- セッションが確立された後，Proxyでは通信内容を復号し暗号化する必要がないため，Proxyの負荷を軽減することが可能である．

一方，以下のような問題点がある．

- Proxyが不正侵入された場合や管理者が不正を行う場合を考える．ユーザ端末では，Proxyの署名付きアプレットに対して，ユーザの秘密鍵へのアクセスを許可する設定となっている．このため，悪意を持ってアプレットを書き換えて署名を付けることにより，ユーザの秘密鍵を取得できてしまう．また，ProxyではWebサーバとクライアント間の暗号通信時の共有鍵を生成できるため，暗号通信を解読することができる．
- セッション開始時にProxyとクライアント間で独自の処理を行うために時間がかかる．

7. 関連研究

Peter Buhlerらは，DH-EKEをSSLに組み込むことにより証明書ベースのクライアント認証を行わない方式を提案している[13]．この方式では，サーバの厳密認証を行うことができない．また，Webサーバの変更が必要となる．我々の方式では，Webサーバに変更を加える必要はない．

鍛らは，分散オブジェクトシステムでの SOA（Seamless Object Authentication）アプローチを提案している[14]．この方式では，モバイルコードに付加された署名情報をサーバ側でも評価することにより，管理者が信頼していない人物が作成したモバイルコードにはサービスを提供しない．ユーザだけではなく管理者によるチェックを加えるという点では同じだが，我々の方式は，モバイルコードでなくHTTPS上のWebページに関するものである．

証明書と秘密鍵を集中管理する方式としては，MyProxy[15]，Netscape Communicatorのローミングアクセスがある．これらの方式では，ユーザの証明書に加えて秘密鍵も集中管理することになる．ユーザ証明書は集中管理されるが，サーバ証明書の検証は，依然としてユーザが行うことになる．また，CRLの入手やサーバ証明書の検証はユーザが行う必要がある．

マイクロソフト社は，企業内で多数のユーザにInternet Explorerを配布するために管理者によるカスタマイズ機能を有した管理者キット[16]を配布している．初期状態で，イントラネット内CA証明書を組み込むのは可能であるが，ユーザ証明書を組み込めない．この方式も，CRLの入手やサーバ証明書の検証はユーザが行う必要がある．

疋田[17]らの方式は，Proxyがユーザの秘密鍵を保持しているにもかかわらず，クライアントにてプレマスターシークレットをサーバ公開鍵で暗号化して送信することにより，サーバとクライアント間のセッション鍵をProxyで共有できないようにする方式である．この方式では，Proxyで現在通信中のセッションの共有鍵が分からないというのみである．Proxyがユーザ秘密鍵を保持しているため，ユーザに成りすまして新たにHTTPSクライアント通信を行うことは可能である．さらに，この方式を行うには専用のクライアントソフトが必要となり，既存のWebブラウザは利用できない．

8. おわりに

HTTPS通信においてサーバ認証を行うには，ユーザの不注意によりセキュリティ上の問題が発生し，失効情報のためのトラフィックが増加する．クライアント認証を行うには，ユーザによる証明書管理作業が必要となる．本論文では，これらの対策として，Proxyを利用した証明書の管理アプローチを示し，実装を行った．そして，通信のオーバヘッドを計測した．

ICカードにユーザ証明書を入れることが一般的になってきても，証明書更新作業が発生するため，本方式は有効と考えられる．今後は，他のプロトコルへの適用を検討していきたい．

参 考 文 献

1) Freier, A.O., Karlton, P. and Kocher, P.C.: The SSL Protocol Version 3.0, IETF Internet Draft draft-freier-ssl-version3-02.txt (1996).
2) Dierks, T. and Allen, C.: The TLS Protocol Ver.1.0, IETF RFC 2246 (1999).
3) Rescorla, E.: HTTP Over TLS, IETF RFC 2818 (2000).
4) 若山公威，中山幹浩，村瀬晋二，鈴木春洋，岩田 彰：ユーザによる証明書管理を軽減するための S/MIME アプレット実装，情報処理学会研究報告 2001-DPS-101，2001-CSEC-12, pp.37–42 (2001).
5) 高須紀樹，若山公威，岩田 彰，村瀬晋二，鈴木春洋：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論文集（第 3 分冊），pp.483–484 (2000).
6) 高須紀樹，若山公威，村瀬晋二，鈴木春洋，岩田 彰：HTTPS におけるユーザ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，pp.585–590 (2001).
7) Gutmann, P.: A Reliable, Scalable General-purpose Certificate Store, *Proc. 16th Annual Computer Security Applications Conference* (2000).
8) Adams, C. and Farrell, S.: Internet X.509 Public Key Infrastructure Certificate Management Protocols, IETF RFC 2510 (1999).
9) Yeong, W., Howes, T. and Kille, S.: Lightweight Directory Access Protocol, IETF RFC 1777 (1995).
10) 若山公威，奥野琢人，岩田 彰，村瀬晋二，鈴木春洋：暗号ライブラリと認証局パッケージの開発，第 59 回情報処理学会全国大会論文集（第 3 分冊），pp.395–396 (1999).
11) Sun Microsystems, Inc.: http://java.sun.com/
12) The OpenLDAP Foundation: http://www.openldap.org/
13) Buhler, P., Eirich, T., Steiner, M. and Waidner, M.: Secure Password-Based Cipher Suite for TLS, *Proc.Network and Distributed Systems Security Symposium* (*NDSS 2000*) (2000).
14) 鍛 忠司，洲崎誠一，梅澤克之，近藤勝彦，手塚 悟，佐々木良一，鬼頭 昭：分散オブジェクトシステムにおけるモバイルコード用セキュリティ機構の提案，情報処理学会論文誌，Vol.42, No.3 (2001).
15) Novotny, J. and Tuecke, S.: An Online Credential Repository for the Grid: MyProxy, *Proc. 10th International Symposium on High Performance Distributed Computing* (*HPDC-10*) (2001).
16) マイクロソフト株式会社：http://www.microsoft.com/japan/ieak/
17) 疋田智治，安田 仁，山本守孝，桑田晶彦，今泉法子，嶋崎 剛：認証代行方法，認証代行サービスシステム，認証代行サーバ装置及びクライアント装置，公開特許公報，特願平 11-317468 (2001).

(平成 13 年 11 月 28 日受付)
(平成 14 年 6 月 4 日採録)

若山 公威（正会員）

平成 5 年名古屋工業大学電気情報工学科卒業．平成 7 年同大学大学院博士前期課程修了．同年沖電気工業株式会社入社．平成 10 年より名古屋工業大学電気情報工学科助手．情報セキュリティに関する研究に従事．電子情報通信学会会員．

高須 紀樹

平成 12 年名古屋工業大学電気情報工学科卒業．平成 14 年同大学大学院博士前期課程修了．同年新日鉄ソリューションズ株式会社入社．在学中，情報セキュリティに関する研究に従事．

村瀬 晋二

平成 5 年名古屋工業大学電気情報工学科卒業．平成 7 年同大学大学院博士前期課程修了．同年株式会社シーティーアイ入社．先端 IT 開発部先端技術グループ所属．現在，PKI 応用技術，ネットワークセキュリティの研究開発に従事．

鈴木 春洋

平成 9 年株式会社シーティーアイ入社．先端 IT 開発部部長．同年より CTI インターネット関連事業を担当．平成 10 年中部電力グループネット責任者として電子認証局運営を開始．現在，中部電力グループ電子認証事業，ブロードバンド関連事業，地域情報配信事業等に従事．

岩田　彰（正会員）

昭和 48 年名古屋大学工学部電気学科卒業．昭和 50 年同大学大学院修士課程修了．同年名古屋工業大学工学部助手．昭和 57 年 4 月より昭和 58 年 10 月までドイツ連邦共和国ギーセン大学医用情報研究所客員研究員．昭和 59 年名古屋工業大学工学部情報工学科助教授．平成 5 年名古屋工業大学工学部電気情報工学科教授．現在に至る．ニューラルネットワーク，生体情報処理，医療情報システム，情報セキュリティ，インターネットコンテンツ開発技術に関する研究．昭和 55 年日本 ME 学会研究奨励賞受賞．平成 4 年郵政省郵政研究所主催文字認識コンテスト奨励賞受賞．平成 5 年電子情報通信学会論文賞受賞．平成 10 年情報処理学会「Best Author 賞」受賞．工学博士．電子情報通信学会，日本 ME 学会，日本心電図学会，日本神経回路学会，日本医療情報学会各会員．IEEE Senior Member．