第XI部

公開鍵証明書を用いた利用者認証技術

第 11 部
公開鍵証明書を用いた利用者認証技術

第 1 章　はじめに

moCA 分科会は、認証技術の一つである CA(Certification Authority) 技術の利用を進める上で CA 運用のノウハウ収集が重要であると考え、1997 年度より moCA という CA を立ち上げ、様々な運用実験を行ってきた。CA 技術は、公開鍵暗号の公開鍵とユーザ識別子の対応を第三者が証明し、その第三者への信頼に基づいてユーザの正当性を保証する技術である。moCA では、S/MIME[45, 120] や SSL(TLS)[41] をアプリケーションとして使える CA という位置づけで、WIDE メンバの公開鍵と WIDE メンバのメールアドレスの対応を証明し、証明書として発行し、WIDE 内での利用につなげる実験を試みてきた。1999 年度、2000 年度は分科会という形式をとらなかったが、運用維持のための活動を続け、個人証明書更新実験を毎年 6 月に行ってきた。

CA 立ち上げ当初は、個人証明書発行時のユーザ認証レベルを CA の理想に合わせようと現実に合わない高いレベルで試したり、個人証明書や CA 証明書の有効期限が切れるまでに証明書の更新を間に合わせるだけで精一杯という試行錯誤の連続であった。しかし、CA 証明書の有効期限を 2006 年まで延ばしてからは、徐々に現実的な運用レベルや、ユーザの利便性の追求にも目が向けられるようになってきた。今年度は、証明書更新時に CA 管理者の手作業を介さずよりスピーディに更新ができるようになった。

今年度は、さらに、JNSA(日本ネットワークセキュリティ協会) が主催する「複数 CA 相互運用性実証実験プロジェクト」にて、他の CA ベンダとともに実験に参加する機会に恵まれた。WIDE CA 系列の CA は CA パッケージ ICAP を利用して階層構造をとってきたが、この ICAP と他の CA ベンダの製品とを比較して相互運用性について実験した。

また、実験という位置づけであった moCA を実運用にするための移行計画の検討が始まった。メンバ管理の手続きといずれ連携させた形で moCA を運用することは、moCA の名称からも推察されるとおり当初からの目標であり、moCA にとって第二フェーズに入ったといえる。

本報告書では、JNSA 実験の概要と実験参加から得られた課題、および moCA の実運用に向けた移行計画の検討状況について述べる。

第 2 章　複数 CA 相互運用性実証実験プロジェクト

2001 年 11 月から、NPO 日本ネットワークセキュリティ協会 (JNSA) による「複数 CA 相互運用性実証実験プロジェクト (通称 Challenge PKI) 」が実施された。このプロジェクトでは、CA(Certificate Authority) の機能を実現するソフトウェア (CA ソフトウェアと呼ぶ) の相互運用性を、実際に証明書の発行/検証を通じて調査するという実験が行われた。WIDE Project のメンバに対して証明書を発行している moCA は、moCA ワーキンググループのメンバによって開発が行われた ICAP(ICAT CA Package) を使って運用されている。この ICAP が複数の CA 製品に対してどの程度の相互運用性を確保することができるのか、また他の CA 製品がどのような機能を持っているのかを知るという動機から、現在 moCA の運用及び開発に携わっているメンバがこの実験に参加した。本章では実験参加を通じてわかってきた相互運用の際の問題と ICAP の持つ課題について述べる。

2.1 複数 CA 相互運用性実験 - Challenge PKI 2001

複数 CA 相互運用性実験 (Challenge PKI 2001) は、日本ネットワークセキュリティ協会 (JNSA) が IPA の「情報セキュリティ関連の調査・開発に関する公募」の「PKI 関連相互運用性に関する調査」に応募したものである。Challenge PKI 2001 は CA ソ

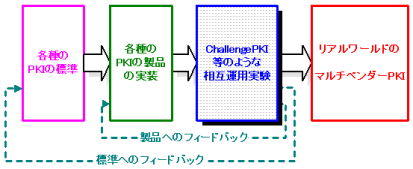

図 2.1.　Challenge PKI 2001 の目標

フトウェアの相互運用性を、実際に証明書の発行と検証を行って調査し相互運用の際に起こる問題を明らかにすることによって、今後の CA ソフトウェアの相互運用に役立てることを目標としている。このフィードバックは、JNSA によって図 2.1 を使って示されている [76]。

PKI(Public Key Infrastracture) は ITU の X.509[1] や IETF の PKIX による RFC2459[65] によって標準化されている公開鍵暗号を使った認証の技術である。PKI 技術を実装したソフトウェアは商用を含めて数多く開発されており、特にクライアント認証やサーバ認証に利用する Web ブラウザやメールクライアントは複数のベンダーの製品が運用されている。標準化文書は、証明書の書式 (証明書プロファイル) や証明書の処理の方法を記述することで、証明書を認証に用いるアプリケーションプログラムだけでなく、証明書を発行する CA ソフトウェアの相互運用性を図っている。しかし CA ソフトウェアごとに発行できる証明書のプロファイルが異なるという現実があるため、実際に異なるベンダーの CA ソフトウェアを相互運用する場合には、問題が起こる可能性がある。相互運用の際に起こる問題には、X.509 で決められている仕様の内、CA ベンダーによって実装を行っている部分や処理の方法の違いからくるものがある。これらの問題を相互運用を通じて明確にし、標準化の段階で回避できるようにするための情報提供を行おうという試みが Challenge PKI 2001 である。

2.1.1 実験の期間と形態

実験は 2001 年 11 月の中旬から 2002 年 3 月までの期間に十数回行われた。

実験会場は工学院大学に設けられた JNSA の実験室で、実験期間中は実験機材が常設されている。実験室に設置された実験ネットワークには、JNSA が用意した参照 CA とサーバ認証実験を行ったり LDAP リポジトリを設置するためのサーバ PC が接続されている。各 CA ベンダは各自が持参した機器をこの

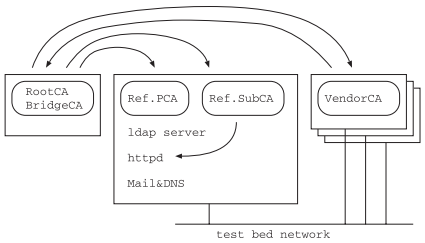

図 2.2.　実験環境のネットワーク

実験ネットワークに接続して実験に参加する。図 2.2 は実験ネットワークと設置された参照 CA を図示したものである。

WIDE Project のチームは認証実用化実験協議会 (moCA ワーキンググループのメンバが多数参加) によって開発された ICAP(ICAT CA Package) を使って実験に参加した。ICAP を動作させる環境の構築のため、ノート PC 一台と VMware を使った仮想のホスト一台を用意した。ICAP は鍵の生成と申請に Web を使ったユーザインターフェースを採用しているため、CA ソフトウェア自身の他に証明書の発行には Netscape ブラウザが必要であるため、これもノート PC にインストールした。

2.1.2 実験への参加の意義

WIDE Project のチームが実験で使用した ICAP は、moCA ワーキンググループのメンバによって現在も開発が進められている。また moCA ワーキンググループで運用している CA、moCA の機能や運用についての議論は moCA メーリングリストを通じて行われている。しかし、現在のところ moCA は WIDE メンバへのサービスを対象とし、WIDE メンバのみに証明書を発行する CA であるため、これまでは他の CA ソフトウェアの機能や相互運用に関する議論があまり行われてこなかった。しかし、WIDE プロジェクトで行われているサービスの中には潜在的に外部の組織と相互運用する可能性を持つものもあり、その際には moCA が提供する認証サービスが他の組織と相互運用をする必要性が生じると考えられる。そのため、Challenge PKI 2001 を通じて ICAP の相互運用性を検証することは、moCA のサービスを拡張することを念頭に置いた ICAP の機能の検討のためには有効であると考えられる。また相互運用を通じて CA 製品などの他の CA ソフトウェアが持っている運用機能を知ることは、ICAP の証明書管理機能の開発のために参考になると考えられる。

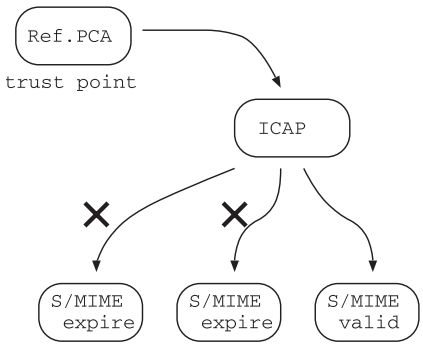

図 2.3. 階層モデル

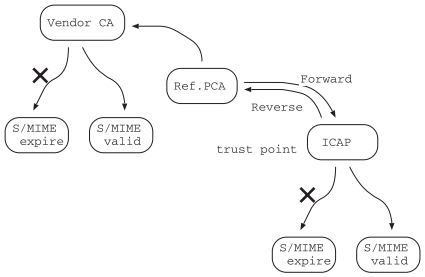

図 2.4. 相互認証モデル

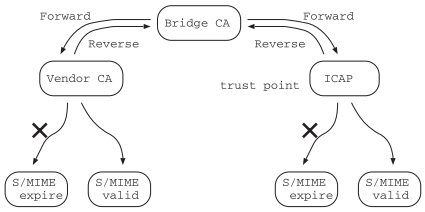

図 2.5. ブリッジモデル

2.2 実験の概要

実験は、各ベンダーが搬入した CA ソフトウェアと JNSA が準備した参照 CA を使って行われた。各ベンダー CA は後述する信頼モデルに従って他の CA に証明書を発行し、証明書ツリーを構成する。更に各ベンダー CA は S/MIME と Web ブラウザ及びサーバのためにユーザ証明書を発行し、他の CA が発行しているユーザ証明書を検証できる環境を作る。

2.2.1 実験の準備作業

ベンダー CA が証明書ツリーを構成するにあたり、以下のような三つの信頼モデルが用いられた。

- **階層モデル**
  各実験参加 CA(ベンダー CA) を JNSA が用意した参照 CA の下位 CA として位置づけ、参照 CA から下位 CA 証明書の発行を受けるもの。この信頼モデルの証明書ツリーを図 2.3 に示す。
- **相互認証モデル (プレブリッジモデル)**
  参照 CA とベンダー CA がそれぞれの自己署名証明書以外に、相互に発行しあった証明書を持つもの。実験は、参照 CA に証明書が発行されているベンダー CA を他の PKI ドメインとして位置づけて、エンドユーザ証明書の検証の検証テストを行う。この際に、全ての PKI ドメインは共通のポリシーを持つと仮定する (ポリシーマッピングを行わない)。この信頼モデルの証明書ツリーを図 2.4 に示す。
- **相互認証モデル (ブリッジモデル)**
  参照 CA が各ベンダ CA と相互に証明書を発行するもの。この際に、個々の PKI ドメインは個別のポリシーを持つと仮定して、他の PKI ドメインとのポリシーマッピングを行う。この信頼モデルの証明書ツリーを図 2.5 に示す。

これらの各々の場合で、ICAP は以下のような証明書ツリーの構成作業を行った。

- **証明書発行リクエストの受付と発行**
  他の CA を下位 CA または相互認証先 CA にするため、証明書発行リクエストを受付け証明書を発行する。
- **他の CA に発行された証明書の組み込み**
  他の CA の下位 CA になるための証明書発行リクエストを作成し、発行された証明書を CA に組み込む。
- **エンドエンティティとサーバのための証明書の発行**
  認証を実際に行うエンドエンティティのための、証明書の発行を行う。それぞれのエンドエンティティに対して、有効な証明書と失効された無効な証明書を発行し、CRL(Certificate Revocation List) を発行する。

上記の作業結果は証明書の種類 (発行元及び先の違い) ごとに記録された。またすべての証明書がどのフィールドをどのような値で持っているかの記録が行われた。

2.3 実験手順

この節では、実験手順について述べる。ユーザ証明書の検証に利用するアプリケーションと環境は以下の通りである。

- **https**
  Web サーバと Web ブラウザのユーザに証明書

を発行する。

- **S/MIME**
  実験用メールアカウントで利用できる S/MIME 用証明書を発行する。

ユーザ証明書は有効であるものと、失効されたものを用意する。証明書を失効した場合、CA は失効した証明書のシリアル番号を含む CRL を発行しておく。CRL は LDAP リポジトリに格納しておくか直接アプリケーションに組み込み、アプリケーションが正常に失効の処理を行うかどうかを観察した。

https を使ったサーバ証明書の検証は以下の手順で行った。

1. **下位 CA 証明書 (WIDE チームの CA 用の証明書の発行)**
   参照 CA に証明書発行要求を送り、WIDE チームの CA 証明書を発行してもらう。予め Web ブラウザに参照 CA の証明書を組み込んでおき、サーバ証明書の検証の際にパス構築を行うことができる状態にしておく。
2. **WIDE チーム Web サーバの証明書発行 (正常系)** WIDE チームの CA を使って Web サーバの証明書を発行し、Web サーバに組み込む。
3. **SSL サーバ認証テスト** Web ブラウザを使って Web サーバにアクセスし、サーバ認証が正常に行われることを確認する。
4. **WIDE チーム Web サーバの証明書発行 (失効系)** WIDE チームの CA を使ってサーバ証明書を発行し、すぐに失効する。更に CRL を発行し、Web ブラウザに組み込んでおく。
5. **SSL サーバ認証テスト** Web ブラウザを使って Web サーバにアクセスし、サーバ認証が証明書の失効を理由に失敗することを確認する。

クライアント証明書を失効してクライアント認証を行う実験は、サーバ証明書の失効を行う実験とは逆にサーバに CRL を組み込んで行った。

S/MIME を使って他ベンダーが発行している証明書を検証する実験は、以下の手順で行った。

1. **下位 CA 証明書 (WIDE チームの CA 証明書の発行)**
   参照 CA に証明書発行要求を送り、WIDE チームの CA のための証明書を発行してもらう。予めメールクライアントに参照 CA の証明書を信頼できる認証機関の証明書として組み込んでおき、メール送信者の証明書を発行している WIDE チーム用の CA 証明書の検証の際にパス構築を行うことができる状態にしておく。
2. **クライアント証明書発行 (正常系)**
   WIDE チームの CA を使って S/MIME で利用できる証明書を発行し、メールクライアントに組み込む。本文にクリア署名を行ったメールを、ベンダー全員に送られるメールアドレスに対して送信する。
3. **クライアント証明書発行 (失効系)**
   正常系と同様に S/MIME 用の証明書を発行するが、すぐにその証明書を失効し、CRL を発行する。その CRL を LDAP リポジトリに格納し、他のベンダーから参照できるようにしておく。
4. **他ベンダーから送られたメールの署名検証**
   手順 2 で行った作業は他のベンダーも行っているので、そのメールを受信する。メールクライアントには参照 CA の証明書が信頼できる認証機関の証明書として組み込まれているので、正常系証明書を使って署名されたメールは正常に検証され、失効系証明書は取得した CRL を参照することで失効として扱われるはずである。各ベンダーが発行した CRL は、メールクライアントによって X.509v3 拡張である cRLDistributionPoints(CRLDP) フィールドの値に従って LDAP リポジトリから取得されるか、直接ファイルで受け渡される。

なお、上記二つの実験手順の最初ではベンダー CA は参照 CA の下位 CA であったが、これは 2.2.1 項で述べた信頼モデルのうちの階層モデルで実験を行ったためである。相互認証モデル (プレブリッジ、ブリッジ) の場合には、WIDE チームの CA 証明書だけをアプリケーションに組んで実験を行う。

2.4 実験結果

上記の要領で行った実験の結果とそのときの証明書ツリーを挙げる。

2.4.1 階層モデルでの実験

- Web サーバ証明書の検証
  - **参照環境**
    Internet Explorer6.0(WindowsXP)

– **証明書ツリー**
参照 CA(トラストポイント)- WIDE チーム CA - Web サーバ
– **失効**
＊参照 CA による WIDE チーム CA 証明書の失効
＊WIDE チーム CA による Web サーバ証明書の失効
– **結果**
＊有効な証明書 (期待値通り)
上記証明書ツリーにより参照環境で Web サーバ証明書が検証できた
＊失効した証明書
・WIDE チーム CA 証明書の失効 (期待値通り)
上記証明書ツリーの WIDE チーム CA 証明書の検証に失敗したため参照環境で Web サーバ証明書の検証に失敗した
・サーバ証明書の失効 (期待値通り)
参照環境で上記証明書ツリーの Web サーバ証明書の検証に失敗した

● Web ブラウザ証明書の検証
– **参照環境**
Apache1.3.20+mod_ssl 2.8.4
– **証明書ツリー**
参照 CA(トラストポイント) - WIDE チーム CA - Web ブラウザ
– **失効**
＊参照 CA による WIDE チーム CA 証明書の失効
＊WIDE チーム CA による Web ブラウザ証明書の失効
– **結果**
＊有効な証明書 (期待値通り)
上記証明書ツリーにより参照環境で Web ブラウザ証明書が検証できた
＊失効した証明書
・WIDE チーム CA 証明書の失効 (期待値通り)
上記証明書ツリーの WIDE チーム CA 証明書の検証に失敗したため参照環境で Web ブラウザ証明書の検証に失敗した
・サーバ証明書の失効 (期待値通り) 参照環境で上記証明書ツリーの Web ブラウザ証明書の検証に失敗した

● S/MIME 証明書の検証
– **S/MIME ユーザ証明書の発行**DN に S/MIME ユーザの電子メールアドレスを入れる。
＊**特記事項**設定の対応ができず subjectAltName に Mail アドレスを入れた証明書が発行できなかった。→Distinguished Name(DN) に含む形式でのみ実験
– **参照環境**
Outlook Express(2000, XP)
– **証明書ツリー**
参照 CA(トラストポイント) - WIDE チーム CA - S/MIME ユーザ
– **失効**
＊参照 CA による S/MIME ユーザ証明書の失効
＊WIDE チーム CA による S/MIME ユーザ証明書の失効
– **結果**
＊有効な証明書 (期待値通り)
上記証明書ツリーにより参照環境で S/MIME ユーザ証明書が検証できた
＊失効した証明書
・WIDE チーム CA 証明書の失効 (期待値通り)
上記証明書ツリーの WIDE チーム CA 証明書の検証に失敗したため参照環境で S/MIME ユーザ証明書の検証に失敗した
・サーバ証明書の失効 (期待値通り)
参照環境で上記証明書ツリーの S/MIME ユーザ証明書の検証に失敗した

● IPsec(IKE) 証明書検証
– **IPsec クライアント証明書の発行**
subjectAltName に dNSName(クライアントの IP アドレス) を入れる。
– **参照環境**
VPN-1, SSH Sentinel
– **証明書ツリー**
WIDE チーム CA(トラストポイント) - IPsec クライアント
– **失効**
WIDE チーム CA による IPsec クライアント証明書の失効

– **結果**
  * 有効な証明書 (期待値通り)
    上記証明書ツリーにより、参照環境で IPsec 証明書が検証でき、通信を行うことができた
  * 失効した証明書 (期待値通り)
    失効により IPsec 証明書が検証できなかったため、通信を行うことができなかった。

2.4.2 相互認証モデル (プレブリッジ) での実験

● 証明書の発行
– **相互認証証明書の発行**
  certificatePolicies にすべての CA で共通のポリシ OID を入れる。(1.3.6.1.3.99)
– **S/MIME ユーザ証明書の発行**
  subjectAltName に S/MIME ユーザの電子メールアドレスを入れる。

● S/MIME ユーザ証明書の検証
– **証明書ツリー**
  他ベンダー CA -相互認証- 参照 CA -相互認証- WIDE チーム CA (トラストポイント) - S/MIME ユーザ
– **失効**
  * 他ベンダー CA による参照 CA との相互認証証明書 (Reverse) の失効
    他ベンダーの S/MIME 証明書の失効に影響
  * WIDE チーム CA による参照 CA との相互認証証明書 (Reverse) の失効
    WIDE チームの S/MIME 証明書の失効に影響
  * WIDE チーム CA による S/MIME ユーザ証明書の失効
    WIDE チームの S/MIME 証明書の失効に影響
– **結果**
  * 有効な証明書 (期待値通り)
    上記証明書ツリーにより、S/MIME ユーザ証明書が検証できた。
  * 失効した証明書
    · S/MIME ユーザ証明書の失効 (一部期待値とならず)
      一部の環境では証明書の失効が検出できず、検証に成功してしまった。証明書に CRLDP が含まれていないことが原因 (CRL は手動でインストール) と考えられる。
    · WIDE チーム CA 証明書の失効 (一部期待値とならず)
      一部の環境では証明書の失効が検出できず、検証に成功してしまった。証明書に CRLDP が含まれていないことが原因 (CRL は手動でインストール) と考えられる。

2.4.3 ブリッジモデルでの実験

● 証明書の発行
– **相互認証証明書の発行**
  * certificatePolicies にベンダ固有のポリシ OID を入れる。
    (0.2.392.200117.1.9.2001.130.1)
  * PolicyMappings にブリッジ CA のポリシ OID を入れる。
    (0.2.392.200117.1.9.2001.10.1)
– **S/MIME ユーザ証明書の発行**
  subjectAltName に S/MIME ユーザの電子メールアドレスを入れる。

● S/MIME ユーザ証明書の検証
– **証明書ツリー**
  他ベンダー CA - ブリッジ認証 - ブリッジ CA - ブリッジ認証 - WIDE チーム CA(トラストポイント) - S/MIME ユーザ
– **失効**
  WIDE チーム CA が発行している S/MIME ユーザ証明書の失効
– **異常系**
  * 参照 SubCA に発行されたポリシー ID が異なるユーザ証明書
    参照 PCA に発行された参照 SubCA 証明書と WIDE チーム CA に発行されたブリッジ認証 (Reverse) 証明書で、ポリシー ID がマッピングされている状態なので、このユーザ証明書は無効になるはず。
– **結果**
  * 有効な証明書 (一部期待値とならず)
    一部の環境では証明書の検証に失敗してしまった。これは検証環境 (OS, クライアント) がポリシーマッピングに対応していないことが原因と考えられる。
  * 失効した証明書
    · S/MIME ユーザ証明書の失効 (期待値どおり)]

すべての環境下で証明書の検証に失敗したが、これは全てのケースで「正しく」失敗したというよりは、環境がポリシーマッピングに対応していないために失敗したものも含まれると考えられる。

2.4.4 オプション実験

- OCSP の証明書
  - **S/MIME ユーザ証明書への証明書発行**
    - ∗ AuthorityInformationAccess(AIA) の id-ad-ocsp(1.3.6.1.5.5.7.48.1) で OCSP レスポンダのロケーションが指定
    - ∗ OCSP レスポンダへの証明書発行
    - ∗ extKeyUsage フィールドに id-kp-OCSPSigning(1.3.6.1.5.5.7.3.9)
    - ∗ 拡張フィールドに id-pkix-ocsp-nocheck(1.3.6.1.5.5.7.48.1.5)
  - **失効**
    WIDE チーム CA による S/MIME ユーザ証明書の失効
  - **結果**
    - ∗ 有効な証明書 (一部期待値とならず)
      一部の環境で OCSP クライアントとレスポンダとの間で正しい通信を行うことができなかった。これは、S/MIME ユーザ証明書と OCSP レスポンダ証明書の Issuer が異なると正しく通信を行うことができない OCSP レスポンダが存在するためであると考えられる。
    - ∗ 失効した証明書 (一部期待値とならず)
      一部の環境で OCSP クライアントとレスポンダとの間で正しい通信を行うことができなかった。これは、S/MIME ユーザ証明書と OCSP レスポンダ証明書の Issuer が異なると正しく通信を行うことができない OCSP レスポンダが存在するためであると考えられる。

2.5 相互運用環境下での ICAP の課題

実験中には、証明書に意図したフィールドを追加したり、他の CA から渡された証明書発行リクエストを使って証明書を発行する必要があった。証明書プロファイルの設定機能は、実際に相互運用を行う際に必ず必要になる機能であるが、現在の証明書プロファイルの管理インターフェースには課題があることがわかった。本節では、ICAP の証明書プロファイル機能についての記述を含め、実験を通じて得られた ICAP の課題について述べる。

- クライアント証明書のリクエスト形式
  ICAP が発行する証明書のフィールドの組み合わせ (証明書プロファイル) は icap_app.profile という設定ファイルで指定される。証明書プロファイルは、同一のフィールドのセットを持つ証明書を複数発行するためには必ず必要となる機能である。しかし現在の証明書発行インターフェースは、証明書プロファイルとリクエストデータの種類が決めうちであるため、同一形式のの申請データでプロファイルが異なる証明書を発行したり異なる形式の申請データで同一のプロファイルの証明書を発行することができない。リクエストデータの書式が決めうちである理由は、ICAP のリクエストページ毎にプロファイル名が指定されるためであり、プロファイル選択機構自体が原因ではない。リクエストデータの書式を受け付けるインターフェースで、自己署名証明書と PKCS#10 の好きなほうを選べるインターフェースにできればいいと思われる。
- DN の一部をきめうちしている問題
  ICAP が発行する証明書の DN の一部は、証明書発行インターフェースで決めうちに設定していた。設定インターフェースを通じて設定できる方が望ましい。
- ICAP が下位 CA となるときの上位 CA への発行リクエスト
  これは ICAP 本体の仕様であるが、自己署名証明書でしかリクエストを発行できない。PKCS#10 の作成機能が必要である。
- subjectAltName 対応
  X.509v3 拡張フィールドの subjectAltName を証明書に含めるためには、設定ファイルの変更が必要であった。また現在 subjectAltName に設定できる値は、証明書のリクエスト時に入力されたメールアドレスだけなので、dNSName 形式などに対応できない。値と型を指定できる証明書インターフェースが望ましい。
- CRLv2 対応
  現在の ICAP はバージョン 2 の形式で CRL を発行することができない。他ベンダーの全ての CA はバージョン 2 に対応していた。
- policyMappings の指定

policyMappings フィールドには、今回の実験中に対応したこともあり、値をひとつしか入れられないなどの制限がある。今後は複数ポリシーマッピングもありえるため (特にメッシュ構造で相互認証する場合)、証明書プロファイル管理機構の改良が望ましい。

2.6 まとめ

本章では 2001 度の後半に参加した JNSA による複数 CA 相互運用性実証実験プロジェクトについて報告した。ポリシーマッピングや S/MIME、OCSP といった相互運用の実験に必要な証明書に含まれるフィールドは、ICAP の若干の変更で対応することができた。実験結果からは概ね期待通りのクライアントの動作が見られた。また実験に参加することを通じて、moCA で運用している ICAP の機能の見直しを図ることができた。この実験を通じて得られた課題を元に moCA として運用する CA の改善が図られることを望む。

第 3 章　moCA 実運用に向けた移行計画

3.1 背景と目的

WIDE プロジェクトでは、メンバ同士で情報共有するための手段として、メーリングリスト (ML)、WWW、WIDE のサービスホスト (sh.wide.ad.jp) などを利用している。

ML については、今のところ投稿できる人に制限を設けている ML はほとんどなく、最近では SPAM が流れてくるために本当に重要なメールを見落とす危険性が増している。これに対して、投稿できる人に制限を設けるべきではないかという議論が持ち上がっている。現状では投稿できる人のメールアドレスをあらかじめ登録する方法などが検討されている。

WWW については、メンバ専用ページを設けており、そのページへアクセスする際のメンバ認証として、メンバ全員が同じパスワードを共有する、共有パスワード方式を利用してきた。この方式は、メンバ数が増えるにつれて秘密管理の難しさや、パスワードを変える難しさが問題視されるようになってきた。しかし、現在のメンバ以外の人にもパスワードが知られていることに対して、潜在的には問題と考えられるものの、思い切って方式を変える動機付けには至らなかった。簡単な認証強化の方法として、共有パスワードを廃止し、メンバ個別のパスワードを利用する方法も考えられるが、WWW サーバ管理者の負担増が見込まれるため実際にはなかなか変更に踏み切れない。

WIDE のサービスホスト (sh.wide.ad.jp) については、メンバがサービスホストを介してファイルの交換をしたり、WWW サーバにコンテンツを置くために利用されている。メンバにはログインアカウントを作成し、サービスホストへのログイン時にメンバ認証を行う。サービスホストにメンバ以外からログインされると、サービスホストにあるリソースの他、踏み台として他の WIDE リソース、他の組織に被害を与えられる可能性があることは明らかである。したがって、ログイン時の認証方式は新しい技術が出てくるごとに見直されてきたし、メンバも必要性を理解した上で方式の変更に従っている。具体的には、パスワード認証、S/Key[59] を用いたワンタイムパスワード認証、SSH の中の公開鍵暗号を用いた認証へと変わってきた。

moCA は実験開始当初から、WWW のメンバ専用ページへアクセスする際のメンバ認証に証明書を利用することをターゲットにしている。しかし、その当時 (1997 年) のサービスホストへのログイン時認証はまだ、ワンタイムパスワード認証であり、公開鍵暗号を利用するまでに至っていなかった。このような状況では、メンバ専用ページを見るためのメンバ認証レベルの方が、サービスホストへのログイン時認証レベルより高くなりアンバランスである。moCA をベースにした証明書によるメンバ認証を、実験ではなく運用として導入するには、他の認証レベルが上がらないと難しい状況であった。

状況は変わってきた。現在は、サービスホストへのログイン時メンバ認証には、SSH の中の公開鍵暗号を用いた認証へと変わり、メンバは SSH で利用するための秘密鍵と公開鍵を作成し、公開鍵をサービスホストに登録する、という経験を積んでいる。一方、WWW のメンバ専用ページを見るには相変わらず共有パスワード方式が利用され続け、認証レベルのギャップが目立ってきて、問題と感じる人が増えてきた。以前よりも、moCA をベースにした証明書によるメンバ認証レベルと、サービスホストへのロ

グイン時メンバ認証レベルとを近づけやすい状況になった。

そこで、WWW のメンバ専用ページを見るためのメンバ認証レベルを、サービスホストへのログイン時メンバ認証レベルに近づけていくことを目的にして、再度、moCA をベースにした証明書によるメンバ認証の導入について検討することになった。これに伴い、moCA としては、実験から実運用に向けての移行計画を立てることにした。

3.2 方針

WWW のメンバ専用ページを見るための認証を、共有パスワード方式から moCA をベースにした証明書によるメンバ認証方式へ移行するにあたり、以下の方針を立てた。

1. メンバ管理者や WWW サーバ管理者にとっての管理コストをあまり増やさないようにすること
2. メンバ登録作業とリンクさせ、メンバにも負担をかけないようにすること
3. サービスホストへのログイン時メンバ認証レベルに近づけ、このレベルを超えないように工夫すること

3.3 移行計画

3.2 節で述べた 3 つの方針に沿って移行計画を立てた。ここでは主な検討内容について述べる。

最初の方針に対しては、以下の工夫が必要になる。

- 証明書発行や取り消しに関してメンバ管理者 (WIDE DB 管理者) の手間ができるだけかからないこと
  - 証明書発行に関する問い合わせがメンバ登録者に集中しないよう手続きを簡単にすること
  - CRL を運用してメンバが脱退したときには CRL から自動的に検出できるようにすること
- メンバなら誰でも見られるページについて、WWW サーバのアクセスコントロールリストをメンバ全員分リストアップしなくても済むようにすること

  ワイルドカードを利用できるようにする。
- メンバを限定して見られるようにするページについて、WIDE 番号のみでアクセスコントロールリストを記述できるようにすること

  WIDE 番号を証明書に記載して発行する。ただし、WIDE 番号だけでは、ブラウザにある複数の個人証明書から選択する際にわかりにくいと予想されるため、表示上の工夫として「WIDE 番号+" "+氏名」の形で記載することになった。

2 番目の方針に対しては、特に証明書発行時にメンバの負担がかからない工夫が必要となる。実験では証明書発行をメンバ各自で行う方法をとっていたが、メンバ登録者がメンバ登録作業のついでにメンバの証明書発行まで行い、メンバに配付した方がよいという結論になった。最初の方針を考え合わせ、メンバの証明書発行をメンバ登録者がコマンド一つで実行できるような方法をとることにする。

3 番目の方針に対しては、CA の存在を意識しなくてすむように工夫する必要がある。サービスホストへのログイン時メンバ認証に利用している SSH の中の公開鍵暗号を用いた認証では、メンバが自分の公開鍵をメンバ登録者に届けることによって運用しており、CA のような第三者は存在しない。したがって、サービスホストへのログインと近い状況にするため、個人証明書のみを配付し、moCA や WIDE ROOT CA などの CA 証明書を Web ブラウザに登録する作業をオプションにする。つまり、個人証明書を SSH の公開鍵と同じ位置づけで扱えるようにする。

このように検討した結果を、実験から実運用に向けての変更概要表として表 3.1 にまとめる。また、証明書フォーマットの仕様である証明書プロファイルを表 3.2、3.3 に示す。メンバの識別子として今まで電子メールアドレスを利用してきたが、今後は WIDE 番号が主体となる。電子メールアドレスは S/MIME を利用したい人が使えるようにするための目的で引き続き証明書に記載するが、WIDE に登録している電子メールアドレス以外にも複数登録できるように検討をさらに進める。

移行開始は、例年の証明書更新時期と合わせ、2002 年の 6 月となる予定である。

表 3.1.　moCA 実運用に向けた変更概要

	moCA の現状	moCA 実運用への移行
CA(信頼点構築/管理)		
CA, RA 構成	・WIDE ROOT CA をルート CA とする階層 CA の下位 CA	・変更なし
	・RA なし	・メンバ登録者が RA となる
CA 証明書配付方式	・Web を用いた配付	・WIDE ROOT CA 証明書、moCA 証明書は Web を用いて配付するがオプション
	・CA 証明書ハッシュ値のアナウンス	・変更なし
名前空間の定義	・メンバ登録時のメールアドレスが中心	・WIDE 番号を中心にする
CA 間連携	・信頼点は WIDE ROOT CA	・変更なし
	・他の系列 CA との連携なし	・変更なし
証明書管理		
証明書プロファイル	(別表参照)	(別表参照)
証明書発行方式	・CA 証明書は半オンライン	・変更なし
	・個人証明書はブラウザから WWW サーバにオンラインで申請	・個人証明書はメンバ登録者がメンバ登録作業とリンクさせて発行し配付する
	・申請時の本人確認にはメールの到達性利用	・証明書配付時にメールの到達性利用
証明書保存方式	・ブラウザの証明書 DB に保存	・変更なし
証明書廃棄管理方式	・CA としては CRL を発行しているが利用していない	・メンバからの申請によりメンバ登録者が証明書廃棄 ・CRL を定期的に発行しクライアント認証時に WWW サーバにて CRL を参照する
秘密鍵管理		
鍵対作成方式	・ブラウザ（ディスク上で作成）	・メンバ登録者がディスク上で作成
秘密鍵保存方式	・ディスク上で保存	・変更なし
	・バックアップ作成は各ブラウザの export 機能を利用	・変更なし
モビリティ	・鍵対バックアップを持ち歩くことで可能	・変更なし
	・moCA の場合、個人証明書は一人につき何通でも発行可能（必要時に証明書発行可）	・個人証明書を一人につき何通発行できるか未定
ディレクトリ		
スキーマ		
アクセス方式	・証明書は HTTP（CGI) で入手可だが利用されていない	・変更なし
	・CRL は HTTP（CGI) で入手可だが利用されていない	・CRL を HTTP で入手可として利用する
公開範囲	・制限なし	・変更なし

表 3.2. 証明書プロファイル表

フィールド名 (フィールドの意味)	moCA 実験時	moCA 運用時（予定）
Certificate		
tbsCertificate		
version(X.509 のバージョン)	v3	v3
serialNumber(証明書シリアル番号)	必須	必須
signature(署名アルゴリズム)	SHA-1 WithRSAEncryption	SHA-1 WithRSAEncryption
issuer(証明書発行者名)		
Country (C=)	○	○
StateorProvince (ST=)	−	−
Organization (O=)	○	○
OrganizationalUnit (OU=)	○	○
CommonName (CN=)	−	−
DomainComponent (DC=)	−	−
EmailAddress (Email=)	−	−
validity(有効期間)	終了期日固定 (1 年)	終了期日固定 (1 年、いずれは数年)
notBefore(有効期間開始)		
UTCTime	○	○
GeneralizedTime	−	−
notAfter(有効期間終了)		
UTCTime	0206302359592	0306302359592
GeneralizedTime	−	−
subject(証明書所有者名)		
Country (C=)	○	○ (いずれ省略へ)
StateorProvince (ST=)	−	−
Organization (O=)	○	○ (いずれ省略へ)
OrganizationalUnit (OU=)	○（複数）	○（複数） (いずれ省略へ)
CommonName (CN=)	○ 氏名	○ WIDE 番号+” ”＋氏名
DomainComponnt (DC=)	−	−
EmailAddress (Email=)	○	−
subjectPublicKeyInfo (証明書所有者の公開鍵情報)		
algorithm	RSA	RSA
subjectPublicKey	1,024bit	1,024bit 以上
issuerUniqueID	−	−
subjectUniqueID	−	−

表 3.3.　証明書プロファイル表 (続き)

フィールド名 (フィールドの意味)	moCA 実験時	moCA 運用時（予定）
extensions(X.509v3 拡張フィールド)		
AuthorityKeyIdentifier (証明書発行者の鍵識別子)	○	○
SubjectKeyIdentifier (証明書所有者の鍵識別子)	−	−
KeyUsage(鍵の利用目的)	−	−
privateKeyUsagePeriod (秘密鍵の利用期間)	−	−
certificatePolicies (証明書のポリシ情報)	−	−
PolicyMappings (ポリシ間のマッピング)	−	−
SubjectAltName(証明書所有者の別名) rfc822Name(電子メールアドレス)	○	○ (一つ、いずれ複数へ)
IssuerAltName(証明書発行者の別名)	−	−
subjectDirectoryAttributes (証明書所有者のディレクトリ属性)	−	−
BasicConstraints (証明書の種別 (CA か EE か))	○	−
NameConstraints (名前の表現に関する制限)	−	−
PolicyConstraints (ポリシに関する制限)	−	−
ExtKeyUsageSyntax (keyUsage では表現しきれない 鍵の利用目的)	−	−
cRLDistributionPoints (CRL 入手のためのアクセス先)	○ http	○ http
authorityInfoAccess (証明書発行者情報へのアクセス先)	−	−
localExtensions(独自拡張)		
ICAT AuthorityInfoAccess	○	−
Netscape　Extensions		
NetscapeCertType(証明書タイプ)	○	−
signatureAlgorithm	SHA-1With RSAEncryption	SHA-1With RSAEncryption
signatureValue	必須	必須

第4章 おわりに

今年度の JNSA 実験参加活動や moCA の運用に向けた移行計画検討活動によって、われわれは二つの道標を得ることができた。

一つは、他の CA パッケージとの相互運用性を加味した ICAP の改良に関する道標である。現実には、JNSA 実験時のように ICAP のようなフリーソフトウェアと製品となっているパッケージの間で連携をとる構成はポリシーとして難しい面がありそうである。しかし、例えば ICAP や他のフリーソフトウェアである程度実験的利用をした後、実運用時に他の CA パッケージに変更することがスムーズにできるようになれば、ICAP のようなフリーソフトウェアにも、CA からなる PKI 技術促進に一役を担う意味が出てくる。

もう一つは、members oriented、つまりプライベート CA の追求に関する道標である。われわれは、WIDE のような、メンバの本人確認レベルが必ずしも厳密とはいえない組織であっても、運用コスト上のメリットが感じられる CA の実現を目指して行きたい。そのためには、CA の理論上のあるべき姿から、現実には削っても問題のないところを削って簡素化して行く姿勢も重要であろう。

今年度検討した計画と目標に沿っての、開発事項、具体的な課題は山積みの状態であり、来年度は、今年度得られた道標を浮かべながら着実に活動を進める予定である。

- WIDE ROOT CA の公開鍵ハッシュ値
  `2B:68:BD:1B:26:28:2A:AC:CF:F3:45:90:1D:6C:2A:9C`
- moCA の公開鍵ハッシュ値
  `C9:6F:22:16:37:5A:8E:D5:F4:E9:74:B0:8C:18:4A:A6`