第 11 部

利用者認証技術

第 1 章

広域認証技術 Interauth-WG

1.1 はじめに

広域認証 WG では，インターネットにおける本人認証手段である公開鍵証明書の安全で実用的な発行方式および配布，検証方式について研究をしている．ここで認識されている課題は，次の 3 つである．

1. 証明書を発行する証明書発行局をいかに安全に，かつ，低コストで運用するか？
2. 分散環境で，発行した証明書を利用者に効率よく配布するにはどうしたらよいか？
3. 署名に利用された証明書の有効性を検証するには，どんな点が問題となるか？

本 WG では，まず課題 1 に対して，オンライン証明書発行局を Web 上に実装し，安全性と運用コストについて定性的な評価を与えた．課題 2 に対して，分散データベースの観点からいくつかの配布モデルを定式化し，待ち行列理論に基づき定量評価を行ない，理想的な配布方式を明らかにした．最後に課題 3 について，従来のよく知られた認証フレームワークには廃棄証明書情報の鮮度と否認防止サービスの欠如の問題点があることを明らかにし，それを解決するオンライン証明書検証サービスを提案した．更に，試験的な実装と運用実験を行い，それらの統計データに基づいて運用規模を同定した．

本章の構成は，次の通りである．
2 章では，オンライン証明書発行局パッケージ (ICAP) の実装と評価を報告する．
3 章では，証明書発行局間の証明書情報共有機構について述べ，平均応答時間の点から最適な設計を示す．
4 章では，X.509 認証フレームワークの問題点とオンライン証明書検証サービスとその評価について報告する．

1.2 オンライン証明書発行局パッケージ ICAP の実装と評価

1.2.1 はじめに

近年の電子商取り引きなどへの関心が高まる中、広域認証技術、とりわけ証明書発行局[?]（以下単に「発行局」と記する）および発行局構築のためのソフトウェアの必要性が増している。

発行局を構築する為のソフトウェアは、すでにいくつか存在しており、実際の運用に用いられているが、その際に問題となっているのが証明書を発行する時の本人認証の手間である。

一般に発行局では、申請者からの依頼を受けて証明書を発行する際、直接対面による方法などで本人の確認を行なっている。つまり、証明書を発行する際の事務的作業量が決して少なくはなく、組織内で発行局を立ち上げる際の障壁となっていた。

そこで本稿では、オンラインによる証明書の発行を実現することにより、従来よりも証明書の発行手続きを簡略化するモデルを提案する。

さらに、提案したモデルを発行局パッケージ (ICAP) の機能として実装した結果について、検討をおこなう。

1.2.2 発行局の課題

A. 証明書発行手続き

最初に、証明書の発行手続きを明確にするために、証明書の発行を希望する者（証明書申請者）が実際に証明書を入手するまでの手順を図 1 に示す。

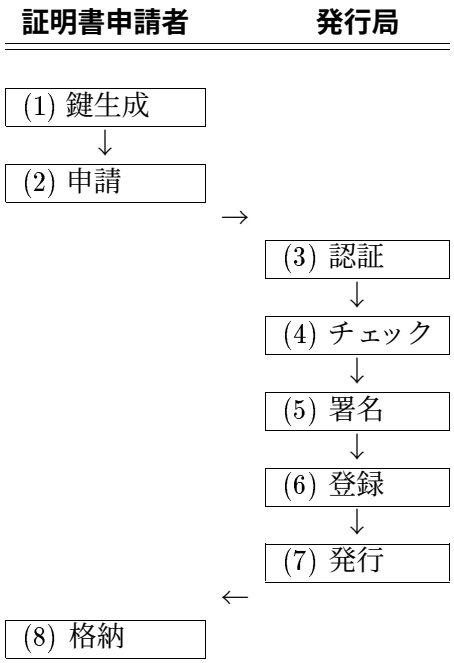

図 1：証明書の発行手続き

図 1 の説明:

1. **鍵生成**　申請者は自分自身が使用する秘密鍵および公開鍵を生成する。
2. **申請**　申請者は (1) で生成した公開鍵に対する証明書の発行を、発行局に申請する。
3. **認証**　発行局は申請者の認証をおこなう。
4. **チェック**　２重申請、あるいは同姓同名などによる DN (Distinguished Name) の重複検査を行なう。
5. **署名**　発行局の秘密鍵で署名を行い、証明書を作成する。
6. **登録**　作成した証明書を、証明書管理のためのデータベースに登録する。
7. **発行**　証明書を申請者に返却する。
8. **格納**　申請者は、発行された証明書を自分自身が使用しているシステムに格納する。

図 1 の手続き (3) における本人の確認方法は、証明書の有効性・信ぴょう性と大いに関連する為、発行局のポリシーによってさまざまな手法が取り得る [?]。

たとえば、ノルウェーの教育研究用ネットワークである UNINETT の場合、下位の発行局が証明書を発行する際は、申請する本人自身が発行局を訪れ、パスポートや運転免許証等を提示することによって、認証が行なわれている [?]。

また、WIDE プロジェクトの発行局は、認証のレベルを 2 つに分けている [?]。一つは、低信頼発行局が発行するペルソナ証明書 (Persona Certificate) で、暗号化を考慮していない通常の電子メイルを用いて発行している。もう一つは、公証局が発行する証明書で、これは本人の確認をきちんと行なう「公証人」を媒介人として証明書を発行している。

B. 証明書発行手続きの問題点

証明書発行手続きにおいて重要な点は、如何に、確実に、証明書の申請者が本人であるかどうかを確認することである。

その為に、UNINETT 発行局 や WIDE 公証局 の場合、証明書の申請者は自らを証明する書類を携えて発行局に行く必要があった。

しかし、これには次のような問題がある。

- 発行局の負担
  - 発行申請件数が増えてくると、それに伴って発行事務に関する作業量が増大する。
- 申請者の不便
  - 証明書を入手する際の本人確認が、対面による方法のため、申請者にとっては煩わしい。
  - 発行局側では、人手作業による認証作業の為、窓口の受け付け時間に制限が設けられることが考えられる。この場合は、証明書発行までに時間を要する場合が考えられる。

1.2.3 証明書発行手続きの簡略化の提案

ここで、証明書の発行を希望する利用者のコンピュータ利用環境について、たとえば、大学の計算センターの利用者が、証明書の発行を希望するケースを想定してみる。

通常、大学の計算センターを利用する為には、事前に利用登録が必要である。利用登録を行なうことができる資格者は、多くの場合、大学の構成員、もしくはそれに準じた者であるようだ。そして利用登録者には、コンピュータを利用するためのアカウントとして、「ログイン名」と「パスワード」が発行される。

大学の計算センターにおける利用登録の手順を図 2 に例示する。

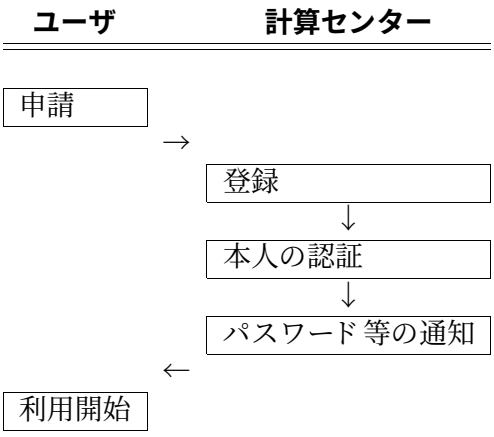

図 2： 利用登録の流れ（例）

つまり計算センターの利用者には、「ログイン名」「パスワード」のような、コンピュータを利用する際の識別情報が与えられており、しかも利用登録を行なった時点で、本人の認証が行なわれているとみなすことができる。

ここで仮に、発行局が計算センターの利用者の識別情報を認証に利用することができるならば、証明書発行時に、再度、前述のような人手による本人認証作業を行なう必然性は少なくなる。

もう一度、図 1 の証明書発行手続きのモデルを眺めてみる。

図 1 において、手続き「(3) 認証」〜「(7) 発行」は、発行局側の作業であるが、「(3) 認証」の際、計算センターの識別情報、たとえば「ログイン名」と「パスワード」を利用できるであろう。そして、識別情報によって本人の認証を行なうのであれば、発行局側の処理は自動化が可能であり、「オンラインによる証明書発行」を実現することができる。これにより、証明書発行に要する発行局側の作業を、大いに軽減することができる。

オフラインによる証明書発行手続きと、オンラインによる証明書発行手続きの概要を、図 3 および図 4 に示す。

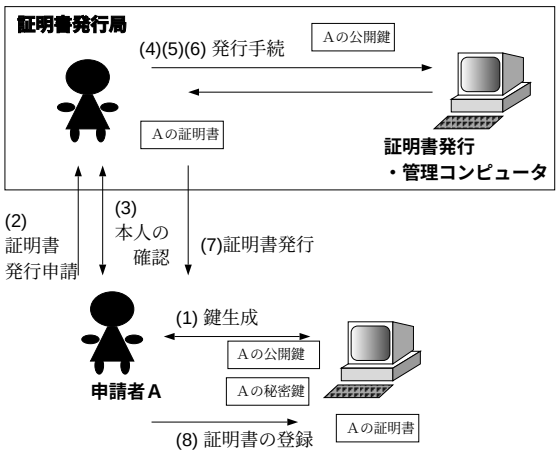

図 3： 従来の証明書発行手続き（オフライン）

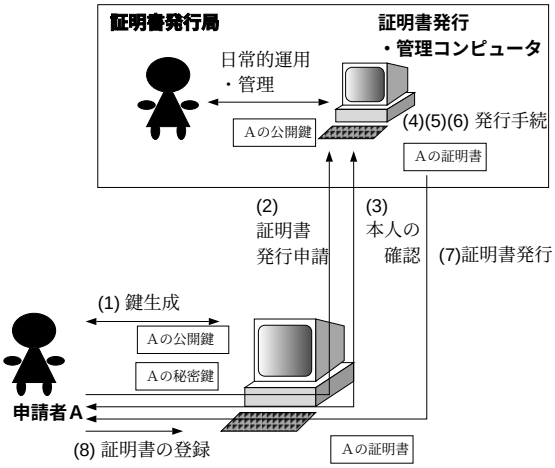

図 4： オンラインによる証明書発行手続き

以上の検討をふまえ、オンラインによる証明書の発行を実現する為の前提条件を、明らかにする。

● 前提条件 1

1. 証明書の発行を希望する者は、どこかの組織や機関（たとえば計算センター）に秘密の識別情報（たとえばパスワード）を登録済みである。

2. 発行局は、証明書発行を希望する者の識別情報を、事前に入手することができる。

ところで、オンライン認証を行なう際には、申請者の識別情報がネットワーク上を流れることになる。
識別情報には、

- （通常の）パスワード
- ワンタイム・パスワード
- チャレンジ／レスポンス

などを用いることが考えられるが、通常のパスワードを用いる場合は、ネットワーク上を流れる情報に対する盗聴対策が必要となる。その場合は、オンラインによる証明書発行を実現する為の前提条件に、次の条件も加える必要がある。

● 前提条件 2

1. 証明書申請者が使用するコンピュータと発行局間の通信は、盗聴への対策を講じている。

1.2.4　ICAP の開発

前節までの検討をふまえ、筆者らは、証明書の発行をオンラインで行なう機能をもつ発行局ソフトウェアー ICAP (ICAT Certification Authority Package) ーを開発した[1]。

ICAP とは RFC1422 に記載されている発行局を実際に立ちあげ、運用するためのソフトウェアパッケージの名称である。

A. 特徴

ICAP には以下の特徴がある。

- オンラインによる証明書発行機能
- オンラインによる証明書有効性の確認機能
- オンラインによる証明廃棄機能
  - 証明書の発行を希望する利用者は、オンラインで証明書の発行、有効性の確認、および廃棄が可能である。
- 簡単なインストール
  - ICAP は簡単にインストールできるよう、(1) インストールスクリプトの充実、(2) バイナリ形式での配布 を行なっている。これにより ICAP のインストールには、面倒な「make」等は不要である。
- 簡単操作
  - 証明書発行に関するほとんどの操作は、Webブラウザから行なうことができる。

B. システム構成

ICAP は、WWW サーバ上で動作する CGI プログラム、および CGI プログラムから起動されるいくつかのプログラムから構成される（図 5）。

[1]ICAP は `http://www.icat.or.jp` より入手可能である。

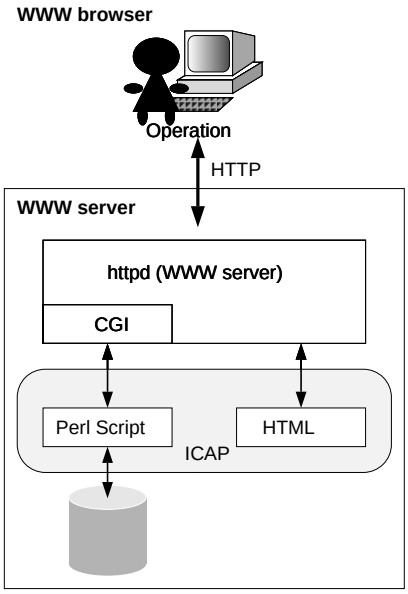

図 5: ICAP のシステム構成

C. 動作環境

ICAP の動作環境は、次の通りである。

- **対象システム**
  - SunOS4.1.X
- **必須ソフトウェア**
  - **WWW サーバ**

    NSCA httpd または Apache
  - **Web ブラウザ**

    netscape navigator ,

    Internet Explore 等
  - **インタプリタ**

    Perl 4.036 以上

  * WWW サーバと Web ブラウザについては、SSL(Secure Socket Layer) [?] のような暗号通信路を用いたアクセスが可能であることが望ましい。

D. 操作とメニュー

ICAP の操作は、Web ブラウザを用いてネットワークを経由して行なう。ICAP は発行局の管理者向けのメニューページと、一般利用者向けのメニューページを用意している（図

6)。管理者向けメニューページは、パスワード認証によって有資格者以外の者によるアクセスを制限している。

ICAP のメニューを以下に示す。

1. 一般ユーザ向けサービス

   (a) 証明書を申請する

   (b) 証明書の有効性を確かめる

   (c) 証明書を廃止する

   (d) 証明書を検索する

   (e) 廃棄証明書リスト (CRL [2]) を見る

   (f) CA のタイムスタンプサービス

2. 管理者向けサービス

   (a) ユーザアカウントの登録/変更/削除

   (b) ユーザの証明書を表示

   (c) ユーザの証明書を発行する

   (d) ユーザの証明書の登録を抹消する

   (e) CRL の発行

   (f) CRL の閲覧

   (g) 管理記録の一覧

   (h) 統計情報

[2]Certificate Revocation Lists

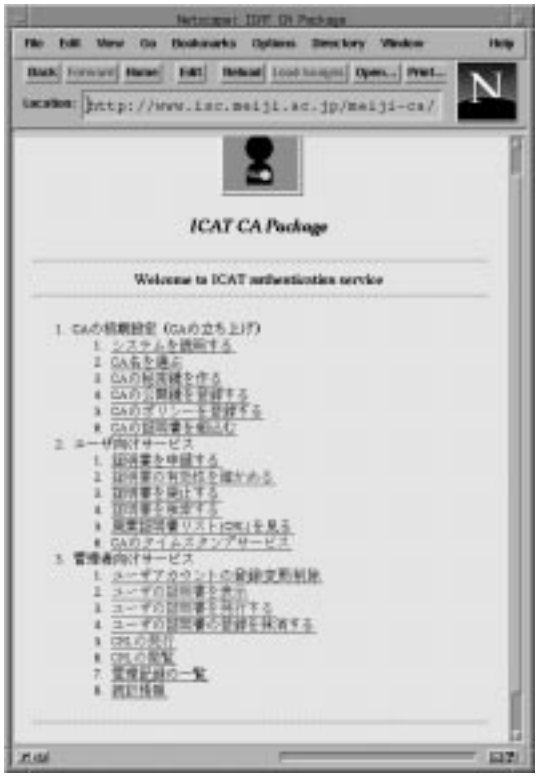

図 6： ICAP 管理者向けメニュー

1.2.5 ICAP を用いた発行局の立ち上げ

ICAP はいずれかの PCA[3]の下位にあたる発行局での利用を念頭においている。本節では、IPRA[4]が証明書を発行した日本で最初の実験的PCA である ICAT (Initiatives for Computer Authetication Technology- 認証実用化実験協議会-) PCA の下位に、ICAP を用いて新たに発行局を立ちあげる際の手順を紹介する。

1. 新規発行局

   (a) ICAP の入手

   (b) ICAP のインストール

   (c) 鍵生成

   発行局自身の鍵を生成

   (d) 発行局自身の証明書の申請

   公開鍵を ICAT PCA に送付、証明書の申請を行なう。

[3]Policy Certification Authority
[4]Internet Policy Registration Authority

2. ICAT PCA

   (a) 発行局運用者の認証

   現在のところ、ICAT 側から電話を使ったコールバックを行なうことにより発行局の運用者の認証を行なっている。

   (b) 証明書を発行

   発行局の公開鍵に署名、証明書を発行する。

3. 新規発行局

   (a) 証明書のインストール

   (b) 発行局の運用開始

ICAT PCA は、1996 年 2 月に IPRA から証明書が発行された。そして、1997 年 1 月現在、ICAT およびその参加組織は、ICAP を用いて図 7 に示す証明書発行局の階層構造を実現している[5]。

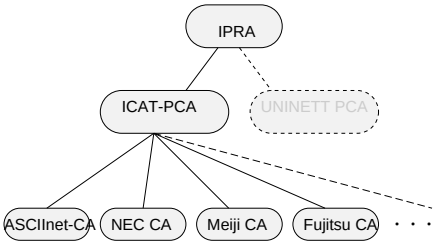

図 7： 発行局の階層構造

1.2.6　ICAP を用いたオンライン証明書発行

証明書の発行を希望する者が、オンラインで証明書の発行申請を行なう手順を紹介する。

前提として、証明書申請者は計算センター等にアカウント登録を事前に行なっており、そのアカウント情報は発行局も保持している必要がある。

1. 発行局

   (a) 申請予定者のアカウント情報を登録する

   ICAP が認証で用いるパスワードファイルは、SunOS4.x が参照するパスワードファイル (`/etc/passwd` や NIS passwd マップ) と同一フォーマットである。

2. 申請者

[5]ICAT PCA の下位には、14 の発行局が存在している。

(a) 秘密鍵と公開鍵を生成。

(b) Web ブラウザを用いて発行局のホームページへ接続。

(c) 「証明書を申請する」というメニューを選択。

(d) 「公開鍵情報 (N)」欄に、(a) で作成した申請者自身の公開鍵を入力する（図 9)。同時に証明書の DN も指定する。

(e) 自分のログイン名およびパスワードを入力する。

3. 発行局

(a) パスワードのチェック。

(b) 申請フォームのチェック。

(c) 証明書の発行。

4. 申請者

(a) 証明書の格納。

ICAP は、一般ユーザが証明書の申請を行なう際に、パスワードによって申請者本人の認証を行なう。従って、申請者のパスワードがネットワーク上を流れてしまう為、第 3 者がネットワークを盗聴することによってセキュリティ上の問題が発生する可能性がある。これを回避する為に、Web ブラウザと ICAP 間の通信は、極力、SSL による暗号通信路を用いて行なう（図 8)。

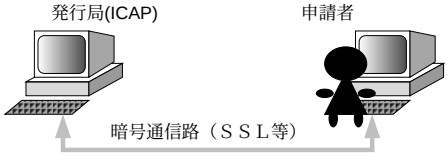

図 8: ユーザ〜ICAP 間の暗号通信路

図 9： ICAP を用いた証明書の発行

1.2.7　考察

A. 認証作業量

オンライン証明書発行方式は、オフライン方式に比べ、発行局側では、証明書発行時の認証作業をオフラインで行なわずに済むという利点がある。

もっともこれは、証明書を発行する時点での作業についてであり、その前提として、申請者本人に対するアカウント発行時の認証作業は、依然として存在する（図 10）。

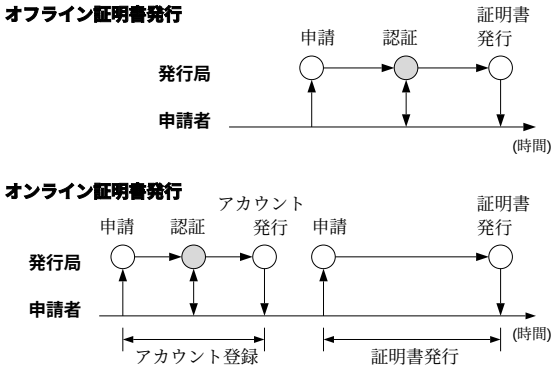

図 10： 認証作業の比較

仮に証明書の発行局と、アカウントの発行を行なう組織が同一であり、発行局がアカウント情報を入手できるのであれば、発行局を新たに立ちあげる際の作業量の増加はあまり無いものと思われる。

しかしながら、これらの組織がまったく別々のものであり、発行局でアカウント情報を入手できないのであるのならば、オフライン証明書発行方式に比較し、作業量の減少は期待できない。

B. 本人認証の確実性

オフラインによる証明書発行方式において、本人であるかどうかの確実性は、対面で認証した際に決まる。

これに対し、オンラインによる証明書発行方式は、

- アカウントを発行する際の本人認証の確実性
- パスワードの確実性

により決定される。ここでパスワードの確実性とは、証明書発行時に申請者が入力したパスワードそのものが、第３者によるものではなく、正しく本人のものであるかどうかを意味している。

オンライン証明書発行方式において、本人であるかどうかの確実性は、この２つの要素が共に確実であるかどうかによって決まる。従って、オンライン証明書発行方式は、オフライン証明書発行方式よりも安全性において劣るといえよう。

さて、それではオンライン証明書発行方式において、安全性を向上するにはどうすればよいであろうか？

根本的な対策ではないが、認証の際に、パスワード以外の情報を、補助的に用いることによって、安全性を向上できるものと思われる。

たとえば証明書を発行する際に、DN の情報として、本人の本名、生年月日、電話番号等の入力も求めるようにする。そして証明書を発行する時は、パスワードによる認証ばかりでなく、これらの情報も認証のためのキーに用いるようにする。

大学の計算センターで発行局を運用する場合、ICAP が学籍データベースや個人情報データベースと連係することにより、このような認証時の補強も可能であると考えられる。

C. オンライン発行局システムの安全性

ICAP はオンラインで証明書を発行するという特徴を持っているが、それゆえに、ネットワークを用いたセキュリティ的脅威についても考慮する必要がある。

- 識別情報の盗聴

  - 証明書申請者と、発行局間のネットワークを盗聴することによって、証明書を発行する際の識別情報が洩れる危険性がある。これについての対策は、暗号通信路を用いてオンライン発行を行なうことが考えられる。
  - 申請者が proxy サーバを経由して証明書の発行を行なう場合は、proxy サーバ上で識別情報が盗聴される恐れがある。従って申請者は、proxy サーバを経由せずに証明書の発行を行う方がよい。

- 発行局への不正アクセス

  - ICAP が稼働しているコンピュータ自身に対する不正アクセス。たとえば、サーバプログラムなどのセキュリティホールを利用した不正アクセスなどが考えられる。従って発行局側は、常に最新のセキュリティ情報に注意をはらい、適切な対策を施す必要がある。

- なりすまし発行局

  - インターネットにおけるコンピュータの攻撃方法の一つに、denial-of-service attack がある [?]。これは、ネットワークを経由して特定のコンピュータを利用不可能にしてしまう攻撃である。仮に発行局がこの攻撃を受け、停止してしまった場合、その間を利用して、他のコンピュータが、発行局になりすましをするという攻撃が考えられる。この攻撃への対策は、SSL 等を用いてサーバクライアント間の認証機能を持たせ、なりすましを困難にすることなどが考えられる。

1.2.8　他の発行局パッケージとの比較

ICAP と他の発行局パッケージとの比較を図 11 に示す。

1.2.9　結論

オンラインによる証明書発行は、発行局側の証明書発行作業を軽減することができるが、本人確認の際の確実性については、発行申請者自身の持つ識別情報の信頼性に依存している。

従って、ログイン名とパスワードを識別情報に利用する場合、発行申請者自身のパスワード管理に問題がある場合は、証明書の信頼性そのものが低くなってしまう。

しかしながら、企業内のイントラネットや、大学計算機センターなど、証明書の発行対象者に対する身元確認が、既に行なわれている場合においては、本稿で述べたオンラインによる証明書発行方式は、組織全体の業務量を増やすことなく提供可能であるため、非常に有効である。

	SecuDE(4.4)	SSLeay(0.6.5)	ICAP(1.0b)
開発国	ドイツ	オーストラリア	日本
利用条件	無償	無償 (商利用可能)	無償
配布形式	ソースプログラム (5.0a はバイナリ)	ソースプログラム	ソース&バイナリプログラム
動作プラットフォーム	UNIX, Windows, DOS	UNIX, Windows, DOS	UNIX(SunOS)
ユーザインタフェース	コマンドライン、専用 X クライアント	コマンドライン	Web ブラウザ
証明書のバージョン	X.509v1 (5.0a は v3)	X.509v3	X.509v1 (1.0 は v3)
証明書要求インタフェース	端末のコマンドライン, GUI	端末のコマンドライン	Web ブラウザ
証明書の署名アルゴリズム	md2withRSAEncryption md5withRSAEncryption shawithRSAEncryption ..	md5withRSAEncryption md2withRSAEncryption shawithRSAEncryption ..	md2withRSAEncryption
対応アプリケーション	PEM	SSLhttpd, SSLtelnet, SSLftp	PEMCAT, FJPEM, Apache+SSL, PEPOP[?]
実働状況	UNINETT PCA および下位 CA で使用中	?	ICAT PCA および下位 CA で使用中
入手先	http://www.darmstadt.gmd.de/secude/	http://www.psy.uq.oz.au/~ftp/Crypto	http://www.icat.or.jp/pages/p6.html

図 11： ICAP と他の発行局パッケージとの比較

1.3　証明書発行局間の証明書情報共有機構の設計

1.3.1　はじめに

インターネットのようなオープンネットワーク上での新しい認証技術として証明書が注目を浴び、証明書の利用を前提とする通信プロトコルの実装が増えてきている [?][?]。

通信プロトコルによっては、通信当事者間で必要となる証明書 (ユーザ、プロセスおよび証明書発行局の証明書) や CRL の入手方法を特に定めていない場合がある。このような場合、通信プロトコルとは独立に、ユーザやプロセスが証明書を検索する機能や、証明書の有効性を確認する機能が必要になる。

証明書発行局が複数ある場合、その運用環境やポリシは様々であり、証明書発行局は組織のファイアウォール内で運用されている可能性もある。このような状況では、全ての証明書発行局が全てのユーザやプロセスと通信できると仮定するのは現実的ではない。

そこで、われわれは証明書発行局間の通信を利用して証明書情報を共有し、組織外のエンティティに対して間接的に証明書情報を提供する機構を提案する。

本節では、まず、複数の証明書発行局と、エンティティおよび CA 間のファイアウォールの存在を前提として一般的な情報共有モデルを示す。さらに、平均応答時間と CA 数に対するスケーラビリティを中心にモデルの考察を行ない、証明書の検索と証明書の有効性確認という 2 つの観点から、適したモデルを選択し設計を行なう。

提案した機構の一部は、われわれが開発した証明書発行局パッケージ (ICAP) の機能として実装中であり、その方針について簡単に述べる。

1.3.2　証明書情報および証明書発行局の概要

本稿における証明書とは、公開鍵暗号における秘密鍵と公開鍵の対を、それを利用するエンティティ(アプリケーション、ユーザ等) と対応づけるためのものである。証明を行なうのは第三者機関である証明書発行局 (以下、CA) であり、有効期限つきで署名を行ない、証明書として発行し配布する。有効期限内に、エンティティの属性や公開鍵の変更が生じた場合は、証明書を取り消す必要があり、CA は有効期限内に取り消された証明書のリストを CRL(Certificate Revocation Lists) として定期的に発行し配布する。

本節では、X.509[?] で定義されている、証明書や CRL の形式、および CA の一般的な機能概要についてまとめる。

ここでは、証明書および CRL をまとめて証明書情報と定義する。

A. 証明書の形式

証明書は、あるエンティティによる署名を受け取って検証したり、あるエンティティだけに配送したい情報を暗号化する際に必要になる。

図 1.1に X.509 バージョン 3 で定義されている証明書の形式のうち、オプショナルとして定義されている拡張フィールドを除いた基本的な部分を示す。

「サブジェクト」は、証明対象となるエンティティの名前であり、X.500[?] で定義されている構造的な名前形式、DN(Distinguished Name) を用いる。バージョン 3 では、拡張フィールドを利用することによって、電子メールアドレスを証明書中に含めることが可能となっている。

証明書の拡張フィールドについては、バージョン 3 で定義されている以外にも、PKIX[?] などによる独自拡張があり、例えば、証明書に関する問い合わせ先を示すフィールドの定義がある。

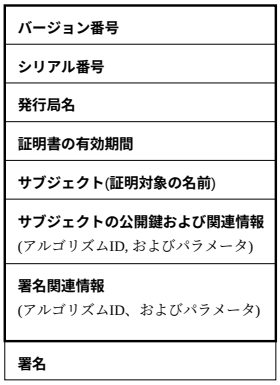

図 1.1: 証明書に含まれるデータ

B. CRL の形式

CRL は、証明書が有効期限内に切れていないかを確認する際に必要になる。

図 1.2に、X.509 バージョン 2 で定義されている CRL の形式を示す。バージョン 3 ではこれらのフィールドに加えてオプショナルな拡張フィールドが定義されている。

CRL の発行間隔は、CA がサポートするアプリケーションの性質を考慮した上で、CA のポリシとして決められる。CRL の発行間隔が長いと、次の CRL が発行されるまでに証明書が取り消されている可能性がある。

CRL のサイズは有効期間内に取り消された証明書の数に比例して大きくなる。CRL のサイズが大きくなると、配送時の通信コストや CRL からシリアル番号を検索する時間が増大するといった問題がある。

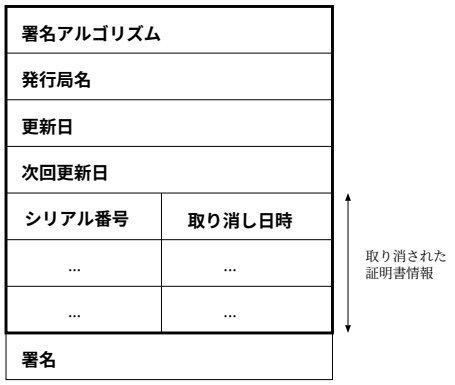

図 1.2: CRL に含まれるデータ

1.3.3　証明書発行局の概要

A. 証明書発行局の機能

一般に CA の機能は、大きくは 3 つに分類される。

- エンティティ(ユーザ、プロセスなど) の認証機能エンティティからの証明書発行要求または証明書取り消し要求を受け付け、そのエンティティが本物かどうかを確認する機能
- 証明書情報の発行機能
  - エンティティと公開鍵の対応を、証明書として発行する機能
  - 発行した証明書が有効期間内に取り消された場合、定期的に CRL として発行する機能
- 証明書情報の配布機能発行した証明書および CRL を必要に応じてエンティティや他の CA に配布する機能。X.509 では X.500 ディレクトリの利用を前提にしている。

B. 証明書発行局の階層モデル

インターネットのような広域分散環境では、単一の CA で全てのエンティティの証明書を管理するのはスケーラビリティに問題があり、複数の CA を分散配置させるのが現実的である。一般的には、CA は図 1.3にあるような階層構造をとり、CA の公開鍵はそれよりも上位の CA が証明する [?]。階層の最上位の CA と公開鍵の対応については信用するものとし、自己署名による証明書を何らかの方法であらかじめ入手することが前提となる。

階層構造では、エンティティの証明書に付加された署名を正しいと確認するまでに、そのエンティティの証明書を発行した CA から最上位の CA まで全ての証明書が必要になる。

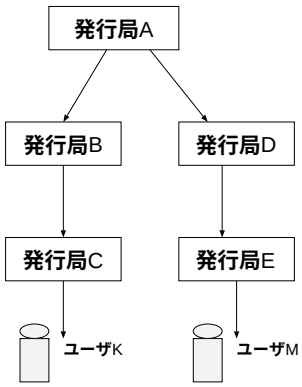

図 1.3: 証明書発行局の階層構造

1.3.4 単一証明書発行局における証明書情報配布機構

あるエンティティが通信相手の証明書を入手するには、通信相手から直接入手する場合と第三者から入手する場合がある。後者の場合、項 1.3.3 で述べたように CA の証明書配布機能を利用することになる。

また、エンティティが通信相手の証明書を既に入手している場合に、その証明書が正しいかどうかを確認するには、証明書の階層上にある CA の証明書を入手する必要がある。

さらに、あるエンティティが通信相手の証明書を既に入手している場合に、過去に利用されたその証明書が現在も有効かどうかを確認したい場合がある。

本項では、階層のない単一 CA における証明書情報配布機構を、証明書の検索機能と証明書の有効性確認機能の 2 つに分けてまとめる。

A. 証明書の検索機能

証明書の検索は、通信相手の証明書を持っていない場合に利用する。X.500 ディレクトリを利用する場合、証明書の検索時の要求は、証明書に含まれるサブジェクト (DN) であり、応答は証明書そのものである。

既存の実装においては、X.500 ディレクトリ以外のツールを利用したり、DN 以外の要求形式を利用している場合など様々である [?]。DN 以外の要求形式を利用する例としては、項 1.3.2で述べたように、証明書中に電子メールアドレスを含めた上で、検索時の要求を電子メールアドレスとする方法がある。

一度入手した証明書をエンティティ側でキャッシュすることによって、証明書の検索による負荷が軽減されることが示されている [?]。このようなキャッシュ機能を前提にすれば検索頻度は通信相手数程度と考えられる。

B. 証明書の有効性確認機能

証明書の有効性確認機能は、既に証明書を入手した状態で利用する。要求は確認対象の証明書であり、その証明書を発行した CA が管理する CRL を利用して有効性を確認することになる。応答は、CRL か、有効か否かの有効性判定情報かの 2 通りがある。

(1) 応答が CRL の場合

有効性確認は各エンティティが行なう。例えば、X.509 では X.500 ディレクトリの CA のエントリに CRL を登録し、各エンティティが CRL を入手する方法が考えられている。

CRL の発行間隔が長い場合、エンティティ側で次の発行日までキャッシュすれば CA に対する有効性確認頻度は低くなる。

(2) 応答が有効性判定情報の場合

有効性確認は CA が行ない、判定情報を配送する。判定情報の要求および応答フォーマットについては新たに定義する必要がある [?][?]。エンティティには CRL を返さないため、エンティティは判定情報を返す CA を認証する必要がある。また、CA とエンティティ間の通信途中で判定情報が改ざんされないようにする必要もある。したがって、判定情報は CA の署名つきである。

1.3.5　複数の証明書発行局における証明書情報共有機構の設計

CA が複数になると、異なる CA から証明書の発行を受けているエンティティ同士が証明書を利用した通信を行なう可能性が出てくる。例えば、PEM[?], S/MIME[?], MOSS[?] などセキュリティ強化電子メールを異なる組織のユーザ間でやりとりする場合である。この場合、エンティティは自分の知らない CA が発行した証明書を検索したり、証明書の有効性確認を行なう必要がある。

本節ではまず、各エンティティが証明書情報を入手する際の、CA 間通信の必要性について述べる。

次に、CA 間通信を使ってエンティティが証明書情報を入手するための機構について一般モデルを列挙する。さらに、証明書の検索機能と証明書の有効性確認機能それぞれに適したモデルを選択し、設計としてまとめる。

1.3.6　証明書発行局間通信の必要性

A. 証明書の検索機能の場合

エンティティが通信相手の証明書を検索したい場合、通信相手の DN や メールアドレスを元にして、通信相手の証明書を発行している CA のアクセス先を見つける必要がある。そのためには、CA のアクセス先と、その CA が発行対象としている (エンティティの) DN やメールアドレスのドメインリストの対応表が管理される必要がある。CA が増えた場合の管理などを考慮すると、この対応表の管理をエンティティが行なうのは繁雑である。そ

こで、CA 間で連携して対応表を管理し、各エンティティは最寄りの CA に 1 回問い合わせるだけで他の CA が発行した証明書を入手できるという機構について考える必要がある。

B. 証明書の有効性確認機能の場合

エンティティが通信相手の証明書の有効性確認を行ないたい場合、その証明書中にある CA の名前を元にして、有効性確認機能へのアクセス先を見つける必要がある。そのためには、CA の名前とアクセス先の対応表が管理される必要がある。証明書中に有効性確認のためのアクセス先が含まれていれば [?]、対応表を管理する必要がなく、エンティティは直接有効性確認の問い合わせ先を調べることが可能である。この場合、CA 間の連携は必要がないように見える。

現実には、全てのエンティティが全ての CA に対してアクセス可能な環境、かつ全ての CA が全てのエンティティからのアクセスを許可できる環境とは限らない。例えば、組織毎に CA が立ち上げられていて、CA が組織のファイアウォール内に設置されている場合である。この場合、組織外のエンティティと証明書を使った通信を行なえるようにするには、CA の証明書情報配布機能を組織外と通信できるように設定する必要がある。CA の証明書情報配布機能をファイアウォール内で運用し続けるならば、組織外からのアクセスを限定できる方がよい (図 1.4)。そこで、CA 間で連携して証明書の有効性確認情報の問い合わせを中継し、組織外のエンティティからの CA へのアクセスを限定する方法が考えられる。

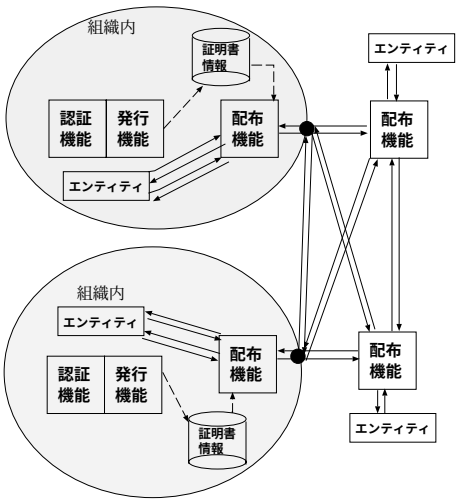

図 1.4: ファイアウォールがある場合の証明書情報共有例

1.3.7　設計上の前提条件

ここでは、機構の設計にあたり、CA、証明書、エンティティに関する前提条件をあげる。

- エンティティは証明書情報の要求を 1 回行なうだけで応答を得る
- エンティティは、任意の証明書の検索および証明書の有効性確認を、自分の証明書を発行した CA に対して行なう
- CA は少なくとも他の 1 つの CA からの通信要求を許可する
- CA 間に階層構造がある場合
  - 最下層の CA のみがエンティティの証明書を発行する
  - 各 CA は自分の親 CA へのポインタを知っている
- エンティティは、自分の証明書を発行した CA (階層がある場合は階層上の全ての CA) の証明書を入手済である、あるいは入手する手段を知っている

1.3.8　証明書情報の共有機構モデルとその比較

ここではまず、各 CA をデータベースサーバ、証明書情報をデータと捉えた場合に考えられるサーバ間の共有機構モデルを列挙する。

モデル 1 各 CA が他の全ての CA の証明書情報をあらかじめコピーし、ローカルに管理する (図 1.5)

モデル 2 CA 間で直接通信を行なって証明書情報を収集する (図 1.6)

モデル 3 CA 間の階層構造を利用し、自分の親 CA とだけ通信し、親 CA が証明書情報を収集する (図 1.7)

モデル 4 CA 間の階層構造を利用し、自分の親 CA と最初に通信し、親 CA が収集して返すポインタ情報を得た後、該当する CA と直接通信して証明書情報を収集する (図 1.8)

モデル 5 CA 間 の階層構造を利用し、自分が階層上の親 CA と直接通信し、該当する CA のポインタ情報を得た後、該当する CA と直接通信して証明書情報を収集する (図 1.9)

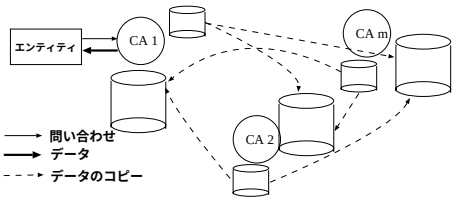

図 1.5: モデル 1

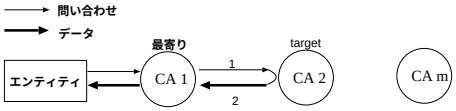

図 1.6: モデル 2

モデル 3〜5 は階層構造を前提としているが、階層構造のない場合でも、親 CA にあたる CA を決めることによって、同様のモデルとして考えられる。

エンティティからは 1 回のデータ要求だけで応答が得られるという条件を満たすためには平均応答時間ができるだけ小さくなるように設計する必要がある。そこで、各モデルの平均応答時間の特徴について調べることにする。

各モデルについて、表 1.1で示す定義および表 1.2で示す仮定を行ない、CA_iへのアクセス率 (λ_i) および CA_i でのサービス率 (μ_i) をモデル毎に計算した。結果を表 1.3に示す。通信時間の計算時、エンティティと CA 間の通信時間を省略している。

さらに表 1.4 に示す環境を想定し、M/M/1 待ち行列モデルを利用して、CA 数を変化させたときの CA 間の平均応答時間 $(T_i = 1/(\mu_i - \lambda_i))$ を求めた結果を図 1.10に示す。データベースの 1 回の検索処理時間の比例定数 a は、同一規模の単一 CA の平均応答時間 (T) が

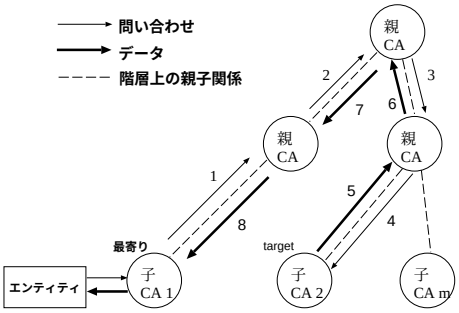

図 1.7: モデル 3

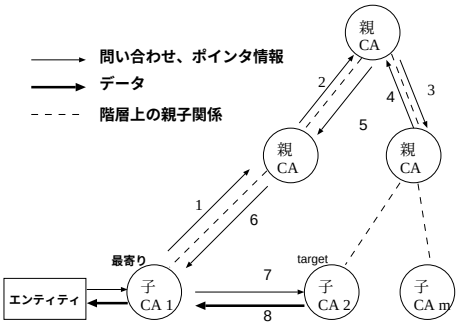

図 1.8: モデル 4

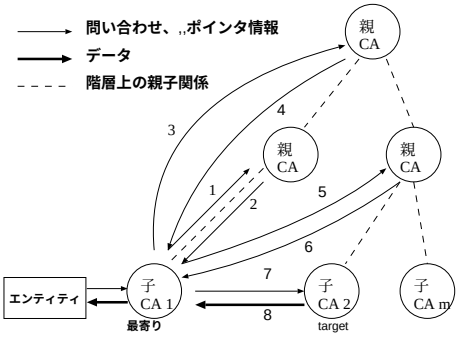

図 1.9: モデル 5

約 2 秒になるような性能を仮定して決めた。図 1.10の “model0” では T を示している。また、図 1.10では、モデル 3,4 で $l = 2$ の 子 CA、および下から 1 段目の親 CA(子の数 $m/2$) についても計算を行なった。

モデル 1 は CA 数に比例してデータベースの大きさが増加するため、平均応答時間は大きくなる。CA が追加された場合は、他の CA のデータベースを全てコピーしなければならず、同じデータを CA 数 (m) 回送ることになる。また、データが追加された場合には、他の全ての CA に追加データを送信しなければならず、データの要求頻度よりもデータの更新頻度が高い場合には結果的に通信量が増えることになる。

モデル 2 は最も CA 数のスケーラビリティに優れている。しかし、モデル 1 と同様に、CA が追加された場合には、他の全ての CA に追加情報を通知しなければならない。CA 数が次第に増えていく場合には管理が困難になる。

モデル 3 は 親 CA の負荷が大きく、CA 数のスケーラビリティには問題がある。しかし、CA の個数に係わらず、階層上直接関係している CA 間のみ (つまり信頼する CA 間のみ) で通信が行なわれるという特徴がある。つまり、CA 間の連携は最も緩やかである。

表 1.1: パラメータの定義

m [個]	CA 数 (階層構造では最下層 CA 数)
n_i[個]	$CA_i(i = 1 \cdots m)$ のエンティティ数 (但し、$N = \sum_{i=1}^{m} n_i$)
p_i[個/s]	1 エンティティの CA へのアクセス率
ν_i	あるアクセスが CA_i の情報に関する確率
$c_{ij}[s]$	CA_iと CA_j間の通信応答時間 (RTT)
l	(階層構造の場合)CA の階層の段数

モデル 4 は 親 CA の負荷がモデル 3 より小さい。また、モデル 2 と比較すると、各 CA は全ての CA に対するアクセス先を知る必要はなく、CA 数が増加したときに管理が容易である。

モデル 5 は 階層の段数 (l) が 1 の場合にはモデル 4 と同一である。階層の段数が大きくなると、子 CA についてはアクセス率が高くなり、モデル 4 よりも平均応答時間が長くなる。親 CA については、他の 親 CA に問い合わせをする必要がないため、サービス率が高くなり、平均応答時間が短くなる。モデル 5 では親 CA が任意の子 CA からの通信を許可する必要がある。

次項では各モデルの特徴を踏まえ、具体的に証明書情報の共有機構の設計について述べる。

1.3.9 証明書の検索機能の設計

CA 数に対する平均応答時間と、CA 数が次第に増える場合の管理の容易さを考慮すると、モデル 4 かモデル 5 が適している。モデル 4 では、親 CA は階層上関係のある CA とだけ通信を行なうため、親 CA の通信を制限すべき状況ではモデル 4 の方が適している。

エンティティが検索キーとするのは、他のエンティティの DN や 電子メールアドレスである。これらと各 CA のアクセス先の対応表についてはあらかじめ 親 CA が 子 CA から収集しておく。

モデル 4 における検索機能の処理の流れを図 1.11に示す。

表 1.2: パラメータに関する仮定

$c_{ij} = c$
$\nu_i = 1/m$
$p_i = p$
CA_i のデータベースの大きさは n_i に比例
データベースの 1 回の検索処理時間は、データの大きさ、つまりエンティティ数に比例 (an_i)
(階層構造で) 親 CA は次の中継点を決めるための CA_i名 とアクセス先の対応表を持つ
対応表の 1 回の検索処理時間は、対応表の大きさ、つまり CA の個数 n_i に比例 $(\alpha m\ )$
データベースのデータとアクセス先対応表の検索効率は等しい $\alpha = \frac{\text{対応表 1 エントリの大きさ}}{\text{データの大きさ}} a$

1.3.10　証明書の有効性確認機能の設計

エンティティが入力する証明書中に、アクセス先の情報が含まれていれば (項 1.3.2参照)、CA 名とアクセス先の情報は自動的に得られ、モデル 2 の欠点が解消される。

このような条件のもとでは、CA 数に対する平均応答時間の最も優れたモデル 2 が適している。

有効性確認機能の処理の流れは図 1.12のようになる。

1.3.11　CA 間通信に制限がある場合

エンティティと CA 間の通信に制限があるように、CA 間の通信にも制限があるとすると、親子関係にある CA 間のみが通信を行なうモデル 3 が可能性として残される。

この場合、親 CA の負荷が高いため、親 CA の配布機能サーバを複数用意するか、CA の構造の一部で利用するといった条件で利用することが望ましい。

表 1.3: 各モデルの比較

	モデル 1	モデル 2	モデル 3(l=1)	モデル 4,5 (l=1)
CA_iへのアクセス率 (λ_i)	$n_i p$	$n_i p + \sum_{j \neq i}^{m} \frac{n_j p}{m}$	$n_i p + \sum_{j \neq i}^{m} \frac{n_j p}{m}$	$n_i p + \sum_{j \neq i}^{m} \frac{n_j p}{m}$
CA_iでのサービス率 (μ_i)	$\frac{1}{aN}$	$\frac{m}{an_i+(an_i+c)(m-1)}$	$\frac{m}{an_i+(an_i+2c)(m-1)}$	$\frac{m}{an_i+(an_i+2c)(m-1)}$

	モデル 3(l=1)	モデル 4,5 (l=1)
親 CA へのアクセス率 (λ_i)	$\sum_{j=1}^{m} \frac{n_j p(m-1)}{m}$	$\sum_{j=1}^{m} \frac{n_j p(m-1)}{m}$
親 CA でのサービス率 (μ_i)	$\frac{1}{an_i+c}$	$\frac{1}{\alpha m}$

表 1.4: モデル比較のための想定環境

n_i	100	既存の実験 CA 規模
p	0.0001	1 週間で 65 人と通信
c	1	[s]
データ	1000	[Byte/個](証明書相当)
対応表	100	[Byte/エントリ]
a	0.02	データベース全体を 2 秒で処理
α	0.002	(表 1.2参照)

1.3.12 キャッシュに関する考察

モデル 3,4,5 で親 CA の負荷を軽減するためには、子 CA と親 CA 間の通信を軽減させる必要があり、CA にキャッシュ機能を持たせることが考えられる。ここでは、各機能でキャッシュを導入する際の影響について考察する。

A. 証明書の検索機能

証明書の検索機能では、結果として得られる証明書は最新であることが求められる。つまり、証明書を入手した直後に有効性確認を行ったときには、有効と判定される必要がある。

証明書をキャッシュする場合には、常に有効な証明書をキャッシュしつづける必要があるため、例えば CRL を入手してキャッシュから不要な証明書を取り除かなければならない。

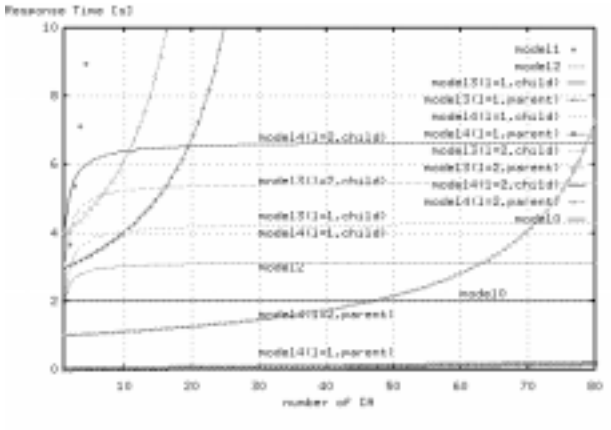

図 1.10: 各モデルの CA 間平均応答時間

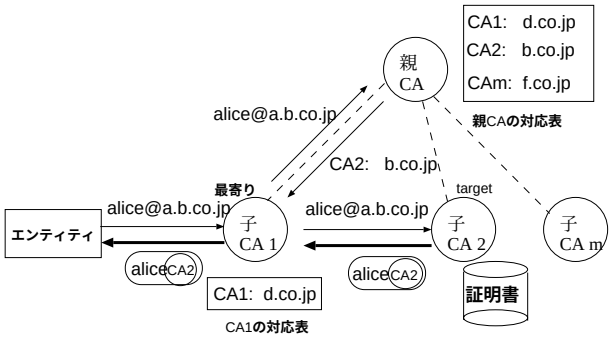

図 1.11: 証明書の検索処理の流れ

これは、キャッシュが利用されなくても必要である。CRL の更新頻度が高い場合、親 CA との通信が頻繁になって結局親 CA の負荷は軽減されない。

モデル 4,5 で CA とアクセス先の対応表をキャッシュする場合には、親 CA と子 CA の通信が減る。用意するキャッシュのサイズは証明書のキャッシュと比較して少量でよく、キャッシュがあっても全体の処理時間への影響は小さいと考えられる。

B. 証明書の有効性確認機能

(1) 応答が CRL である場合

CRL をキャッシュする場合には、CRL 中には次の発行日が含まれるため、データの一貫性を保つのは比較的容易である。有効性確認頻度は、証明書の検索頻度より高いと考えられるため、CRL のキャッシュは効果が高い。CRL の 1 エントリのサイズは証明書と比較すると十分小さいため、キャッシュサイズの増加による処理時間の増加率は比較的低いと予

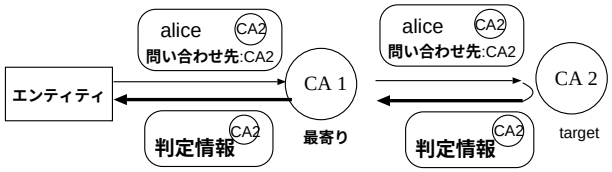

図 1.12: 証明書の有効性確認処理の流れ

想される。

さらに、親 CA が子 CA の CRL をコピーして持つ場合には、証明書の有効性確認時の CA 間の最大通信回数を 1 回分減らし、子 CA の平均待ち時間を下げる効果が期待できる。

(2) 応答が 有効性判定情報である場合

有効性確認情報は、本来は確認対象となる証明書を発行した CA が作成するべきである。

CRL をキャッシュする場合には、有効性判定情報を作成し署名するのはエンティティの最寄りの CA となり、確認対象の証明書の署名と異なるという問題が生じる。証明書情報機構としては、エンティティが有効性確認を行なう場合に、厳密な確認が必要かどうかを指定できるようにすることが考えられる。一般には、有効性確認を頻繁に行なう必要があるアプリケーションほど厳密な確認を必要とすると予想され、その場合 CRL のキャッシュによる効果は期待できない。

有効性判定情報をキャッシュする場合には、有効性判定情報に期限が含まれていれば効果がある。

1.3.13 発行局パッケージ (ICAP) における実装状況

われわれが開発した発行局パッケージ (ICAP) では、エンティティと CA 間は HTTP を利用して通信を行なっており、ICAP 単体での証明書検索機能と有効性確認については一部実装済である。今回の設計をもとにして、現在、単体の場合と同一のインターフェイスで CA 間通信を利用した拡張をおこなっている。

CA 間通信は WWW サーバ上で動作する CGI(プログラム) から HTTP を使ってリモートの CGI を呼び出す方針である。この方法で実際に、NEC EWS4800/320(CPU R4000, メモリ 32MB) を使って、ローカルに CRL(約 800byte) を転送したところ、平均応答時間は 4.9(s)、そのうちの通信時間の平均が 2.2(s) となった。通信に入る前に平均 2.6(s) かかっていることから、CGI の呼び出しに予想以上に時間がかかることがわかった。さらに実験を行ない、HTTP と CGI だけで運用可能かどうか検討する必要がある。

1.3.14　結論

CA が複数になり、エンティティが CA の構造について知らなくても、CA がファイアウォール内で運用される環境であっても、CA 間通信によって証明書情報を共有できるような機構を提案した。方針として、エンティティは 1 回の要求だけで証明書情報の応答を得られることとし、5 つのモデルの検討を行なった。そして、平均応答時間と CA 数のスケーラビリティを評価の対象とし、証明書検索と有効性確認に適したモデルの選択と設計について述べた。さらに、CA 間の通信に制限のある場合や CA でのキャッシュ機能に関する考察を行なった。

現在の CA の運用動向としては、階層構造のある CA よりも単体の CA が複数立ち上がっている状況である。今後、証明書を利用するサービスが増えてくると、CA 間の相互認証を利用して、証明書情報を互いに共有したいという要求が高まると予想される。今回は、CA 全体で 1 つのモデルを選択することを検討したが、いくつかのモデルが混在した状況についても考慮する必要があるだろう。

今後は、今回設計した方式の実装をおこない実用性について検討した後、大規模な環境での利用に適していると言われている証明書の機構の検証を進めていく予定である。

1.4 X.509 認証フレームワークの問題点と解決案

1.4.1 はじめに

大規模ネットワークでのセキュリティ確保には公開鍵暗号技術の利用が必須であり、通信相手の正しい公開鍵を入手することが重要である。その手段の一つとして X.509 認証フレームワーク [?] があり、電子メールや World Wide Web での通信安全性の確保 [?][?] の他、クレジットカード決済の保護 [?] にも使われている。

X.509 認証フレームワークでは認証局 (Certification Authority) という信頼できる第三者が発行する公開鍵証明書を用いて、ユーザの公開鍵を配布する。一旦発行した公開鍵証明書が無効になった場合、廃棄証明書リスト (Certificate Revocation List、CRL) を用いて無効になった証明書をユーザに通知する。しかし、CRL を使った廃棄証明書の通知には、(1) 廃棄証明書リストのサイズが大きくなり通信・処理コストがかかる、(2) 廃棄されてからユーザに通知されるまでの時間的ギャップがある、といった問題がある。

上記の問題点を解決するサービスとして証明書の有効性を確認するサービス [?][?] がある。しかし、これらのサービスは、(3) 否認防止サービスに適用できない、(4) 認証局が階層化された場合にコストがかかる、という問題を残していた。

本稿では、これらの問題を報告した後、証明書有効確認サービス [?] を拡張したサービスについて報告する。

1.4.2 X.509 認証フレームワークの問題

公開鍵暗号アルゴリズムを用いたディジタル署名やデータ暗号化には、通信相手の正しい公開鍵を入手する必要がある。X.509 認証フレームワークでは、認証局という信頼できる第三者が、公開鍵の所有者の認証を行った上で、公開鍵証明書を発行、ユーザに配布することで、正しい公開鍵の入手手段を提供している。公開鍵証明書は、所有者の名前・証明書の有効期間・公開鍵・証明書のシリアル番号などを含んでおり、認証局のディジタル署名が施されている。認証局を信用し、その公開鍵を持っているユーザは、通信相手の公開鍵証明書を入手、ディジタル署名を確認することで、正しい公開鍵を入手することができる。

公開鍵証明書は単なるデータであるが、認証局のディジタル署名で保護されているので、改竄や偽造が検知できる。しかし、秘密鍵の盗難や所有者の所属や名前の変更により、証明書が廃棄になってもユーザにはわからない。このため X.509 認証フレームワークでは、廃棄になった証明書のシリアル番号を含み認証局が署名した CRL を発行することで、ユーザに証明書の廃棄を通知する方式を採用している。しかし、CRL を用いた廃棄の通知には、廃棄情報と否認防止に関わる問題がある。

1.4.3　A. 廃棄情報のサイズと鮮度

まず最初に CRL 自体の問題として次の二点がある。

1. **リストのサイズ**

   廃棄した全ての証明書のシリアル番号は、証明書の有効期間中 CRL 含まれるので、リストが大きくなる。このためリストの生成・通信・検証・蓄積コストが大きくなる。

2. **情報の鮮度**

   一方 CRL は、例えば月一回のように、定期的に発行される [?] ため、廃棄になってからユーザが廃棄を知るまでの時間ギャップがある。このため廃棄になっている公開鍵を使ってディジタル署名を検証したり、データを暗号化することが起こる。

新しい X.509 認証フレームワークでは差分 CRL[?] を導入し、最初の問題を解決しようとしている。差分 CRL は全ての廃棄証明書のシリアル番号を含むのではなく、前回発行した CRL から新たに廃棄した証明書のシリアル番号を含んだ CRL である。差分 CRL は廃棄になった理由別に発行することもできるので、CRL のサイズを小さくすることができる。このためユーザのコンピュータが「安全な」記憶装置を持っている場合は、そこに廃棄証明書の情報を格納することで通信・処理コストを小さくできる。

しかし、安全な記憶装置を持っていない場合には逆にコストが大きくなる。つまり、証明書が廃棄された場合は、その廃棄証明書のシリアル番号を含んだ差分 CRL を入手し、CRL の署名を確認すれば良いので、コストは小さくなるが、逆に廃棄されていない場合には、どの CRL にもその証明書のシリアル番号が含まれていないことを確認しなければならないので、全ての CRL を入手、署名を確認しなければならないからである。差分 CRL と同時に通常の CRL を発行する方法 [?] もあるが、大きな CRL を入手・検証・蓄積するコストがかかる。

証明書有効確認サービス [?] を使用することで、CRL の問題を解決することができる。このサービスは、ユーザからの問い合わせに対して、発行認証局が運営する証明書有効確認サービスサーバが、ある時刻における証明書の有効性 (有効や廃棄済みなど) をディジタル署名付きで返答するものである。本サービスを通じて、文書作成日時における証明書の有効性を確認することで、正しく文書のディジタル署名を検証することができる。しかし、このサービスでは以下の問題点がある。

1.4.4　B. 否認防止サービスの欠如

証明書有効確認サービスを利用することで、ユーザはディジタル署名つきのデータを受け取った時に、そのデータの発信元や完全性を確認することができる。しかし、データを受け取った後に証明書が廃棄された場合には、データの署名の有効性がなくなってしまう。つまり、後日に署名を確認しようとしても、署名確認に必要な公開鍵を含んだ証明書が廃

棄されているからである。鍵の所有者が故意に証明書を廃棄することで、過去のディジタル署名の有効性をなくし、メッセージの発信や内容を否認することができる。

1.4.5 C. 階層的認証局への不適用

認証局が階層を成している場合、証明書有効確認サービスを階層数の回数使用しなければならない。例えば、インターネットの認証局 [?] では、最低でもルート認証局、ポリシ認証局、組織認証局が運営する証明書有効確認サービスサーバに、それぞれ問い合わせを行う必要がある。さらに、ルート認証局が運営する証明書有効確認サービスサーバには、問い合わせが集中することになる。

1.4.6 タイムスタンプと認証パス有効性確認サービスの提案

以上報告した問題を解決するためには、認証局の階層まで考慮に入れた証明書有効確認サービスと、データと署名がある時点で存在したことが確認できれば良い。そこで筆者は、この二つのサービス、認証パス有効性確認サービスとタイムスタンプサービス [?] をユーザに提供することで問題が解決すると考えた。以下本稿では、提案したサービスを説明し、試作したプログラムの性能評価を行い、インターネットような大規模なネットワークで実用可能かどうか検証する。

A. 全体構成

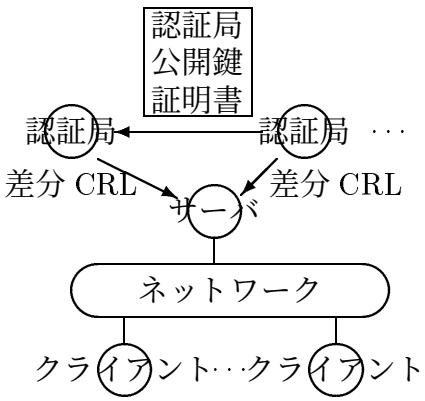

図 1: 全体構成

新サービスの全体構成を図 1 に示す。各認証局は証明書の廃棄依頼が来ると、依頼を確認した後に差分 CRL を使ってサーバに伝える。サーバは差分 CRL の署名を確認した後、ローカルな廃棄証明書データベースに廃棄情報を格納する。廃棄情報の中身は、差分 CRL と同じであり、証明書のシリアル番号、廃棄日時、廃棄理由などである。

サーバは上位の複数の認証局から差分 CRL を受けとる。つまり、ユーザの公開鍵証明書発行認証局、その認証局に対して証明書を発行した認証局、さらに上位の認証局と、ルートの認証局までの CRL を受けとり、認証パス [?] に含まれる全ての公開鍵証明書の有効性確認に必要な情報を収集する。

ユーザのクライアントコンピュータは、ディジタル署名の付いた文書やメッセージのメッセージダイジェストと署名確認に必要な公開鍵証明書の識別情報をサーバに送る。識別情報は、証明書発行認証局の名前と証明書のシリアル番号である。

クライアントからの要求に応じて、サーバは証明書の最新状態、つまり、廃棄されている/いない等の結果とともに、メッセージダイジェストと現在時刻などの返答データに署名をつけて返す。

通信プロトコルには TCP/IP を用い、要求・返答データは ASN.1 型を BER エンコードして交換する。以下このデータについて報告する。

なお本サービスは、サーバとクライアント間でやりとりされるデータの構成において既存の証明書有効確認サービス [?] と似ているが、認証局とは別のオーソリティによって運営され、複数の認証局から差分 CRL を受けとるという点で根本的に異なっている。

1.4.7　B. 要求データ構成

```
Request ::= OPTIONALLY SIGNED SET {
  version   [0] INTEGER default 0,
            SEQUENCE OF SEQUENCE {
    issuer       [1] Name,
    serialNumber [2] INTEGER } OPTIONAL,
  requestVerificationTime
        [3] UTCTime OPTIONAL,
  dataToBeTimeStamped
        [4] OCTET STRING OPTIONAL,
  messageDigestAlgorithm
        [5] AlgorithmIdentifier OPTIONAL,
  requestOriginator
        [6] Name OPTIONAL,
  requestIdentifier
        [7] OCTET STRING OPTIONAL }
```

各フィールドの意味は次の通り。

1. version: 要求データのバージョン番号。
2. issuer、serialNumber: 認証パスに含まれる公開鍵証明書を識別する。issuer は証明書の発行認証局、serialNumber はシリアル番号。

3. requestVerificationTime: ユーザが要求する公開鍵証明書の有効性を確認する日時。フィールド値がない場合は、最新の CRL 情報に基づいて証明書の有効性を確認する。このフィールドはユーザがある特定時刻、例えば文書作成者が署名した時刻[6]、の証明書の有効性を確認したい場合などに使用する。

4. dataToBeTimeStamped: タイムスタンプするデータのメッセージダイジェスト。

5. messageDigestAlgorithm: データのメッセージダイジェストを生成するのに使用した、メッセージダイジェスト (一方向性) 関数の識別子と付加的に使用したパラメータ。

6. requestOriginator: サービスの要求者を示す。サービスを受ける資格があるかどうかの判断や課金のために使用する。

7. requestIdentifier: ユーザが割り振る要求データの識別子。

C. 返答データ構成

```
Reply ::= SEQUENCE { SIGNED SET {
  version   [0] INTEGER default 0,
            SEQUENCE OF SEQUENCE {
    issuer       [1] Name,
    serialNumber [2] INTEGER } OPTIONAL,
  verificationResult
        [3] VerificationResult,
  invalidCertificate
            SEQUENCE {
    issuer        [1] Name,
    serialNumber  [2] INTEGER } OPTIONAL,
  revokedReason
        [4] RevokedReason OPTIONAL,
  revokedOrHoldTime
        [5] UTCTime OPTIONAL,
  holdExpirationTime
        [6] UTCTime OPTIONAL,
  holdInstructionCode
        [7] OBJECT IDENTIFIER OPTIONAL,
  invalidityTime
        [8] UTCTime OPTIONAL,
```

[6]鍵の使用がサスペンドされ、証明書が一時的に無効になる場合がある。署名確認時では証明書が有効であっても署名時に鍵使用がサスペンドされていた場合はその署名は無効と考えるべきある。

```
  requestedTime
        [9] UTCTime,
  verificationTime
        [10] UTCTime OPTIONAL,
  dataToBeTimeStamped
        [11] OCTET STRING OPTIONAL,
  messageDigestAlgorithm
        [12] AlgorithmIdentifier OPTIONAL,
  requestIdentifier
        [13] OCTET STRING OPTIONAL,
  replyIdentifier
        [14] OCTET STRING OPTIONAL },
        CertificationPath OPTIONAL }

VerificationResult ::= ENUMERATE {
  notRevoked (0), revoked (1), hold (2) }

RevokedReason ::= ENUMERATE {
  unspecified (0),  keyCompromise (1),
  caCompromise (2), affiliationChanged (3),
  superseded (4),   cessationOfOperation (5) }
```

各フィールドの意味は次の通り。

1. version: 返答データのバージョン番号。
2. issuer、serialNumber: 有効性を確認した証明書を識別する。
3. verificationResult: 証明書確認の結果。
   notRevokedは全ての証明書が未廃棄であることを示す。ユーザは証明書の署名と有効期間を確認した上で公開鍵証明書を使えば問題ないことを示す。holdは invalidCertificate に示す証明書が、未確認の廃棄要求があったり秘密鍵が盗まれた可能性があるとの理由で、一時的に使用中止であることを示す。holdExpirationDate に使用中止終了予定日時を、holdInstructionCode に証明書の取り扱い方法を示す。
4. invalidCertificate: 認証パスに含まれる証明書の内、どの証明書が無効になったかを示す。
5. revokedReason: 無効になった理由。
6. revokedOrHoldTime: 廃棄、または一時使用中止となった日時。

7. invalidityTime: 当該証明書が廃棄になった日時。
8. requestedTime: サービス要求を受け付けた日時。
9. verificationTime: 証明書の有効性を確認した日時。
10. dataToBeTimeStamped、
    messageDigestAlgorithm、
    requestIdentifier: 要求データにあったものと同じ。
11. replyIdentifier: サーバが割り振る返答データの識別子。
12. CertificationPath: 返答の署名確認に必要な公開鍵証明書の認証パス。

1.4.8 性能評価

サービスの規模性を評価するため、表 1 の環境でサーバとクライアントを実装、性能を評価した。

表 1: 評価環境

1	CPU	SuperSPARC-II/75
2	メモリ	320MB
3	OS	SunOS5.4
4	開発ツール	ISODE-8.0 + OSISEC-1.0 + SSLeay-0.6.4
5	RSA 鍵長	1024 bits
6	データ数	10,000
7	DB	OS 附属の NDBM

ISODE[?] は OSI アプリケーションを開発するための環境で ASN.1 ツールを含んでいる。OSISEC[?] は ISODE を利用した OSI セキュリティアプリケーション開発環境であり、ディジタル署名の生成・検証や公開鍵暗号・復号のためのモジュールを含んでいる。OSISEC は鍵長 512 ビットの RSA をサポートしているが、サービスの性質からみて 512 ビットは短いと判断、RSA 暗号は SSLeay[?] の暗号モジュールを利用した。

サーバがクライアントからの一件の要求を処理するのに 0.17 秒かかった。処理の 8 割以上は署名の生成であり、他はデータベース処理、メッセージの生成・エンコードなどである。

サービスの規模性を求めるために、M/M/1 待ち行列モデルを使って、どの程度の数の電子メールユーザまで一つのサーバで賄えるか計算する。ここではサーバの処理能力に注目し、ネットワーク上の転送時間は無視する。平均サービス率 (μ) は $1/0.17 = 5.9$ である。

サーバへの要求の平均到着率 (λ) を求めるために、ある組織とその組織内の一部署でのある月の電子メールの受信数を調査した。

1. 部署への到着メール

   平成 8 年のある月には 93 人のユーザが 22,020 通の電子メールを受信した。その月の営業日数は 20 日であり、一日の営業時間は 8 時間であったので、$\lambda = 22020/(60 \times 60 \times 8 \times 20) = 0.0382$ と計算できる。

2. 組織への到着メール

   平成 8 年ある月には 2,200 名の電子メールユーザがおり、組織外から受信した総数は約 245,000 通だったので、$\lambda = 245000/(60 \times 60 \times 8 \times 20) = 0.425$ と計算できる。

$T = 1/(\mu - \lambda)$ より平均応答時間を計算すると、結果は表 2 となる。

表 2: 平均応答時間

#	グループ	人数	λ	T(秒)
1	部署	93	0.0382	0.17
2	組織	2,200	0.425	0.18

この数字をもとに、平均応答時間が 3 秒、つまり平均到着率λが 5.6 となる人数を求める。人数と平均到着率が比例するとすると、人数は表 3 の通りになる。

表 3: 平均応答時間が 3 秒となる人数

#	グループ	人　数
1	部署	14,000
2	組織	29,000

1.4.9　結果の検討

A. 規模性の検討

前章の計算によると、一つのサーバで数万人程度の電子メールユーザにサービスを提供することができる。これはインターネットの規模を考慮すると、性能不足と思われる。しかし、全てのメッセージがサーバを使ったタイムスタンプや証明書の確認が必要とは思えない。実際、電子メールの一割に必要とすると数十万人規模のユーザに一つのサーバで対応できる。

インターネットのような大規模なネットワークになった場合、複数のサーバを用意してユーザにサービスを提供することは避けられない。この場合、ユーザからの要求をどのようにして複数のサーバに振り分けるかが問題となる。一つの解決方法として、Domain

Name System (DNS) サーバとの組合せがある。DNS サーバは DNS クライアントからのドメイン名を含んだ要求に対して、その IP アドレスを返答する。返答する IP アドレスは一つのドメイン名に対して一つとは限らず、複数の IP アドレスを順番に返すことが可能である。サーバのドメイン名を証明書中の拡張フィールド [?] を使って記述し、本 DNS サーバの機能を使って、ユーザクライアントのアクセス先を複数のサーバに分散することができる。DNS サーバは、クライアントとサーバの距離やサーバの負荷を考慮して IP アドレスを返答するのではないので、要求先にばらつきがでるのは避けられないが、有効な方法と考えられる。

1.4.10 B. 第三者の攻撃に対する安全性

認証局からサーバへの CRL とサーバからクライアントに対する返答は、それぞれ認証局とサーバの署名で保護されているので、なりすまし・改竄などの攻撃を防ぐことができる。

C. 認証局の不正行為に対する安全性

不正行為の一つに、CRL や公開鍵証明書の署名の否認がある得るが、認証局の性格からして可能性は薄い。

1.4.11 D. サーバの不正行為に対する安全性

提案サーバの不正行為の一つに、タイムスタンプのバックデートがある。対策方法として返答中にリンク情報 [?] を含める方式がある。これはタイムスタンプ対象データと時刻情報と一つ前の返答データ中のリンク情報とのメッセージダイジェストであり、結果的にサーバが発行した全ての返答データ中のタイムスタンプ対象データと時刻情報のメッセージダイジェストとなる。このため、バックデートしてタイムスタンプした場合、それ以降のリンク情報を参照することで、バックデートを検知できる。

1.4.12 結論

本節は、X.509 認証フレームワークにおける公開鍵証明書の廃棄情報に関わる問題と否認防止サービスの欠如を述べ、これらを解決する手段として認証パス有効性確認サービスとタイムスタンプサービスの組合せサービスを提案した。サービスの構成、サービス要求・返答データの報告の後、試作したサーバの性能評価を行い、一台のサーバで数万人規模の電子メールユーザにサービスできることを示した。

否認防止サービスは、本稿で述べたタイムスタンプサービスだけでなく、メッセージの発信確認、受信確認にも必要である。情報化社会の基盤であるネットワークのセキュリティ確保のため、これらのサービスを含めた安全で信頼できる技術や仕組みが必要である。

1.5　おわりに

本章では，証明書発行方式，配布方式，検証方式について考察した．Web に基づく証明書発行方式の有効性を明らかにし，管理者が分散しているインターネットにおいて実用的な証明書配布機構を設計した．従来の証明書検証方式における本質的な問題点を指摘して，それを解決するオンライン証明書検証サービスを実装して，評価を行った．

証明書に基づく情報基盤は，提案されて研究者のコンセンサスが得られてから数年を費やしていまだ実用レベルに至っていない．その間，インターネットを取り巻く技術にはいくつもの大きなブレークスルーがあったというにもかかわらず．一つには，本稿で議論した分散環境における認証や検証の技術的困難さがあるだろう．輸出規制や特許に関する技術以外の障害も大きかった．しかしながら，最も大きな原因は，実用的な認証基盤が実現されていないということから逆説的に導かれる，それを本当に必要とするアプリケーションが一般化していないこと，ではないだろうか．本 WG での活動が，この現状を少しでも改善し，より実用的なインターネット環境に貢献することを期待したい．