電子政府情報セキュリティ相互運用支援技術の開発
(PKI 相互運用フレームワークの開発)

GPKI アプリケーション実装ガイド報告書

平成 15 年 3 月

情報処理振興事業協会

セキュリティセンター

― 目 次 ―

1 はじめに

1.1 はじめに

PKI 相互運用の促進は、PKI が真の認証基盤となるための最も重要な課題のひとつだと考えられる。特に、政府認証基盤 GPKI のような広い認証ドメインにおいて、色々なアプリケーションを動作させる必要がある PKI にとっては、非常に重要な課題である。

GPKI を認証基盤とした電子政府の成功の鍵のひとつに、使いやすくセキュアな電子政府対応アプリケーションが流通するとことが考えられる。相互運用性が確保された PKI/GPKI 対応電子政府アプリケーションが多数開発され、また、COTS (Commercial Off-The-Shelf) 製品の流通も必要である。

しかし、いざ GPKI Ready なアプリケーションを開発しようとすると、現状では色々な困難に遭遇する。参考になる実装がない、テスト環境、開発環境がない、テストの方法が分からない等の GPKI の相互運用技術に由来する困難な課題に遭遇する。

GPKI の相互運用性の課題の解決は、GPKI に対応した電子政府アプリケーションの開発を促進することになり、結果として、GPKI を基盤とした広範囲なセキュリティを適正なコストで実現することにつながる。

本報告書は、これら課題に取り組むべく、情報処理振興事業協会（IPA）が「情報セキュリティ関連の調査・開発」のテーマのひとつである「電子政府情報セキュリティ相互運用支援技術の開発」として公募した成果物である。公募の採択は、NPO 日本ネットワークセキュリティ協会（JNSA）の提案が採択され本報告書をまとめたものである。

1.2 報告書及びプロジェクトの目的

報告書の作成以外に、同じく委託された、GPKI テストケース設計書の作成、PKI 相互運用テストスイートの開発、GPKI アプリケーションサンプル実装に開発などと合わせて Challenge PKI 2002 プロジェクトと称している。

昨年度の、同じく情報処理振興事業協会（IPA）の委託を受け実施したプロジェクトである Challenge PKI 2001 では、複数のベンダーの CA 製品を集めマルチベンダーの PKI 相互運用実験を行った。この実験は、ブリッジモデルも対象にした、マルチベンダーPKI、マルチドメイン PKI の相互運用性を確認すものであった。Challenge PKI2001 は、CA 中心の実験であったが、ブリッジモデルなどでは、CA

が発行する証明書の解釈を行う PKI アプリケーションの相互運用性を確保することの技術的な難しさなどが確認された。

政府認証基盤 GPKI が採用しているブリッジモデル（または、ハイブリッドモデル）で要求される多くの仕様は、基本的に、PKI の標準である ITU の X.509、IETF の RFC3280 などに含まれている。従って GPKI で要求されるアプリケーションは、これらの標準が実装されていれば動作するということになる。しかし、X.509 や RFC3280 は、非常に汎用的な仕様であり、これらを全て実装することは非常に困難だという事に注意しなければならない。

GPKI に限らず、広く相互運用性を確保する必要がある PKI のシステムは、X.509、RFC3280 などの標準からさらに絞り込んだプロファイルを使用することにより、その枠組みの中での相互運用性を達成している。GPKI では、政府認証基盤相互運用性仕様書（GPKI 相互運用性仕様書）などで相互運用性を確保するための仕様を示している。GPKI は、マルチベンダーPKI、マルチドメイン PKI といった拡張性がある構成となっているが、その反面、高度な相互運用技術が要求される。特に、ブリッジ認証局と新たに相互認証するドメインの認証局が発行する証明書の検証が一番難しい。この証明書の検証では、一般に認証パス構築、パス検証といったことを行う。このパス構築、パス検証の仕様は、GPKI のアプリケーション共通の要求となる。

これらの GPKI のアプリケーションへの要求は、GPKI 相互運用性仕様書に記述されているわけであるが、この仕様があるだけでは、実際の開発は難しいものがある。GPKI のブリッジ認証局は、今後、地方公共団体における組織認証基盤(LGPKI)、公的個人認証基盤との相互認証が予定されている。また、更に、海外との相互認証も行われるかもしれない。

この様に、証明書検証を行う署名検証者(リライングパーティ)にとっては、検証するドメインが広がっていくことになるが、これがブリッジモデルの利点でもあり、技術的な課題ともなる。

リライングパーティは、ブリッジ認証局と相互認証した先のドメインに対する証明書検証が要求されることになる。この時、広いドメインで色々な証明書が混在して発行される中、リライグパーティは、目的にあった証明書のパスを構築し、そして、パス検証ができる必要がある。このパス構築、パス検証を行う GPKI に準拠したアプリケーションの実装は簡単ではない。また、LGPKI、公的個人認証基盤などを取り込んだ、より広範囲なドメインでの相互運用性を確保したアプリケーションの開発は更に難しいと思われる。

こうした中で、GPKI 相互運用性仕様書に準拠したアプリケーションの開発を促進するためには、仕様以外に、以下のものがあるべきだと考えられる。

- GPKI の仕様の根拠となる標準の説明

- GPKI の開発環境
- GPKI のテスト環境
- GPKI のテストケース
- 参照となる実装

Challenge PKI 2002 プロジェクトでは、正に、これらを提供しようとしている。更に、GPKI だけでなく、LGPKI、公的個人認証基盤、更に海外との相互認証なども視野に入れている。

本報告書では、GPKI に準拠したアプリケーションの共通の要求事項である、パス構築、及び、パス検証の標準、パス検証に関する標準の動向、マイクロソフトの暗号フレームワークである Microsoft CryptoAPI を使用した実装、Java 環境での実装、テストの方法などについて記述する。そして、Challenge PKI 2002 プロジェクトの他の成果物である、相互運用テストスイート、GPKI テストケース設計書、Microsoft CryptoAPI と Java によるサンプル実装などを使って説明する。

図 1-1 に、Challenge PKI 2002 のプロジェクトの範囲、及び、本報告書の範囲を示す。

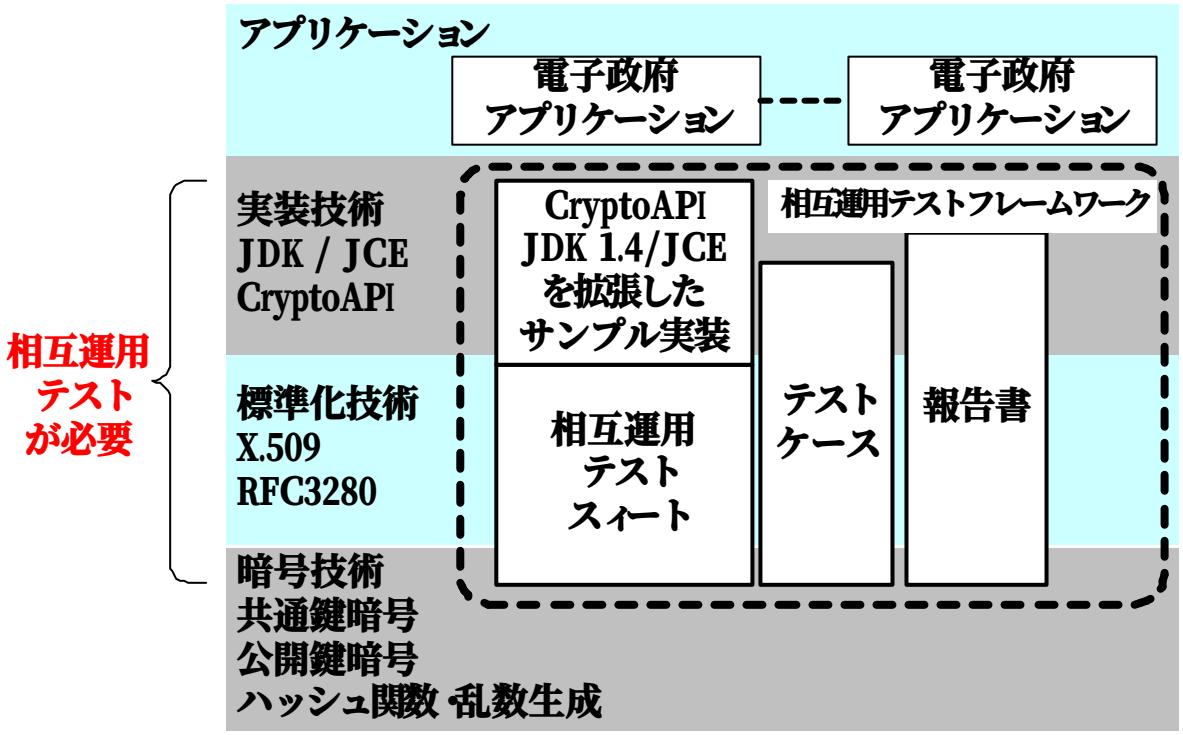


図 1-1 Challenge PKI 2002 の範囲

電子政府に関しての暗号技術の議論や成果物の作成、また、電子政府のアプリケーションの実証実験などは、これまでにも行われてきている。しかし、このChallenge PKI 2002の範囲に関する議論や成果は極めて少ない。本報告書では、この空白を埋めることが大きな目的となる。

1.3 パス構築・パス検証の標準

GPKIのようなブリッジモデル（またはハイブリッドモデル）では、柔軟な認証ドメインの拡張を許す反面、そのアプリケーションの高度な相互運用技術が要求される。特にリライングパーティ側のパス構築、パス検証に関しては、これらの実装、及び、テストが難しい。

GPKIで発行される証明書のプロファイルは、GPKI相互運用性仕様書に記述されている。このGPKIの証明書プロファイルは、X.509、RFC3280などのサブセットだと考えられる。以下にX.509、RFC3280などとGPKIの証明書プロファイルの関係を示す。

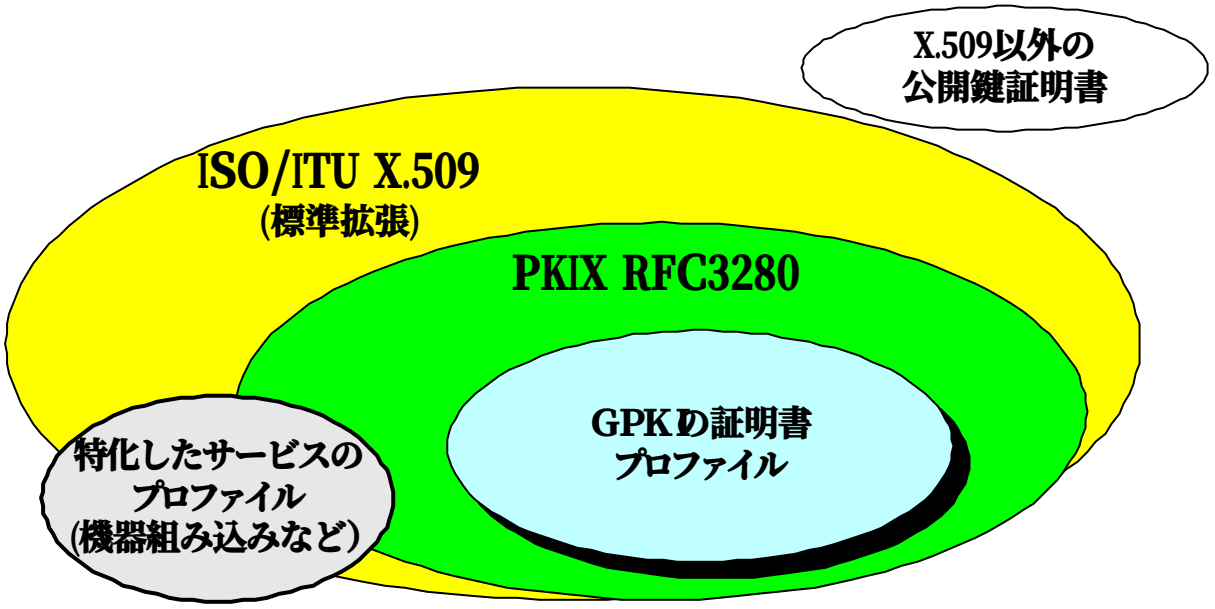


図 1-2 X.509、RFC3280およびGPKI証明書プロファイルの関係

GPKIのアプリケーションは、このGPKIの証明書プロファイルの解釈が要求されるが、この実装は、RFC3280などで要求される実装のサブセットと言うことになる。しかし、サブセットといっても、RFC3280に記述された仕様のかなりの部

分の実装を要求している。そして、その実装の要求の多くは、X.509 証明書 v3 フォーマットの証明書と CRLv2 フォーマットの失効リストの処理である。

X.509 証明書 v3 フォーマットと CRLv2 フォーマットは、1997 年度版の X.509 3rd Edition で定義されたフォーマットであるが、このフォーマットでは X.509v3 拡張、及び、CRLv2 拡張と呼ばれる拡張領域を持つ。そして、この拡張領域に関して、それぞれ標準拡張と呼ばれるものが定義されている。

GPKI のアプリケーションの実装者は、この標準拡張の理解が要求されることになる。しかし、GPKI 相互運用性仕様書が参照している X.509 3rd Edition と現在の最新版である X.509 4th Edition、そして RFC2459 と現在の最新版である RFC3280 などは、かなりの量のドキュメントであり、これらを正確に把握することはかなりの技術力を要する。

本報告書では、2 章の「認証パス構築・パス検証の概要」で、これらの仕様の説明を記述する。

1.4 証明書検証サーバモデル

複雑な証明書検証の処理が要求されるブリッジモデルでは、証明書の検証をサーバに依頼するようなモデルが考えられている。これは、単にパス構築、パス検証などの処理が複雑で難しいという理由だけでなく、検証するドメインが広がることによる仕様の変更などにより、クライアントのソフトウェアを入れ替えるといった影響を最小限にするといった理由もある。実際、GPKI においても、こうした理由から GPKI 証明書検証サーバが用意されている。

しかし、現状の GPKI の証明書検証サーバはいくつかの問題がある。

(1) GPKI で採用されている証明書検証サーバは、府省側にしか用意されておらず、電子政府全体で使用できるものではない。

(2) GPKI で採用されている証明書検証サーバに対するアクセスプロトコルは、標準化されたものではない

これらには、それぞれ理由がある。(1)に関して、証明書検証サーバの応答メッセージ自体は、証明書検証サーバにより署名される。この証明書検証サーバの署名を信頼するのは同一ドメインのユーザである必要がある。つまり証明書検証サーバを民間側で使うには、民間認証局なりが、証明書検証サーバを用意することが必要になる。

(2)に関しては、そもそも、現時点では標準自体が存在しないということがある。

標準が存在しないことが、民間側も含めGPKI全体（電子政府全体）で、このような証明書検証サーバを使ったモデルが採用されにくい状況を作っている。

関連する問題に、GPKI証明書検証サーバのプロトコルが独自仕様であるため、証明書検証サーバを使ったアプリケーションの開発を困難にしている。ただし、GPKI証明書検証サーバは、府省ドメインの中で使用されているため、例えば海外との相互認証時など、海外からの証明書検証などに影響を与えるものではない。

IETFでは、GPKIの証明書検証サーバのような目的のためのプロトコルの標準化を進めている。IETF以外でも、W3Cでは、XMLのフレームワークでの鍵管理であるXKMSのX-KISS（XML Key Information Service Specification）といった、似たような目的に仕様が提案されている。

3章では、GPKIの証明書検証サーバ自体の説明や、これらの標準化の動向を中心に説明する。

1.5 テスト環境の重要さ

GPKIは、複数の異なる主体者のPKIドメイン(認証ドメイン)から構成されており、それぞれPKIドメインは、それぞれのCP/CPSなどに従って証明書が発行される。複数のPKIドメインを持つPKI、すなわちマルチドメインPKIは、PKIドメイン（認証ドメイン）を拡張していくことができる。しかし、その反面、マルチドメインPKIは、そのテスト環境を整えることが難しいという問題がある。

GPKIの場合、マルチドメインPKIというだけでなく、複数のベンダーのPKIコンポーネント、複数のアークテチャが混在したハイブリッド環境となっており、このテスト環境を構築することは容易ではない。全く同じ環境を整えるとすると、各ベンダーの製品を取り揃えることになってしまう。

マルチベンダーPKI、マルチドメインPKIの相互運用性実験であるChallenge PKI 2001では、多くのCA製品が参加し、一定の成果を得ることができた。しかし、テスト環境の構築に多くの工数を消費したにも関わらず、実験終了後は、テスト環境を取り壊した。

以上のことから、Challenge PKI 2002では、汎用的な複数の任意の仕様の認証局を模擬することができる相互運用テストスイートの開発を行った。相互運用テストスイートは、GPKIで要求される任意のプロファイルの証明書、CRL、OCSPレスポンダメッセージを生成することができ、GPKIのブリッジ認証局、既存の府省認証局、民間認証局、商業登記認証局なども模擬することができる。

図 1-3 に、相互運用テストスイートの構成を示す。

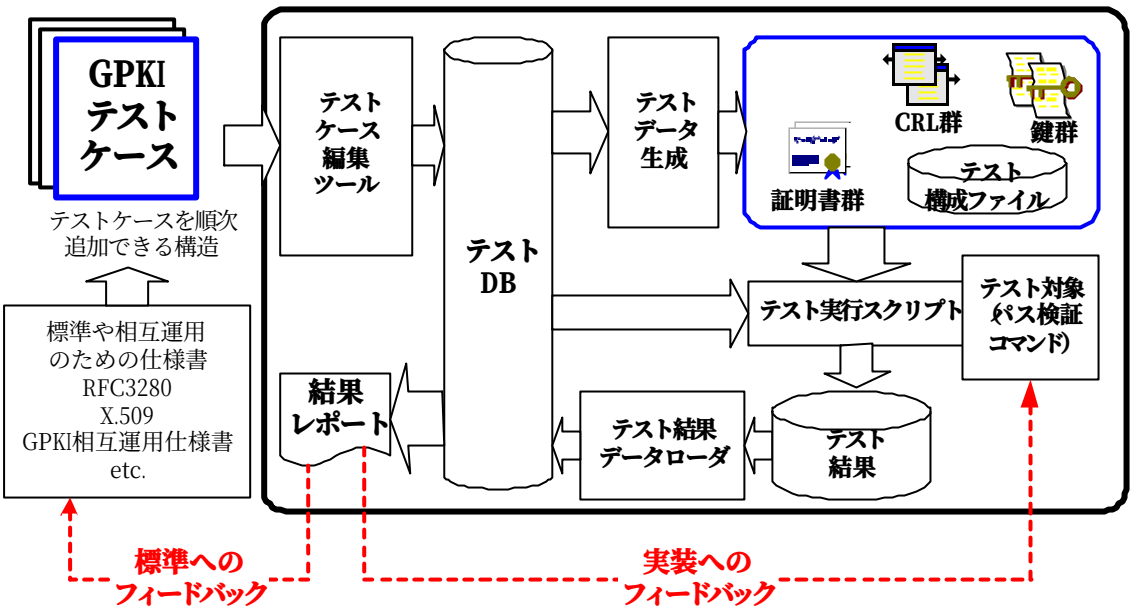


図 1-3 相互運用テストスイートの構成

7 章で開発した相互運用テストスイートの説明を行う。

1.6 テストケースやテストクライテリアの重要さ

PKI を使ったシステムのセキュリティを論じるとき、その暗号強度や、認証局の運用、監査の重要さといったことが、よく指摘される。その反面、例えば、電子政府アプリケーションが、GPKI の仕様に準拠して動作するかといったことは深く議論されていないように思われる。当然のことながら、実際の電子政府アプリケーションが GPKI の要求する仕様を満足しなければ、セキュリティを確保できているとは言いがたい。特に、GPKI のようなマルチドメイン PKI では、パス検証における各種の制約拡張の処理が重要になる。

マルチドメイン PKI では、広いドメインで色々な証明書が発行される中、パス検証で、各種の制約拡張の処理が行われる。この制約拡張は、X.509v3 証明書拡張で設定されるが、この制約拡張の処理により、目的にあった認証パスの処理が行われることが重要になる。この処理に問題があると、例えば、保障レベルが高い証明書のみを受け入れなければならない場面で、保障レベルが低い証明書を受け入れてしまうといったことが生じる。

これらの GPKI の仕様に対する準拠性をテストを行う場合、汎用的に使用できるテストケースがあれば、テストが容易になる。しかし、テストケースの設計は、非常に難しい。テストケースの設計を行うには、２章で説明する多くの標準や、GPKI

相互運用性仕様書などを熟知する必要がある。また、標準や仕様の理解だけでなく、テスト環境の問題がある。すなわち、テストを実行するテスト環境自体の構築が非常に面倒なことである。

マルチベンダーPKI の相互運用実験である Challenge PKI 2001 でも、このプロジェクトで構築したテスト環境でのテストケースを、わずかではあるが設計した。しかし、このテストケースも、Challenge PKI 2001 のテスト環境に依存したものであり、このテスト環境自体も、実験終了後、取り壊した。そのため、Challenge PKI 2001 で設計したテストケースでの再現テストも容易ではない。

Challenge PKI 2002 では、容易にテスト環境を構築できる相互運用テストスイートを開発することで上記の問題の解決をはかった。その上で、再利用できるテストケースの設計を行った。テストケースの設計や、テストクライテリアの作成は、海外でも盛んに行われており、Challenge PKI 2002 のテストケースの設計でも、これらを参考にした。

4 章では、海外での PKI 相互運用テストクライテリアの事例を説明する。また、8 章では、Challenge PKI 2002 で設計したテストケースの概要を説明する。テストケースの詳細自体は、別冊の GPKI テストケース設計書に記述される。

1.7 パス構築、パス検証の実装

実装が伴わない、標準や仕様は、絵に描いた餅のような側面を持つ。過去の IETF などで開発されたプロトコルは、簡易なプロトコルの仕様がドラフトとして提案されると同時に実装も行われることで、その有効性が確かめられてきた。そして、更に精度の高い標準へ持ち上げられるという過程を経てきた。しかし、今日の高度なセキュリティへの要求は、誰もが実装できるというレベルを超えている。

Challenge PKI 2002 では、既存のコンポーネントを利用し、パス構築、パス検証のサンプル実装方法を示している。こうした実装を示した上で、報告書では、GPKI において要求される仕様の実装の説明を行う。実装を示すことにより標準や仕様自体を、より深く理解することができると思われる。

RFC3280 の以前のバージョンである RFC2459 に準拠した多くのパス検証の実装は、存在する。しかし、これらは全て RFC2459 のサブセットの実装に過ぎない。Windows-2000 までの CryptoAPI、Windows-XP の CryptoAPI、JDK 1.4 から加わった Certificate Path Library と GPKI で要求される主な仕様の関係を以下に示す。

表 1-1 実装の種類と提供する機能の関係

	Microsoft CryptoAPI Win-2000	Micosoft CryptoAPI Win-XP	JDK1.4 Cert. Path lib.	サンプル実装	GPKIの要求 (パス構築、パス検証)
基本制約拡張	○	○	○	○	必須
ポリシ制約拡張	×	○	○	○	必須
ポリシマッピング拡張	×	○	○	○	必須
名前制約拡張	×	○	○	○	必須
AIA 拡張 / OCSP	×	×	×	○	必須(官側のみ)
動的パス構築	×	△	○	○	必須
CRL IDP *1	×	○	×	○	必須

これらの Java/JDK の実装、Microsoft CryptoAPI の実装は、それぞれセキュリティフレームワークにおけるプロバイダといった形で、パス構築、パス検証が実装されている。そして、GPKI の対応は、これらのプロバイダを GPKI 対応のプロバイダに置き換えることにより行う。このことにより、PKI アプリケーションの GPKI の対応を、アプリケーション自体の変更を行うことなく、実装することができる。

Challenge PKI 2002 では、この様に GPKI に対応したサンプル実装を GPKI 対応したプロバイダに置き換えるといった手法で実装した。

置き換えた GPKI 対応プロバイダは、GPKI 相互運用性仕様書で要求される、X.509 証明書 v3 拡張、同じく CRLv2 拡張、そして、OCSP による証明書失効検証を実装した。

Java の JDK 1.4 では、PKIX(すなわち RFC3280)に対応したパス検証ライブラリのレファレンス実装が提供されている。図 1-4 に、パス検証ライブラリと、GPKI に対応したプロバイダの構成を示す。

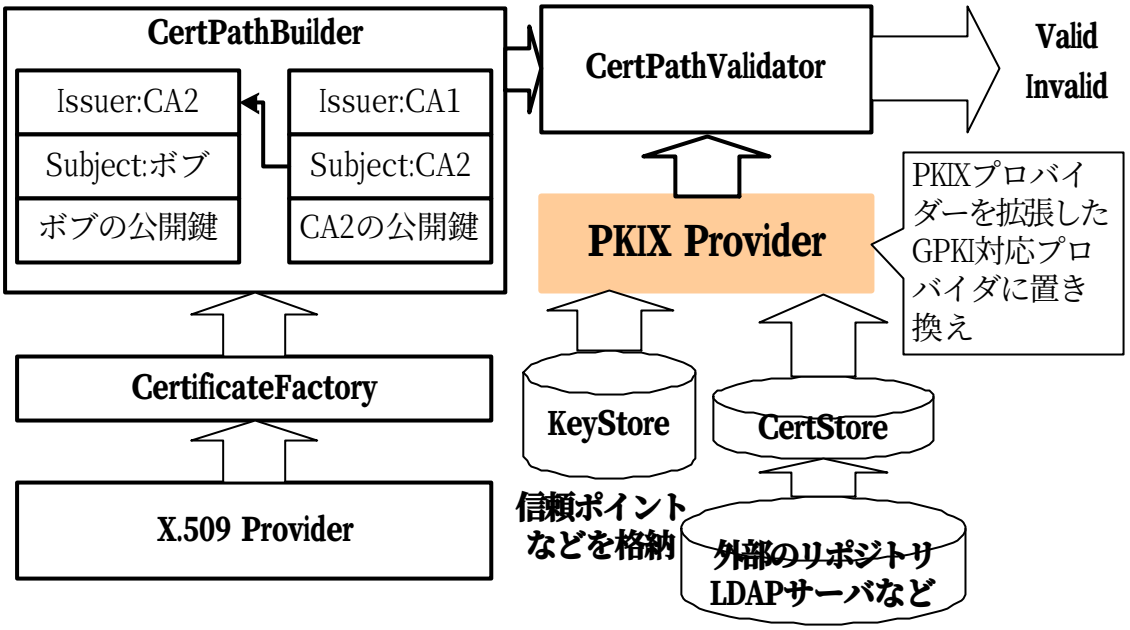


図 1-4 Java パス検証ライブラリ

Microsoft の CryptoAPI では、PKI を利用するアプリケーションに依存しない形で暗号モジュールの入れ替えができるようになっている。例えば、IC カードなどのハードウェアトークンを使用する場合、その IC カードに依存したモジュールを、CSP(Cryptographic Service Providers)として組み入れるといったことができる。同じ様に、証明書検証に関しては、Revocation Providers なるものを置き換えることができ、サンプル実装でもこの方法を取った。

図 1-5 に CryptoAPI での実装を示す。

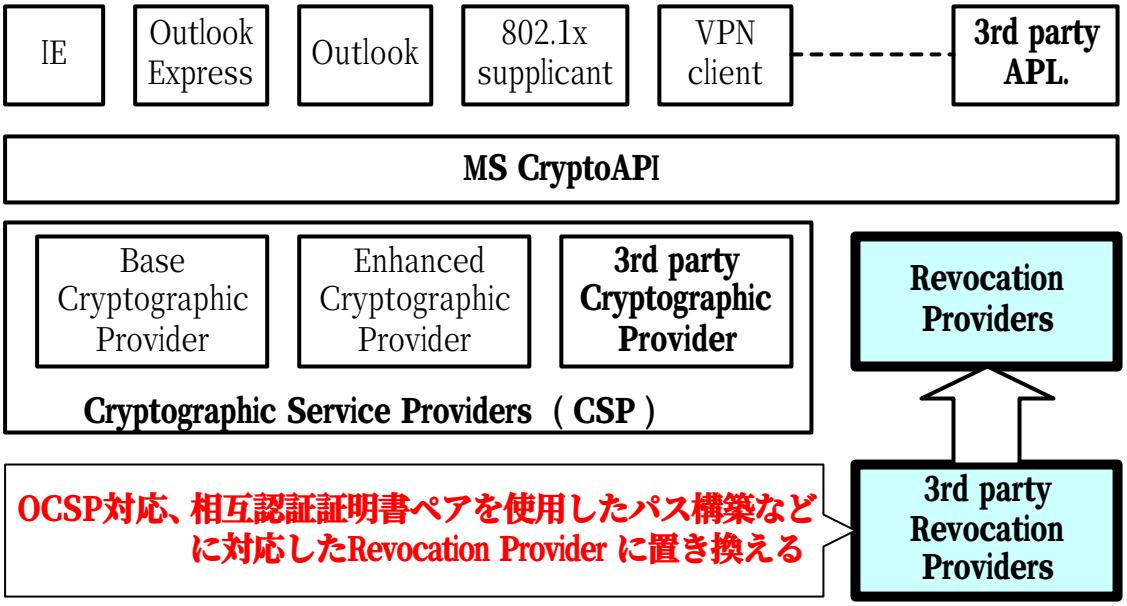


図 1-5 CryptoAPI による実装

5 章では、現状の Java で標準的に提供されるパス検証ライブラリや、java でのサンプル実装について説明する。

6 章では、現状の MicorsofのCryptoAPI、そしてサンプル実装について説明する。

1.8　その他の目標とまとめ

マルチドメインPKIの問題点	解説	Challenge PKI 2002 の目標
標準の曖昧さ	マルチドメイン PKI などの場合、新しい標準を参照しており、その曖昧さが問題になる。	実装、実験を通じ曖昧な部分を明確にしてIETF/PKIX などにフィードバックする。54thIETF 横浜、55th アトランタ、56th サンフランシスコなどでの発表
テスト・クライテリア	マルチドメイン PKI に対応した実装があってもテストケースが少なく標準への準拠性が分からない	主に GPKI に対応したテストケースを設計し提供する。また、テストケースを容易に拡張できるものを提供し GPKI 以外でもテストを可能にする。
テスト環境	マルチドメイン PKI のテスト環境を構築することは非常に困難。	マルチドメイン PKI をターゲットにした PKI 相互運用テストスイートを開発し提供する。1台のLinuxマシンで多くのCA環境をシミュレート。
レファレンス実装	ブリッジモデル(GPKIなど)で必要とされる証明書検証の実装が分かりづらい	Microsoft のプラットホームと Java の環境でレファレンスとなるサンプル実装を提供する。また、テストを行いその結果を報告書に記述する。
分かり易い解説書	標準、テスト、実装、そして将来の方向性の関係が分かりづらい。	X.509、RFC3280 などの標準、色々なテストケース、実装を解説した報告書を作成する。
共通のテストプラットホーム	マルチドメイン PKI のテスト環境の難しさは日本に限ったことではない。世界で共通で使用できるテストプラットホームが必要。	テストスイートなどを海外へも配布できることを検討。また、IETF などでもテストの標準化などを提言し、テスト DB の Informational RFC 化なども検討。

2 認証パス構築・パス検証の標準や仕様の説明

2.1　認証パス構築・パス検証の概要

証明書はある公開鍵をその用途に応じて内容を決定し、認証局（CA）が署名し発行するものである。使用例としては以下がある。

1．　署名者があるデータに署名する際に自身の秘密鍵で署名
2．　秘密鍵と対になる公開鍵に対して発行された証明書を添付する。
3．　署名データ転送
4．　署名検証者は署名されたデータを検証する際、添付されている証明書内の公開鍵を使って署名検証を行う。署名が正しければ、添付されていた証明書を検証する。

上記４の証明書を検証する際、以下のことをする必要がある。

1. 認証パスの構築
   署名検証者のトラストアンカから検証対象証明書までの証明書チェーンを作る。また、証明書チェーンに含まれる各証明書が失効されているか否かを判断するための情報を集める。
2. 認証パスの検証
   認証パスの構築により作られた証明書チェーンに含まれる全ての証明書について
   1. 検証時に有効期限内であること
   2. 失効されていないこと
   3. 認証パス中の各種制約条件(パス長や名前制約等)に適合していること
   4. 有効な証明書ポリシが存在するか否か

   を確かめる。

この章では認証パス構築・パス検証に関して各種標準や仕様の説明を行う。

2.1.1 パス構築　概説

認証パス構築とは、ある署名者の署名を検証したいとするとき、署名を検証したい人(署名検証者)の信頼ポイント(トラストアンカ)の証明書(または鍵情報)と、署名者の公開鍵証明書を入力として、証明書の発行者(issuer)と主体者(subject)の名称お

よび、署名と署名鍵が繋がるようにリポジトリと呼ばれるデータベースより中間の証明書を取得しながら、トラストアンカから署名者の証明書に電子署名の連鎖が形成されるまでの証明書のチェーンを得ることである。

パスが作ることができなければ、その時点で署名者の署名は無効であると判断される。

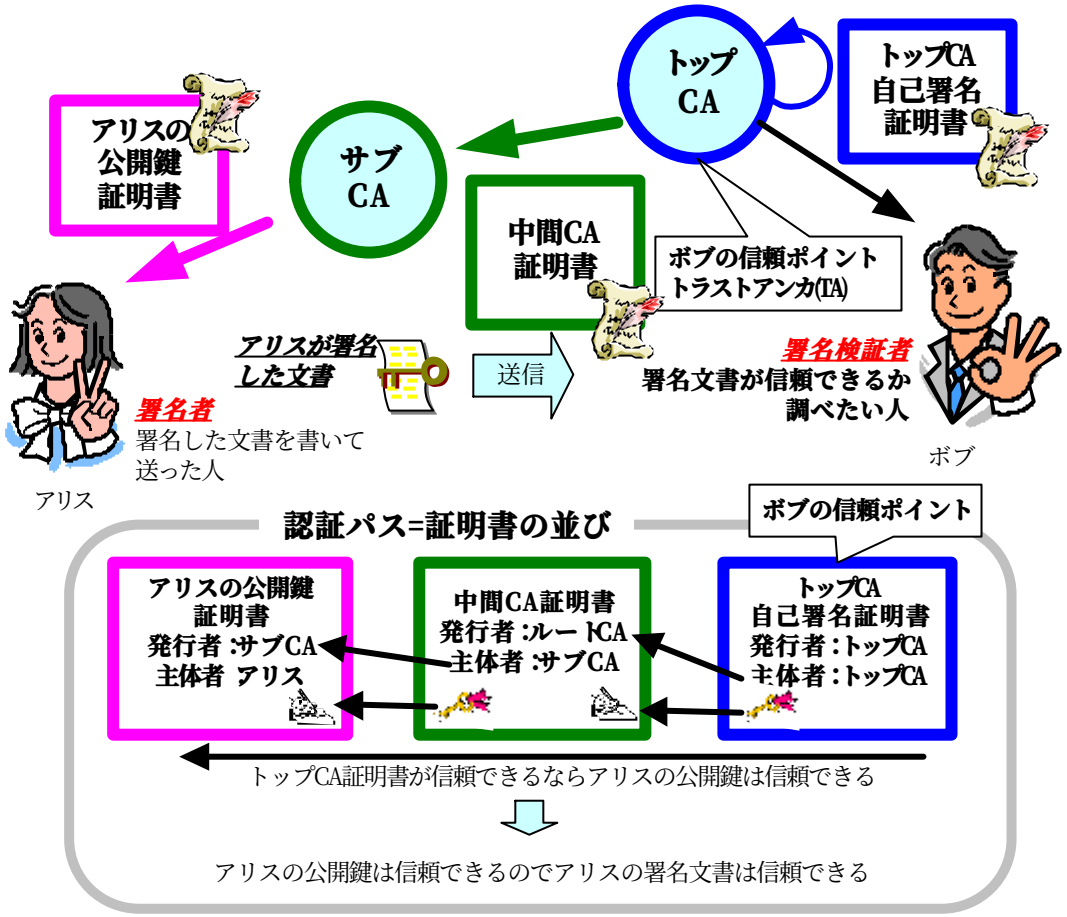


図 2-1 認証パス

パス構築とパス検証の流れを示したのが図 2-2 である。署名検証者の信頼ポイントであるトラストアンカ(TA)証明書と、検証したい署名者の証明書(エンドエンティティ(EE)証明書)を入力として、パス構築アルゴリズムにかけると、証明書の発行者と主体者の名前が繋がるように証明書のチェーンを構築する。狭義のパス構築では、証明書のチェーンを得るだけであるが、広義にはそれらの証明書の失効情報を得るための ARL、CRL、OCSP などの情報を含める場合もある。

検証を行う前提条件として「検証ポリシ」を、生成されたパスを入力としてパス検証アルゴリズムにかけることにより、パスを構成する個々の証明書の有効性を検証し、結果として署名者の証明書の有効性を判定できる。

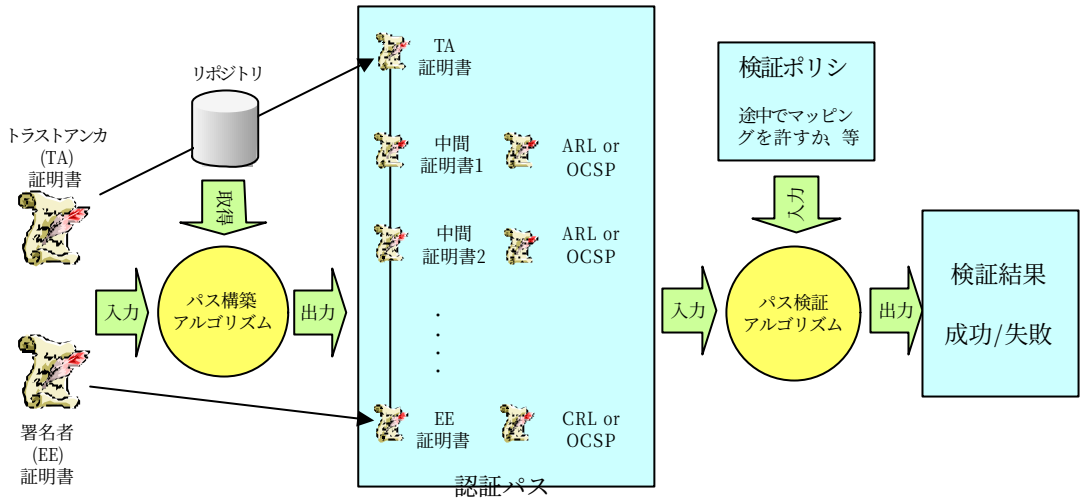


図 2-2 パス構築とパス検証の流れ

2.1.2 パス検証　概説

認証パス検証とは、前節の認証パス構築で得られた証明書チェーンの、各証明書に対し､以下のチェック項目の全てを満足しているか検証する手続きのことを指す。

(1) 署名の検証(名前のチェーンを含む)

名前のチェーンが繋がるか、署名が正しい署名であるか、等

(2) 有効期限のチェック

証明書は証明書有効期限内のものであるか

(3) ポリシ制御

証明書の発行ポリシが要求されているクライテリアに合うか、認証局毎に違うクライテリアの対応がとれるか(＝ポリシマッピング)、等

(4) その他の制約(CA フラグ、パス長制約、名前空間制約、鍵使用目的、等)

正しい認証局用証明書であるか、証明書主体者の名前が指定された名前空間の制約を満足するか、等

(5) 失効検証

その証明書が証明書失効リスト(CRL)に含まれたものでないか、

2.1.3 証明書プロファイル・CRL プロファイル

証明書および CRL は、構造をもった一つのデータファイルとなっている。証明書の構造の概略を 図 2-3 に示す。

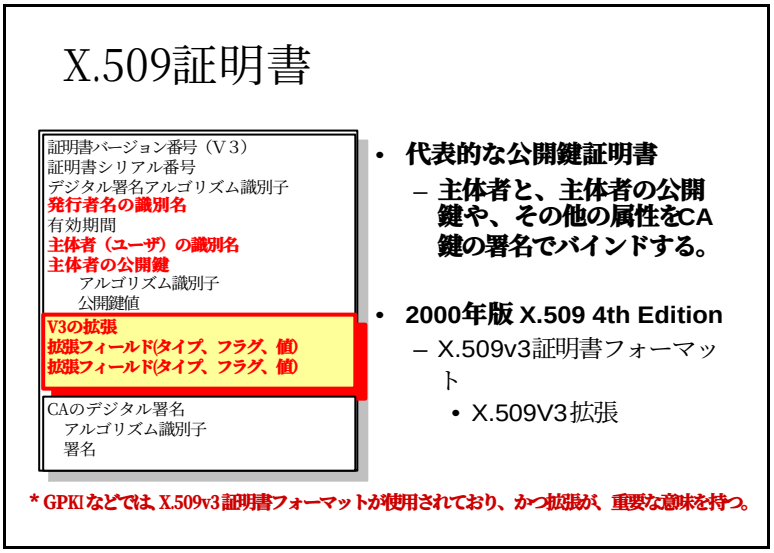


図 2-3 X.509v3 証明書の構造

PKI におけるパス構築、パス検証においては拡張領域にある拡張が重要な意味を持つ。証明書を構成する基本領域・発行者の署名、そして拡張領域にどの拡張を持たせることができるかを定めたものを証明書プロファイルと呼ぶ。証明書プロファイルを定めることによって、汎用的な X.509 証明書の使用法を限定し、アプリケーションの実装を厳密かつ容易にすることができる。

また、CRL は次のような構造になっている。CRL も同様にプロファイルがあり(CRL プロファイルと呼ぶ)、証明書と同様にプロファイルにより CRL の使用法を限定し、アプリケーションの実装を厳密かつ容易にすることができる。

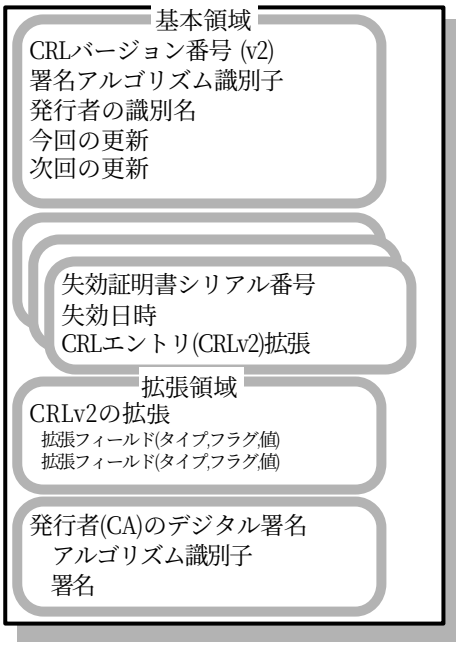


図 2-4 CRLv2 の構造

2.2 ITU-T と IETF PKIX ワーキンググループ

PKI を扱う標準仕様には、ITU-T(International Telecommunication Union Telecommunication Standardization Sector:国際電気通信連合 電気通信標準化部)の定めた国際標準仕様と、インターネット上で PKI をどう使うかを定めた IETF(Internet Engineering Task Force)の PKIX-WG で定めたものが著名である。

表 2-1 ITU-T と IETF RFC の特徴

組織	文書	特徴
ITU-T	X.509 4th	PKI 利用の基礎となる
IETF PKIX	一連の RFC	X.509 4th をインターネット向けに少し簡素化・拡張

ITU-T の最新の仕様が X.509 4th Edition (文献[X509.4])であり、PKIX の最新の仕様が RFC3280, RFC2560 などの一連の RFC として公開されている。

ITU-T の X.509 では、X.500 ディレクトリを基本とした仕様となっているが、IETF PKIX-WG の RFC は、よりインターネットを意識した PKI の利用の観点から仕様が策定されている。ほぼ同じ目的で作られた仕様であるが、パス構築、パス

検証の実装を進める上で、両者の微妙な違いを正しく把握しておくことが重要である。

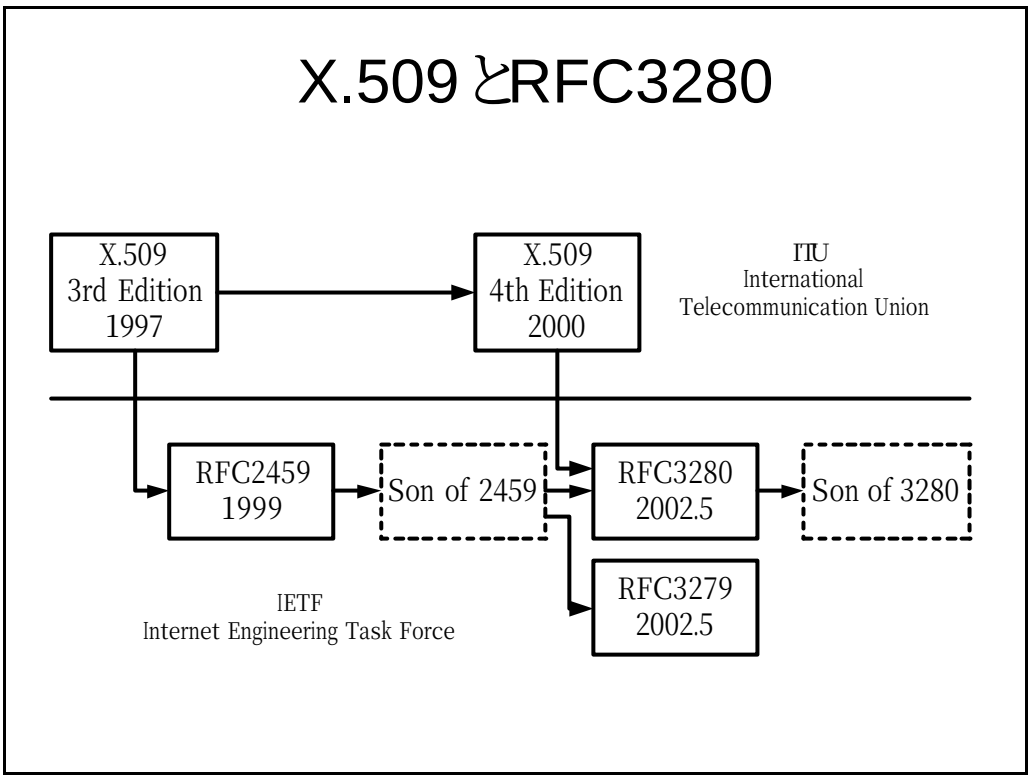


図 2-5 ITU-T と IETF の仕様の系譜

2.2.1 ITU-T X.509 4th Edition

X.509 4th Edition([X509.4])は 2001 年 10 月に ITU-T より公開（本文では 2000 年 3 月付けとなっている）された勧告で、以下の内容でまとめられている。

- 公開鍵証明書フレームワーク
- 属性証明書フレームワーク
  （PMI：Privilege Management Infrastracture）
- 公開鍵証明書と属性証明書を利用する際のディレクトリフレームワーク

表 2-2 X.509 のバージョン

X.509 Edition	証明書フォーマット	CRL フォーマット	備考
1st Edition	V1	V1	古いトップ CA の証明書に V1

1988			フォーマットのものがある
2nd Edition 1994	V2	V1	ほとんど使用されていない
3rd Edition 1997	V3	V2	14 個(v3)標準拡張フィールド
4th Edition 2000	V3	V2	標準拡張フィールドが 8 個追加された※1

※1：X509 4th Edition で追加された標準拡張フィールド
【証明書】inhibitAnyPolicy, freshestCRL
【CRL】crlScope,orderedList,crlStreamIdentifier,
statusReferrals, baseUpdateTime,deltaInfo

2.2.2 IETF PKIX-WG

IETF PKIX ワーキンググループは 1995 年に設立され、X.509 に基づく PKI のインターネット上での利用を目的として活動している。PKIX におけるパス構築・検証を理解するためには幾つかの RFC を参照しなければならない。特に重要なものをまとめたのが表 2-3 である。

表 2-3 PKI に必要な RFC

RFC	内容
3280	証明書・CRL プロファイル、パス検証アルゴリズム
2459	(旧)証明書・CRL プロファイル、パス検証アルゴリズム
2587	LDAPv2 PKI スキーマ
2585	証明書リポジトリとしての HTTP、FTP の利用
2560	OCSP(オンラインによる証明書の状況確認サービス)
[pkix-ldap3-draft]	LDAPv3 PKI スキーマ(ドラフト)
[pkix-rmap]	PKIX ワーキンググループの今後の展開

2.3 認証パス構築

認証パス構築では、署名検証者にとっての信頼ポイントである認証局(＝トラストアンカ)の自己署名証明書と、署名者の公開鍵証明書を入力として、これらをチェーンの両端とする発行者と主体者を参照する証明書のチェーンを作ることである。

認証パス構築の概要を以下に示す。

- 証明書中の主体者名、発行者名を調べる
- リポジトリよりチェーンを成す証明書を取得する
  - ✧ ディレクトリより証明書を取得　(または)
  - ✧ HTTP や FTP など他のプロトコルにより証明書を取得

証明書の内容を知るためには証明書プロファイルを、ディレクトリから証明書を取得するにはディレクトリスキーマを、ディレクトリ以外のリポジトリからは、その取得方法を理解しておく必要がある。本節では、これらについて説明する。

2.3.1 パス構築に必要な証明書プロファイル

証明書プロファイル中で、認証パス構築に使われる基本領域属性および拡張は次の通りである。

表 2-4 パス構築に必要な証明書属性

	基本領域属性名/拡張名	内容
基本領域	Issuer	証明書発行者名
	Subject	証明書主体者名
	AuthorityKeyIdentifier	発行者署名鍵の鍵識別子
	SubjectKeyIdentifier	主体者公開鍵の鍵識別子
	authorityInformatinoAccess ※1	発行者情報の取得先
	subjectInformationAccess ※1	主体者情報の取得先
	cRLDistributionPoints ※2	CRL 配布点
	Freshest CRL　※2	Delta CRL 配布点

※1：IETF のみの拡張([RFC3280] 4.2.2.1 節、4.2.2.2 節)
※2：ARL、CRL を取得する広義のパス構築に使われる

証明書の主体者名と発行者名により証明書のチェーンを作ることが目的となるので、名称および鍵に関する属性が主となる。リポジトリに複数の証明書がある場合には AuthorityKeyIdentifier、SubjectKeyIdentifier 拡張があれば、その値によりパスを構成する証明書を絞り込むことができる。

証明書プロファイルに関しては 3 章を参照の事

[RFC2459]より、新たに authorityInformationAccess(AIA)拡張が追加された。この属性値には、認証局の証明書を直接取得するための属性値を保持したり、オンラインで証明書の検証ができる OCSP レスポンダ(後述 2.6　節参照)の場所を指定することができるようになった。IETF の規格でしか利用できない拡張であるが、この拡張によりパス構築を格段に容易にすることができる。AIA の属性値を以下にまとめる。(詳細は[RFC3280]4.2.2.1 節参照)

表 2-5 authorityInformationAccess の属性値

AIA 属性値種別	内容
cAIssuer	証明書を発行した CA 情報。ディレクトリ名が指定された場合、そのディレクトリ名のエントリの crossCertificatePair 属性の値には CA 証明書が含まれている
ocsp	証明書を検証できる OCSP レスポンダの URI

2.3.2 ディレクトリスキーマ

証明書や失効リストを得るためのリポジトリは、ディレクトリサービスを用いるのが一般的である。

PKI スキーマ定義に関する記述は、ITU-T の X.509 4th Edition[X509.4]では 11 章に、IETF の LDAPv2 PKI スキーマについては文献[RFC2587]の 3 章にかかれている。

PKIX-WG では、[RFC2587]と[RFC2256]で書かれていた内容を一つにまとめ、LDAPv3 に対応させたスキーマ定義のドラフト[pkix-ldap3-draft]を策定中であり、その 3 章において新しいスキーマが定義されており、[X.509.4]のスキーマ定義に合わせるように改訂されている。

パス構築では CA 証明書および、失効情報に関する属性に注目する必要がある。

表 2-6 パス構築に必要なスキーマ

策定団体			ITU-T	IETF PKIX-WG	
仕様			X.509 4th	RFC2587+ RFC2256	ドラフト
ディレクトリサービス			X.500	LDAPv2	LDAPv3
	用途	ディレクトリ属性名			
パ	EE 証明書	userCertificate ※2	○	○	○

ス構築用	CA 証明書	cACertificate ※3	○	○	○
		crossCertificatePair	○	○	○
	パス一時保持	pkiPath	○		○
パス検証用	失効情報	authorityRevocationList	○	○	○
		certificateRevocationList	○	○	○
		deltaRevocationList	○	○	○
	ポリシ	certificatePolicy	○		○
		cpPointer	※1		○
		cps,certificationPracticeStmt	○		○
		cpsPointer	※1		○
	その他	supportedAlgorithms	○		○

※1：属性としては存在しないが上段の属性から取得可能である
※2：この属性を持つオブジェクトクラスは LDAP の WG の発行した[RFC2256]では strongAuthenticationUser となっているが、PKIX の RFC および[X509.4]では、pkiUser である。
※3：この属性を持つオブジェクトクラスは LDAP の WG の発行した[RFC2256]では certificationAuthority となっているが、PKIX の RFC および[X509.4]では、pkiCA である。

pkiPath 属性は[X509.4]/[pkix-ldap3-draft]において新たに追加された属性であり、頻繁に使われる他の CA への認証パスの相互認証証明書の並び(またはその一部)を保存しておくことで効率的にパス構築に利用可能である。

[RFC2587]2 章では、LDAP のオブジェクトクラスが動的に変更される仕様により、[X.509.4]で必須とされている属性は PKIX ではオプショナルとされており、属性が必ずしもエントリにあるとは限らないので注意が必要であると記されている。
その後継である[pkix-ldap3-draft]には、そのような記述は見当たらない。

2.3.3 使用するリポジトリの違い

トラストアンカとエンドエンティティの証明書より、それらがチェーンとして繋がるような認証パスを構築するためには、中間の証明書や失効リストをリポジトリより取得しなければならない。(リポジトリの用語定義:[RFC2585]1.1 節、等)

リポジトリよりパス構築に必要な証明書/失効情報を取得するのに使われるプロトコルを比較したのが 表 2-7 である。インターネット利用を意識した PKIX では、

[RFC2585]1.2 節中において、「通常リポジトリはディレクトリを想定しているが、インターネット上では利用できない場合もあり、その代替として FTP、HTTP の利用法も規定している」、と書かれている。

表 2-7 リポジトリをアクセスするプロトコル比較

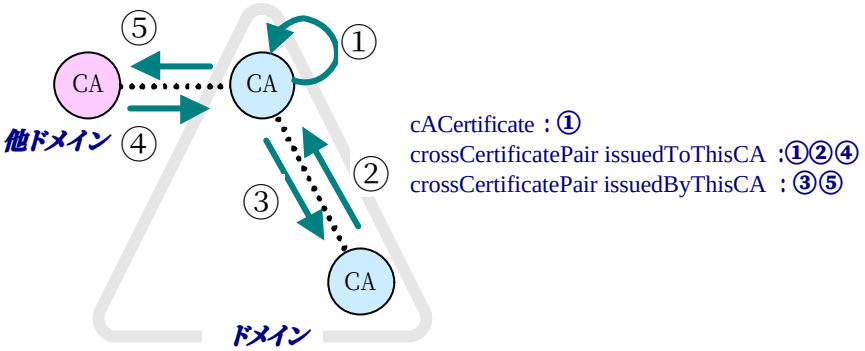
	ITU-T X.509.4th	IETF PKIX RFC
証明書、CRL のリポジトリ	X.500 ディレクトリ ※失効リストは URI も指定できるが、どのプロトコルが使えるかは言及せず	LDAPv2(schema rfc2587,protocol rfc2559) 利用可能な URI について言及 HTTP (RFC2585) 例：http://www.your.org/pki/id22.{cer,crl} FTP(RFC2585) 例：ftp://ftp.your.org/pki/id48.{cer,crl}

2.3.4 IETF PKIX-WG のロードマップ

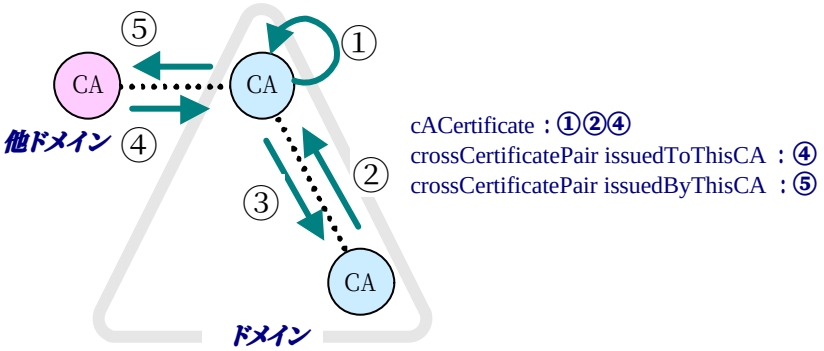
PKIX-WG ではロードマップという形で WG の計画をまとめている。(文献 [pkix-rmap]) 5.1.3 節では、認証パス構築についての 2 つの決定事項について述べている。

- 証明書の Authority Information Access 拡張について
  証明書の主体者名から、CA や CA のリポジトリを簡単に得られない場合で効率的に CA 証明書を取得する方法について議論されてきたが、証明書の Authority Information Access 拡張を使うことにより「CA の情報やサービスへのアクセス方法や場所」を持たせることが決定された。これにより、主体者名とディレクトリ名が一致しないような場合でも、発行者の証明書を取得することができ、パス構築が容易になる。(参考：[RFC3280]4.2.2.1 節 AIA)

- 同一ドメイン証明書登録先の cACertificate と crossCertificatePair の選択
  認証局(CA)の公開鍵証明書(PKC)を、ディレクトリ中の cACertificate 属性に置くか、crossCertificatePair(CCP)置くかについて多くの議論がなされ、2 つの方法が提案されてきた。
  - [方法 1]
    - ✧ cACertificate には自己署名証明書のみ
    - ✧ CCP の issuedToThisCA 要素には、この CA への証明書
    - ✧ CCP の issuedByThisCA 要素には、この CA からの証明書

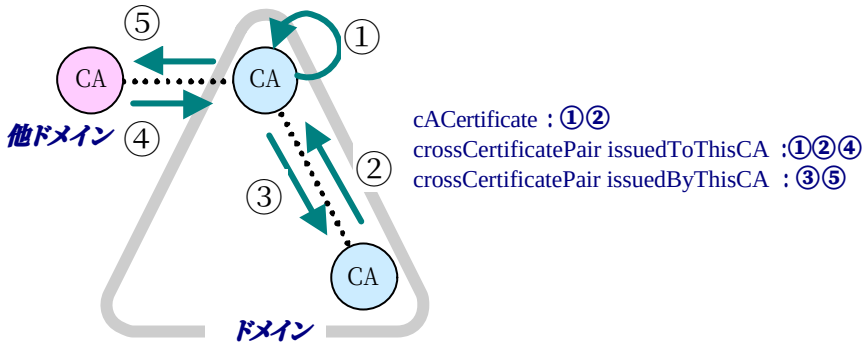


- [方法 2]
  - ✧ cACertificate には、この CA への全ての証明書
  - ✧ CCP の issuedToThisCA 要素には、他ドメインから、この CA への証明書
  - ✧ CCP の issuedByThisCA 要素には、この CA から他ドメイン CA への証明書



方法 2 の方が効率的には優れているが、両者とも既に広く流布してしまっている。そこで、PKIX-WG では以下の折衷案的な方法 3 の採用を決定した。これにより、方法 1、2 のどちらを想定した実装でもサポートされることになる。

- [方法 3: 採用案]
  - ✧ cACertificate には、CA の自己署名と同一ドメインからの証明書
  - ✧ CCP の issuedToThisCA 要素には、この CA への全ての証明書
  - ✧ CCP の issuedByThisCA 要素には、この CA からの全ての証明書



2.4　認証パス検証

パス検証では、署名の検証、有効期限のチェック、ポリシ制約処理、その他制約処理および失効検証を行う。失効検証については、別途 2.5　節において述べる。

X509 4th Edition の認証パス検証のアルゴリズムについては、文献[X.509.4]の 10 章で詳しく説明されている。一方、RFC3280 におけるパス検証アルゴリズムは文献[RFC3280]の 6 章に記述されている。6.1 節では基本的な検証アルゴリズムについて述べられており、6.2 節では、「前節のアルゴリズムはあくまでも最小限の要件を満たすためのものであり、アプリケーションによってはアルゴリズムの拡張をしても構わない」と記している。

X.509 4th Edition と RFC3280 の入力、状態変数、処理内容、出力を示すことにより、両者の違いをまとめたのが 表 2-8 である。(注：差異が重要な箇所に○印をつけた。)

表 2-8 X.509 4th/RFC3280 のパス構築の違い

	ITU-T(X.509 4th Edition)	IETF PKIX(RFC3280)
入力	パスを構成する証明書集合 トラストアンカ情報 ポリシ検証用初期値 現在時刻(必要に応じ)	パスを構成する証明書集合 トラストアンカ情報 ポリシ検証用初期値 現在時刻 予期される認証パス長: n
状態変数	ポリシ検証用 ○ポリシ状態は表形式 ○クリチカルフラグ処理無	ポリシ検証用 ○ポリシ状態は木構造 ○ノードにクリチカルフラグ有

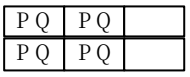
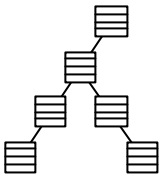
	 名前制約用 パス長管理	 名前制約用 パス長管理 ○作業中の発行者と公開鍵 issuerUniqueIdentifier を処理 (※GPKI では使用しない)
処理内容	証明書の有効性検証 基本制約検証 ポリシ検証 名前制約検証	証明書の有効性検証 基本制約検証 ポリシ検証 名前制約検証
出力	成功失敗フラグ ポリシ検証の状態(表形式) エラーコード	成功失敗フラグ ポリシ検証の状態(木構造) エラーコード(記述無し)
細かな違い	keyUsage 処理の記述無し クリチカルフラグ処理無し 名前制約で subjectAltName について明記せず	

表 2-8 が示す通り、細かい違いはあるものの、入力、状態変数、処理内容、および出力は殆ど同じであり、ポリシの状態を保持するためのデータ構造が表形式か木構造であるかの違いだけで､結果は全く同じになることに注意しなければならない。

2.4.1 パス検証に必要な証明書プロファイル

パス検証に必要となる証明書プロファイル中の基本領域属性、拡張をまとめたのが 表 2-9 である。

表 2-9 パス構築に必要とされる基本属性領域および拡張属性領域

	領域	属性名	内容
署名の検証	署名		証明書発行者の署名
	基本	Signature	署名アルゴリズム
	基本	Issuer	証明書発行者の識別名
	基本	Subject	証明書主体者の識別名

	基本	subjectPublicKeyInfo	主体者公開鍵情報
	拡張	authorityKeyIdentifier	認証機関鍵識別子
	拡張	subjectKeyIdentifier	主体者鍵識別子
	拡張	subjectAltName	主体者別名
	拡張	issuerAltName	発行者別名
有効期限	基本	Validity	有効期限
ポリシ制約	拡張	certificatePolicies	証明書ポリシ
	拡張	policyMappings	ポリシマッピング
	拡張	policyConstraints	パス中のポリシ影響範囲
	拡張	inhibitAnyPolicy	anyPolicy を許可しない
その他制約	拡張	basicConstraints	基本制約(CA,パス長)
	拡張	nameConstraints	名前空間制約
	拡張	keyUsage	鍵使用目的
	拡張	extendKeyUsage	拡張鍵使用目的
失効検証	基本	serialNumber	証明書シリアル番号
	基本	Issuer	証明書発行者の識別名
	拡張	cRLDistributionPoints	CRL 配布点
	拡張	freshestCRL	Delta CRL 配布点
	拡張	authorityInformationAccess	発行者情報(OCSP の時)

2.4.2 X509 4th Edition のポリシ検証

X.509 のパス検証アルゴリズムでは、ポリシ検証に表形式のデータを使うのが大きな特徴である。表形式のデータを用いた簡単なポリシ検証を図示したものが 図 2-9 である。(注：理解しやすくするためにポリシ修飾子の処理は省略した。)

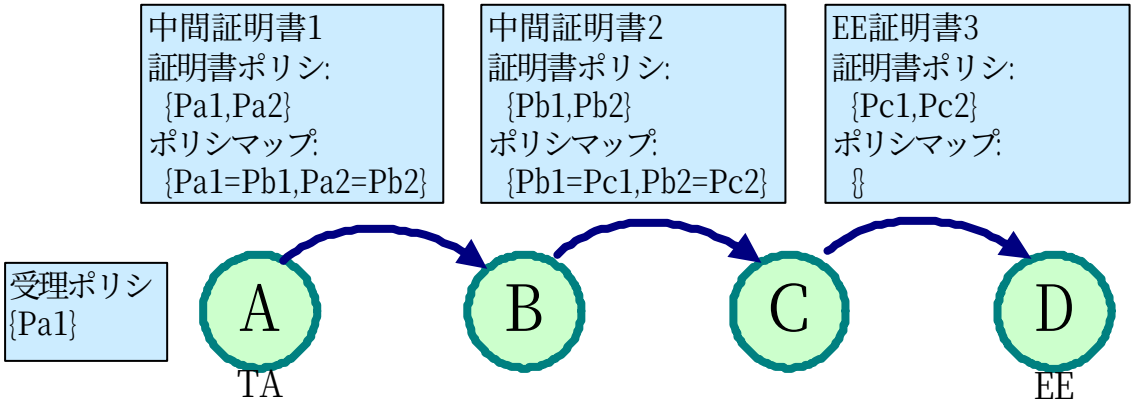


図 2-6 X.509 4th のポリシ検証(認証パス例)

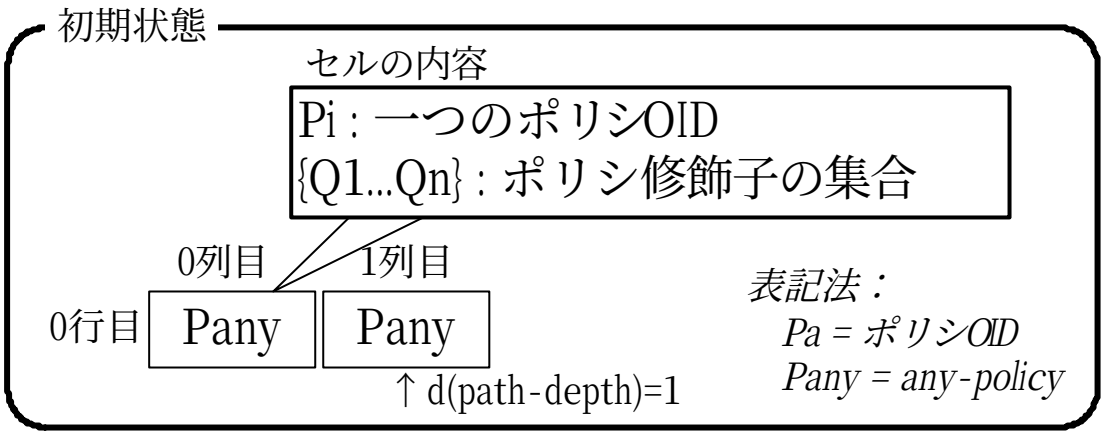


図 2-7 X.509 4th のポリシ検証(初期状態)

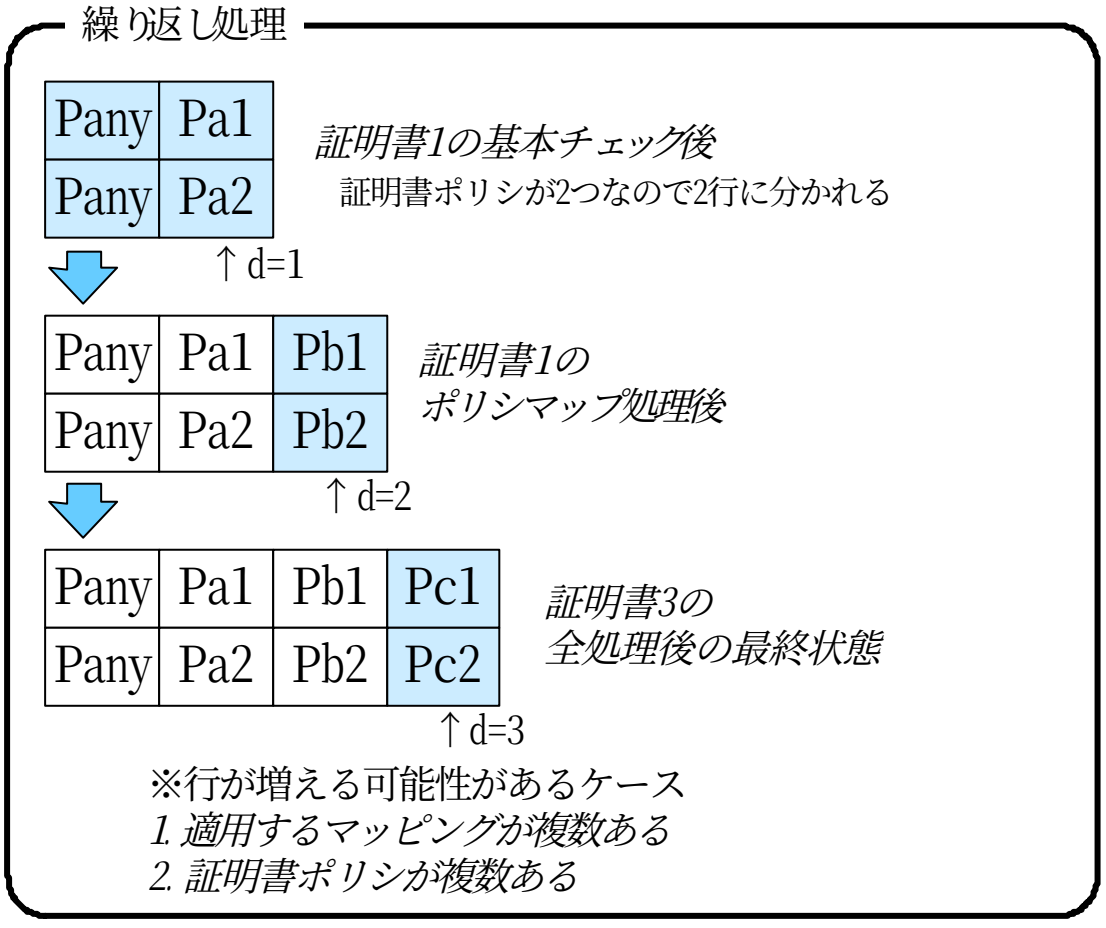


図 2-8 X.509 4th のポリシ検証(ループ)

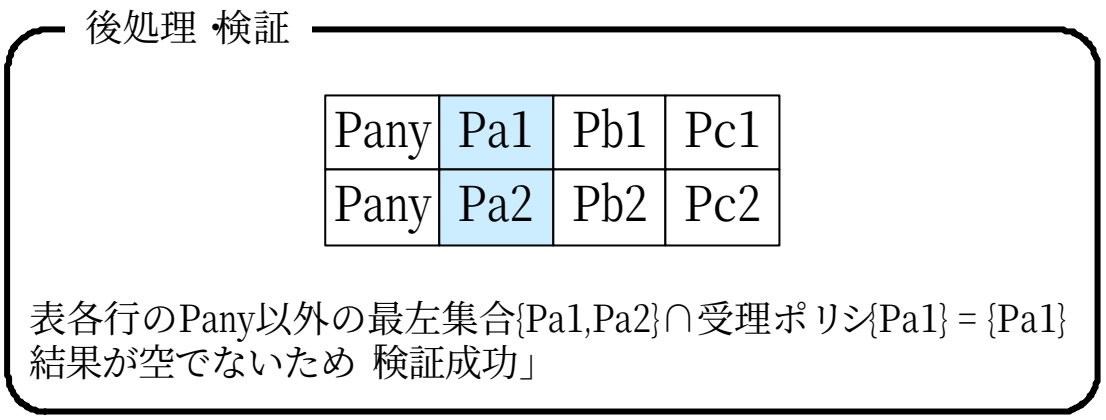


図 2-9 X.509 4th のポリシ検証(後処理)

[X.509.4] 10.2 **のパス処理手続きの出力の節の末尾には、このアルゴリズムによる検証が成功したとしても、ポリシ修飾子や証明書中の他の情報を元に、この結果を受け入れるかどうか判断しなければならないと明記されている。**[RFC3280] **ではポリシ修飾子からの結果判断の必要性は述べられていない。**

2.4.3 RFC3280 のポリシ検証

RFC3280 **のパス検証アルゴリズムは、ポリシ検証に木構造のデータ形式を使うことが特徴である。簡単なポリシ検証の処理を図示したものが 図** 2-13 **である。(注:わかりやすくするため、クリチカルフラグとポリシ修飾子の処理は省略している。また、見やすさのため木構造を左から右へ枝が伸びるようにしている。)**

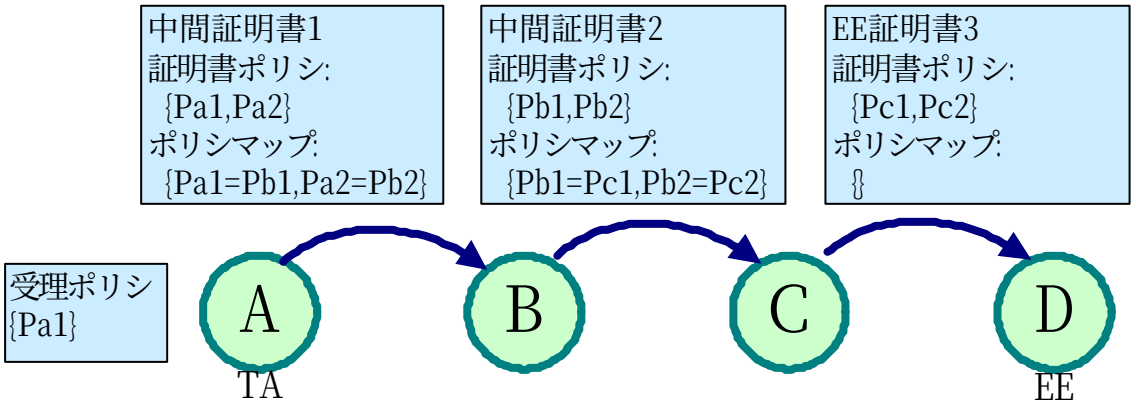


図 2-10 RFC3280 のポリシ検証(認証パス例)

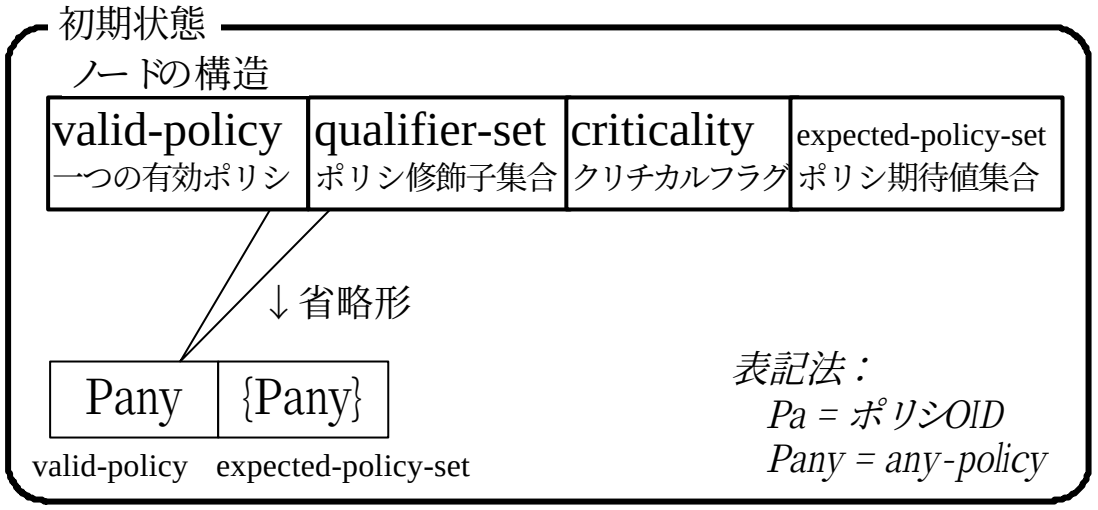


図 2-11 RFC3280 のポリシ検証(初期状態)

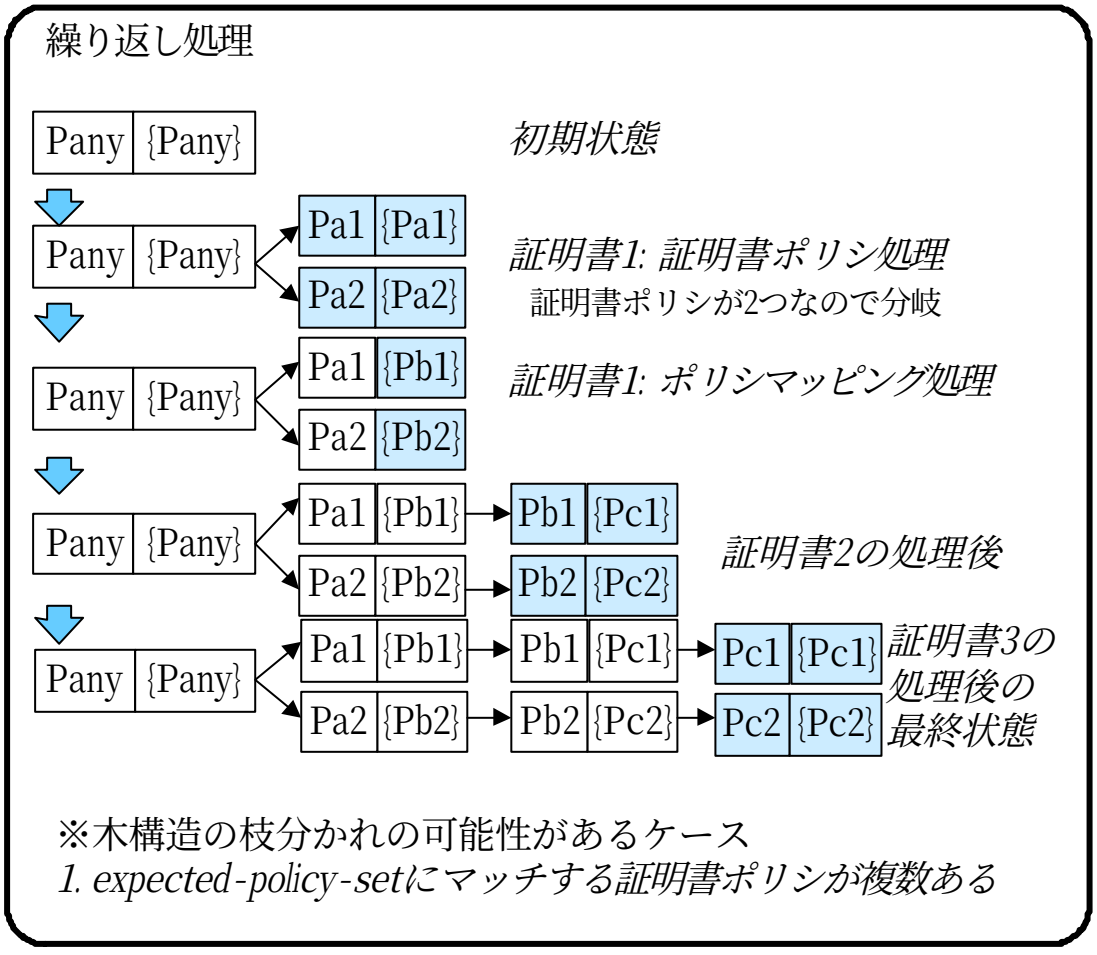


図 2-12 RFC3280 のポリシ検証(ループ)

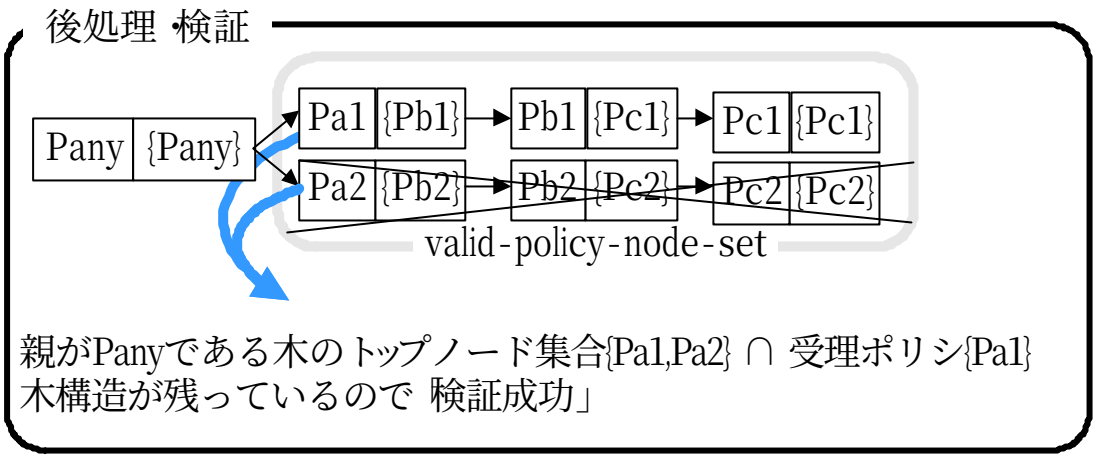


図 2-13 RFC3280 のポリシ検証(後処理)

2.5 CRL による失効検証

証明書の失効検証には、CRL を使う方法と OCSP を使う方法がある。OCSP については、2.6 節で詳しく述べる。本節では CRL を用いた失効検証について述べる。

[X509.4]Annex B では CRL の種別判定、適切な CRL の取得、CRL の有効性の検証について述べられている。また、[RFC3280] 6.3 節では、CRL を用いた証明書失効検証アルゴリズムについて述べられている。本節ではこれらについて詳しく述べる。

2.5.1 X.509 4th Edition における CRL 種別の判定と CRL 有効性の検証

署名検証者が証明書が信頼できるかどうかを判定するための仕組みとして CRL があるが、検証したい証明書のための CRL を取得することは以下のような理由により単純にはならない。([X509.4]による CRL の種別のまとめについては詳しくは 10.5 節を参照)

- **認証局によって CRL の肥大化を防ぐため以下の手法を行っている場合がある**
  - ✧ **差分 CRL を使う場合**
  - ✧ **認証局用とエンドエンティティ用を分けている場合**

- 単独の認証局用 CRL の場合もあれば、複数用の場合もある

CRL が対象毎に分かれている場合があるため、どの CRL に検証対象の証明書の CRL があるか判定が難しいのである。そのため、[X509.4]では、CRL に対し crlScope 拡張を用いることにより CRL の管理している証明書の範囲に関する情報を持たせることができる。(※[RFC3280]ではこの拡張は定義されていない。)

[X509.4] Annex B の節では、crlScope 拡張がない場合でも、CRL の種類を判定し、適切な CRL を取得するためのアルゴリズムが示されている。

発行局の、CRL の発行に関するポリシによって、証明書利用者は以下の種類の基本 CRL を保持するかもしれない。

- Complete CRL for all entities;
- Complete EPRL
- Complete CARL
- Distribution Point Based CRL/EPRL/CARL

CRL がどの種類に属するかを判別する方法については、[X509.4] Annex B 5.1 に記述されている。
また、証明書の freshestCRL 拡張が critical であり、証明書利用者のポリシがデルタ CRL の確認をサポートしている場合、証明書利用者はデルタ CRL についての処理も可能性がある。デルタ CRL が適切なものであるかどうかを確認する方法については、[X509.4]Annex B 5.2 に記述されている。
基本 CRL の失効検証を行うためには、その CRL が配布されてから一度も変更されておらず、かつ、次の条件を満たす必要がある。

(1) 証明書利用者が、CRL の発行者の公開鍵を取得でき、公開鍵を用いて、CRL の署名を検証できる。　（CRL の署名検証）
(2) nextUpdate フィールドがある場合、そのフィールドに記された時刻が、現在時刻よりも後である。（最新の CRL であることの確認）
(3) CRL に書かれた発行者名が、失効を検証しようとしている証明書の発行者名と一致する。※

※ただしその CRL が、証明書の CRL DP によって示された場所から取得されており、CRL DP 拡張が CRL issuer 要素を含む場合を除く。この場合、CRL issuer 要素が CRL の発行者名と一致することが条件となる。

また、デルタ CRL の失効検証を行うためには、その CRL が配布されてから一度も変更されておらず、かつ、上記(1)(2)及び、次の条件を満たす必要がある。

(4) デルタ CRL に含まれる発行者名が、失効を検証しようとしている証明書の発行者名と一致する。（デルタ CRL の検証）

条件(1)～(4)についての詳細は、[X509.4]Annex B 5.3～5.4 節に書かれている。

2.5.2 RFC3280 における CRL を用いた証明書失効検証アルゴリズム

CRL を用いた証明書の失効検証を行うアルゴリズムは、以下の入力を必要とする。

・証明書（証明書シリアル番号と発行者名）
・デルタ CRL が使用されるか否か（use-deltas フラグ）

失効検証アルゴリズムは、与えられた証明書が「失効していないこと」を前提として検証処理を開始する。そして証明書が失効していることが判明するか、あるいは全ての理由コードをカバーするのに十分なCRLのチェックが完了するまで、CRLのチェックを続ける。
失効検証は、証明書 CRL 分配点拡張の各々の分配点(DP)に対する、ローカル CRL キャッシュの対応する各々の CRL に対して、次の手順で行われる。

手順(1) 失効情報（CRL、デルタ CRL）の取得
手順(2) 失効リストの検証
手順(3) 失効理由（reasons_mask）の検証
手順(4) 失効リスト発行者のパス検証

手順(1)では、必要に応じて CRL・デルタ CRL を取得し、ローカルの CRL キャッシュを更新する。
手順(2)では、CRL の発行者とその範囲を検証する。また、デルタ CRL が使用される場合（use-deltas フラグがセットされている場合）は、デルタ CRL の発行者と範囲を検証する。
手順(3)では、その CRL によってサポートされている失効理由の集合（interim_reasons_mask）を計算する。そしてこのセットが、これまでに処理された失効理由の集合（reasons_mask）に含まれない 1 つ以上の理由を含むことを検証する。

手順(4)では、CRL の発行者に対する認証パスを取得・確認し、CRL 上の署名を確認する。

以上の処理を行うことで失効状態が決定され、cert_status が返される。失効状態が決定されなかったならば、分配点で明示されていないが証明書発行者により発行されたすべての利用可能な CRL を用いて、上記のプロセスが繰り返される。それでも失効状態が決定されなければ、UNDETERMINED が返される。

手順(1)～(4)の詳細については、[RFC3280] 6.3 節 CRL Validation の(a)～(k)に書かれている。

2.6 RFC2560 OCSP

本節では OCSP による検証の方法を記述する。

2.6.1 OCSP による有効性確認

認証パスの検証作業において、認証パスを構成する証明書が現時点で有効かどうかを確認する必要があり、一般的に証明書の有効性確認の手段として認証局が発行する CRL を利用する。しかし、CRL を用いた有効性確認では、以下のような問題点が発生しうる。

- パフォーマンス
  CA が発行する証明書が増加すると、一般的にそれに比例して失効した証明書の数も増加し、CRL のサイズが大きくなる。CRL のサイズが大きくなると、CRL のダウンロードに掛かる時間が増大し、結果として認証パス構築・パス検証に掛かる時間が増大する。
  また、認証パスの検証には、認証パスを構成する全ての中間証明書およびユーザ証明書の有効性確認を行う必要があるため、複数の CRL を取得・検索する必要が発生し、認証パス検証に掛かる時間が更に増大する。

- リアルタイム性
  CRL は定期的に CA によって発行される。そのため、一度 CRL を取得したら、その CRL が更新されるまで CRL をキャッシュし、認証パス検証のパフォーマンスを向上させることが可能となる。その反面、現在の CRL が発行された後で失効された証明書の情報は、次回の CRL 更新まで CRL に反映されず、リアルタイムに証明書の状態を確認することができない。

これらの問題を解決するために、CRL を複数の小片に分割することでダウンロー

ドするCRLのサイズを小さくするCRL-DP（CRL Distribution Point）や、CRLの更新部分のみを配信するDelta CRLといった手法が考案され、利用されている。

これらの手法が、CRL による有効性確認方法を基本として、ダウンロードするCRL のサイズを小さくするための対処方法として利用されているのに対し、CRLとは全く異なる手法による証明書の有効性確認手段として、OCSP (Online Certificate Status Protocol)がRFC2560として標準化されている。

OCSPは、署名検証者がOCSPレスポンダ（OCSPサーバ）に証明書の状態を問い合わせるリクエストを送信し、OCSPレスポンダがそれに対するレスポンスを返答すると言う、クライアント/サーバ型の有効性確認手段である。

OCSP の利点としては、リクエストおよびレスポンスのサイズが小さく、またCA認証局とOCSPレスポンダを密に連携させることによりリアルタイムな証明書の有効性確認を可能とする点があげられる。また、一つのリクエスト/レスポンスで複数の証明書の有効性確認を一度に行うことができるため、OCSPレスポンダが複数の認証局に対応している場合、認証パスを構成する証明書の有効性確認を行うためのリクエスト/レスポンス数を減らすことができる。

以下に、OCSPのリクエストとレスポンスの詳細、およびOCSPを実現する上で重要となるOCSPレスポンダの信頼性モデルについて説明する。また、現在IETFのPKIX WGに提案され議論されているOCSPv2についても簡単に説明する。

2.6.2 OCSP リクエスト

図 2-14 にOCSPリクエストのメッセージフォーマットを示す。

tbsRequest		リクエスト	
	version	バージョン	規定値 V1(0)
	request	リクエスタの名称	オプション
	requestList	個別リクエスト（複数）	
	reqCert	リクエストする証明書の情報	
	hashAlgorithm	ハッシュアルゴリズム	
	issuerNameHash	IssuerのDN名のハッシュ値	
	issuerKeyHash	Issuerの公開鍵のハッシュ値	
	serialNumber	証明書のシリアル番号	
	singleRequestExtensions	個々のリクエストに対する拡張領域	オプション
	requestExtensions	リクエスト全体に対する拡張領域	オプション
optionalSignature		リクエスタの署名	オプション
	signatureAlgorithm	署名アルゴリズム	
	signature	署名データ	
	certs	証明書（複数）	オプション

図 2-14 OCSP リクエスト・メッセージフォーマット

OCSPでは、メッセージのサイズをできるだけ小さくするため、確認対象の証明

書そのものではなく、証明書を特定しうる最小限の情報のみで確認対象の証明書を指定する。具体的には、CertID という構造体において、確認対象の証明書を発行した CA 認証局の名前（issuerName）のハッシュ値（issuerNameHash）と CA 認証局の公開鍵のハッシュ値（issuerKeyHash）、ならびに確認対象の証明書のシリアル番号（serialNumber）の 3 つの情報を指定する。

確認対象の証明書情報（requestList）は、一つのリクエスト内に複数指定することができる。

OCSP リクエストでは、確認対象証明書情報毎およびリクエスト全体の 2 種類の Extension（拡張領域）を指定することができる。RFC2560 で規定されている OCSP リクエスト用拡張領域としては、リプライ攻撃を防ぐための Nonce や、確認対象証明書の AIA Extension 情報を OCSP レスポンダに伝えるための Service Locator などがある。AIA Extension とは、RFC2459 および RFC3280 で規定されている証明書に対する Extension であり（X.509 4th Edition では規定されていない）、ある認証局が OCSP レスポンダを使用する場合、認証局が発行する証明書の AIA Extension に OCSP レスポンダへの URI を示すことができる。それにより、その認証局が発行した証明書を受け取った署名検証者が、有効性確認するためにはどこに対して OCSP リクエストを発行すれば良いのかがわかる仕組みになっている。

リクエストには署名検証者が署名することもできるが、一般的には用いられない。

2.6.3 OCSP レスポンス

図 2-15 に OCSP レスポンスのメッセージフォーマットを示す。

responseStatus	レスポンスの状態	successful (0) malformedRequest (1) internalError (2) tryLater (3) （4は未使用） sigRequired (5) unauthorized (6)
responseBytes	レスポンス本体	オプション
responseType	レスポンスの形式	id-pkix-ocsp-basic
response	レスポンスデータ（OCTET STRING）	
tbsResponseData	レスポンス（全体）	
version	バージョン	規定値 V1(0)
responderID	レスポンダのID	
byName	レスポンダの名称	選択(String)
byKey	レスポンダの公開鍵のハッシュ値	選択(OCTET STRING)
producedAt	レスポンス生成時刻	
responses	個別レスポンス（複数）	
certID	証明書のID	
hashAlgorithm	ハッシュアルゴリズム	
issuerNameHash	IssuerのDN名のハッシュ値	
issuerKeyHash	Issuerの公開鍵のハッシュ値	
serialNumber	証明書のシリアル番号	
certStatus	証明書の状態	
good	good	選択(NULL)
revoked	revoked	選択(SEQUENCE)
revocationTime	失効時間	
revocationReason	失効理由	オプション
unknown	unknown	選択(NULL)
thisUpdate	状態の有効期間（from）	
nextUpdate	状態の有効期間（to）	オプション
singleExtensions	個々のレスポンスに対する拡張領域	オプション
responseExtensions	レスポンス全体に対する拡張領域	オプション
signatureAlgorithm	署名アルゴリズム	
signature	署名データ	
certs	証明書（複数）	オプション

図 2-15 OCSP レスポンス・メッセージフォーマット

レスポンスメッセージの先頭には、レスポンスの状態を示す responseStatus があり、responseStatus が successful の場合のみ実際のレスポンスメッセージ（responseBytes）が後に続く。

レスポンスメッセージにはまずレスポンスメッセージのタイプを示す responseType があるが、現時点では id-pkix-ocsp-basic のレスポンスタイプ（Basic レスポンス）のみが定義されている。

Basic レスポンスでは、ヘッダとしてレスポンスフォーマットのバージョン（version）、OCSP レスポンダの情報（responderID）、また OCSP レスポンダがこのレスポンスを作成した時刻情報（producedAt）があり、その後に個々の確認対象証明書に対するレスポンス情報（response）が列挙される。

証明書毎のレスポンス情報では、確認対象の CertID とその証明書の状態（certStatus）、および状態の有効期限（thisUpdate、nextUpdate）が記載される。証明書の状態は、good（有効）、revoked（失効）、unknown（不明）のどれかが指定される。unknown 状態とは、例えば OCSP レスポンダがその証明書を発行した

認証局に対応していない場合などに発生する。

OCSP レスポンスでもリクエストメッセージと同様に、証明書毎の拡張領域とレスポンス全体の拡張領域を指定することができる。RFC2560 で規定されている拡張領域としては、OCSP レスポンダが状態を判定する際に参照した CRL の情報を返す CRL References などがある。

また、OCSP レスポンスには必ず OCSP レスポンダによる署名が付与される。これは、信頼できる OCSP レスポンダからのレスポンスかどうかを OCSP クライアントが判別するために利用される。

2.6.4 OCSP レスポンダの信頼性モデル

OCSP レスポンダを構築する上で重要な検討事項として、OCSP レスポンダの信頼性モデルがある。OCSP レスポンスに付与された署名を、OCSP クライアント（署名検証者）が信頼するための構成モデルである。

OCSP レスポンダの信頼性モデルとして、大きく以下の 3 種類がある。

- Direct CA Trust Model（図 2-16）
  OCSP クライアントが直接 CA のみを信頼するモデルである。OCSP レスポンダは、対応する CA の秘密鍵でレスポンスに署名する。
  法務省も商業登記認証局の OCSP レスポンダはこのモデルである。
  Direct CA Trust Model では、OCSP レスポンダが対応する CA と秘密鍵を共有する必要があるが、秘密鍵をセキュアに管理するための問題が発生しうる。一般的に CA および CA の秘密鍵は外部から直接アクセスできない箇所で管理されているが、OCSP レスポンダは外部から直接アクセスできなければ用を成さない。従って、OCSP レスポンダを外部に公開しつついかにその秘密鍵をセキュアに管理するかが問題となりうる。
  また一般的に CA の証明書は、有効期限を長くするために長い鍵を利用する場合が多いが、その場合 OCSP レスポンスの署名に時間がかかり、OCSP レスポンダのサーバのパフォーマンスが悪くなることが考えられる。

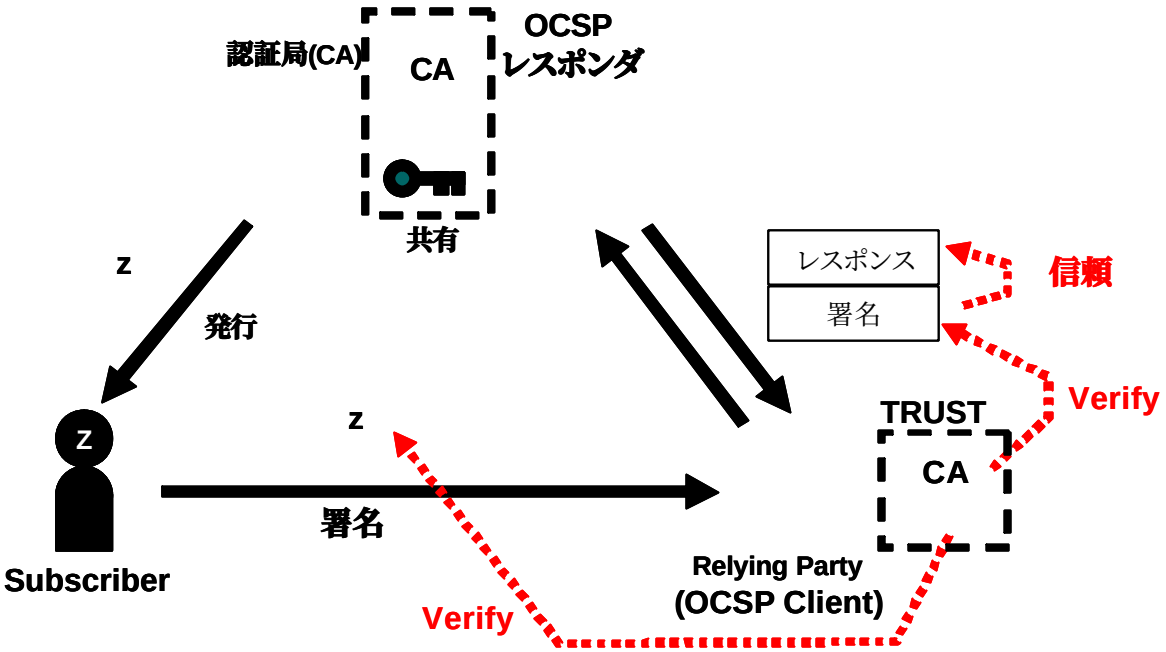


図 2-16 Direct CA Trust Model

- Direct VA Trust Model（図 2-17）
  OCSP レスポンダが自己署名証明書を発行しており、OCSP クライアントが直接 OCSP レスポンダ（VA）の証明書を信頼するモデルである。
  OCSP レスポンダの証明書を直接信頼させるためには、OCSP レスポンダの鍵の管理が同様に問題になる。また、OCSP レスポンダの証明書の信頼性を高めるためには鍵長を長くする必要があり、同様にパフォーマンスの問題が発生しうる。

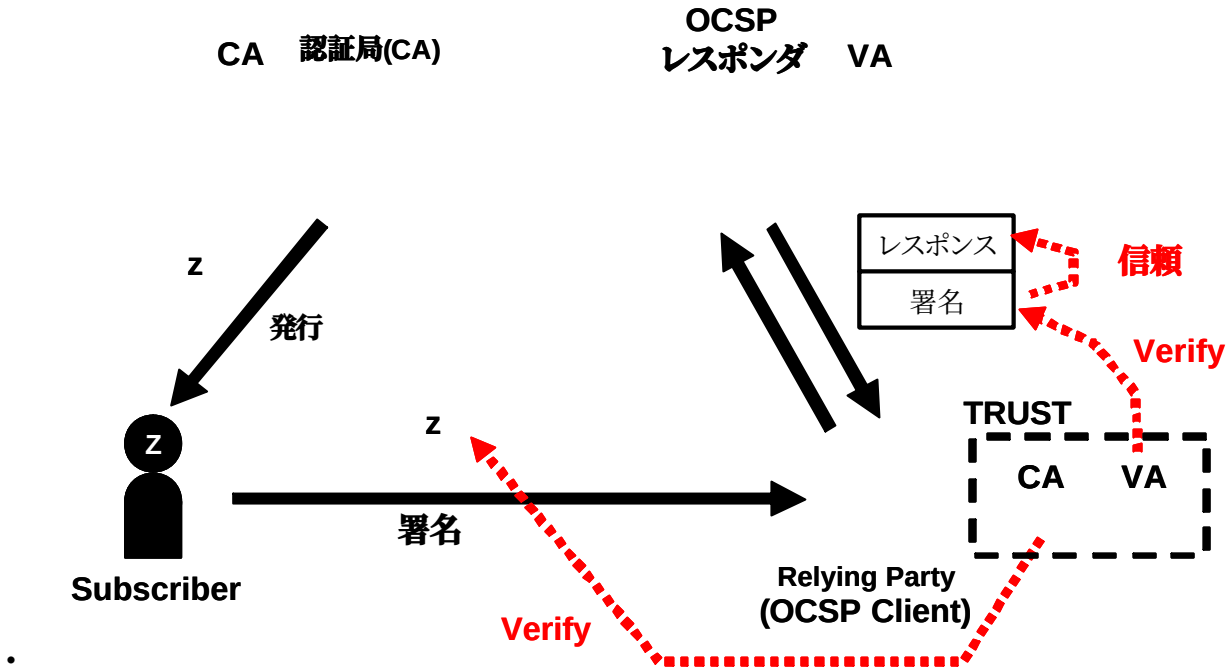


図 2-17 Direct VA Trust Model

- CA Delegated Model（図 2-18）
  OCSP クライアントは CA のみを直接信頼し、その CA から発行された OCSP レスポンダの証明書を間接的に信頼するモデルである。この場合、CA が発行した証明書の状態を OCSP レスポンダが返答することを CA 自身が委譲したことを明示的に示すために、OCSP レスポンダに対して発行する証明書の Extended Key Usage Extension に OCSPSigning という特殊な印をつける必要がある。
  CA Delegated Model の場合、OCSP レスポンダの証明書が更新されても特にクライアント側の変更は必要ないため、OCSP レスポンダの証明書の有効期限を短く設定して頻繁に更新することで、OCSP レスポンダの証明書の信頼性を高めることができる。
  ただし、図に示すように OCSP クライアントはまず OCSP レスポンスの署名に使用された証明書が自身が信頼する CA から発行されたものであり、かつ OCSPSigning がついていることを検証し、次に署名に使用された証明書で署名を検証するという 2 段階の検証作業が必要となる。

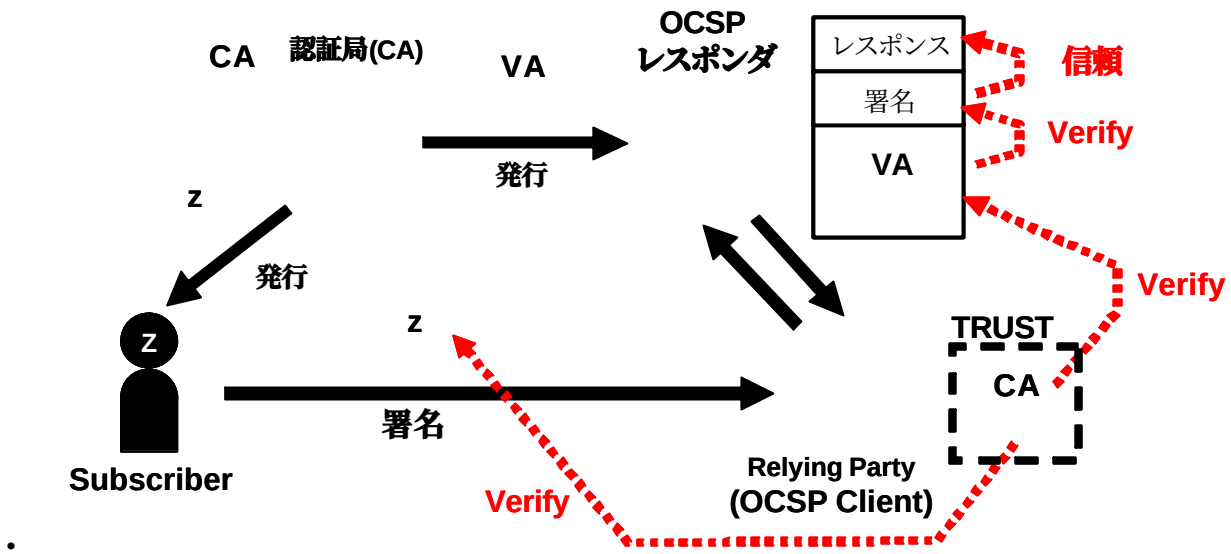


図 2-18 CA Delegated Model

2.6.5 OCSPv2

OCSP は IETF の PKIX WG にて 1999 年 6 月に RFC2560 として標準化されたが、その後実際の運用において幾つかの問題点が指摘されてきた。また、新たに規定された属性証明書（Attribute Certificate）への対応も必要とされている。

そこで、OCSP の問題点の克服および新しい機能の取り込みを目指し、2002 年 12 月に OCSP を拡張する OCSP Version2（OCSPv2）が PKIX に提案されている。

OCSPv2 の特徴は以下の 3 点である。

- 属性証明書への対応
  認証証明書だけではなく属性証明書に対する有効性確認も行えるようにする（特にプロトコル上の拡張はなし）。

- CertID の拡張
  確認対象の証明書を指定する方法として、CertID 以外にも証明書全体を送信する方法、および発行者の名前とシリアル番号、証明書の tbsCertificate 構造体のハッシュ値、証明書の署名を送る方法の 2 種類を選択肢として追加。

- CRL の参照情報の指定
  crlLocator という Extension を追加することで、OCSP クライアントが対象証明書の失効状態を格納している CRL の参照情報を証明書の CRL Distribution Point Extension（CDP）からコピーすることで OCSP レスポンダに指定することを可能とする。

2.7 GPKI 相互運用性仕様書

本節では GPKI の相互運用性仕様書に関しての解説を行う。

2.7.1 GPKI 概要

GPKI の特徴としては府省 CA 及び民間 CA は相互認証するために BCA(ブリッジ CA)を介して相互認証を行っている。 異なるドメインを経由して、相互認証を行うため、GPKI という枠組みの中で共通の仕様に従った処理が必要となる。

次の表に、BCA を介した異なるドメイン同士の相互運用性の概念図を 図 2-19 に示す。

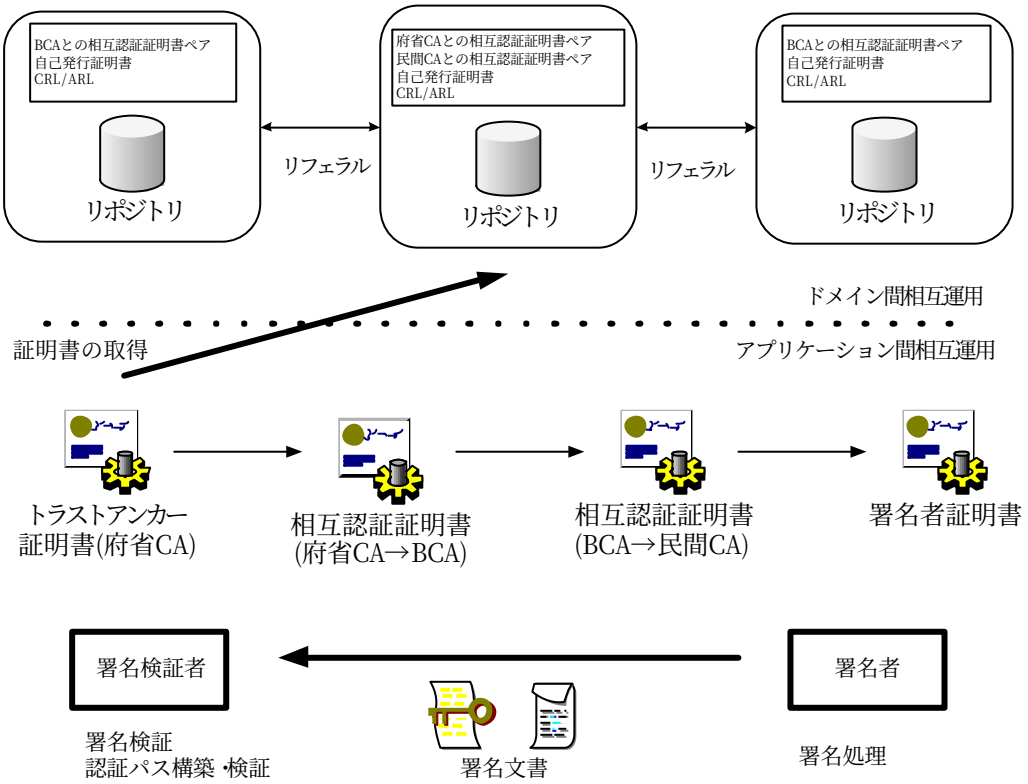


図 2-19 BCA を介する相互運用性概念図

2.7.2 コンポーネント仕様、構成

ここではGPKIを構成するコンポーネントがGPKIを利用するアプリケーションについて相互運用性を確保するために必要である主な機能要件を記述する。

・機能要件

1. BCA
   - ✧ ―自己署名証明書、CRL/ARL、相互認証証明書の発行
   - ✧ ―自己署名証明書、CRL/ARL、相互認証証明書ペアのリポジトリへの格納
   - ✧ ―CA 鍵更新機能
2. 府省・民間 CA（PCA）
   - ✧ ―自己署名証明書、CRL/ARL、相互認証証明書、EE 証明書の発行
   - ✧ ―自己署名証明書、CRL/ARL、相互認証証明書ペアのリポジトリへの格納
   - ✧ ―CA 鍵更新機能(オプション)
3. リポジトリ
   - ✧ ―証明書、CRL/ARL の検索
   - ✧ ―他のリポジトリとのリフェラル
4. OCSP レスポンダ
   - ✧ ―証明書所有者からの証明書検証要求の受付とその検証結果の返答
5. 証明書検証サーバ
   - ✧ ―署名検証者からの証明書検証要求の受付と検証結果の返答
   - ✧ ―認証パス構築・検証
6. EE(エンドエンティティ)
   - ✧ ―データに対して署名・署名検証
   - ✧ ―リポジトリから証明書 CRL/ARL の取得
   - ✧ ―自ドメインの PCA までのパス構築(オプション)
   - ✧ ―OCSP レスポンダや証明書検証サーバへの問い合わせ機能

図 2-20 に、GPKI を構成する PKI コンポーネントの概念図を示す。

GPKI を構成する各コンポーネントの詳細については GPKI 相互運用性仕様書「2.PKI コンポーネント仕様」に記述されている。

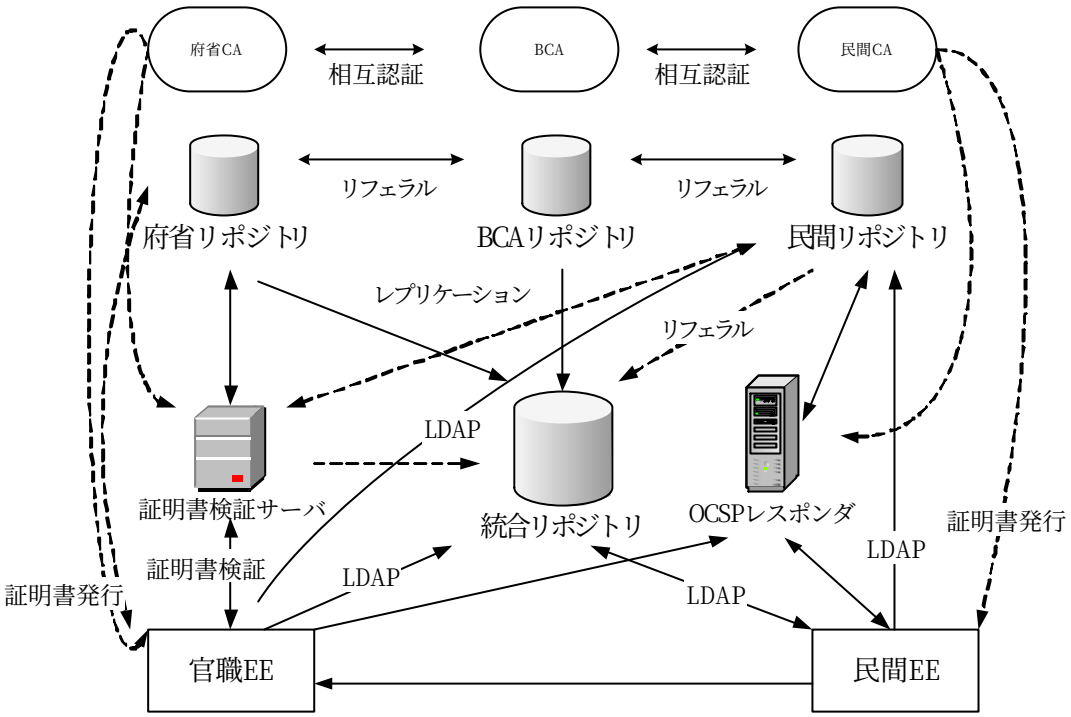


図 2-20 GPKI を構成するコンポーネント

2.7.3 アプリケーション間相互運用

ここでは、GPKI において利用するアプリケーションが相互運用性を確保するために必要な機能や推奨モデルを示す。

(1) 認証パス構築、検証仕様概要

GPKI において異なるドメイン間で署名検証を行うアプリケーションは、以下のことを行う必要がある。

1. **パス構築**
2. **パス検証**
3. **証明書ポリシ等の各種制約条件**

各項目の詳細は GPKI 相互運用性仕様書「3.7 認証パスの構築・検証方法」に記述されている。

(2) リポジトリ

認証パスの構築・検証を行う際、パス中に含まれる証明書及び失効情報を、LDAPv3 を利用し、リポジトリから取得する必要がある。リポジトリアクセスでの検索内容によっては LDAPv3 Referral を返すリポジトリもある。アプリケーション等は検索結果として返ってきた Referral を解釈できる必要がある。

詳細は、GPKI 相互運用性仕様書「3.6 リポジトリへのアクセス」に記述されている。

(3) 認証パスの構築

証明書チェーンを構成する全ての証明書を取得できる必要があり、そのためにパスの構築を行う。証明書の取得方法としては、署名者側から署名文書に添付して送付する方法や、署名検証者がリポジトリにアクセスする方法が考えられる。署名検証者が署名者証明書までの認証パスを構築するとき、その方法はいくつか存在し、署名検証者のトラストアンカーから見て順方向、逆方向に署名者証明書までの認証パスを構築する方法が GPKI 相互運用性仕様書「3.7.2 認証パスの構築」に記述されている。

(4) 鍵更新後のパス構築

CA 秘密鍵の定期的な更新を行うことで、複数の新旧の CA 秘密鍵が存在する。その環境下においても既存の有効な証明書による認証パスの構築を可能にするためにリンク証明書(自己発行証明書)を発行する。これにより、既存の証明書を、リンク証明書により新旧の自己署名証明書がリンクすることでパス構築が可能になる。

詳細は GPKI 相互運用性仕様書「3.7.2.6 鍵更新」を行った後のパス構築に記述されている。

(5) 認証パスの検証

基本的に RFC3280 にある「Certification Path Validation」に準じている。認証

パスの検証は一般的に署名検証者が最も信頼しているCAの証明書から始まり署名者のエンドエンティティ証明書の方向へ検証する方法が最も効率が良く、好ましい。検証項目は、証明書チェーンや証明書の有効期限や署名検証、失効検証等がある。

詳細は、GPKI相互運用性仕様書「3.7.3 認証パスの検証」に記述されている。

(6) EEモデルと証明書検証サーバモデル

GPKIでは、異なるドメイン間を介して署名の検証を行うので複雑になる。
GPKIでは署名検証を実装するアプリケーションモデルとしてEEモデル(図2-21)と証明書検証サーバモデル(図2-22)がある。

- **EEモデル**
  署名検証、認証パスの構築・検証の複雑な処理をすべてエンドエンティティで行うモデル。

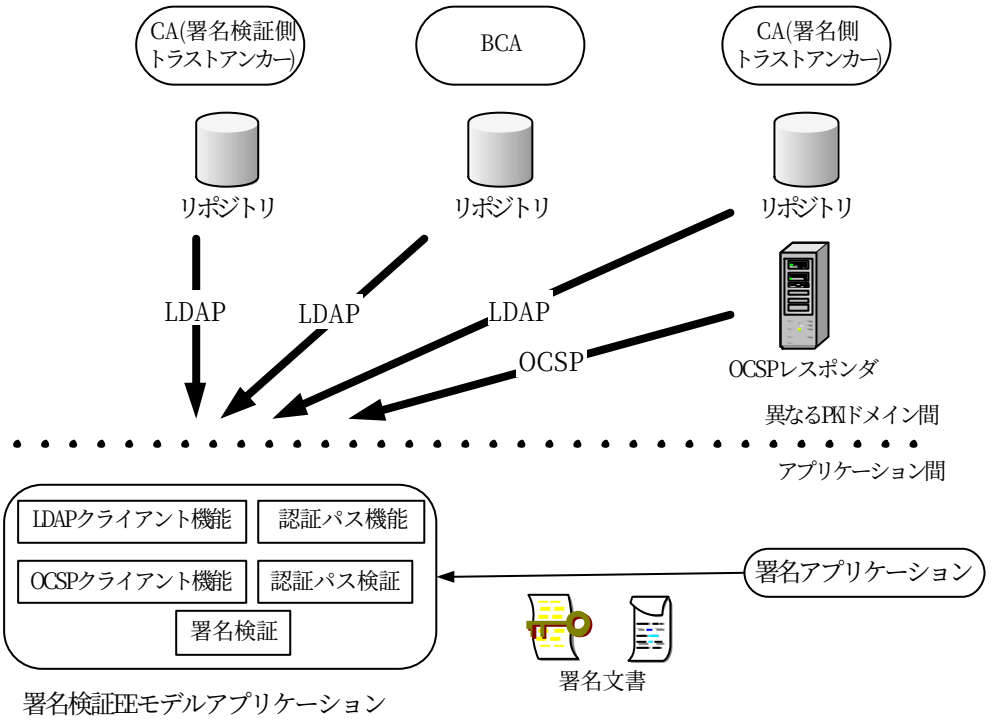


図 2-21 EEモデル

- **証明書検証サーバモデル**

認証パスの構築・検証をするサーバとして証明書検証サーバを設け、アプリケーションは証明書検証サーバのクライアントとして動作するモデル。

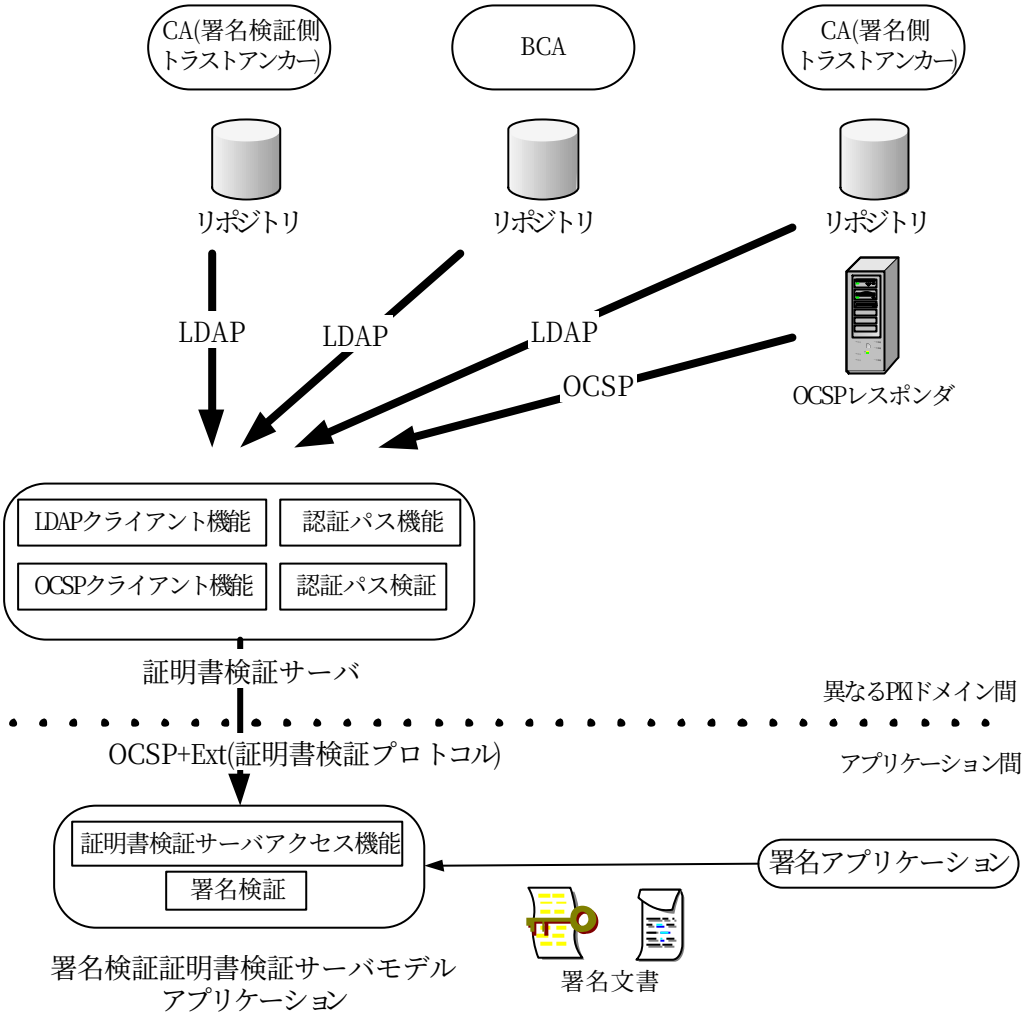


図 2-22 証明書検証サーバモデル

2.7.4 PKI ドメイン内仕様

GPKI において BCA と相互接続を行うには次に述べる 2 つの仕様を満足しなければならない。

1. **鍵更新**
   自己署名証明書の有効期限を考慮し、定期的に CA 秘密鍵を更新する必要がある。
2. **リンク証明書**

鍵更新に伴い新旧の自己署名証明書が複数存在するときに、新しい世代の鍵ペアと古い世代の鍵ペアの関係を保証し、更新しても既存の有効な証明書による認証パスの構築を可能にするため、BCA 等はリンク証明書を発行する。

詳細は、GPKI 相互運用性仕様書「5.2.1 鍵更新」に記述されている。

2.7.5 プロファイル

ここでは、GPKI における証明書プロファイルの特徴をいくつか示す。

GPKI の環境において必須なフィールド

- policyMappings
- keyUsage
- certificatePolicies
- authorityKeyIdentifier(keyIdentifier)

その他

・issuerUniqueID、subjectUniqueID のフィールドが存在する証明書は無視する。

詳細は、GPKI 相互運用性仕様書「7 プロファイル」に表が記述されている。

3 証明書パス構築・パス検証サーバなどの新しいモデルの説明

3.1 証明書パス構築・パス検証サーバの概要

第 2 章で述べたように、電子証明書等を受け取った署名検証者が証明書の検証を行う場合、認証パスの構築と認証パスの検証という作業を行う必要がある。

一般的に認証パスの構築および認証パスの検証作業は、署名検証者自身で証明書や CRL/ARL 等の必要な情報を収集して行う必要があるが、これらの作業全てまたは一部をサーバ側で行う方法が検討および一部利用されている（以下、認証パスの構築・検証を行うサーバを証明書検証サーバ、またそのクライアントを証明書検証クライアントと呼ぶ)。

認証パス構築・パス検証をサーバサイドで行うことには、以下のような利点が考えられる。

- 証明書検証クライアントの実装の簡略化
  認証パスの構築・パス検証をサーバサイドで行わせることにより、署名検証者（証明書検証クライアント)では単に証明書検証サーバとの通信プロトコルにのみ対応すればよくなり、証明書や失効情報取得のためのプロトコルや、認証パス構築・パス検証の実装を行う必要なくなる。それにより、従来リソース（処理能力やメモリ、通信能力）に制限のあるため PKI の適用が難しいとされた PDA や携帯電話といった端末においても、正確な認証パス構築・パス検証の作業が行えるようになるといった利点がある。

- パフォーマンス
  認証パス構築・パス検証の作業において、必要な情報の取得やパスの構築、パスの検証の作業を能力の高い証明書検証サーバで行わせることにより、認証パス構築・パス検証に掛かる時間を短縮できる。また、サーバサイドで取得した証明書や失効情報をキャッシュすることで、より効率的にパス構築・パス検証のリクエストを処理できるようになると考えられる。

- パス構築・パス検証方法の均一化
  認証パス構築・パス検証作業を署名検証者で行った場合、各署名検証者の実装によって動作が異なってしまう可能性があり、それらを統一することは非常に難しい。また、認証パス構築やパス検証の仕様に変更があった場合、既に稼動している全署名検証者を新しい仕様にあわせて修正することには多大なコストと時間がかかる。それに対して、認証パス構築・パス検証をサーバサイドで

行うことにより、パス構築・パス検証の方法を均一にすることができると同時に、仕様の変更が発生した際にも証明書検証サーバのみ対応すればよくなり、運用のコストが低減される。

このようなことから、IETF/PKIX では認証パス構築・パス検証のサーバサイドで行うためのプロトコルやその仕様について数年にわたり議論されてきている。また、実際に GPKI では各府省に証明書検証サーバが設置され、認証パスの構築・検証作業の全てをサーバ側で統一的に行うことが可能になっている。

この章では、認証パス構築・パス検証作業をサーバサイドで行うために必要なクライアント・サーバ間のプロトコルの標準化状況と標準化において議論されている点について説明する。また、実際に利用されている GPKI の証明書検証サーバの機能とプロトコルについて説明し、標準化作業中の仕様との比較を行う。

3.2 DPV/DPD REQ

本節では、IETF の PKIX Working Group（以下 IETF/PKIX）において標準化された RFC3379「Delegated Path Validation and Delegated Path Discovery Protocol Requirements（以下、DPV/DPD REQ)」について詳細を説明する。

まず DPV/DPD REQ の成立の過程について簡単に説明し、次に DPV/DPD REQ の内容について説明する。

3.2.1 DPV/DPD REQ への過程

PKIX において証明書検証サーバに関する議論が 1999 年 6 月頃から始まり、これまでに表 3-1 に示すような多くの新規プロトコルや OCSP の拡張案が提案されてきた。

表 3-1 PKIXにおける提案プロトコル

名称	略称	提案時期	現状
Data Validation and Certification Server Protocols	DVCS	1998年9月	RFC3029
Simple Certificate Validation Protocol	SCVP	1999年6月	Version 11
OCSP Extensions	OCSP-X	1999年9月	(Expired)
Delegated Path Validation	OCSPv2+DPD+DPV	2000年8月	(Expired)
Online Certificate Status Protocol, version 2		2000年9月	(Expired)
Delegated Path Discovery with OCSP			
Delegated Path Validation and Delegated Path Discovery Protocols	DPV/DPD	2001年7月	(Expired)
Delegated Signature Validation Delegated Path Validation and Delegated Path Discovery Protocol Requirements	DVS/DPV/DPD REQ	2001年10月	(Expired)
Delegated Path Validation and Delegated Path Discovery Protocol Requirements	DPV/DPD REQ	2001年11月	RFC3379
Certificate Validation Protocol	CVP	2002年6月	Version 1
DPV and DPD over OCSP	DPV/DPD OCSP	2003年1月	Version 0

ここで重要なのは、OCSPv2+DPV+DPD において、証明書検証サーバのサービスとしてDPV（Delegated Path Validation）とDPD（Delegated Path Discovery）という2種類のサービスが提案され、DPV/DPD REQにおいてDPVとDPDに求められる機能要件が定義されたことである。

DPVとは、認証パスの構築から検証までの全ての作業をサーバサイドで行い、証明書検証クライアントでは証明書検証サーバへ問い合わせるだけで署名に使用された証明書の検証が行える。

DPDでは、認証パスの構築およびパスを構成する証明書の失効情報（CRLやOCSPレスポンス）の収集のみをサーバサイドで行い、ポリシーのマッチングなどの認証パスの検証作業はクライアントサイドで行う必要がある。DPDを利用することにより、証明書の検証作業は自身で行いたい署名検証者は、オーバーヘッドの掛かる認証パスの構築と失効情報の収集作業のみをサーバサイドに委託することができる。

DPV/DPD REQは2002年9月にRFC3379として標準化され、現在PKIXにおいてDPV/DPD REQを要件定義として実際にDPV/DPDを実現するプロトコルの標準化について議論がなされている。現在DPV/DPDの候補となっているプロトコルには、SCVP、CVP（Certificate Validation Protocol）、DPV and DPD over OCSP、DVCS（Data Validation and Certification Server Protocols）がある。

また、W3CにおいてもXKMS（XML Key Management Specification）のX-KISS（XML Key Information Service Specification）として、XMLを利用して証明書の有効性検証を行うためのプロトコルの策定が行われている。

次節以降にて、DPV/DPD REQ の詳細について説明し、また 3.5 章においてDPV/DPD を実現するプロトコルの一つである SCVP について詳細を説明する。また、3.6 章においてその他の仕様についても概略を説明する。

3.2.2 DPV の要件

DPV は、認証パスの構築および検証作業の全てをサーバサイドで行うためのプロトコルである。

DPV の利点として、上記にあげたように、証明書検証クライアントの実装の簡略化、証明書検証パフォーマンスの向上、証明書検証方法の統一化が挙げられる。

DPV/DPD REQ では、DPV の要件として以下のようなことが挙げられている。

- バリデーションポリシーに従った証明書の検証
- 過去のある時点における証明書の有効性の検証
- レスポンスへの署名

バリデーション・ポリシーとは、認証パスの構築およびパスの検証を行う際の条件セットであり、クライアントは証明書の検証を依頼する際にバリデーション・ポリシーを指定する。バリデーション・ポリシーに含まれるべき条件としては以下のような規定されている。

- 信頼する認証局（Trust Anchor）
- 各種制約（Name Constraints、Policy Constraints）
- RFC3280 の証明書検証アルゴリズムにおける初期値
- レスポンスへの失効情報の有無
- 検証対象の証明書が満たすべき条件（KeyUsage など）

また、DPV では現時点だけではなく、過去のある時点における証明書の有効性検証が行える必要がある。これは、過去になされた署名に対して、署名された時点で署名に使用した証明書が有効であったかどうかを確認する場合などに利用される。

DPV のレスポンスは、証明書検証クライアントが信頼できる DPV サーバから受け取る必要があるため、DPV レスポンスに DPV サーバは署名する必要がある。

3.2.3 DPD の要件

DPD は、認証パスの検証作業はクライアントで行うが、認証パスを構成する証明書や証明書の失効情報（CRL や OCSP レスポンス）の収集のみをサーバに依頼する際に使用するプロトコルである。

DPD の利点として、証明書検証クライアントの実装の簡略化が挙げられる。

DPV/DPD REQ では、DPD の要件として以下のようなことが挙げられている。

- パス発見ポリシーに従った認証パスの構築
- 一つ以上の認証パスの返答

パス発見ポリシー（Path Discovery Policy）は、バリデーション・ポリシーまたはバリデーション・ポリシーの一部で構成され、クライアントは DPD サーバに対してパス発見ポリシーを指定する必要がある。

また、DPD サーバでは認証パスが複数発見された場合、複数の認証パスを返答することも可能でなければならない。

DPD サーバは認証パスを構成する証明書やそれらの失効情報を収集することだけが目的であり、実際の認証パスの検証において証明書や CRL に対する認証局の署名や OCSP レスポンスに対する OCSP レスポンダの署名をクライアント側で検証し、認証パスの検証を行う。そのため、DPD のレスポンスには特に信頼性は求められず、DPD レスポンスへの署名はオプションとされている。

3.2.4 DPV and DPD Policy Query

DPV/DPD REQ では、その他に DPV/DPD サーバが利用できるバリデーション・ポリシーおよびパス発見ポリシーをクライアントが取得するためのプロトコルが定義されている。

DPV/DPD サーバでは、クライアントから要求があった場合、DPV/DPD サーバで利用できるポリシーの全てのリストまたはデフォルトのポリシーを返答する必要がある。

3.3 SCVP

本文書の執筆中である 2003 年 2 月 21 日に、PKIX WG のメーリングリストにて SCVP（Simple Certificate Validation Protocol）を PKIX WG における DPV/DPD のプロトコルの唯一の候補とすることが発表された。ここでは、PKIX WG における DPV/DPD のプロトコル候補であり、今後 RFC として標準化される SCVP について解説する。

3.3.1 SCVP の概要

SCVP は、DPV/DPD の候補の中でもっとも長く議論されたプロトコルである。SCVP は 1999 年 6 月に提案され、その後 DPV/DPD REQ の標準化にあわせて修正されるなど、修正が続けられ、現在（2003 年 2 月末時点）のドラフトの最新バージョンは 11 である（本稿はバージョン 11 について解説する）。SCVP は、基本的に DPV を実現するプロトコルであり、クライアントがサーバに求める処理機能を指定することで、DPD としても利用することができるようになっている。また、DPV/DPD REQ にあわせるために、バリデーションポリシーやパス発見ポリシーを取得するためのプロトコルも追加定義された。SCVP では、認証証明書だけではなく、属性証明書も検証の対象としている。

SCVP は、証明書検証サーバ・クライアント間でやり取りするプロトコルのフォーマットとして CMS（Cryptographic Message Syntax）を採用しており、CMS に格納する ContentType として新たに以下の 4 種類を定義している。

- SCVP Request
- SCVP Response
- Validation Policy Request
- Validation Policy Response

従って、リクエストやレスポンスに署名を行う必要がある場合、CMS に従って署名を行う。

3.3.2 SCVP リクエスト/レスポンス

図 3-1 に、SCVP リクエストのメッセージフォーマットを示す。

scvpVersion			バージョン	
query			リクエスト内容	
	queriedCerts		対象証明書（複数）	
		pkc	公開鍵証明書情報	
			cert	公開鍵証明書	
			pkcRef	ESSCertID	
		ac	属性証明書情報	
			attrCert	属性証明書	
			acRef	ESSCertID	
	checks		リクエストタイプ（複数）	
	wantBack		レスポンスに含める情報（複数）	
	serverContextInfo		サーバコンテキスト	オプション
	valPolicy		バリデーションポリシー	オプション
		valPolicyId	ポリシーID	
		parameters	パラメータ	
	validityTime		検証時刻	オプション
	trustAnchors		トラストアンカー（複数）	オプション
		anchor	トラストアンカー情報	
			certRef	公開鍵証明書情報	
				cert	公開鍵証明書	
				pkcRef	ESSCertID	
			rawInfo	公開鍵証明書情報	
				name	名前	
				algorithm	アルゴリズム識別子	
				pubKey	公開鍵	
		certPolicies	ポリシーID（複数）	オプション
	intermediateCerts		中間証明書（複数）	オプション
		cert	公開鍵証明書	
		pkcRef	ESSCertID	
	revInfos		失効情報（複数）	オプション
		riType	失効情報タイプ（OCSP/CRL/DeltaCRL）	
		riValue	パラメータ	
	queryExtensions		拡張領域	オプション
requestor			クライアント情報	オプション
requestNonce			Nonce	オプション
reqExtensions			拡張領域	オプション

図 3-1 SCVP リクエスト・メッセージフォーマット

リクエストの query 構造体にてリクエストの内容（検証対象証明書やトラストアンカー、ポリシー等）を指定する。また checks にて SCVP サーバに依頼する処理（DPV/DPD）を指定する。更に、wantBack にてレスポンスに含める情報（認証パスの証明書や失効情報を含めるか）を指定する。

図 3-2 に、SCVP レスポンスのメッセージフォーマットを示す。

scvpVersion	バージョン	
producedAt	レスポンス生成時刻	
responseStatus	レスポンスの状態	
statusCode	状態コード	okay (0) skipUnrecognizedItems (1) tooBusy (10) badStructure (20) unsupportedVersion (21) abortUnrecognizedItems (22) unrecognizedSigKey (23) badSignature (24) unableToDecode (25) not
errorMessage	エラーメッセージ（UTF8）	オプション
requestRef	リクエストへの参照	
requestHash	リクエストのハッシュ値	
algorithm	ハッシュアルゴリズム	
value	ハッシュ値	
fullRequest	リクエスト全体	
requestor	クライアント情報	オプション
responder	サーバ情報	オプション
replyObjects	レスポンス内容（複数）	オプション
cert	証明書情報	
pkc	公開鍵証明書情報	
cert	公開鍵証明書	
pkcRef	ESSCertID	
ac	属性証明書情報	
attrCert	属性証明書	
acRef	ESSCertID	
replyStatus	レスポンスの状態	success (0) unrecognizedCheck (1) unrecognizedWantBack (2) malformedPKC (3) malformedAC (4) unrecognizedCertPolicy (5) unrecognizedValPolicy (6) unrecognizedExtension (7) unavailableValidityTime
replyValTime	検証時刻	
replyChecks	レスポンス（複数）	
check	リクエストタイプ	
status	検証結果	
replyWantBacks	クライアント要求情報（複数）	
wb	レスポンスに含める情報	
value	データ	
valPolicy	バリデーションポリシー	
valPolicyId	ポリシーID	
parameters	パラメータ	
nextUpdate	証明書状態の次回更新予定時刻	オプション
certReplyExtensions	拡張領域	オプション
requestNonce	Nonce	オプション
serverContextInfo	サーバコンテキスト	オプション
respExtetnsions	拡張領域	オプション

図 3-2 SCVP レスポンス・メッセージフォーマット

レスポンスではヘッダとしてレスポンスの状態（responseStatus）やサーバの情

報（responder）、レスポンスの生成時刻（producedAt）を指定し、replyTypeOfCheck にて証明書の検証結果を示す。また、replyWantBacks にてクライアント側から指定された情報を送り返す。

SCVP では、一つのリクエストに対して一つの認証パスのみを返答することとなっている。しかし DPV/DPD REQ では、複数の認証パスが発見された場合には複数の認証パスを返答できる必要性がある。そこで SCVP では、レスポンスに serverContextInfo を格納し、クライアントが同じ証明書に対するリクエストに前回のレスポンスに含まれていた serverContextInfo をコピーして送信することで、サーバは前回とは異なる認証パスを返答する仕組みを提供している。これにより、クライアント側で異なる認証パスが必要となった際に、別の認証パスを要求することが可能となる。

3.3.3 バリエーションポリシー・リクエスト/レスポンス

図 3-3 にバリデーションポリシー・リクエスト/レスポンスのメッセージフォーマットを示す。

Validation Policies Request

ContentInfo		
contentType	コンテンツのタイプ	id-ct-scvp-valPolRequest
content	リクエストの中身	
scvpVersion	バージョン	

Validation Policies Response

ContentInfo		
contentType	コンテンツのタイプ	id-ct-scvp-valPolResponse
content	リクエストの中身	
scvpVersion	バージョン	
valPolicies	ポリシーのOID（複数）	

図 3-3 Validation Policies Request/Response メッセージフォーマット

バリデーションポリシー・リクエスト／レスポンスのメッセージフォーマットは図 3-3 に示す通りである。バリデーションポリシー・リクエストでは単にリクエスト用の OID と SCVP のバージョンを送付し、SCVP サーバからのレスポンスでは、SCVP サーバが対応するポリシーの OID のリストを返答する。

3.4 その他の標準案

本文書の作成中、2003 年 1 月中旬より PKIX WG のメーリングリストにおいて、DPV/DPD を実現するプロトコルを SCVP、CVP、DPV/DPD for OCSP および DVCS の 4 候補から一本化するための投票が行われ、2003 年 2 月 21 日に PKIX WG

としてSCVPをDPV/DPDを実現するプロトコルとして選択したことが発表された。今後、SCVP に対して最終的な議論が行われ、多少の修正を経て SCVP が標準化される。しかし、ここでは DPV/DPD の他の候補であった CVP、DPV/DPD for OCSP および DVCS についても簡単に解説する。また、W3C において議論が行われている XKMS についても簡単に解説する。

3.4.1 CVP

CVP（Certificate Validation Protocol）は、DPV/DPD のために定義されたプロトコルの一つである。CVP は DPV/DPD REQ の著者が中心となって提案しているプロトコルであり、DPV/DPD のためにプロトコルを新規の設計している。また、CVP のクライアント・サーバ間で CVP のリクエストやレスポンスを通信するためのプロトコルとして、HTTP 上での通信手段意外に、直接 CVP プロトコルを通信する方法と Email で CVP リクエスト・レスポンスを通信する手法を定義している点が特徴である。現在（2003 年 1 月末時点）の最新バージョンは 3 であり、本文書はバージョン 3 を参考としている。

CVP では、一つのリクエスト・レスポンスのセットで DPV、DPD およびポリシー要求の 3 つの機能を提供している。DPV や DPD のレスポンスに認証パスの証明書や失効情報を含める場合、レスポンスデータが大きくなり、レスポンスに対する署名やその検証に掛かる時間が増大しがちである。CVP では、レスポンス全体に署名する代わりに、追加情報である証明書や失効情報のデータのハッシュ値のみをレスポンスの署名対象領域に含めることで、この問題を回避することができるようになっていることが大きな特徴である。また、複数の認証パスをユーザに回答する手段として、SCVP と同様に serverContextInfo を使用している。

3.4.2 DPV/DPD OCSP

DPV/DPD OCSP（DPV and DPD for OCSP）は、OCSP に対する最小限の拡張で、OCSP を DPV または DPD のためのプロトコルとして利用することを目指して、2003 年 1 月に提案されたばかりのプロトコルである。DPV/DPD OCSP では、OCSP リクエストや OCSP レスポンスに対する拡張領域（Extension）を定義しており、またそれらに格納する情報を利用形態に従って解説している。既存の OCSP を利用することで、最小限の実装の修正で、OCSP クライアントを DPV/DPD のクライアントとして拡張することを可能としている。

DPV/DPD OCSP では、リクエストに対する拡張領域として ExtendedOCSPRequest を、またレスポンスに対する拡張領域として ExtendedOCSPResponse を定義している。ExtendedOCSPRequest では、適用するバリデーションポリシーやパス発見ポリシーや、トラストアンカーの情報、レス

ポンスに格納すべき情報を指定するフラグ、および DPD レスポンスで格納可能なパスの上限値等を指定できる。これらの構造体を DPV で利用する場合、および DPD で利用する場合に使い分ける。また、リクエストする証明書の情報は OCSP の標準のリクエスト内にて指定する。ExtendedOCSPResponse では、パスを構成する証明書と失効情報のセットやタイムスタンプのトークン、適用したポリシーの OID 等を指定できる。DPV における証明書の検証結果は OCSP 標準の CertStatus を利用する。

DPV/DPD OCSP の特徴として、DPD で利用する際にクライアントは numPaths フィールドでレスポンスに含めるパスの上限を指定することができる。それにより、サーバ側でポリシーに合致するパスが numPaths を超えて発見された場合、numPaths 個のパス情報のみをクライアントに返答する。numPaths が指定されていない場合には、デフォルトとして一つのみパス情報を返答する。受け取ったレスポンスに目的とするパスが含まれない場合には、クライアントは numPaths の値を大きくして、再度リクエストを投げることで、最終的に目的のパスを見つける事ができる。

3.4.3 DVCS

DVCS（Data Validation and Certificate Status）は、TTP（Trusted Third Party）が否認防止サービス（Non-repudiation Services）を実現するためのプロトコルとして提案され、2001 年 9 月に RFC3029 として標準化された。DVCS は DPV/DPD を実現するためのプロトコルとして定義されたものではないが、DVCS の一つのサービスとしてある時刻の証明書の有効性を検証するサービスを提供しており、他のデータの所有権を証明するサービスや電子署名を検証するサービスなどと同一のプロトコルで証明書の検証も行うことができるため、DPV/DPD の候補として挙げられている。

DVCS サーバは以下の 4 つのサービスを提供し、その結果を保証する証明書としてデータ検証証明書（DVC : Data Validation Certification）を返答する。

- Certification of Possession of Data (CPD)
  要求者がデータの所有者であることを証明する。要求者は DVCS サーバに対してデータを送付し、DVCS サーバは要求者がデータをその時点で所有していたことを証明するデータ検証証明書を発行する。
- Certification of Claim of Possession of Data (CCPD)
  要求者がデータの所有権を有することを証明する。CPD との違いは、要求者はデータ自身ではなく、データのハッシュ値を DVCS サーバに送付する。
- Validation of Digitally Signed Document (VSD)
  電子署名の検証を行い、その結果を証明するデータ検証証明書を発行する。

電子署名の検証時には、電子署名に使用された証明書の検証も行う。

- Validation of Public Key Certificates (VPKC)
  指定された時刻における証明書の有効性を検証し、その有効性を証明するデータ検証証明書を発行する。

DVCS サーバは DVCS サービス規定とポリシーによって、どのような情報を使用して証明書の検証を行うのかが定義されている。クライアントはポリシーや追加情報を指定することで、証明書検証方法を指定することができる。

DVCS サーバでは検証が成功した場合、データ検証証明書を発行してクライアントに返答する。データ検証証明書には以下のような情報が含まれる。

- 検証結果
  認証または検証の結果
- シリアル番号
  各データ検証証明書には単純増加する一意なシリアル番号を付与する。
- 時刻情報
  検証を行った時刻を示すため、時刻時刻、またはタイムスタンプのトークンを付与する。
- ポリシー
  証明書の検証時に利用したポリシーを示す OID を付与する。

また、電子署名データ検証証明書には DVCS サーバによる電子署名が施される。その際利用する証明書には、DVCS サーバの署名用証明書であることを示す OID を Extended Key Usage Extension に付与されている必要がある。

データ検証証明書は他のユーザ（検証者）に対して、ある時刻にあるデータを保有していたことや、電子署名が正しいことを証明する目的で利用される。検証者は受け取ったデータ検証証明書自身を DVCS サーバを使用して検証することができる。

3.4.4 XKMS/X-KISS

X-KISS（XML Key Information Service Specificadtion）は、W3C において XML を利用した証明書管理のためのプロトコルである XKMS（XML Key Management Specification）の一部として標準化が進められている。W3C では現在 XKMS 2.0 に仕様が策定中であるが、そこではその他にも認証局に対して証明書の発行や失効を依頼する X-KRSS（XML Key Registration Service Specification）や、複数の証明書の一括発行を依頼する X-Bulk（XKMS Bulk Operation）が規定されている。

X-KISS は、XML ファイルに対する署名や暗号化の仕様として標準化された XML

Signature および XML Encryption において使用された証明書を検証するための仕組みとして規定されている。XML Signature や XML Encryption では、暗号化や署名に使用した鍵の情報を<KeyInfo>という構造で指定するが、そこで指定する情報としては証明書の名前（Subject Name）や公開鍵の値だけを指定して、証明書自体を格納しない場合がおおい。また、電子証明書は ASN.1 という言語で規定されているため、XML のクライアントで証明書を直接扱う場合、XML とは別に ASN.1 を解析できる必要があり、クライアントに負担が掛かる。そこで、X-KISS として<KeyInfo>から証明書を取得する手段や<KeyInfo>で指定された証明書の有効性を検証する仕組みをサーバサイドで実装し、クライアントとの間は XML で通信する仕組みを提供する。

X-KISS では、Tier0、Tier1、Tier2 の 3 種類のサービスレベルが用意されている。

Tier0 では、<KeyInfo>の<RetrievalMethod>で指定されたサーバから証明書を取得することのみを XKMS サーバに依頼する。

Tier1 では、<KeyInfo>を指定して、対応する鍵の名前と公開鍵を XKMS サーバが XML フォーマットで返す。

Tier2 では、Tier1 のサービスに加えて証明書の有効性確認まで XKMS サーバで行い、その結果も含めてクライアントに返す。

3.5 GPKI の証明書検証サーバ

GPKI ではブリッジモデルが採用されている。このブリッジモデルでは柔軟性がある反面、ブリッジ CA を介した署名検証を行うためには証明書パス構築・検証が複雑になる。そのため、府省側には各府省毎に証明書検証サーバを設けている。

証明書検証サーバはクライアントからの要求に応じて、クライアントが指定した証明書の検証（証明書パス構築・検証）を行い、クライアントへ結果を返す。こういった証明書検証サーバの必要性は、世界的にも認識されており IETF でも議論されているが、実例がない時期に OCSP のプロトコルを拡張することで、プロトコル等の仕様を定め実装したものである。

3.5.1 証明書検証サーバの機能

証明書検証サーバは概ね以下の機能を持つ。(表 3-2)

表 3-2 証明書検証サーバの機能

機能名	機能説明
認証パス構築・検証	GPKI 相互運用性仕様書に則った認証パスの構築・検証を行う。これにより検証したい証明書が信頼できる認証局から発行されたものかどうかが分かる。
有効性の確認	認証パス中の個々の証明書が有効なものかどうか確認する。CRL と OCSP（RFC2560）をサポートしている。
鍵ペアの生成	GPKI 証明書検証サーバの応答データは OCSP の署名を行うが、そのために署名鍵を生成するための鍵ペア生成機能を持つ。
証明書発行要求データの生成	府省認証局から GPKI 証明書検証サーバへの証明書を発行するために、PKCS#10 の形式で証明書発行要求データを作成する機能を持つ

図 3-4 に、GPKI 証明書検証サーバの、大まかな構成を示す。

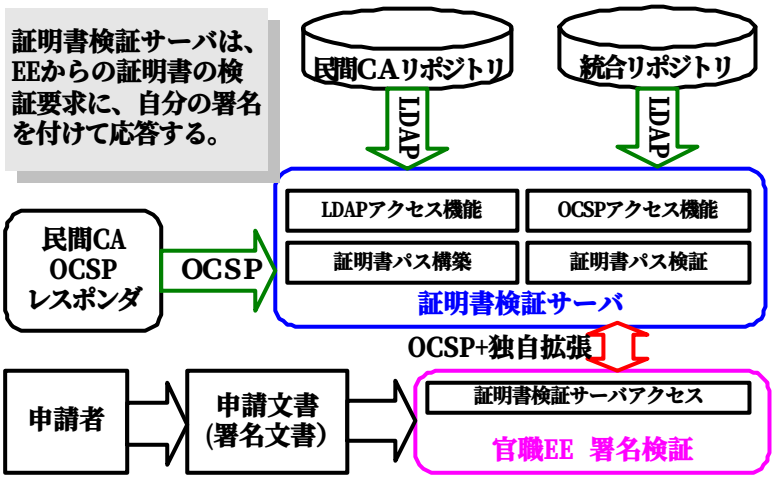


図 3-4 GPKI 証明書検証サーバの構成

図 3-5 に、GPKI 証明書検証サーバを、簡易なモデルの PKI で構成した場合の概念図を示す。

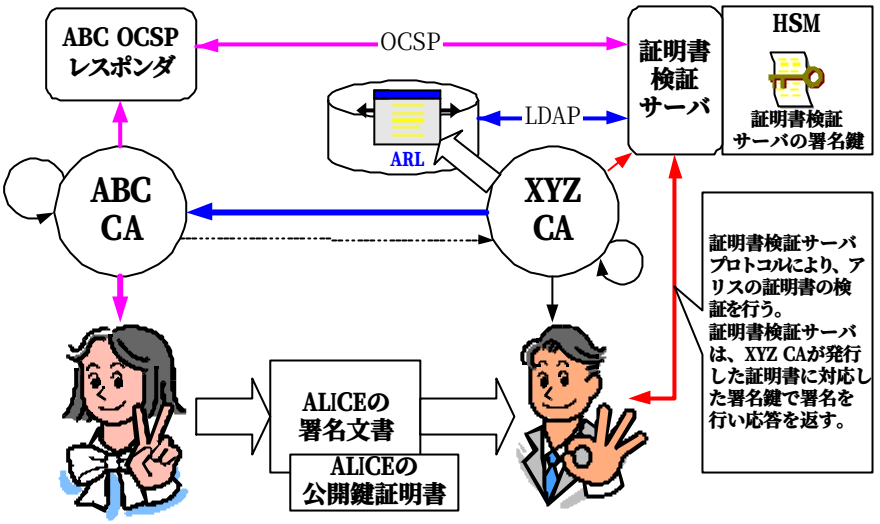


図 3-5 GPKI 証明書検証サーバの利用

3.5.2 GPKI の中での証明書検証サーバの構成

証明書検証サーバは、府省認証局毎に用意されている。証明書検証サーバは、応答データに署名を施すが、その署名鍵に対応した公開鍵に、府省認証局から証明書が発行される。

証明書検証サーバと、証明書検証サーバを利用するアプリケーションは、同じ PKI ドメインに存在しなければならないことに注意しなければならない。すなわち、ある府省のアプリケーションが証明書検証サーバを利用する場合、その証明書検証サーバからの応答の署名の検証は自分で行う必要がある。この署名は、アプリケーションが持つトラストアンカーと、証明書検証サーバのトラストアンカー（信頼ポイント）は同じである必要がある。

3.5.3 証明書検証サーバの応答メッセージの署名検証

証明書検証サーバからの応答には、署名が施されている。この署名の検証は、証明書検証サーバのクライアントが行う必要がある。また、この署名に対応した証明書の検証も、クライアントが行う必要がある。すなわち、リライングパーティの EE と同じトラストアンカーからの証明書チェーンの検証は、クライアント側で行う必要がある。

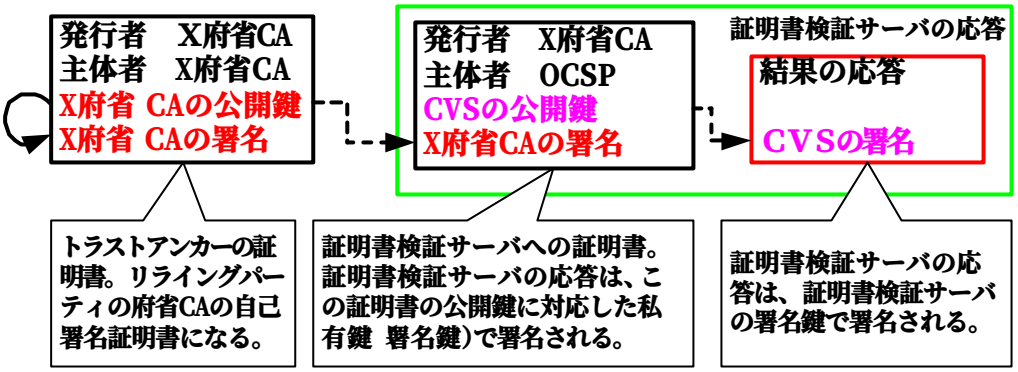


図 3-6 証明書検証サーバを使った認証パス

3.5.4 GPKI 証明書検証サーバへの証明書

GPKI 証明書検証サーバへのアクセスプロトコルは OCSP(RFC2560)を拡張したものを使用している。この RFC2560 には、OCSP レスポンダに発行される証明書プロファイルについての記述がある。GPKI 証明書検証サーバへの証明書の発行は、この RFC2560 に準拠した形で行われる。GPKI 相互運用性仕様書では以下のように規定している。

「GPKI 証明書検証サーバの証明書の extendedKeyUsage に id-kp-OCSPSigning を設定する必要がある。また、id-pkix-ocsp-nocheck 拡張を指定しても良いこととする。この場合、証明書の有効期間は一年以内程度とすることを推奨する。」

また、この証明書自体の失効チェックは、以下のように規定されている。

「署名の検証はレスポンスデータに添付される GPKI 証明書検証サーバの証明書により行うが、証明書内に id-pkix-ocsp-nocheck 拡張がない場合、証明書の検証もあわせて行うことを推奨する。
証明書検証としては、他の証明書に対する検証項目に加えて、extendedKeyUsage 拡張に id-kp-OCSPSigning を設定されているかどうかを確認する必要がある。また、GPKI 証明書検証サーバ証明書は、府省 CA が発行する CRL により失効されるため CRL を取得し有効性の確認を行うものとする。」

3.6 GPKI の証明書検証サーバプロトコル

GPKI 証明書検証サーバのプロトコルは、OCSP(RFC2560)をベースとしている。OCSP では、その要求メッセージや、応答メッセージに、X509v3 証明書のような任意の数の拡張を許している。拡張は、X509v3 証明書の拡張と同じく、その拡張を示す OID（オブジェクト識別子)、その拡張の解釈を強制するか否かを決めるフラグ（クリチカル、ノンクリチカル)、そして、その値から構成させる。

OCSP では、複数の要求を一つの要求メッセージに入れることができるが、GPKI 証明書検証サーバでは、一つの要求だけを受け付ける。また、応答も同じく複数の要求があった場合、複数の応答を一つの応答メッセージに入れることが出来るが、GPKI 証明書検証サーバでは一つの応答のみを返す。

OCSP の拡張には、個々の要求に対しての拡張（singleRequestExtensions）と、要求全体についての拡張（RequestExtensions）があるが、GPKI 証明書検証サーバアクセスプロトコルでは、singleRequestExtensions を使って拡張を行っている。RequestExtensions では、OCSP の標準的な拡張である NONCE を入れている。

また、応答に対しても、拡張（singleRequestExtensions）と、応答全体についての拡張（RequestExtensions）があるが、GPKI 証明書検証サーバアクセスプロトコルでは、singleRequestExtensions を使用して拡張している。また、要求と同じく、NONCE を使用している。

OCSP では、要求メッセージに署名を付けることができるが、GPKI 証明書検証サーバプロトコルでは、署名を付けた要求、及び、その検証はサポートしていない。

GPKI 証明書検証サーバアクセスプロトコル用の拡張のベースは次の OID を使用している。

OID：1.2.392.200010.10

これは、GPKI 証明書検証サーバの製造、開発元の日立製作所の OID を使用している。

3.6.1 証明書検証要求

(1) 要求の内容

tbsRequest	リクエスト	
version	バージョン	規定値 V1(0)
requestorName	リクエスタの名称	オプション
requestList	個別リクエスト 複数）	
reqCert	リクエストする証明書の情報	
hashAlgorithm	ハッシュアルゴリズム	
issuerNameHash	IssuerのDN名のハッシュ値	
issuerKeyHash	Issuerの公開鍵のハッシュ値	
serialNumber	証明書のシリアル番号	
singleRequestExtensions	個々のリクエストに対する拡張領域	*1
requestExtensions	リクエスト全体に対する拡張領域	nonceを使用
optionalSignature	リクエスタの署名	使用しない
signatureAlgorithm	署名アルゴリズム	
signature	署名データ	
certs	証明書 複数）	オプション

***1 証明書検証サーバプロトコルで大幅に,この拡張領域を使用している。**

図 3-7 証明書検証サーバへの検証要求

(2) OCSP 基本領域（OCSPRequest）

RFC2560 の規定に従っているが、requestExtensions に NONCE を必須としている。

(3) OCSP 拡張領域（singleRequestExtensions）

GPKI 証明書検証サーバのクライアントが以下の各拡張を指定する場合、各拡張を Critical で指定する。

表 3-3 証明書検証要求と OCSP の拡張領域

拡張領域名	機能説明	オプション
サブスクライバの証明書	OCSP では、リクエストする証明書の情報を基本領域に設定するが、GPKI 証明書検証サーバアクセスプロトコルでは、検証対象の証明書自体を送る。そして、これは OCSP の拡張で実現している。 GPKI 証明書検証サーバとしてこれは、必須のため拡張のフラグはクリチカルに設定される	必須
中間証明書	サブスクライバ側のルート認証局（トラストアンカー）からサブスクライバのエンドエンティティまでの証明書等、認証パスの一部となる証明書を指定する。ただし、GPKI 証明書検証サーバは、中間証明書を認証パス構築のヒント情報として利用する	オプション

	だけであり、必ずしも指定した証明書を含むパスを常に構築するわけではない。この拡張はオプションであり、複数指定可能である。また、複数指定する場合は、認証パスにおいてトラストアンカーに近い証明書から順に指定する。	
CA（トラストアンカー）の識別情報	リライングパーティが信頼するルート認証局（トラストアンカー）を識別するための情報として CA 証明書を送る。 この拡張はオプションである。省略した場合、検証依頼者（リライングパーティ）が信頼するルート認証局（トラストアンカー）は、GPKI 証明書検証サーバ側で設定したルート認証局となる。ただし、GPKI では、必ず自府省ルート認証局証明書を指定することとし、それ以外が指定された場合は、エラー（エラーコード：901）を返すとなっている。 GPKI/LGPKI では、GPKI 証明書検証サーバと、リライングパーティは、同一ドメインにいることが前提なので、省略すべきである。	オプション
満たすべきポリシ	リライングパーティが、認証パスに対して満たすべきポリシを要求する場合に、そのポリシの OID を指定する。満たすべきポリシをが指定された場合、GPKI 証明書検証サーバは、構築した認証パスが指定されたポリシを満たすかどうかを検査する。 この拡張はオプションであり、複数指定可能である。 現在の GPKI/LGPKI 環境においては、官職の証明書ポリシと同じ OID をひとつ指定するのがよいと思われる。	オプション
require-explicit-policy の初期値	リライングパーティが、認証パスに対して require-explicit-policy を要求する場合に、範囲を指定する。require-explicit-policy が指定された場合、GPKI 証明書検証サーバは、require-explicit-policy の範囲外の証明書にポリシが存在することを検査する。 GPKI では、満たすべきポリシが指定された場合、この Extension は常に”0（ゼロ）”を指定することとする。	オプション
inhibit-policy-mapping の初期値	検証依頼者（リライングパーティ）が、認証パスに対して inhibit-policy-mapping を要求する場合に、範囲を指定する。inhibit-policy-mapping が指定された場合、GPKI 証明書検証サーバは、inhibit-policy-mapping の範囲外の証明書にポリシマッピングが存在しないことを検査する。 GPKI ではポリシマッピングを必須とするため、この Extension は常に指定しないこととする。	オプション
応答フォーマット	リライングパーティが、証明書検証応答として返却すべき情報のレベルを指定する。省略時は証明書の検証結果のみ返却する。指定可能な情報レベルを以下に示す。 0: 証明書の検証結果のみ返却する（デフォルト） 1: 証明書の検証結果に加えて、認証パス及び CRL/ARL を返却する。 GPKI 証明書検証サーバのクライアントは、GPKI 証明書検証サーバに証明書の有効性を問い合わせるのだが、その根拠となるデータを保存しなければならない場合などに、GPKI 証明書検証サーバからデータ一式を要求することができる。	オプション

3.6.2 証明書検証応答

(1) 応答の内容

responseStatus	レスポンスの状態	successful (0) malformedRequest (1) internalError (2) tryLater (3) (4は未使用) sigRequired (5) unauthorized (6)
responseBytes	レスポンス本体	オプション
responseType	レスポンスの形式	id-pkix-ocsp-basic
response	レスポンスデータ（OCTET STRING）	
tbsResponseData	レスポンス（全体）	
version	バージョン	規定値 V1(0)
responderID	レスポンダのID	
byName	レスポンダの名称	選択(String)
byKey	レスポンダの公開鍵のハッシュ値	選択(OCTET STRING)
producedAt	レスポンス生成時刻	
responses	個別レスポンス（複数）	
certID	証明書のID	
hashAlgorithm	ハッシュアルゴリズム	
issuerNameHash	IssuerのDN名のハッシュ値	
issuerKeyHash	Issuerの公開鍵のハッシュ値	
serialNumber	証明書のシリアル番号	
certStatus	証明書の状態	
good	good	選択(NULL)
revoked	revoked	選択(SEQUENCE)
revocationTime	失効時間	
revocationReason	失効理由	オプション
unknown	unknown	選択(NULL)
thisUpdate	状態の有効期間（from）	
nextUpdate	状態の有効期間（to）	使用しない
singleExtensions	個々のレスポンスに対する拡張領域	＊2
responseExtensions	レスポンス全体に対する拡張領域	Nonceを使用
signatureAlgorithm	署名アルゴリズム	
signature	署名データ	
certs	証明書（複数）	オプション

図 3-8 OCSPの応答

(2) OCSP基本領域（OCSPResponse）

RFC2560の規定に従う。ただし、responseExtensionsにnonceを必須とする。また、nextUpdateは使用しない。

GPKI証明書検証サーバの応答では、OCSPの基本領域の多くの情報は意味を持たない。

(3) OCSP 拡張領域（singleExtensions）

GPKI 証明書検証サーバは「証明書検証結果」は Critical、その他の拡張は Non-Critical で応答する。

表 3-4 証明書検証応答と OCSP の拡張領域

拡張領域名	機能説明	オプション
証明書検証結果	OCSP レスポンダでは、失効チェックの結果を OCSP の基本領域で応答するが、GPKI 証明書検証サーバの結果は、OCSP の拡張領域を使用して、証明書検証（認証パスの構築及び検証）の結果コードが設定される。GPKI 証明書検証サーバの基本的な機能なので、クリチカルで応答メッセージには必ず含まれる。 証明書検証の結果コードは次の通り。 0：認証パスの構築が成功し検証結果が正しい（good） 101：認証パス構築不可 202：認証パスに署名が不正である証明書が含まれる 203：認証パスに失効した証明書が含まれる 204：認証パスにポリシが一致しない証明書が含まれる 205：認証パスに制約に違反している証明書が含まれる 206：認証パスに OCSP での CertStatus が unknown と応答される証明書が含まれる 901：GPKI 証明書検証サーバ側で要求の受付けを拒否した 902：要求がタイムアウトとなった	必須
認証パス	構築した認証パスの全証明書を設定する。証明書検証要求の応答フォーマットで「返却」が要求されている場合に限り応答メッセージに含まれる。	オプション
CRL/ARL	構築した認証パスの証明書について、証明書発行者（OCSP レスポンダを除く）が発行した失効リスト（CRL/ARL）を設定する。証明書検証要求の応答フォーマットで「返却」が要求されている場合に限り応答メッセージに含まれる。	オプション
OCSP レスポンダからの応答	構築した認証パスの証明書について、証明書発行者（OCSP レスポンダに限る）から取得した OCSP 応答情報を設定する。 証明書検証要求の応答フォーマットで「返却」が要求されている場合に限り応答メッセージに含まれる。	オプション
適合したポリシ	構築した認証パスの証明書について、検証結果として有効なポリシが設定される。証明書検証要求の応答フォーマットで「返却」が要求されている場合に限り応答メッセージに含まれる。 適合したポリシは、プロトコル上、複数のポリシが返却される可能性がある。しかし、現状の GPKI のブリッジ認証局は、ひとつクラスのポリシマッピングしか行っていないため、ひとつ以上のポリシが返却されることはない。	オプション

	将来的には、複数のポリシマッピングがあるかもしれない。その場合には、サブスクライバ側の証明書ポリシが、自ドメインのポリシにマッピングされた中で、そのポリシによりアプリケーションの振る舞いを変えるといったことも考えられる。	

4 証明書パス構築・パス検証のテストクライテリアの動向

4.1 証明書パス構築・パス検証のテストクライテリア

ある証明書パスを構築・検証する場合には、どのような PKI アプリケーションであっても同じ検証結果を得られる必要がある。しかしこれまでの章で述べてきたように、証明書パス構築・パス検証は非常に複雑であり、PKI アプリケーションによっては異なる検証結果を返すことがあり得る。

このため近年世界各国では、どのような PKI アプリケーションを使っても同じ検証結果が得られるように、いくつかの相互運用実験が行われている。これらの相互運用実験では、まず最初にテストクライテリアを設計し、次に各 PKI アプリケーションがこれらのクライテリアに準拠したテスト結果を返すかどうかを検証することで、PKI 相互運用性を維持してる。

本章では、諸外国で実施されている主な証明書パス構築・パス検証についての相互運用実験の事例を説明する。**7 章の相互運用テストスイートや 8 章の GPKI テストケースは、これらの各テストクライテリアを参考に、GPKI に沿ったテストクライテリアを目指して設計を行った。**

4.2 EEMA pkic

4.2.1 活動紹介

PKI チャレンジ(pkic)は、EEMA イニシアチブが推進する、PKI の相互運用を目指したプロジェクトである。EEMA 自体は 32 ヶ国、273 団体から構成される業界団体で、pkic には Entrust, Baltimore のような PKI ベンダー、Globalsign のような認証プロバイダ、UK Post のようなユーザ、KPMG のような監査コンサルタント、その他大学など幅広い組織が参加している。

pkic は 2001 年 Q1 から 2 年間のプロジェクトで、図 4-1 にあるような「**異なるポリシを持つ組織間でセキュアなメッセージ交換**」を実現するためのフレームワーク作成を目的としている。

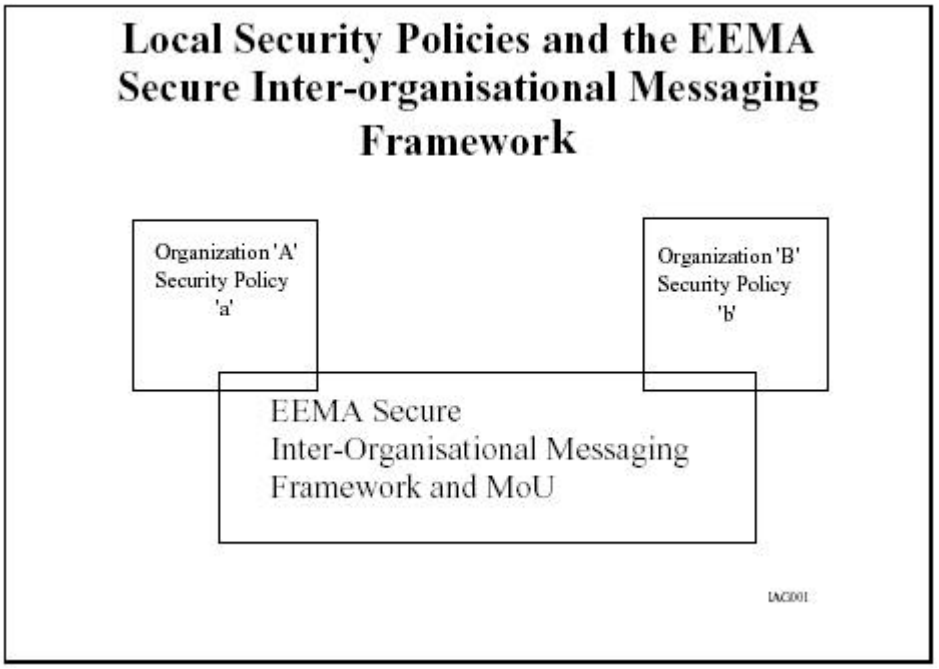


図 4-1 EEMA における pkic 相互運用実験

pkic は、このフレームワークを確立するにあたって以下の３点を具体的な目標として据えた。

- セキュアな必須(Mandatory)アプリケーションの開発
- PKI でディレクトリ(X.500, LDAP)を扱う際の問題点の分析
  A) ディレクトリ(X.500, LDAP)自身の仕様が抱える問題点
  B) PKI で利用する際に発生する問題点
- PKI の相互運用性に関するクライテリアの確立

そして、これらの目標を達成・検証するために、大きく分けて以下の２つの作業が進められることになった。

a) テストクライテリアの作成とモデル実験
b) 実験に基づくデモンストレーション

4.2.2 テストクライテリアの特徴

(1) テストクライテリアの着眼点

pkic では、PKI 相互運用性を保つためには、以下の要素について検討する必要があるとしている。

- スマートカード
  - RA からの証明書発行

  - PKI アプリケーションとの I/F
- CA 間の証明書発行
  - マニュアル相互認証
  - オンライン相互認証
- EE 証明書登録・発行手続き
  - Web ベースまたはメールベースでのオンライン発行
  - CA 側で鍵生成する場合
  - EE 側で鍵生成する場合
- オンライン証明書検証 I/F
  - CA から VA への証明書ステータスの提供
  - PKI アプリケーションから VA への検証要求
  - VA 間プロキシ
- メタディレクトリ I/F
  - ディレクトリ間の連携
  - PKI アプリケーションによる証明書・CRL の取得
- セキュアアプリケーション
  - セキュア E-mail(S/MIME)

これらを解決するために、pkic では以下の各項目について、**相互運用要件**と**準拠すべき(または参照すべき)標準仕様**を示している。

(a) プロファイル

メッセージングアプリケーションとして S/MIME を想定しているため、**S/MIME** や **QualifiedCertificate** をリファレンスとした証明書プロファイルを定義している。
CRL プロファイルについては、一般的に **X.509 v2 CRL** や **RFC2459** をリファレンスとした一般的なものとなっている。
また、ディレクトリスキーマについては**一般的なスキーマのみ**を使用、としており、実際には **RFC2256**(LDAPv3 Schema)に準拠している。

(b) アプリケーション

デジタル署名された電子メールを交換する仕組みとして **S/MIME**, **CMS** をリファレンスとしている。特徴としては**デルタ CRL** を用いた検証、相互認証を含め少なくとも **3 レベル以上の認証パス**の検証を要件として挙げている。
またスマートカードについては、その相互運用性は pkic の主目的ではないとし

ながらも、PKCS#11 を想定してテストを行うとしている。

(c) 証明書発行

pkic では、異なるポリシを持つ組織間にまたがった形での３種類の証明書(相互認証証明書、下位 CA 証明書、EE 証明書)の発行要件について検討している。
これらの証明書を発行する際のメッセージフォーマットとして PKCS#10, CRMF(Certificate Request Message Format), PKCS#7, CMS(Cryptographic Message Syntax)についてそれぞれ検討している。
また、メッセージ転送プロトコルとして CMC(Certificate Management Message over CMS), CMP(Certificate Management Protocol)について検討している。

(d) ディレクトリサービス

ディレクトリに関しては、pkic では相互運用性に関する問題提起をするにとどまっている。

(e) 検証サービス

pkic では検証サービス(VA)として OCSP レスポンダを想定しており、その VA に対する失効情報の配布プロトコルとして LDAP(pull)と HTTP(push)を想定している。また VA の用法として proxy/forwarding についても検討している。

http://www.eema.org/pki-challenge/criteria.asp

(2) テストシナリオ

pkic では、以下のカテゴリでテストシナリオを用意している。証明書パス構築・パス検証に関して特徴的なのは、EE モデルと VA モデル、さらには単純 VA モデルとプロキシ VA モデルまで明確に定義しており、いずれの場合も、証明書パス構築・パス検証の結果は同じものにならなければならない点である。

(a) 他の CA との相互認証(TG1)

pkic ではリファレンスとなる CA を明確にしている。テスト対象となる CA は、全てこのリファレンス CA と相互認証することで、客観的な相互接続性を検証することができる。
リファレンス CA と相互認証する際には、以下のテストを実施することになる。

- マニュアルでのオフライン相互認証
- オンラインでの自動相互認証
- MAC を用いた CMP での相互認証
- 署名を用いた CMP での相互認証

(b) EE への証明書発行(TG3)

pkic では、EE 自身で鍵生成する場合と、CA で EE の鍵生成する場合のそれぞれについて、以下の発行パターンを検証している。

- クライアントを用いたオンライン証明書発行
- マニュアルでの証明書発行

(c) 下位 CA に対する証明書発行(TG2)

テスト対象となる CA は、全てこのリファレンス CA と互いに下位 CA 証明書を発行し合うことで、客観的な相互接続性を検証することができる。

この時、テスト CA とリファレンス CA との関係は以下の 2 通りが考えられる。

- テスト CA がリファレンス CA に下位 CA 証明書を発行するケース
- リファレンス CA がテスト CA に下位 CA 証明書を発行するケース

テスト CA は、それぞれについて以下のテスト項目を検証することになる。

- マニュアルでの証明書発行
- オンライン(CMC)での証明書発行
- オンライン(CMP)での証明書発行

(d) 証明書の検証(TG4, TG5)

pkic では、証明書の検証について、特に失効検証の部分に着目して以下の 2 つのテストシナリオを用意した。

- EE モデル(必須)
  証明書の失効検証に必要な CRL を EE 自身が取得して失効検証する。
- VA モデル(オプショナル)
  EE は証明書の失効状態を OCSP レスポンダに問い合わせ、OCSP レスポンダは失効検証した結果を EE へ返信する。

(i) 証明書パス構築・パス検証(TG4)

本テストでは EE モデル、VA モデルそれぞれについて、以下の証明書検証を行っている。従って pkic のクライテリアでは、EE モデルであっても VA モデルであっても、同じ検証結果が得られる必要がある。

- 正常な証明書
- 失効している証明書
- 有効期限の切れた証明書
- ポリシが一致しない証明書
- パス構築できない証明書

(ii) OCSP を用いた証明書検証(TG5)

本テストでは、特に VA モデルについて、さらに詳細なシチュエーションを想定して検証を行っている。従って pkic のテストクライテリアでは、単純 VA モデルであってもプロキシ VA モデルであっても、同じ検証結果が得られる必要がある。

- 単純 VA モデル
  VA は失効情報を得るためにディレクトリから CRL を取得する。
- プロキシ VA モデル
  VA は失効情報を得るために、他の VA へ問い合わせる。

4.3 DoD BITS

4.3.1 活動紹介

(1) 背景と目的

米国政府はブリッジ CA モデルによる政府と民間との PKI の相互運用性の可能性に着目し、政府とベンダは協調してブリッジ CA モデルの PKI 環境を開発している。また、その可能性の検証事業として、NIST を中心とした Federal Bridge CA における「Electronic Messaging Association (EMA) Challenge 2000」、DoD の Bridge CA における「Bridge Certification Authority (BCA) Technology Demonstration」などが実施されている。このうち、DoD BCA Demonstration と、その成果のひとつである、テストスイート DoD BITS(BCA Interoperability Test Description)について、ここで紹介する。DoD BCA Demonstration の成果報告は以下のサイトに公開されている。

http://www.anassoc.com/BCA.html

http://bcatest.atl.getronicsgov.com/index.htm

DoD BCA Demonstration は Phase Ⅰ と PhaseⅡに分けて実施された。Phase Ⅰは 1999 年から 2000 年に掛けて実施され、PhaseⅡは 2000 年から 2001 年に掛けて実施された。

Phase Ⅰの要件は以下のようである。

- 異なる PKI ベンダよりなる 3 つの PKI ドメイン間での BCA を介した信頼関係の確立
- 3 つのドメイン間でのディレクトリアクセス
- 「border directory」の確立.
- BCA を介したパス構築
- BCA をはさんだ署名データの通信
- 署名アルゴリズムは md5withRSAEncryption のみ
- 証明書ポリシ関連の処理は含まない。

PhaseⅡでは Phase Ⅰに加えて以下がポイントとなった。

- 異なる PKI ベンダよりなる５つの PKI ドメイン間での BCA を介した信頼関係の確立
- 5 つのドメイン間でのディレクトリアクセス
- BCA をはさんだ署名データ、暗号データの通信
- 署名アルゴリズムの混在
- 証明書ポリシに関わる処理を行う。

このBCA Demonstrationを通し、以下のような目的と要件を満たすことが実証されている。

- 既存の CA 製品で、若干の改良で、階層型、メッシュ型 PKI を含む BCA 環境を実現できる。
- 証明書パス構築のフリーのリファレンス・ライブラリを提供する。
- S/MIME バージョン 3 を使った電子メール環境において、証明書パス構築・パス検証を行うソフトウェアを提供すること
- **市販の既製品（COTS）が利用可能であること**
- BCA 環境で電子メールクライアントにて電子署名が適切に処理できることの検証
- BCA 環境で電子メールクライアントにて暗号化が適切に処理できることの検証
- 暗号アルゴリズムの適合性の検証

(2) PhaseⅡデモンストレーションの参加企業と製品

- **ソフトウェア開発**

Getronics Government Solutions：

X.500 **証明書管理ライブライ** (CML).

S/MIME **ライブラリ**(SFL).

アクセスコントロールライブラリ(ACL).

Entrust CygnaCom：**証明書パス構築ライブラリ**(CPDL).

- **ＰＫＩ製品**

Baltimore Technologies：「UniCERT™」

Entrust Inc.：「Entrust/Authority 5.0.2.」

SETECS：「OnePKI™」

SPYRUS：「S^2CA 」（DoD SDN **準拠の** PKI **製品**）

Motorola：「CAW」 DoD SDN(Secure Data Network)**準拠の認証局ワークステーション** 、Fortezza **の初期化などにも使われる**。

– CygnaCom：「Bridge CA」NIST **の** MISPC(Minimum Interoperability Specification for PKI Components)**のリファレンス実装を元に開発**

- **ディレクトリ製品**

Entegrity：X.500 「Safepages directories」**管理ツール**、border directory **と同期ツール**、Web **ベースのデモツール**

Critical Path：「LiveContent directory servers」

- **ユーザ・アプリケーション**

Baltimore Technologies：「MailSecure security enhancements for MS Outlook」

Entrust Inc.：「Entrust/Express security enhancements for Microsoft Outlook」

CygnaCom：「Qualcomm Eudora」**へのプラグイン**（SFL, CML, CPDL,ACL **を元に開発**）

(3) PhaseⅡデモンストレーションのモデル

次図のように、DoD **内に位置する** Bridge CA **と、**DoD **内の４つの** PKI **ドメイン** (Armed Force：**軍隊**、DISA：**国防情報システム局**、DFAS：**国防財務会計局**、DLA：**国防兵站局**)**が相互認証し、**DoD **外部の** PKI **ドメインである司法省の** CA **とも相互認証ししている**。Armed Force **内では、さらにその下位に陸軍、空軍、海軍の** CA

が位置する。DISA や DFAS は多段の CA 階層を持ち、DLA はポリシを異にする 2 つの CA よりなる。司法省の CA は、FBI、沿岸警備隊の CA とそれぞれメッシュ状に相互認証している。それぞれの PKI ドメインの持つ CA の数、ポリシを表 4-1 PKI ドメインと証明書ポリシに示す。また、相互認証におけるポリシのマッピング状況を図 4-3 証明書ポリシのマッピングに示す。この図でわかるように、沿岸警備隊のポリシはマッピングが行われていない。沿岸警備隊のポリシのみを持つ証明書は他のドメインでは拒否されるはずだ。

構成で各 PKI ドメインには異なる PKI ベンダの製品が配置されている。

ディレクトリサーバ間の同期は“border directory”というコンセプトで内部と外部のアクセス可能な内容を区別することにより、DoD の内部のディレクトリサーバと DoD 外部のディレクトリサーバが同期することにより、検証に必要なデータは BCA のディレクトリサーバが全てを統合する形である。

このような環境で各 PKI ドメインのユーザが S/MIME を使った電子メールにて電子署名あるいは暗号を送受信して検証が行なわれた。

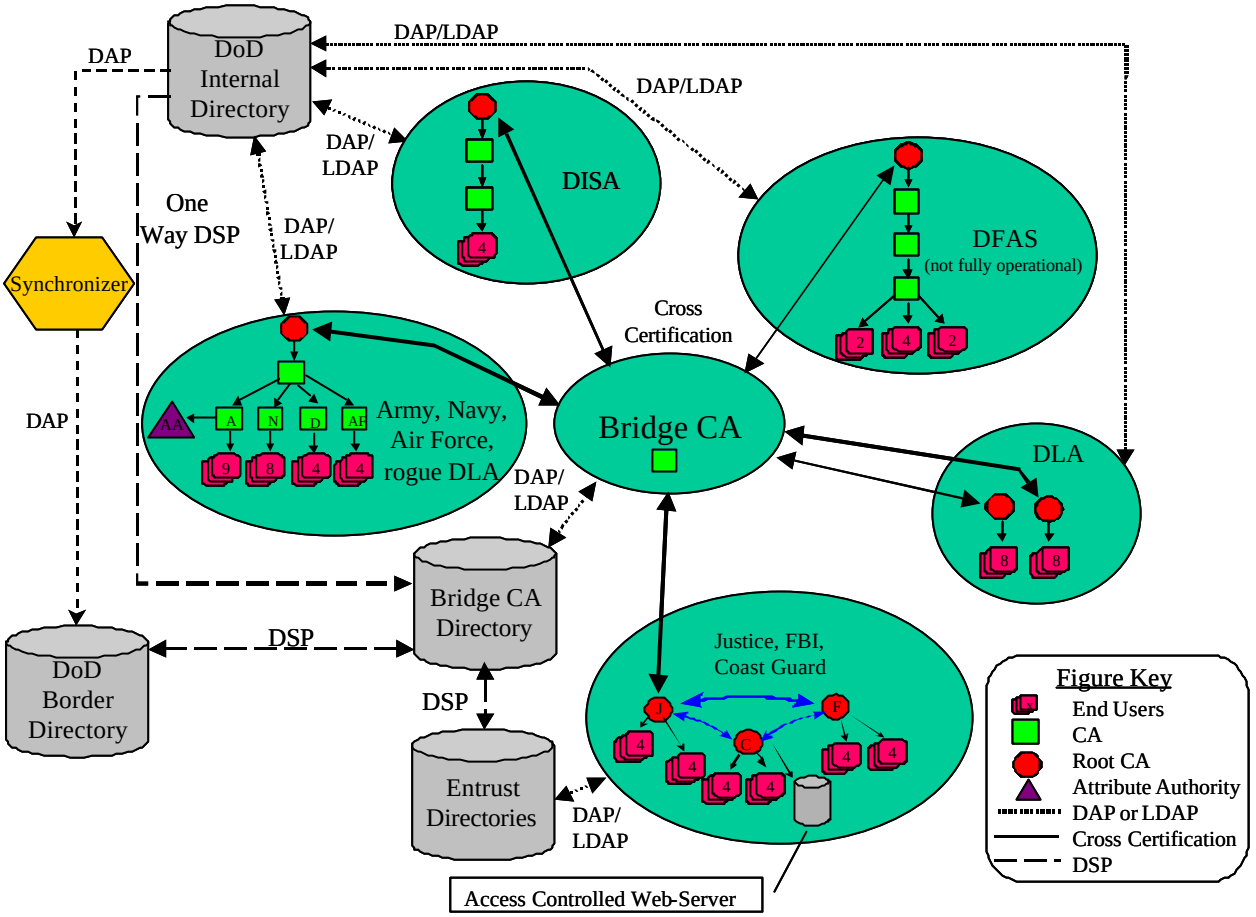


図 4-2 PhaseⅡDemonstration の PKI モデル

表 4-1 PKI ドメインと証明書ポリシ

ドメイン	ベンダ	タイプ	CA数	証明書ポリシ
Defense Logistics Agency (DLA)	Baltimore Technologies	階層	2	Logistics High Logistics Medium
Law Enforcement: Justice,Coast Guard, FBI	Entrust	メッシュ	3	Law Enforcement High Lawn Enforcement Medium Coast Guard High
Armed Forces (Army, Navy, Air Force)	SPYRUS	階層	6	DoD Class 3 DoD Class 4
Defense Information System Agency (DISA)	Motorola	階層	2	DISA Policy
Defense Finance &	SETECS	階層	4	Finance Policy

Accounting (DFAS)				
BCA	CygnaCom	BCA	1	BCA High BCA Medium

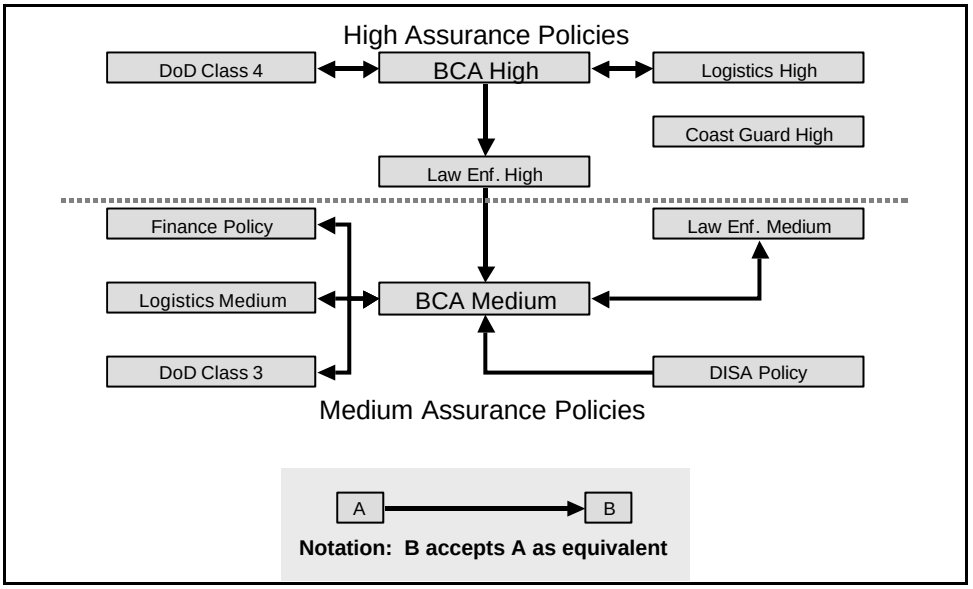


図 4-3 証明書ポリシのマッピング

(4) テストスイート"BITS"

PhaseⅡデモンストレーションの成果を基に、RFC 2459 に規定された機能のうち以下の点に重点を置いたテストスイートが BITS（Bridge Certification Authority Interoperability Test）として公開されている。

- 名前制約 (許可と除外ともに)
- 証明書ポリシ
- 証明書ポリシマッピング
- CRLにより失効検証
- 相互認証
- CRL-DP処理
- 証明書パス中に複数署名アルゴリムの混在

PhaseⅡBCA Demonstration で構築された PKI 環境から、DFAS を除く PKI ドメインによる環境を前提としたデータが用意されている。データは、以下のサイトより入手可能である。

http://bcatest.atl.getronicsgov.com/bits.htm

また、オンラインによるパス構築が可能であり、以下のディレクトリサーバが現在、稼動している。

ホスト名：bcatest.atl.getronicsgov.com

ポート：389

配布されているテストデータはディレクトリごとに以下のような内容となっている。

- certpairs：クロスサーティフィケートペア
- certs ：CA、BCA、EE の証明書
- crls ：CRL(次回更新日は 2003 年 10 月)
- pkcs12：全証明書について秘密鍵が配布される（パスワード:password）
- smime：テストにて生成された S/MIME メールメッセージ

(5) BITS テスト手順

別の PKI ドメインとの S/MIME メールメッセージの送受信を想定して行われ、S/MIME の処理として以下のように分類されたテストを行う。

署名処理： テストケース数 39 コ

検証処理：テストケース数 48 コ

暗号化処理：テストケース数 26 コ

復号処理：テストケース数 6 コ

署名および暗号化処理：テストケース数 5 コ

署名処理以外は、以下の検証初期パラメータをセットした上で、パス構築、パス検証をあわせて行う。

- トラストポイント証明書のセット
- 受け入れポリシのセット
- Initial-explicit-policy のセット
- Initial-policy-mapping-inhibit のセット
- Initial-inhibit-policy のセット

配布される証明書で使われているオブジェクト ID を次表に示す。

表 4-2 証明書ポリシのオブジェクト ID

オブジェクト ID	登録名	識別名
2.5.29.32.0	*any-policy*	*any-policy*
2.16.840.1.101.2.1.12.1.1	Infosec test-certificate-policy-1	DISA
2.16.840.1.101.3.2.1.48.1	NIST test-policy-1	DoD Class 3
2.16.840.1.101.3.2.1.48.2	NIST test-policy-2	DoD Class 4
2.16.840.1.101.3.2.1.48.3	NIST test-policy-3	Law Enforcement Medium
2.16.840.1.101.3.2.1.48.4	NIST test-policy-4	Law Enforcement High
2.16.840.1.101.3.2.1.48.5	NIST test-policy-5	Coast Guard High
2.16.840.1.101.3.2.1.48.6	NIST test-policy-6	Bridge CA Medium
2.16.840.1.101.3.2.1.48.7	NIST test-policy-7	Bridge CA High
2.16.840.1.101.3.2.1.48.8	NIST test-policy-8	Finance
2.16.840.1.101.3.2.1.48.9	NIST test-policy-9	Defense Logistics Medium
2.16.840.1.101.3.2.1.48.10	NIST test-policy-10	Defense Logistics High

5 Java による証明書パス構築・パス検証の実装の説明

5.1 Java JDK 1.4 におけるパス構築/検証　概説

JAVA はもともと PKI 技術の利用を考えられて作られており、証明書の扱い、検証の機能を持った作りとなっている。

JDK の最新バージョンである JAVA JDK 1.4(以下 JDK 1.4 と略する)の証明書関連のフレームワークの実装は RFC3280 ベースのパス構築/検証アルゴリズムを実装しており単純なシングルドメイン PKI での動作およびマルチドメイン PKI においても証明書の内容によって問題なく動作する。しかしながら、証明書の失効検証処理に関しては、十分とは言えず GPKI のように CRL 内の IssuerDistributionPoint などの CRL 関連の処理に関しては処理できない。もともと RFC3280 では CRL 内の IssuerDistributionPoint の扱いに関しての記述が不明確であり(昨年の JNSA Challenge PKI 2001 にて報告済み)、この部分の検証が出来ないのは RFC3280 の実装に忠実であるという言い方も出来る。

しかしながら、単純なシングルドメイン PKI では CRL の配布に関しての不明確のままでも処理可能であるが、マルチドメイン PKI では CRL の配付に関して不明確のままでは証明書の検証を行うことが出来ない。

また、SUN の実装では証明書の扱いに関して不十分な点もままあり RFC3280 を準拠するだけの実装であるともいえる。

これらのパス構築/検証の機能は、X.509 証明書処理部分の拡張する形で java.security.cert 以下に実装されている。

パス構築/検証部分の実装は、以下に示す 3 つの部分に仮想化されている。

a) CertStore
   証明書の格納および検索、必要に応じて取得を行う。JDK 1.4 において証明書の取得は CertStore に対して取得の要求を行うことにより行う。CertStore は要求に応じて証明書を検索し取得を行う。
b) PathBuilder
   証明書のパス構築を行う。
c) 証明書の検証処理を行う。

これらのインターフェイスを組み合わせることにより、証明書のパス構築/検証を

行うこととなる。

最低限の RFC3280 ベースのパス検証のためのコードを以下に示す。

```
import java.io.IOException;
import java.security.InvalidAlgorithmParameterException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertPath;
import java.security.cert.CertPathValidator;
import java.security.cert.CertPathValidatorResult;
import java.security.cert.CertPathValidatorException;
import java.security.cert.PKIXBuilderParameters;
import java.security.cert.PKIXCertPathValidatorResult;
import java.util.Set;
import java.util.HashSet;

static public final String DEFAULT_IMPLEMENTATION_NAME    = "PKIX";
public CertPathValidatorResult validate(
   String         implementationName,
   Set            trustAnchor,
   CertPath       path,
   boolean        anyPolicyInhibit,
   boolean        policyMappingInhibit,
   boolean        explicitPolicy,
   Set            initialPolicy)
throws CertPathValidatorException {
   PKIXBuilderParameters     params = null;
   CertPathValidator         validator = null;
   try { // パラメータの用意をします
      params = new PKIXBuilderParameters(trustAnchor, null);
      params.setAnyPolicyInhibited(anyPolicyInhibit);
      params.setExplicitPolicyRequired(explicitPolicy);
      params.setPolicyMappingInhibited(policyMappingInhibit);
      params.setInitialPolicies(initialPolicy);
      // 適当なリポジトリを設定してください
   } catch(InvalidAlgorithmParameterException iape) {
      throw new CertPathValidatorException(iape);
   }
```

```
    try {  // Validator を動かし、結果をもらいます
      validator = CertPathValidator.getInstance(implementationName);
      return validator.validate(path, params);
    } catch(NoSuchAlgorithmException nsae) {
      throw new CertPathValidatorException(nsae);
    } catch(InvalidAlgorithmParameterException iape) {
      throw new CertPathValidatorException(iape);
    }
}
```

図 5-1 Java による RFC3280 ベースの簡単なパス検証のコード

実質 20 行に満たないコードでパス構築/パス検証を行うことが出来る。多少、凝ったパス検証を行うにしても以下に示すサンプルコードで実現できる。

```
import  java.io.IOException;
import  java.security.InvalidAlgorithmParameterException;
import  java.security.NoSuchAlgorithmException;
import  java.security.cert.CertPath;
import  java.security.cert.CertPathValidator;
import  java.security.cert.CertPathValidatorResult;
import  java.security.cert.CertPathValidatorException;
import  java.security.cert.PKIXBuilderParameters;
import  java.security.cert.PKIXCertPathValidatorResult;
import  java.util.Set;
import  java.util.HashSet;

// パス生成オブジェクトを生成
CertPathBuilder  builder  =
CertPathBuilder.getInstance("GPKI");
CertPathValidator  validator  =
CertPathValidator.getInstance("GPKI");
// 検査対象となる証明書を取得
X509Certificate  targetCertificate  =
getCertificate(certificateStore, argv[1]);
// トラストアンカーとする証明書を取得
X509Certificate  trustCertificate  =
```

```
getCertificate(certificateStore, argv[2]);
// 構築パラメータ生成
CertPathParameters buildParameter = getCertPathParameters(
      argv[0], validator, trustCertificate,
      targetCertificate,  certificateStore);
try {
  CertPathBuilderResult buildedPathResult =
    builder.build(buildParameter);
  certificatetionPath = buildedPathResult.getCertPath();
  printCertPath(certificatetionPath);
  if(buildedPathResult instanceof PKIXCertPathBuilderResult) {
    PKIXCertPathBuilderResult result =
      (PKIXCertPathBuilderResult)buildedPathResult;
    PolicyNode policy = result.getPolicyTree();
    if(policy != null) {
      printPolicy(policy);
    }
  }
} catch(Exception e) {
  e.printStackTrace();
}
try {
  CertPathValidatorResult result =
    validator.validate(certificatetionPath, buildParameter);
  printCertPath(certificatetionPath);
  if(result instanceof PKIXCertPathValidatorResult) {
    PKIXCertPathValidatorResult r =
      (PKIXCertPathValidatorResult)result;
    PolicyNode policy = r.getPolicyTree();
    if(policy != null) {
      printPolicy(policy);
    }
  }
} catch(Exception e) {
  e.printStackTrace();
}
```

図 5-2 Java によるやや凝ったパス検証のコード

いずれのサンプルにおいても、パス構築/パス検証の知識があれば JDK 1.4 においてパス構築/パス検証を行うことが出来る。

しかしながら、SUN による JDK 1.4 の実装（およびフレームワーク）には不便な点もある。

第一は、JAVA の証明書操作ライブラリィの制限によるものかもしれないが、証明書に含まれている各種情報(特に X.509v3 拡張)に対しての操作に制限が多いことである。
例えば、CRLDistributionPoint などといった拡張の情報を取得しようとすると Java.security.cert.X509Extension .getExtensionValue(oid)という形で取得せざるを得ない。

5.2 サンプル実装の説明

5.2.1 API 仕様について

実装を行う上で「SUN 実装のプロバイダと同一のインタフェイスを持っている」ようにした。これはプロバイダが動作する上で以下のクラスを直接・間接的に使用することを意味しており、互換性を持たせる事を目的としている。

(1) java.security.cert.CertStore
(2) java.security.cert.PKIXBuilderParameters
(3) java.security.cert.PKIXCertPathBuilderResult
(4) java.security.cert.PKIXCertPathChecker
(5) java.security.cert.PKIXCertPathValidatorResult

ただし、SUN の実装がサポートしていない範囲の規格に関して、設定項目などが不足している場合は、上記のクラスを継承し新しいクラスも使用できることとする。

(1) java.security.cert.CertStore

このクラスはJAVAのパス生成時などで使用される証明書リポジトリを表す抽象クラスである。全ての証明書と CRL はこのクラスを通じて取得される。SUN 標準の実装では LDAP を使用したものと JAVA の VM 内に存在する

java.security.cert.X509Certificate/java.security.cert.X509CRL のインスタンスを保持する実装が用意されている。

なお、SUN 標準実装では証明書内の CRLDistributionPoint の処理は出来ない。本実装では CRLDistributionPoint の処理を行うため以下の拡張を行っている。

URLObject で指定された CRL の獲得手段を実装では CRLDistributionPoint で想定される HTTP での獲得を考え HTTP プロトコルでの獲得の実装を追加し、CRLDP を扱うプロバイダを実装し追加している。

(2) java.security.cert.PKIXBuilderParameters

このクラスはパス生成/パス検証時に与えるパラメータを表現している。パラメータの詳細に関しては SUN のドキュメントを参照のこと。

本実装ではこのクラスのインスタンスを使用することも可能であるが、OCSP などに関するパラメータは与えることが出来ないので、派生クラスである GPKIBuilderParameters を使用するようにしている。

(3) java.security.cert.PKIXCertPathBuilderResult

このクラスはパス生成プロバイダが返す実行結果を表すオブジェクトである。内包するデータに関しては SUN のドキュメントを参照のこと。本実装では、手を加えずに使用している。

(4) java.security.cert.PKIXCertPathChecker

このクラスはパス生成やパス検証時に使用することが出来る証明書チェックオブジェクトの基底クラスである。SUN の実装における使用法ではユーザ定義の証明書チェッカーを実装するために存在しており、パス生成や検証にはプロバイダ組み込みのチェッカーを固定的に使用している。

本実装では、PKIXCertPathChecker のインスタンスも使用できるが、このクラスを継承したクラス GPKICertPathChecker を規定とするようにしている。理由は、PKIXCertPathChecker のインタフェイスだけでは実行時に与えられるデータが少ないので十分なデータが渡るように拡張する必要があるからである。

(5) java.security.cert.PKIXCertPathValidatorResult

このクラスはパス検証プロバイダが返す実行結果を表すオブジェクトである。内包するデータに関しては SUN のドキュメントを参照のこと。本実装では、手を加えずに使用している。

5.2.2 パス生成プロバイダの実装

(1) パス生成で使うアルゴリズム

本実装に置けるパス生成プロバイダは、検証対象から TA(トラストアンカー）方向へパスを生成する。これはフォーカスしている証明書の署名者名から署名者の証明書群を検索するというアルゴリズムで実現する。

署名者の証明書を取得すると複数の証明書が取得できる場合が存在する。複数の証明書が取得できると言うことは、複数の証明書パスが存在することになり、検証対象証明書を頂点にした木構造、もしくは網目構造が形成される。

このようなデータ構造から目的の経路を検索する問題に対する解は大まかに以下の 2 種類が存在する。

証明書パス生成に関する問題は、検証対象証明書からトラストアンカーへ到達する証明書のリンクをたどる問題であり、基本的には有向グラフにおける経路検索問題に置き換えることが可能である。

この種の問題を解決する研究は古典的なものであり、大まかに以下のようなアルゴリズムが存在する。

(a) 深さ優先検索
(b) 幅優先検索

(a) 深さ優先検索

グラフ中の任意の点を出発点とし、その点と隣接する点へ移動していく検索アルゴリズムである。

このアルゴリズムでは、注目している点は常に一点だけであり、間違った点に移動していると判断できる場合は手戻りを発生させる。このような動作を一般的にバックトラックと呼ぶ。

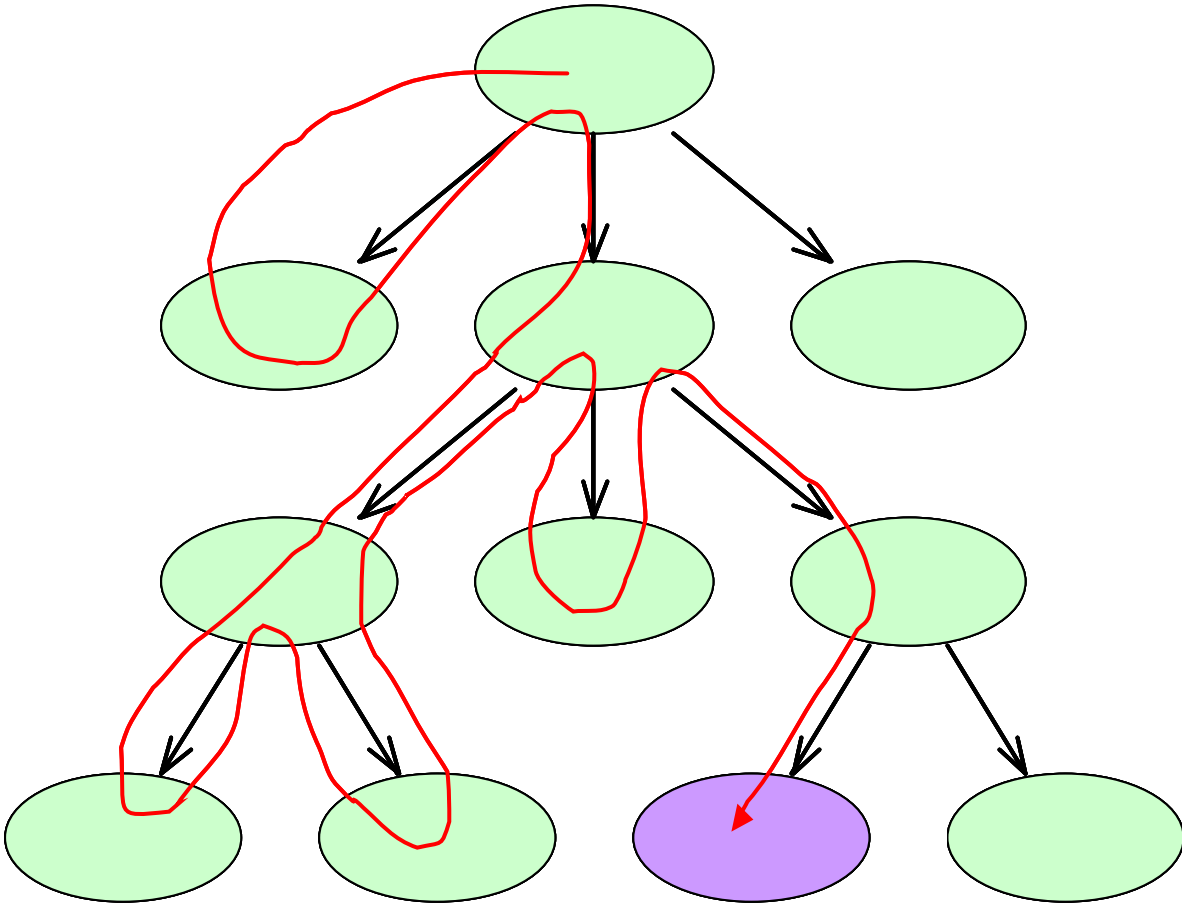


図 5-3 深さ優先探索

このアルゴリズムの最大の利点は、理解の容易さと動作中のコンテキストの追跡が楽な点が上げられる。しかし、問題空間と期待する解の関係によってはバックトラックが大量に発生し、計算時間が爆発的に増加する。予想される計算量(文献[uniformed serach])は

O(b^m)

ただし、

b　　各点が有する平均リンク数
m　　グラフ中における最大距離

となる。

(b) 幅優先検索

とはグラフ中の任意の点を出発点とし、その点と隣接する点全てを集合とする経路を同時に検索するアルゴリズムである。

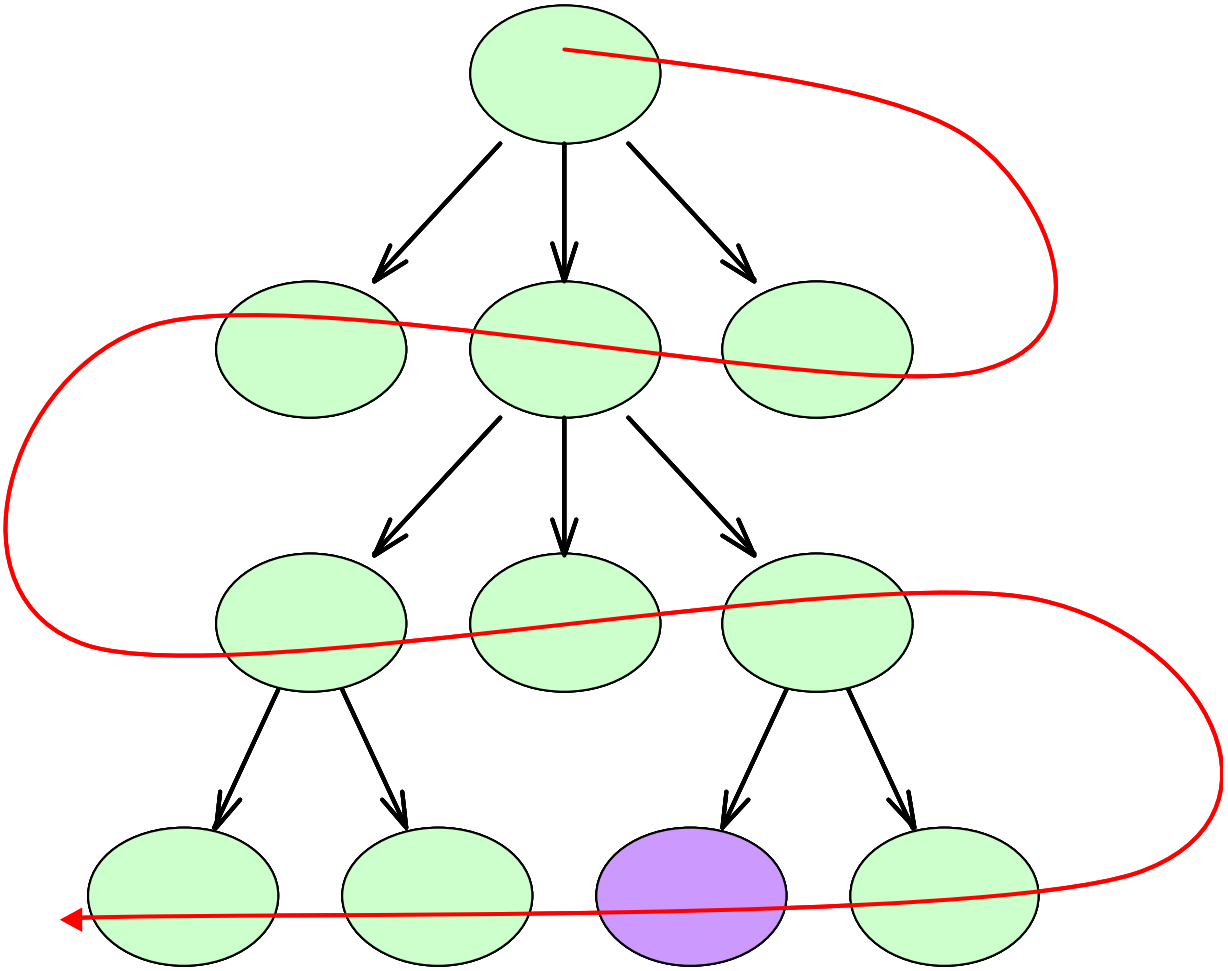


図 5-4 幅優先探索

このアルゴリズムでは深さ優先アルゴリズムのようにバックトラックが発生することは無く、解が存在する深さのみによって計算量が決定される。予想される計算量は

O(b^d)　　：(d を解の深さとする)

となる。

深さ優先検索と幅優先検索の計算量を比較したすると、同一の問題空間に対する検索を行った場合は b は定数となるので指数 d と m が計算量を決定することになる。

また、d と m の関係は d ≦ m が成り立つので、論理的には幅優先検索の方が計算

コストが低いと言える。

(c) サンプル実装において採用したアルゴリズム

本実装においては幅優先検索を採用する。採用理由は以下のとおりである。

a) 解がどの位置に存在しても一定の性能が期待できる。
b) 候補に上げることが出来る解を同時に発見できる。それ故、解の評価に対する拡張が行いやすい。

しかし、幅優先検索を単純に適用した場合、大量のメモリを必要とする欠点が存在する。特に、証明書をノードと見なし、ノードへ結合可能な証明書全てにリンクを作成した場合、分岐点の平均保有数の深さ乗に比例することになる。

この問題に対して、エンティティをノードと見なし、発行されている（動的に発見された）証明書をリンクと見なした。これにより、使用するメモリ量は前記方式に対して分岐点が保有する証明書の数に反比例することになる。

また、エンティティ単位へ問題空間を分割することが可能となるので、大きな問題空間に対する実行速度低下も最小限にとどめられる。

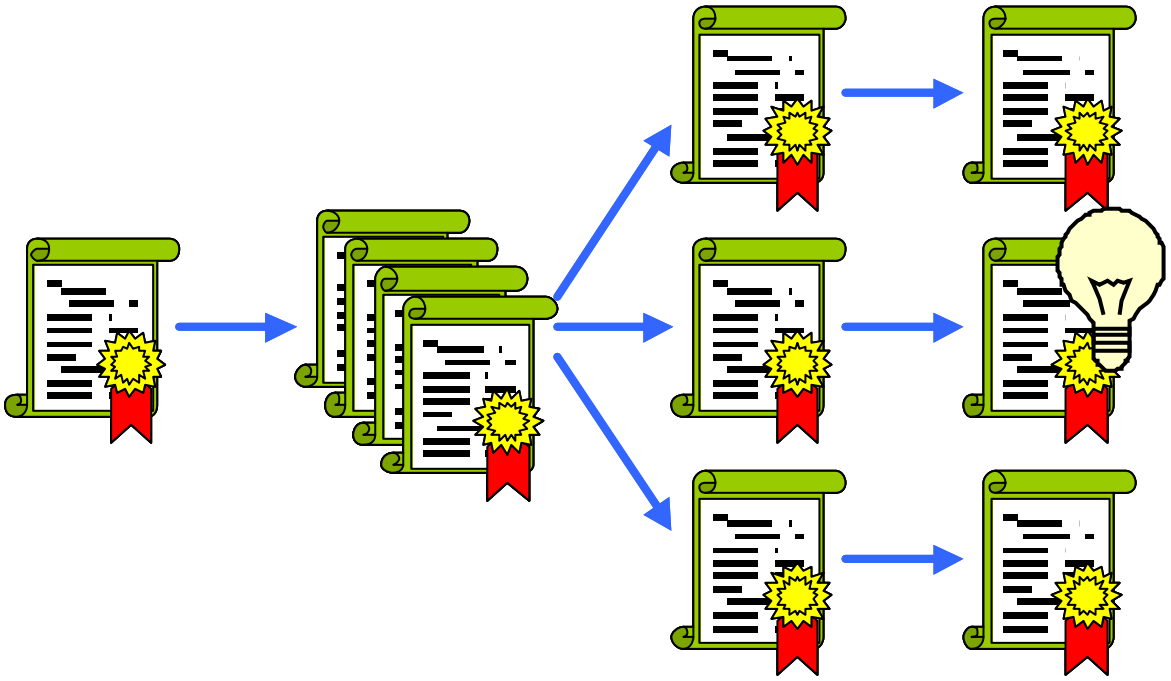


図 5-5 証明書と探索木

5.2.3 パス検証プロバイダの実装

本実装におけるパス検証プロバイダの機能は RFC3280 に準拠している。プロバイダは RFC3280 における検証アルゴリズムを垂直方向に分割し、順次検証していく「汎用検証ブロック」と、RFC3280 が要求する出力を作成する valid_policy_tree を生成する「ポリシー処理ブロック」から構成される。

「汎用検証ブロック」は RFC3280 が定義しているアルゴリズムを分割、整理することによって構成されている。各項目は PKIXCertPathChecker を最基底クラスとしたインスタンスで表現し、複数のチェッカーが集まることによって規格を満たしている。

また、プロバイダのユーザが独自に定義したい検証項目もこの部分で同時に処理する。

以下にデフォルトで使用される(組み込まれている)チェッカーは以下のとおり。

□KeyUsageChecker
本チェッカーは証明書パス中の KeyUsage 拡張に関する妥当性を検証する。
□BasicConstraintChecker
本チェッカーは証明書パス中の BasicConstraints に関する妥当性検証する。
□CRLRevocationChecker
CRL を使用して証明書が破棄されていないか検証する。この際、CRL 自身の検証も行う。
□SignatureChecker
パス中の署名や発行者名の妥当性確認を行う。
□OCSPChecker
OCSP を使用して証明書が破棄されていないか検証する。この動作は可能な限り行うが、OCSP へのアクセスに必要な情報が得られない場合は検証を行わない。
□NameConstraintsChecker
名前制約に関する妥当性を検証する。
□DateChecker
証明書の有効期限に関する妥当性を検証する。

「ポリシー処理ブロック」は RFC で定義されているアルゴリズムが出力するデータを作成する。担当する拡張は以下のとおりである。

□ポリシー
□ポリシーマッピング
□ポリシー制約
□ANY POLICY INHIBIT

また、処理結果は java.security.cert.PKIXCertPathValidatorResult が要求する java.security.cert.PolicyNode を出力とする。

＊java.security.cert.X509CertSelector は大文字小文字を区別しているように見えます。本来はしてはいけません。少なくとも PrintableString であるならば。

see sun.security.x509.AVA

5.3 考察

5.3.1 CRL を発行せず失効情報を OCSP でしか供給しない場合(商業登記認証局など)の扱いについて

OCSP のみで失効情報を提供している認証局の場合(日本国における商業登記認証局など)において CRLDP などで CRL を取得した場合、CRL が一つも取れないことになる。

システムとして警告が出来る場合は、警告を出すことで正常に処理することが出来るが、警告が出せないフレームワークの場合はエラーにするのが安全と見なすケースが考えられる。

しかし、OCSP の使用が予想される場合はエラーにしてはならないので、CRL の失効機能が OCSP に関する予想が出来なくてはならなかった。

5.3.2 自己署名証明書を EE としたパス生成に関して(商業登記認証局)

パス生成機能において、一度通過したノードは再度通らないようにチェックすることは良くあるパターンだと思われる。

この場合、発行者の候補を検索しても自身が発行していることになり、それ以上の展開を抑制する。この状態では cross certificate が存在してもトラストアンカーには到達できないことになり、パスの生成に失敗する。

実運用においては、そのようなエンティティを無視すると言う選択もあるかと思うが、出来るだけパス生成を行える方向で検討した。

方法論としては複数考えられるが、今回は 1 段目の中間 CA を検索する場合に限定し既に通過したノードを再度通過できるように許可した。

これにより「自己署名証明書」⇒「Cross Certificate」というパスが生成され、トラストアンカーへ到達する可能性が生まれることになる。

6 CryptoAPI による証明書パス構築・パス検証の実装の説明

マイクロソフト Windows においては、セキュリティのプラットホームとして CryptoAPI が用意されている。開発者はプログラムに CryptoAPI を利用することにより、暗号プリミティブや、証明書を利用することができる。

本プロジェクトでは、CryptoAPI を利用した「電子署名プログラム」と「電子署名および証明書検証プログラム」の 2 種を開発した。

「電子署名および証明書検証プログラム」は、検証対象ファイルを入力とする、ユーザインターフェースのプログラムと証明書パス構築／パス検証を行うプログラムの 2 つよりなる。

6.1 CryptoAPI 概説

6.1.1 API の分類

CryptoAPI の構造を次図に示す。図では、左側が証明書の関連する関数群、右側は暗号メッセージつまり、 CMS（PKCS#7）に関連する関数群、下側が暗号プリミティブを提供する関数群となっている。

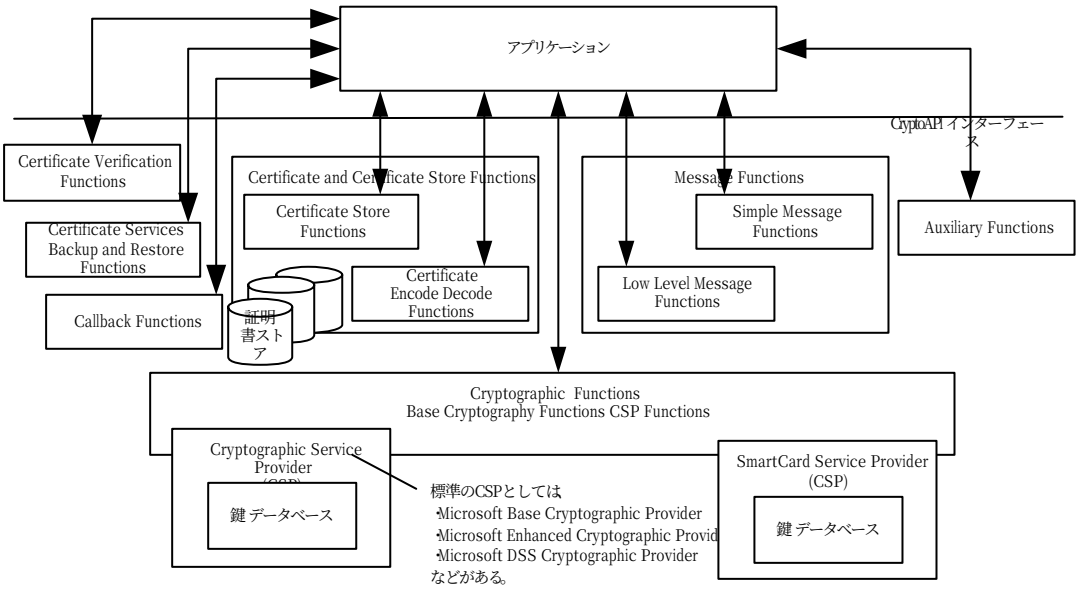


図 6-1 CryptoAPI の構造

なお、CryptoAPI では、PKCS#7 に関して、上位互換を持つ後継仕様である CMS (Cryptographic Message Syntax)を前提として処理を行っている。そのため、本章では、PKCS#7 ではなく、CMS と表記する。

これらの関数群はさらに以下のように分類することができる。

表 6-1 CryptoAPI の関数の分類

Base Cryptography Functions	暗号アルゴリズムを利用するための関数
Certificate and Certificate Store Functions	証明書ストアにアクセスする関数
Certificate Verification Functions	証明書パスの獲得、検証の関数
Message Functions	CMS の生成、解析する関数
Auxiliary Functions	関連するデータを扱うにあたって、補足的な関数
CSP Functions	CryptoServiceProvider との I/F 関数
Certificate Services Backup and Restore Functions	証明書ストアのバックアップ関係の関数
Callback Functions	証明書ストアに関するコールバック関数

6.1.2 証明書ストアと構造体

Windows システムでは、PKI で扱う公開鍵証明書は「証明書ストア」と呼ばれる、ローカルディスクやレジストリ、メモリを用いたリポジトリに保管されており、CryptoAPI を使ってアクセスする。システムが標準で持つ「証明書ストア」は、「カレントユーザ」のもの「ローカルコンピュータ」のものに分けられる。それぞれは、以下のよう証明書の種類ごとの証明書ストアを持つ。

「個人」：秘密鍵とペアになった証明書

「ほかの人」：他の EE 証明書

「中間証明機関」：中間 CA の証明書

「信頼されたルート証明期間」：トラストポイントとなるルート CA の証明書

PKI の主な要素を CryptoAPI で扱う場合の構造体には以下のようなものがある。

表 6-2　CryptoAPI の構造体の例

構造体	役割
CERT_INFO	解析された証明書の各フィールドの情報を持つ
CERT_CONTEXT	証明書コンテキスト、証明書のバイナリデータと CERT_INFO を持つ

CRL_INFO	解析されたCRLの各フィールドの情報を持つ
CRL_CONTEXT	CRL コンテキスト、CRL のバイナリデータとCRL_INFOを持つ
CTL_INFO	解析されたCTLの各フィールドの情報を持つ
CTL_CONTEXT	CTL コンテキスト、CTL のバイナリデータとCTL_INFOを持つ

次にCryptoAPIを使って中間CAの証明書ストアから証明書を取得する流れを示す。

```
HCERTSTORE  hCertStore;  /*証明書ストアハンドル*/
PCCERT_CONTEXT  pCertContext  /*証明書コンテキスト*/
hCertStore = CertOpenSystemStore(NULL, “CA”);
while (pCertContext = CertEnumCertificatesInStore(hCertStore, pCertContext))
{
        char szNameString[256];
      CertGetNameString(pCertContext, CERT_NAME_RDN_TYPE,0,
                              NULL, szNameString, 256);
      printf(“subject : %s¥n”, szNameString);
}
CertFreeCertificateContext(pCertContext);
CertCloseStore(hCertStore, CERT_CLOSE_STORE_CHECK_FLAG);
```

6.1.3 証明書信頼リスト CTL（Certificate Trust List）

CryptoAPI では信頼できる証明書の配布形式として、CTL と呼ばれるものがある。これは、証明書とその使用目的を利用者に通知するためのCryptoAPI独自のデータフォーマットである。これには、管理者が電子署名を行うことが可能であり、安全に証明書を配布することが可能である。本実装でもトラストポイントの証明書のモジュール間での引き渡しにCTLを使った。

6.1.4 証明書パス検証のための関数

(1) Certificate Verification Functions

通常、CryptoAPIを使った証明書パスの獲得と、パスの失効検証は「Certificate Verification Functions」にある関数を使って行う。本プロジェクトでもこれら関数を使った開発を行った。これについては、6.2.3 節で解説する。

(2) CertVerifyRevocation による証明書ステータスの取得

「Auxiliary Functions」に分類される関数 CertVerifyRevocation 使って、証明書のステータスを確認できる。前述の「Certificate Verification Functions」の関数が証明書パスに対する処理を行うのに対し、この関数では、個々の証明書コンテキストを入力とし、単一の証明書のステータスを取得するのが目的である。この関数の特筆すべき点は、任意の失効検証ルーチンを開発し Revocation Providers としてシステムに登録しておくと、関数内部で、登録されている失効検証ルーチンを呼び出すことだ。Revocation Providers とは関数“ CertDllVerifyRevocation”をエクスポートした DLL ファイルとして実現される。つまり CertVerifyRevocation が内部で DLL を動的にロードし、CertDllVerifyRevocation へパラメータをそのまま引き渡し、処理結果をまたパラメータに受ける。

システムへの登録はレジストリのキー

HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Cryptography¥OID¥EncodingType 1¥CertDllVerifyRevocation¥DEFAULT¥Dll が使われる。

複数の Revocation Providers が登録可能で、DLL の実装によって、呼び出される順番を指定可能である。OS の標準では cryptnet.dll、mscrlrev.dll などが登録されている。本プロジェクトでも、パス構築、パス検証を実現する Revocation Providers を開発した。これについては、6.2.3 節で解説する。

(a) 既存のアプリケーションについて

一般的な CryptoAPI を使ったアプリケーションでも、CertVerifyRevocation を使った失効検証を行っている。その場合、アプリケーションでの設定が必要となる場合がある。以下に設定について記す。

- Outlook Express の設定

ツール」メニューの「オプション」の「セキュリティ」タブで「セキュリティの詳細設定」を開く。そこで、「デジタル ID が取り消されているか確認する」の「オンラインのときのみ」をチェックする。

- Internet Explorer の設定

インターネットオプションの「詳細設定」で「サーバ証明書の取り消しを確認する」をチェックする。

- IPsec/ISAKMP での PKI 利用時の設定

IPsec/ISAKMP でのネゴシエーションに公開鍵証明書を利用する場合、CRL-DP を元に CRL を取得し失効検証をさせるためには設定が必要である。レジストリにキー

HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥PolicyAgent¥Oakley

を追加して、
DWORD 値の値「StrongCrlCheck」を追加

- StrongCrlCheck が 0 の場合、CRL による検証を行わない。
- StrongCrlCheck が 1 の場合、CRL による検証を行う。
- StrongCrlCheck が 2 の場合、CRL による検証を行い、かつ、適正な CRL が取得できないような場合、無効な証明書として扱う。

値を追加、変更した後は、コマンドプロンプトから
>net stop policyagent
>net start policyagent
を実行すると、設定が有効となる。

(b) CRL のキャッシュ

標準の cryptnet.dll では、取得した CRL をキャッシュする。キャッシュに使う場所は、Internet Explorer の Temporary Internet Files などである。

6.1.5 参考

Revocation Providers については、以下が参考になる。
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsecure/html/rpcrypto.asp
CryptoAPI での証明書検証に関しては、以下が参考になる。
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/prodtech/dbsql/tshtcrl.asp

6.2 サンプル実装の解説

CryptoAPI を利用してブリッジ CA モデルの PKI 環境においてデジタル署名および署名検証を適切に処理するサンプル実装を開発した。

6.2.1 開発環境

本プログラムの開発は以下の環境で行った。
コンパイラ : Microsoft Visual C++ V6.0 SP5

CryptoAPI ライブラリ: Microsoft Platform Software Development Kit (SDK)

6.2.2 テストプログラム

CryptoAPI を使ったサンプル実装として、以下の 2 つのテストプログラムを開発した。

- 電子署名プログラム pkiicsig.exe
- 電子署名およびの証明書検証プログラム pkiicsver.exe

これら 2 つの実装について解説する。

(1) 電子署名プログラム pkiicsig.exe

コマンドラインで受けたクレデンシャルファイルを署名者として、ファイルに電子署名を行う。次図に署名の処理フローを示す。

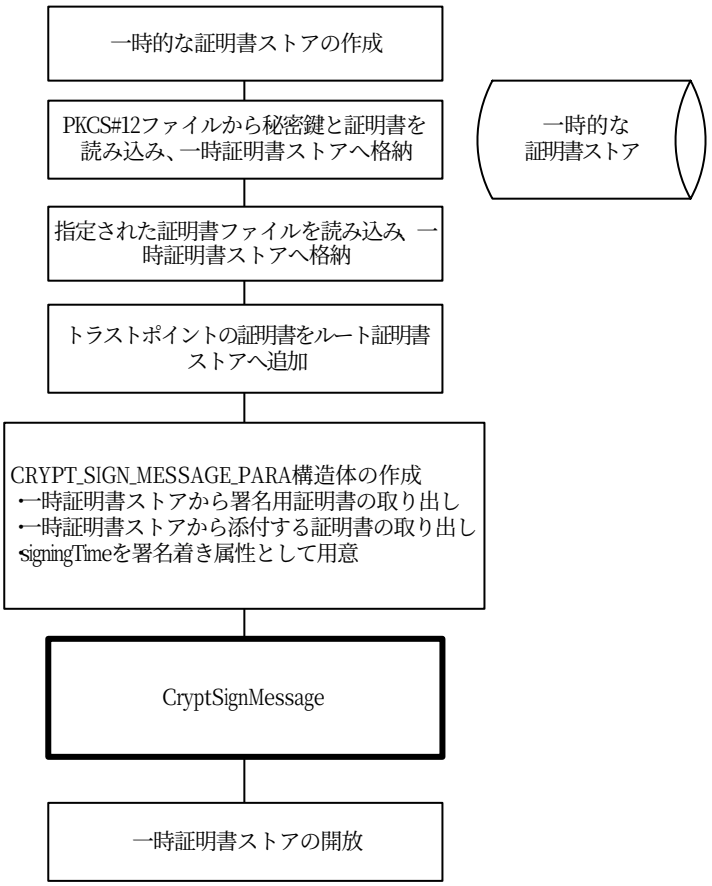


図 6-2 署名処理フロー

(2) 電子署名および証明書検証プログラム pkiicsver.exe

検証対象のファイルが CMS の signedData の場合、電子署名の検証をまず行う。CMS signedData ファイルに署名者の証明書が添付されていた場合、あるいは、検証対象のファイルが X.509 の DER エンコードされた証明書ファイルの場合、証明書パス構築とパス検証を行う。パス構築、パス検証機構の詳細は別項に記載するが、ここでは、前処理について解説する。

Windows にはシステムで標準の証明書ストアがあり、トラストポイントのルート証明書ストア、中間 CA 証明書ストア、他人としての EE 証明書ストア、秘密鍵をペアでもつ自身の証明書ストアがあり、CA の証明書ストアには CRL や CTL も格納されている。本プロジェクトでは、証明書パスの構築に使う証明書を限定する必要があるため、これら標準の証明書ストアとは別に一時的な証明書ストアを作成し、プログラムが参照することとした。パス構築、パス検証への前処理の主な役割は、この一時証明書ストアの作成である。前処理は、以下のような手順となる。

① トラストポイント用一時証明書ストアの作成
② トラストポイント証明書から CTL の作成
③ トラストポイント証明書を一時証明書へ追加
④ CTL を一時証明書へ追加
⑤ 中間 CA 用一時証明書ストアの作成
⑥ 中間 CA 証明書を一時証明書へ追加
⑦ 中間 CA 発行の CRL、ARL を一時証明書へ追加

また、ポリシ制約関連のパラメータを、設定ファイル（pkiic.dat）に、以下のようなキーにて保存する。この設定は後述する「パス構築、パス検証機構」における「独自のパス構築、パス検証ルーチン」が利用するものである。

[common]セクション
InitialPolicyMappingInhibit = false
InitialExplicitPolicy =false
InitialAnyPolicyInhibit = false
InitialPolicy0= 0.2.392.200117.1.9.2002.2.1.2.392.100595.8.5.1.1.10

6.2.3 パス構築、パス検証機構

電子署名およびの証明書検証プログラムが証明書の検証のために利用するパス構築、パス検証の実装として、以下の 2 つを開発した。この 2 つは、テストプロ

グラムの設定ファイルにて選択する。

- CryptoAPI の「Certificate Verification Functions」を利用した実装
- 独自のパス構築、パス検証ルーチンを CryptoAPI の Revocation Provider とした実装

次にこれら 2 つの実装について解説する。

(1) CryptoAPI の「Certificate Verification Functions」を利用した実装

「Certificate Verification Functions」のうち、「Certificate Chain Verification Functions」として分類される関数を利用して検証を行う。ここでは、「証明書パスエンジン」を作成し、対象の証明書を検証する。次に処理の内容を解説する。

(a) 証明書パスエンジン

パス構築、パス検証には「証明書パスエンジン」を作成し、「証明書パスエンジン」を指定して、証明書ストアから証明書パス上の証明書を獲得し、失効の検証を行う。

「証明書パスエンジン」には、システムのルート証明書ストアのサブセットを一時的な証明書ストアとして作成し、トラストポイントの証明書を限定することが可能である。また、中間 CA の証明書については、システムの中間 CA 証明書ストアとは別に一時的な証明書ストアを作成して、証明書、CRL を格納しておくことにより、パス構築処理の際に追加、あるいは限定的に参照させることも可能である。また、拡張鍵使用法、CTL の使用法など検証方法を指定できる。

(b) 検証処理

検証時のパラメータとしては、EE 証明書の失効検証の有無、パス上の証明書の失効検証の有無や、キャッシュの使用法、日時を特定しての検証が可能である。

失効検証では、内部で関数 CertverifyRevocation が呼ばれて、登録されている RevocationProvider が働く。

なお、この方法では、ポリシー制約関連の初期パラメータを参照することはできない。

処理の結果として証明書パスとともに、証明書の信頼情報として、以下のようなエラーステータスと参照情報が獲得される。

表 6-3 CryptoAPI におけるパス検証の診断値

#	エラーコードの定義名
	意味
検証対象の証明書または証明書パスに関するエラー	
1	CERT_TRUST_NO_ERROR
	証明書およびパスに問題はない。
2	CERT_TRUST_IS_NOT_TIME_VALID
	検証対象の証明書かパス上の証明書で有効期限が妥当ではないものがある。
3	CERT_TRUST_IS_NOT_TIME_NESTED
	パス上の証明書で有効期限の関連が適切ではない
4	CERT_TRUST_IS_REVOKED
	トラストポイントの証明書かパス上の証明書で失効しているものがある。
5	CERT_TRUST_IS_NOT_SIGNATURE_VALID
	検証対象の証明書かパス上の証明書で有効な署名でないものがある。
6	CERT_TRUST_IS_NOT_VALID_FOR_USAGE
	検証対象の証明書かパス上の証明書で指定された鍵使用法と合致しないものがある。
7	CERT_TRUST_IS_UNTRUSTED_ROOT
	証明書パスが信頼されないルート認証局に基づいている。
8	CERT_TRUST_REVOCATION_STATUS_UNKNOWN
	検証対象の証明書かパス上の証明書で、失効状態が不明なものがある。
9	CERT_TRUST_IS_CYCLIC
	証明書パスが循環している。
10	CERT_TRUST_INVALID_EXTENSION
	無効な拡張がある。
11	CERT_TRUST_INVALID_POLICY_CONSTRAINTS
	検証対象の証明書かパス上の証明書で、ポリシー制約拡張を持つものがあり、プリシマッピングが禁止されているか、ポリシーが要求されているのにない証明書がある。
12	CERT_TRUST_INVALID_BASIC_CONSTRAINTS
	検証対象の証明書かパス上の証明書で、基本制約拡張を持つものがあり、CA制約かパス長制約に反する証明書がある。
13	CERT_TRUST_INVALID_NAME_CONSTRAINTS
	検証対象の証明書かパス上の証明書で、無効な名前制約拡張がある
14	CERT_TRUST_HAS_NOT_SUPPORTED_NAME_CONSTRAINT

	検証対象の証明書かパス上の証明書で、サポートしていないフィールドを持つ名前制約拡張を持つものがある。minimum フィールドと maximum フィールドはサポートしていない。したがって、minimum 値は常に 0 として、maximum 値はあってはいけない。otherName としては MicrosoftUPN(1.3.6.1.4.1.311.20.2.3)がサポートされている。x400Address、ediPartyName、registeredID はサポートされていない。
15	CERT_TRUST_HAS_NOT_DEFINED_NAME_CONSTRAINT
	検証対象の証明書かパス上の証明書で、名前制約拡張があるが、証明書の名前に使われているフィールドに関する制約がない。
16	CERT_TRUST_HAS_NOT_PERMITTED_NAME_CONSTRAINT
	検証対象の証明書かパス上の証明書で、名前制約拡張を持つものがあり permittedSubtrees フィールドがあるが、許可されていない名前を持つ証明書がある。
17	CERT_TRUST_HAS_EXCLUDED_NAME_CONSTRAINT
	検証対象の証明書かパス上の証明書で、名前制約拡張を持つものがあり、excludedSubtrees フィールドがあるが、受け付けられない名前を持つ証明書がある。
18	CERT_TRUST_IS_OFFLINE_REVOCATION
	検証対象の証明書かパス上の証明書に関する失効情報は、オフラインでものであるか古い。
19	CERT_TRUST_NO_ISSUANCE_CHAIN_POLICY
	ポリシー制約拡張でポリシーが要求されているが、要求されたポリシーを満たさないものがある。
証明書パスに関するエラー	
20	CERT_TRUST_IS_PARTIAL_CHAIN
	証明書パスが完結していない。
21	CERT_TRUST_CTL_IS_NOT_TIME_VALID
	この証明書パスを構築するために使った CTL が古い。
22	CERT_TRUST_CTL_IS_NOT_SIGNATURE_VALID
	この証明書パスを構築するために使った CTL に有効な署名がない。
23	CERT_TRUST_CTL_IS_NOT_VALID_FOR_USAGE
	この証明書パスを構築するために使った CTL の使用法は適切ではない。
参照情報（エラーではない）	
24	CERT_TRUST_HAS_EXACT_MATCH_ISSUER
	適切な発行元の証明書が存在する。
25	CERT_TRUST_HAS_KEY_MATCH_ISSUER
	適切な鍵を持つ発行元の証明書が存在する。

26	CERT_TRUST_HAS_NAME_MATCH_ISSUER
	適切な名前を持つ発行元の証明書が存在する。
27	CERT_TRUST_IS_SELF_SIGNED
	自己署名の証明書である。
28	CERT_TRUST_IS_COMPLEX_CHAIN
	証明書パスは複数ある。
29	CERT_TRUST_HAS_PREFERRED_ISSUER
	適切な発行元を持つ。
30	CERT_TRUST_HAS_ISSUANCE_CHAIN_POLICY
	ポリシーを持つ。
31	CERT_TRUST_HAS_VALID_NAME_CONSTRAINTS
	有効な名前制約を持つ。

(c) **処理フロー**

CryptoAPI **の**「Certificate Verification Functions」**を利用した場合のフローは次図のようである。**

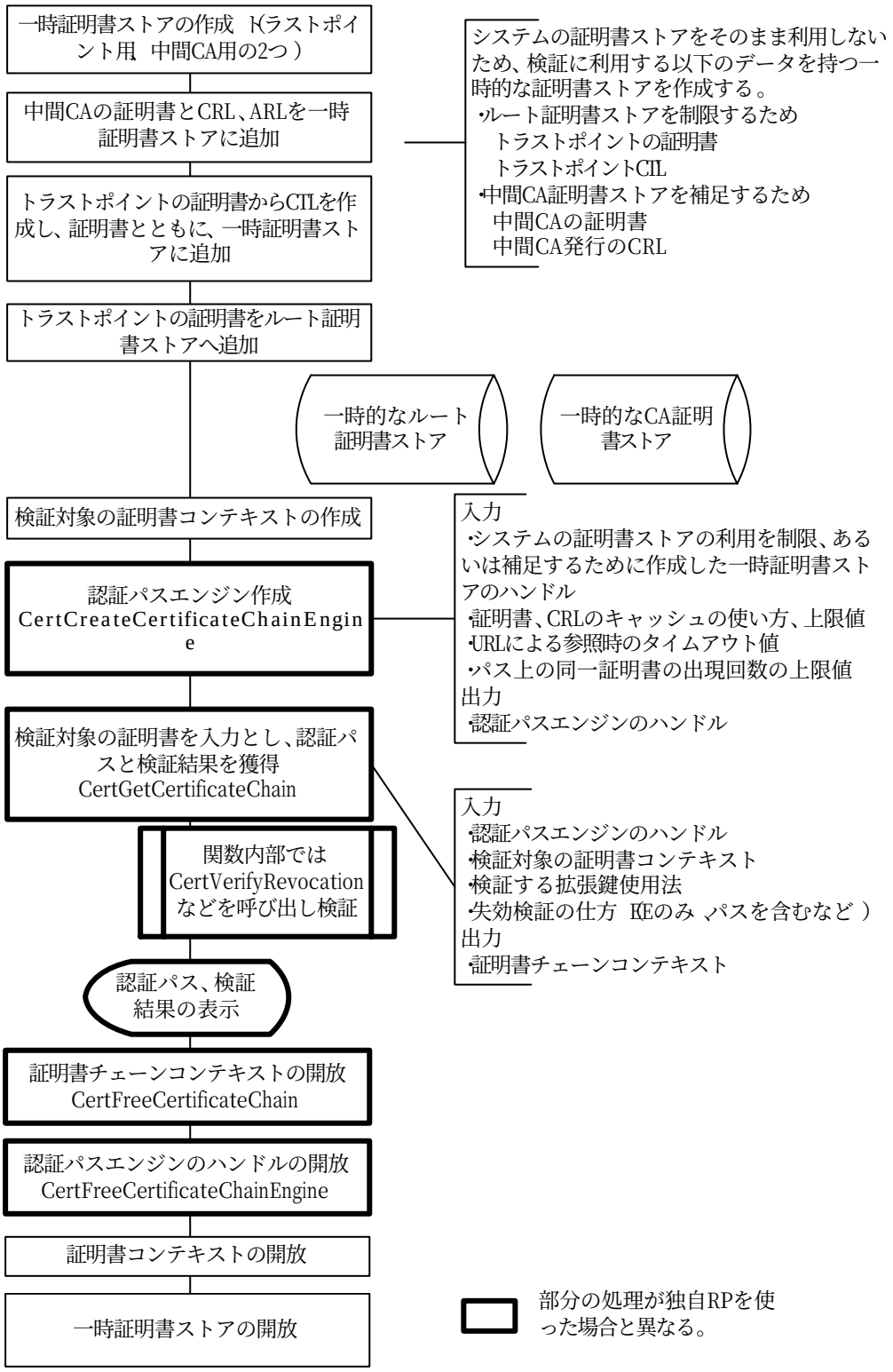


図 6-3 CryptoAPI の「Certificate Verification Functions」を利用した実装のフロー

(2) 独自のパス構築、パス検証ルーチンを CryptoAPI の Revocation Provider とした実装

(a) 概要

本プロジェクトでは、パス構築、パス検証の開発のベースとして、オープンソースのライブラリ Certificate Management Library (CML)を改良した。このライブラリは米国 FPKI に関連する「DoD Bridge CA Technology Demonstration」プロジェクトの成果物の一部である。

次図に CML のモジュールの関連を示す。

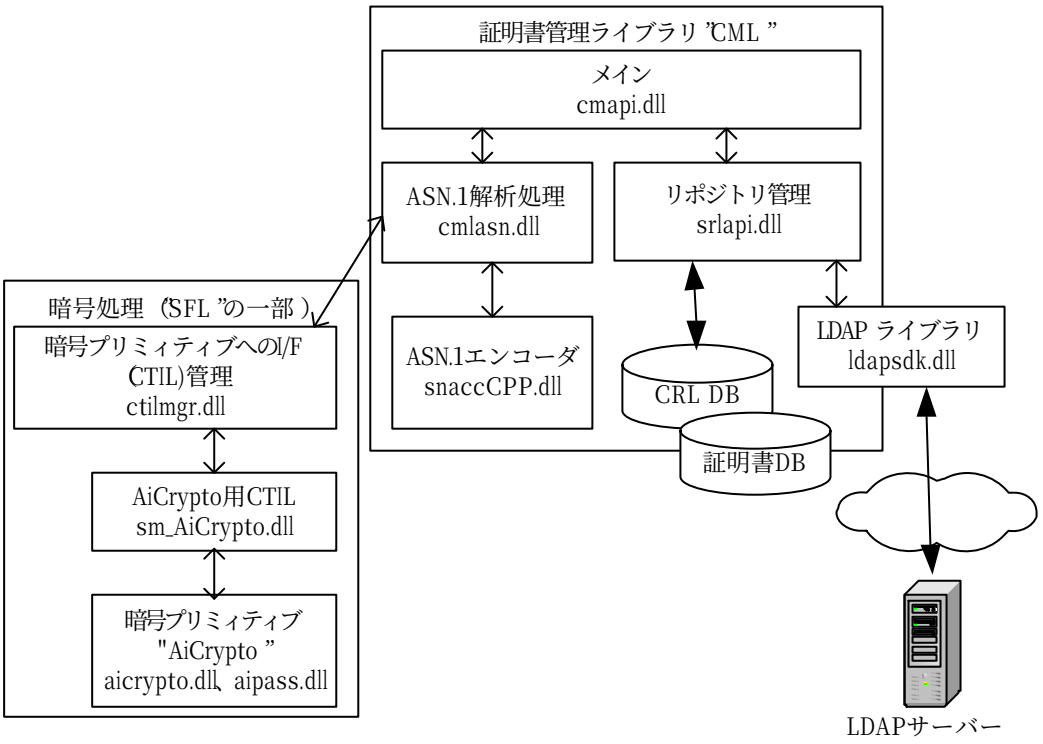


図 6-4 CML のモジュール関連図

次に、本プロジェクトで開発したモジュールの関連を示す。CML の修正の主として以下のようである。

- CML 自体の改造
  OCSP の実装
  HTTP による CRL の取得
- CML へ指定可能なコールバック関数の作成
  CML の DB を使わず、一時証明書ストアを参照する。
  LDAP 検索を標準機能とは別にして、きめ細かな検索を可能にする。

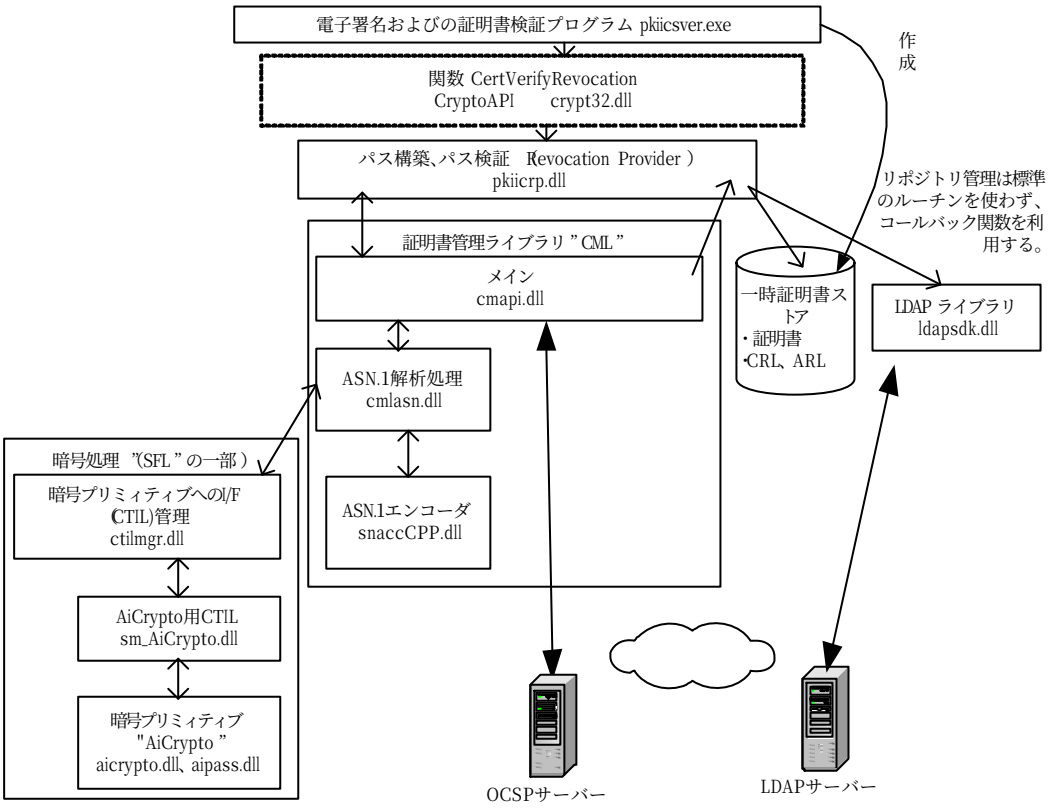


図 6-5 パス構築、パス検証ルーチンモジュール

(b) 処理フロー

独自のパス構築、パス検証ルーチンを CryptoAPI の Revocation Provider とした実装の処理フローを次図に示す。

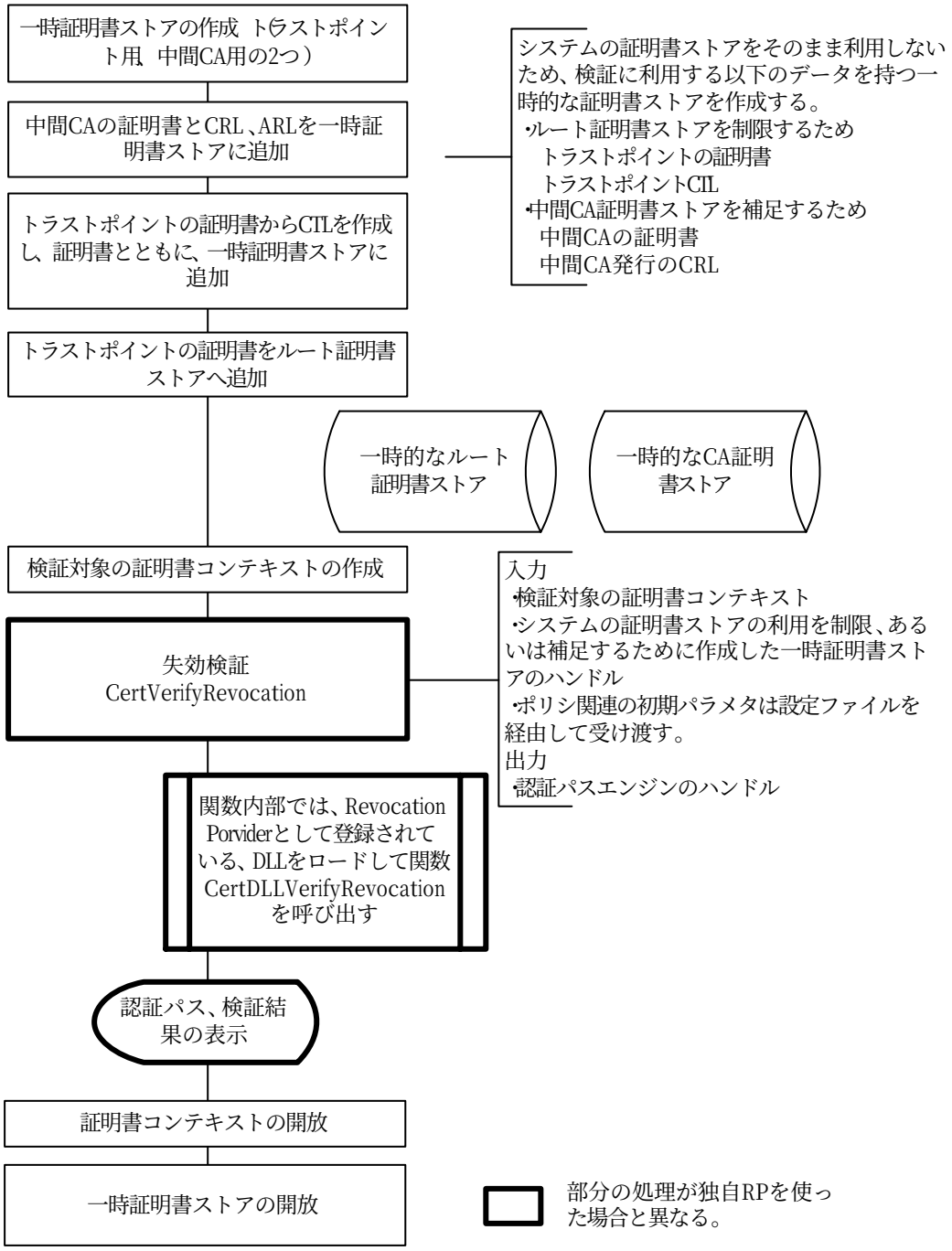


図 6-6 CryptoAPI Revocation Providier による処理フロー

(c) パス構築とパス検証の処理アルゴリズムについて

1.パス構築検証動作(開始する証明書)
while 検証成功 or リトライ制限まで 　if 最初のパス構築 　　構築したパス ? パス構築(開始する証明書) ……[2] 　else 　　構築したパス ? 次のパス構築(開始する証明書) ……[3] 　if パス構築成功? 　　パス検証(構築したパス) ……[6] 　if 検証成功? 　　return 構築したパス return パス構築失敗

2.パス構築(構築する証明書), 3.次のパス構築
if 構築する証明書がトラストポイントなら 　return 構築する証明書 1 つを含むパスを返す 証明書リスト ? 証明書を探す(構築する証明書の Issuer 名) ……[4] 優先順序をつけてソート(証明書リスト) ……[5] foreach 証明書 in 証明書リスト 　パス ? パス構築(証明書) 　if パス構築成功なら 　　return [パス, 証明書] …… パスの末尾に証明書を追加 　　ここで返したパスが検証失敗したら,次の foreach ループから開始 return パス構築失敗

4. 証明書を探す(Issuer 名)
オフラインモードの場合 　一時証明書ストアの, issuer 名が subject 名と一致する証明書をリストにして返す。 オンラインモード（デフォルト）の場合 　DB 上で, issuer 名が subject 名と一致する証明書をリストにする 　リポジトリから, DN=issuer 名の証明書を取り出しリストする 　DB とリポジトリから取り出した証明書をマージして返す

5. 優先順序をつけてソート
優先順位は次のポイントの合計

1. SKI と AKI が一致する +8
2. SKI と AKI が一致しない –6
3. SKI あるいは AKI がない +2
4. 証明書がトラストポイント +4
5. 証明書が CA 証明書 +3
6. 証明書が EE 証明書 +2
7. 証明書が相互認証証明書+1

（その他 V2 の UniqueID を見たり、キャッシュを優先したりなど）

6. パス検証(パス上の証明書のリスト)

トラストポイントの証明書から,有効なポリシーを初期化
while 証明書 in パス上の証明書のリスト
　有効性検証(証明書)　……[7]
　証明書のポリシーマップを計算
　有効なポリシーを更新
return 検証成功

7. 有効性検証(証明書)

以下の点を検証する
1. 証明書の署名アルゴリズムと issuer の署名アルゴリズムの一致を検証
2. 署名の検証
3. 有効日付の検証
4. issuer と subject の DN の一致を検証
5. 失効検証(証明書)　……[8]
6. BasicConstraint の検証(CA の場合 CA フラグを確認)
7. KeyUsage の検証 (CA の場合,certSign を確認)
8. KeyUsage が crtical の場合,未知の KeyUsage がないか検証
9. 名前制約,パス長制約の検証
10. subjectAltName が crtical の場合,未知のフォーマットがないか検証
11. subjectAltName の名前制約の検証
12. ポリシー制約の検証
13. 未知の crtical な拡張がないか検証

8. 失効検証(証明書)

if OCSP の検証モードが[URL 指定]で URL が空文字列でなければ
　OCSP 検証（指定 URL）　……[10]

```
    if 有効なら
      return 有効
    if 失効なら
      return 失効
if AIA が存在して,OCSP の検証モードが[AIA に従う]なら （デフォルト）
   foreach AIA1 in AIA
   if  AIA1 の OID が OCSP なら
      OCSP 検証（AIA1 の URL） ……[10]
       if 有効なら
            return 有効
      if 失効なら
         return 失効
if CRLDP が存在するなら
  foreach 名前 in CRLDP
       CRL 検証(名前) …… [9]
else
   CRL 検証(Issuer 名) …….[9]
```

9. CRL 検証(URL or DN)
一時証明書ストアから CRL を取り出す if CRL が存在して,次の発行日付を経過していない return CRL を検索して失効結果を返す if CRL をリモートに取り出さないモード return 最新の CRL がない or CRL が取り出せない if URL 指定 URL から CRL を取り出す else ディレクトリサーバから DN 指定で，CRL を取り出す if 取り出し失敗なら return CRL が取り出せない if CRL の次の発行日付が経過していたら return 最新のＣＲＬがない return CRL を検索して失効結果を返す

10.OCSP による検証（サーバ URL）
OCSP レスポンス ？ サーバ URL に HTTP で問い合わせ レスポンスの署名検証

パス構築検証動作(レスポンスの証明書)　　.....　[1] ? ただし，パス構築，CRL 参照はオフライン（一時証明書ストア）のみに限定 ? レスポンスに id-pkix-ocsp-nocheck がある場合は，失効検証[8]を行わない

(d) リポジトリ検索の方法について

証明書や CRL をディレクトリサーバから取り出す際の LDAP API の利用方法についてしめします。

ldap_init で anonymous bind し、ldap_set_option で以下を指定している。

LDAP_OPT_PROTOCOL_VERSION : LDAP_VERSION3

LDAP_OPT_REFERRALS : LDAP_OPT_ON

LDAP_OPT_DEREF : LDAP_DEREF_ALWAYS

ldap_search_st の引数は以下のようである。

証明書の検索	
ベース DN	Issuer の DN
Scope	LDAP_SCOPE_BASE
filter	objectclass=*
attrs	以下の属性を指定 cACertificate crossCertificatePair cACertificate;binary crossCertificatePair;binary
attrsonly	FALSE
CRL の検索	
ベース DN	Issuer の DN
Scope	LDAP_SCOPE_BASE
filter	objectclass=*
attrs	以下の属性を指定 certificateRevocationList authorityRevocationList certificateRevocationList;binary authorityRevocationList;binary
attrsonly	FALSE

6.3 考察

6.3.1 本来、失効確認が目的である CertVerifyRevocation

CertVerifyRevocation は戻り値が論理型で失効済みであれば、FALSE を、未失効であれば、TRUE を返し、引数を使って失効理由を出力するように定義されている。しかし、本実装での目的は、失効確認のみに留まらない、パス構築、パス検証の結果の証明書の有効性の確認であるため、戻り値に関しては、パス検証の結果有効であれば、TRUE を、無効であれば、FALSE を返すこととした。

6.3.2 ポリシー関連の初期パラメータ

CryptoAPI ではポリシー制約にかかわる初期パラメータに対する考慮がなされていない。

6.3.3 CryptoAPI の OS による相違点

Windows の 95/98/Me 系列と NT4/2000/XP 系列ではサポートされている関数に相違があり、95/98/Me 系列ではいくつかの機能が利用できない。また、NT4/2000/XP 系列のなかでも Windows XP とそれ以外でも、相違点がある、例えば、以下のようだ。

(1) パス検証ルーチン CertGetCertificateChain

証明書ストアより証明書パスを獲得し、その適切な失効検証を行い、パスの正当性検証する関数である(本プロジェクトでも利用しており、詳細は後述する)。証明書ストアに、相互認証した CA の証明書とその CA の自己署名の証明書があり、その CA の発行した証明書を検証する場合、Windows XP では、証明書パスを適切に処理するが、Windows 2000 では、証明書パスが CA の自己署名の証明書で終わり、トラストポイントまで到達しない。

(2) 証明書プロパティのダイアログ表示 CryptUIDlgViewContext

Windows XP では、cryptoui.dll でエクスポートされているこの関数に、任意の証明書、CRL、CTL のコンテキストを引数として渡し、Windows で標準的に用いられている、証明書や CRL,CTL のプロパティを参照するダイアログを表示可能であるが、Windows 2000 以前の OS にある cryptoui.dll ではこの関数がない。

7 テストスイートの概要

本章では、PKI 相互運用テストスイート（以下、単にテストスイートと書く）の概要及びサポートしているテストケース、テストの実行方法について簡単に説明する。また、テストスイートで利用している証明書検証プログラムの使用方法についても言及する。

7.1 テストスイートの概要

テストスイートは、大きく分けて以下の２つの要素によって構成されている（図 7-1）。

(1) テストを実行するために必要なデータを格納するデータベース（以下、DB と書く）。必要であれば、テストの実行結果を格納することもできる

(2) DB の内容を元にテスト実行に必要なデータ群（証明書、ＣＲＬ、OCSP/GCVS 応答データなど）と、テストを実行するスクリプト群などの各種ファイルを生成するためのスクリプト・プログラム群。

各種スクリプト・プログラムの仕様については、それぞれの仕様書に書かれているため、本章では説明しない。

テストの実行には、オンラインモードとオフラインモードの、２つの実行形態がある。オンラインモードでは、事前にテストの実行に必要な証明書、CRL 及び ARL が LDAP サーバ（リポジトリ）に登録された上で、パス検証が行われる。従ってテスト対象プログラムは、パス検証に必要となる証明書、CRL 及び ARL は、リポジトリから取得する。

これに対しオフラインモードでは、証明書、CRL 及び ARL は LDAP サーバへは登録されない。テスト対象プログラムはこれらの情報を、ローカルファイルから取得する。

テストの実行内容がモードによってどのように異なるかについては、次節で詳しく示す。

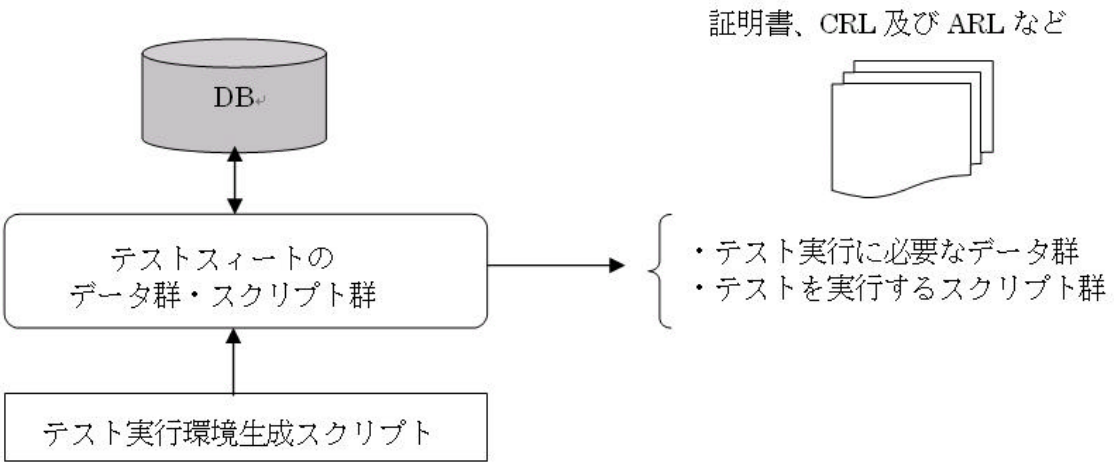


図 7-1 テストスイートの全体図

7.2 テストの分類

DB の詳細については、DB 設計書で示す。本節では、テストスイートのサポートしているテストケースについて概説する。

DB には、表 7-1 に示す通り、大きく分けて 3 種類のテストケースが納められている。この表で、項目「オンライン（オフライン）」が「可」である場合、オンライン（オフライン）モードでテストを実行できることを示す。

表 7-1 テストケースのカテゴリー

カテゴリ番号	カテゴリ	オンライン	オフライン	OCSP を利用したテストケース	GCVS を利用したテストケース
1	NIST テストケース	不可	可	無	無
2	GPKI テストケース	可	可	有	有
3	オリジナルテストケース	可	可	有	無

項目「OCSP を利用したテストケース」または「GCVS を利用したテストケース」が「有」となっている場合、OCSP または GCVS（GPKI 証明書検証サーバ）を利用するテストケースが一部含まれていることを意味する。

OCSP または GCVS を利用したテストケースをオフラインモードでテストすると、OCSP レスポンダシミュレータまたは GCVS サーバシミュレータへの問い合わせが行われず、テスト結果が期待通りにならないことがある。

以下で、各カテゴリについて順に説明する。

(1) NIST テストケース

NIST テストケースは、米国国立標準技術研究所の公表している X.509 Path Validation Test Suite（http://csrc.nist.gov/pki/testing/x509paths.html 参照）を元に作成されたテストケースである。NIST の各テストにはテストレベル（テストの難易度）が設定されている。NIST テストケースの詳細については、GPKI テストケース設計書の３章「NIST パス検証と同等のテストケース」を参照のこと。

(2) GPKI テストケース

GPKI テストケースは、日本の政府認証基盤（http://www.gpki.go.jp/）における様々なパス検証のシチュエーションを考慮して作成したテストケースである。テストケースによっては、OCSP を利用するものや、GCVS を利用するものも存在する。GPKI テストケースの詳細については、GPKI テストケース設計書の２章「GPKI 模擬テストケース」に書かれている。

(3) オリジナルテストケース

オリジナルテストケースは、NameRollover や KeyUpdate といった作業を含む、高度なパス検証を実験するためのテストケースである。テストケースによっては、OCSP を利用するものが存在する。オリジナルテストケースの詳細については、GPKI テストケース設計書の４章「オリジナルテストケース」に示す。

7.3 テストの流れ

テストスイート実行の開始から終了までの流れについては、テスト実行環境設計書で詳しく述べられている。本節ではテスト実施者（以下、ユーザと書く）からの視点で、テストの手順と、使用するコマンドについて簡単に説明する。

7.3.1 テストの実行手順

ユーザは、以下の３つの手順でテストを実行する。

(1) ユーザは初めに、DB が起動しているマシン上（以下、サーバと書く）で、**テスト実施環境生成スクリプト**を起動する。テスト実施環境生成スクリプトは、テストスイートの各種スクリプト・プログラム群を順番に起動することで、指定されたテストの実行に必要なデータ群、及びテストを実行するスクリプト群（以下、これらをまとめてデータ群・スクリプト群と書く）を、テストデータ領域に生成する（図 7-1）。従ってユーザは、テストデータ

領域の生成に際しては、テスト実施環境生成スクリプトのみを使用すれば良く、逆にそれ以外の各種スクリプトを直接起動する必要は無い。

(2) 次にユーザは、テストデータ領域に生成されたデータ群・スクリプト群を用いて、テストを実行する。テストを実行するマシンをクライアントと呼ぶ。サーバマシンとクライアントマシンとが異なる場合、ユーザは ftp コマンドなどを用いて、あらかじめテストデータ領域をコピーする必要がある。

(3) クライアント上でのテストの実行には、**テスト実行スクリプト**を用いる。テスト実行スクリプトは、個々のテストケースごとに用意されたテストケース実行スクリプトを順番に起動し、テスト結果を保存する機能を持つ。従ってユーザは、テストの実行に際し、テスト実行スクリプトのみを使用すれば良い。

一連の手順において、ユーザが起動するテストスイートのプログラムは、**テスト実施環境生成スクリプト**及び**テスト実行スクリプト**の２つのみである。次項からは、これらのスクリプトがどのように動作しているかを概説する。

7.3.2 テスト実施環境生成スクリプト

ユーザは、テスト実施環境生成スクリプトを起動することによって、テスト実行に必要なデータ群・スクリプト群を生成できる。起動の際、代表的なオプションとして表 7-2 が挙げられる。

表 7-2　テスト実施環境生成スクリプトの代表的なオプション

形式	説明
-c　対象テストカテゴリ ID	複数指定する場合はスペースで区切る、指定のない場合は全て
-l　対象テストレベル	複数指定する場合はスペースで区切る、指定のない場合は全て
-m　アクセスモード(on | off)	LDAP などを用いたテストをするか選択する、デフォルトは on

オプション「-c」、「-i」あるいは「-l」で、データ群・スクリプト群を生成する対象となるテストケースを指定できる。何も指定の無い場合は、全てのテストケースが対象となる。

図 7-2 に、テスト実施環境生成スクリプトの動作フローを示す。テスト実施環境生成スクリプトからは、テストデータ生成スクリプト、テストスクリプト生成スクリプト、汎用証明書生成プログラム、及び汎用 CRL 生成プログラムが外部コマンドとして起動され、結果として、指定されたテストケースに関するデータ群・スクリプト群が生成さ

れる。

テスト実施環境生成スクリプトの詳細については、テスト実施環境生成スクリプト仕様書に詳しく書かれている。

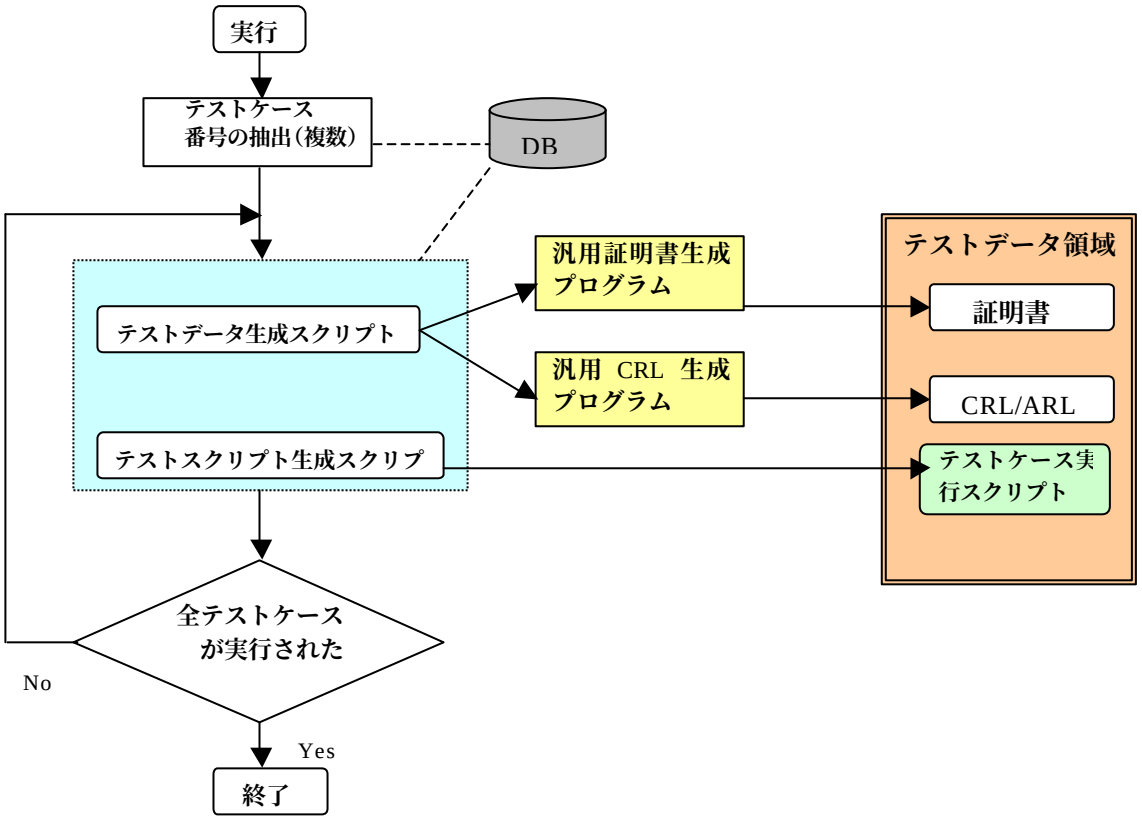


図 7-2 テスト実施環境生成スクリプトのフロー

7.3.3 テスト実行スクリプト

ユーザは、**テスト実行スクリプト**を実行することによって、**テスト実施環境生成スクリプト**によって生成されたデータ群・スクリプト群を元に、テストを実行することができる。起動の際の代表的なオプションとして表 7-3 が挙げられる。

表 7-3 テスト実施環境生成スクリプトの代表的なオプション

形式	説明
-lh LDAP ホスト名	オンラインでのテスト実行時に LDAP ホスト名を指定
-s DB サーバ名	テスト結果を DB へ格納する場合に指定

テスト実行スクリプトは、テストケース毎に用意された、**テストケース実行スクリプト**を順番に起動することで、テストの実行を行う。さらに必要であれば、データ

ベースへの実行結果の登録を行う。

テストケース実行スクリプトは、テスト実施環境生成スクリプトによって自動的に生成されたスクリプトであり、モード（オンライン・オフライン）によってその内容が異なる。図 7-3 に、オンラインモード用に生成されたテストケース実行スクリプトの動作フローを、図 7-4 に、オフラインモード用に生成されたテストケース実行スクリプトの動作フローを、それぞれ示す。

オンラインモード（図 7-3）では、テストケース実行スクリプトは、テスト対象プログラムを起動してパス検証を行う直前に、サーバ側でテスト環境生成プログラムを起動する。テスト環境生成プログラムは、OCSP レスポンダシミュレータの応答に必要な情報を準備したり、LDAP 登録スクリプトを起動することで、証明書、CRL 及び ARL を LDAP サーバへ登録する。

テストケース実行スクリプトは次に、テスト対象プログラムを起動する。テスト対象プログラムは、証明書、CRL 及び ARL を LDAP サーバへ問い合わせて取得する。またテストケースによっては、OCSP レスポンダシミュレータへ問い合わせることもある。

最後に、必要に応じてテスト結果ローダを起動することで、テスト対象プログラムの実行結果が DB へ登録される。

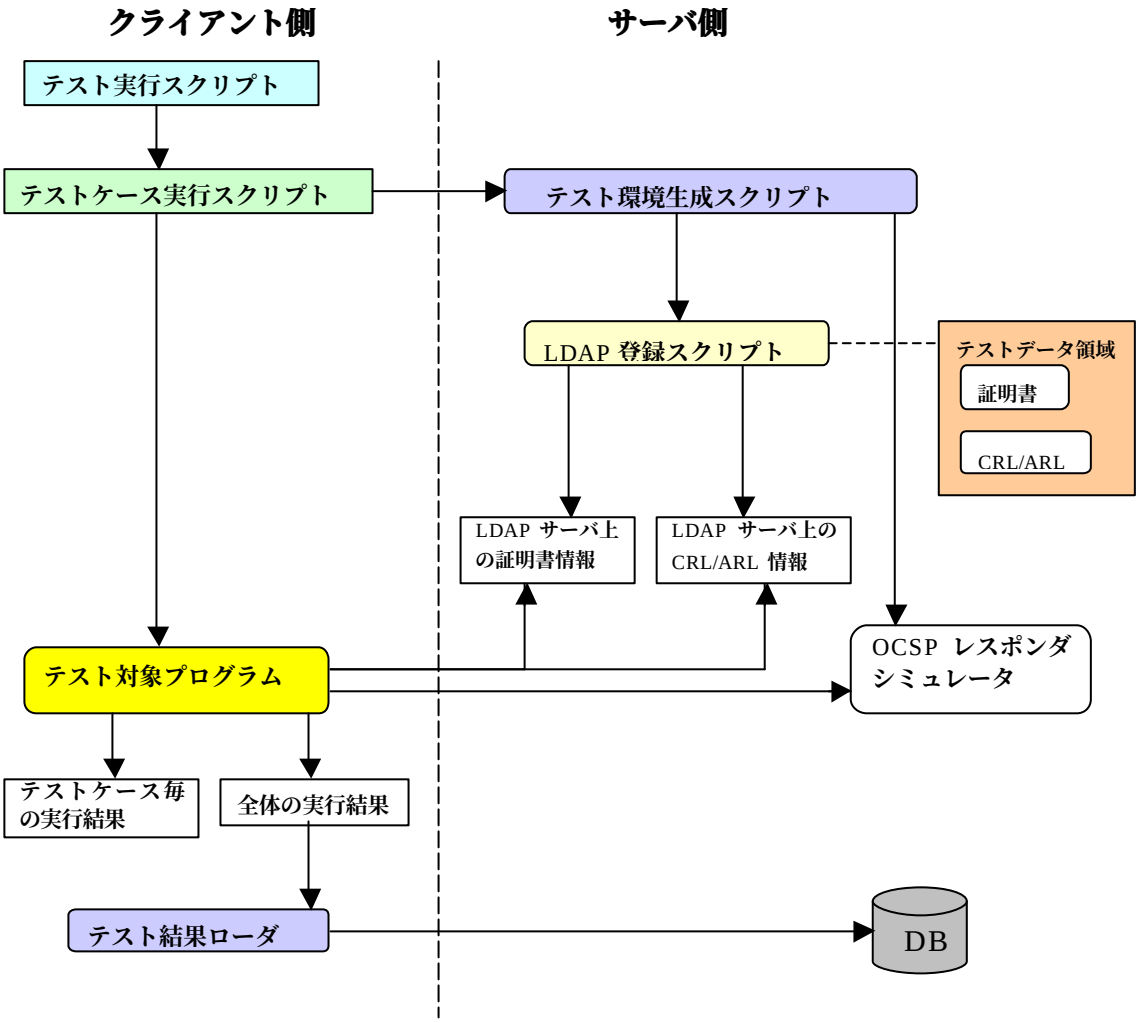


図 7-3 テストケース実行スクリプトのフロー（オンライン）

オフラインモード（図 7-4）では、テストケース実行スクリプトは、オンラインモードのようにテスト環境生成スクリプトの起動は行わずに、テスト対象プログラムを起動する。テスト対象プログラムは、証明書、CRL 及び ARL を LDAP サーバへは問い合わせず、ローカルファイルから取得する。

テスト対象プログラムを起動した後の動作はオンラインモードと同一であり、必要に応じてテスト結果ローダを起動することで、テスト対象プログラムの実行結果が DB へ登録される。

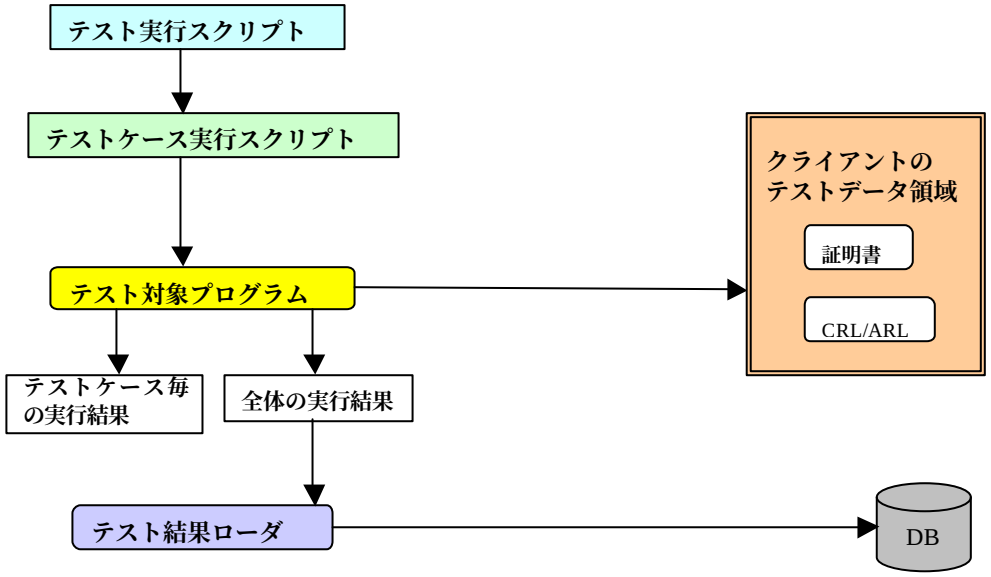


図 7-4 テストケース実行スクリプトのフロー（オフライン）

7.3.4 テスト対象プログラム

テスト対象プログラムとは、証明書、CRL 及び ARL を、ローカルファイル（オフライン）あるいはリポジトリ（オンライン）から取得し、必要に応じて OCSP レスポンダを利用し、証明書有効性の検証を行う機能を有するプログラムである。

本項では、テスト対象プログラムの使用方法について簡単に説明する。なお、テストスイートを利用した一連のテスト実行手順の中では、ユーザが直接テスト対象プログラムを起動することは無い。テスト対象プログラムの詳細な仕様については、「テスト対象プログラム外部仕様書」に書かれている。表 7-4 に、テスト対象プログラムの概要を示す。

表 7-4　テスト対象プログラムの概要

形式	gpkverify [オプション] ipcy1 ipcy2 … ipcy<n>
機能	指定された証明書及びポリシーを利用し、対象データの署名検証・証明書検証を行う。
終了ステータス	0　正常終了 ＞0　検証エラーが発生した。 ＜0　内部エラー（システムエラー）が発生した。

テスト対象プログラムでは、以下の通りに引数を設定することで、initial-policy-set を任意数指定することができる。

gpkverify init-policy1 init-policy2 …… init-policy<n>

このように、ポリシーを複数指定する場合は、各引数間を空白（スペース）で区切る。ポリシーは省略可能であり、省略時は、NULL を示す。規定値として、以下の値を使用可能とする。

"ANY"　　：全てのポリシーを受け入れる場合(any-policy)。
"NULL"　　：ポリシーを指定しない場合(NULL)。
ただし、-ie オプションが有効な場合は、引数として NULL 以外の値を指定しなければならない。

表 7-5 に、テスト対象プログラムで利用可能なオプションの一覧を示す。

表 7-5　テスト対象プログラムのオプション一覧

オプション	オプションの引数	説明
-v	なし	このテストプログラムが基づいているテストプログラム仕様のバージョン番号を表示する。実装必須。
-d	File 名	署名検証検証を行う対象となるデータファイルを指定する。
-f	Path 名	署名検証・証明書検証を行う対象となるデータファイルが格納されているフォルダ/ディレクトリを指定する。-f オプションによりデータが指定される場合は、path で指定されたフォルダ/ディレクトリ以下全てのファイルが検証対象となる。
-c	File 名	検証対象証明書ファイル（der 形式）を指定する。-c オプションが指定された場合は、署名検証を行わず、証明書検証（証明書パス構築・検証）のみを行う。
-t	File 名	署名・証明書検証側トラストアンカーの自己署名証明書ファイル（der 形式）を指定する。

-P	[File 名]	中間 CA の証明書ファイルを指定する。中間 CA の証明書ファイルが１つも無い場合は、「-P」のみを与える。ファイルを複数指定する場合は、コロン「:」で区切る。このオプションが指定された場合はオフラインモードで動作し、リポジトリや OCSP への問い合わせは行わない。※
-ip	なし	initial-policy-mapping-inhibit フラグの有効/無効を指定する。-ip オプションがある場合、有効となる。
-ie	なし	initial-explicit-policy フラグの有効/無効を指定する。-ie オプションがある場合、有効となる。その場合、コマンド引数（initial-policy-set を指定する）は、NULL 以外でなければならない。
-ia	なし	initial-any-policy-inhibit フラグの有効/無効を指定する。-ia オプションがある場合、有効となる。
-l	Host 名	LDAP サーバのホストとポート番号を「IP アドレス:ポート番号」もしくは「FQDN:ポート番号」の形式で指定する。

※各 CA が発行する CRL/ARL については、各 CA 証明書（トラストアンカー、中間 CA 証明書）のファイル名の拡張子が crl あるいは arl となっているものを読み込む。これらのファイルは全て-f で指定されたパス配下にあるものとする。

テスト対象プログラムを使用した署名検証・証明書検証の例を以下に示す。

＜例１＞　署名検証

署名者 cn=test,o=gpki,c=jp、検証側トラストアンカー証明書 tanchor.cer、証明書リポジトリとして ldap.gpki.go.jp を使用し、initial-explicit-policy フラグを有効にして、ファイル test.p7m の署名検証を行う。

```
gpkverify -d test.p7m -t tanchor.cer -ie -l ldap.gpki.go.jp ANY

Signature verification: ng[cert expire] - signer certificate is expired.
```

＜例２＞　オンラインモードでの証明書検証

/project/date 配下にある target.1.1.1crt を検証対象証明書ファイルとし、

trust.1.1.1.crt をトラストアンカー証明書ファイルとして、証明書検証のみをオンラインモードで行う。署名検証は行わない。

gpkverify -f /project/data -c target.1.1.1.crt -t trust.1.1.1.crt -l ldap.gpki.go.jp:389

＜例３＞　オフラインモードでの証明書検証

/project/date 配下にある target.1.1.1crt を検証対象証明書ファイルとし、trust.1.1.1.crt をトラストアンカー証明書ファイルとして、証明書検証のみをオフラインモードで行う。
intermediate.1.1.1.crt 及び intermediate.1.1.2.crt を中間 CA の証明書ファイルとし、トラストアンカーや中間 CA の CRL/ARL については、それぞれのファイル名の拡張子が crl または arl となっているファイルを参照する。

gpkverify -f /project/data -c target.1.1.1.crt -t trust.1.1.1.crt -P intermediate.1.1.1.crt:intermediate.1.1.2.crt

7.3.5 GPKI 証明書検証サーバを用いたテスト

テストスイートにおいて、GPKI 証明書検証サーバシミュレータ（以下、GCVS シミュレータと書く）を用いたテストを実行するためには、上で示した手順とは異なる手順が必要である。

GCVS シミュレータを用いたテストが可能なのは、表 7-6 に示す９つのテストケースのみである。個々のテストケースについては、GPKI テストケース設計書に詳しく書かれている。

表 7-6　GPKI 証明書検証サーバを用いたテストケース一覧

テストケース番号	トラストアンカ	検証対象	検証対象証明書（EE）の状態
2000700	MPHPT	METI	正常
2001000	MPHPT	Asign2	正常
2001200	MPHPT	MOJ	正常
2003400	MPHPT	METI	失効
2003700	MPHPT	Asign2	失効
2003900	MPHPT	MOJ	失効
2006100	MPHPT	METI	期限切れ
2006400	MPHPT	Asign2	期限切れ

2006600	MPHPT	MOJ	期限切れ

GCVS シミュレータを用いたテストを実行するためにはまず、**GPKI 証明書検証サーバシミュレータ設定スクリプト**を起動することで、テストの事前準備を行う。このスクリプトは、GCVS クライアントによる問い合わせに対して、事前に応答のために必要なデータを用意する機能を持つ。簡単に言えば、テストスイートにおける、テスト実施環境生成スクリプトのようなものである。GPKI 証明書検証サーバシミュレータ設定スクリプトの詳細については、GPKI 証明書検証サーバシミュレータ設定スクリプト仕様書に書かれている。

ユーザは GPKI 証明書検証サーバシミュレータ設定スクリプトを実行した後に、GCVS クライアントを実行することで、テストを行うことができる。GCVS クライアントの詳細については、GPKI 証明書検証クライアント仕様書に詳しく書かれている。

7.4 テスト結果から見た考察

7.4.1 GPKI 模擬テストケースの結果と考察

(1) サンプルプログラム実行結果

CryptoAPI サンプルプログラム、Java サンプルプログラム、JDK 1.4 オリジナルのテスト実行結果を示す。(表 7-7) 期待値通りならば結果は○、そうでなければ結果は×とする。

表 7-7 GPKI 模擬テスト実行結果

テストケース	期待値	CAPI sample	Java sample	Sun JDK 1.4	Java サンプル実装備考
2000100	成功	○	○	×	CRL include IDP, But provider can not process like this.
2000200	成功	○	○	×	CRL include IDP, But provider can not process like this.

2000300	成功	○	○	×	CRL include IDP, But provider can not process like this.
2000400	成功	○	○	×	CRL include IDP, But provider can not process like this.
2000500	成功	○	○	×	CRL include IDP, But provider can not process like this.
2000600	成功	○	○	×	CRL include IDP, But provider can not process like this.
2000700	成功	○	○	×	CRL include IDP, But provider can not process like this.
2000800	成功	○	○	×	CRL include IDP, But provider can not process like this.
2000900	成功	○	○	×	CRL include IDP, But provider can not process like this.
2001000	成功	○	○	×	CRL include IDP, But provider can not process like this.
2001100	成功	○	○	×	CRL include IDP, But provider can not process like this.
2001200	成功	○	○	×	CRL include IDP, But provider can not process like this.
2001300	成功	○	○	×	CRL include IDP, But provider can not process like this.
2001400	成功	○	○	×	CRL include IDP, But provider can not process like this.
2001500	成功	○	○	×	CRL include IDP, But provider can not process like this.
2001600	成功	○	○	×	CRL include IDP, But provider can not process like this.
2001700	成功	○	○	×	CRL include IDP, But provider can not process like this.
2001800	成功	○	○	×	CRL include IDP, But provider can not process like this.
2001900	成功	○	○	×	CRL include IDP, But provider can not process like this.
2002000	成功	○	○	×	CRL include IDP, But provider can not process like this.
2002100	成功	○	○	×	CRL include IDP, But provider can not process like

					this.
2002200	成功	○	○	×	CRL include IDP, But provider can not process like this.
2002300	成功	○	○	×	CRL include IDP, But provider can not process like this.
2002400	成功	○	○	×	CRL include IDP, But provider can not process like this.
2002500	成功	○	○	×	CRL include IDP, But provider can not process like this.
2002600	成功	○	○	×	CRL include IDP, But provider can not process like this.
2002700	成功	○	○	×	CRL include IDP, But provider can not process like this.
2002800	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2002900	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2003000	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2003100	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2003200	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2003300	失敗	○	×	○	CRL include IDP, But provider can not process like this.
2003400	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2003500	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2003600	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2003700	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2003800	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2003900	失敗	○	×	○	CRL include IDP, But provider can not process like this.

2004000	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2004100	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2004200	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2004300	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2004400	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2004500	失敗	○	×	○	CRL include IDP, But provider can not process like this.
2004600	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2004700	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2004800	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2004900	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2005000	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2005100	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2005200	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2005300	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2005400	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2005500	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2005600	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2005700	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2005800	失敗	○	○	○	CRL include IDP, But provider can not process like

					this.
2005900	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2006000	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2006100	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2006200	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2006300	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2006400	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2006500	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2006600	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2006700	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2006800	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2006900	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2007000	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2007100	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2007200	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2007300	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2007400	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2007500	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2007600	失敗	○	○	○	CRL include IDP, But provider can not process like this.

2007700	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2007800	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2007900	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2008000	失敗	○	○	○	CRL include IDP, But provider can not process like this.
2008100	失敗	○	○	○	CRL include IDP, But provider can not process like this.

7.4.2 NIST テストケースの結果と考察

(1) サンプルプログラム実行結果

CryptoAPI サンプルプログラム、Windows 2000 標準の CryptoAPI、Wdinwos XP 標準の CryptoAPI、Java サンプルプログラムおよび JDK 1.4 オリジナルのテスト実行結果を示す。(表 7-8)

表 7-8 NIST テストケース実行結果

テストケース	期待値	CAPI sample	Win 2000	Win XP	Java sample	Sun JDK 1.4	Java サンプル実装備考
1000100	成功	○	○	○	○	○	
1000200	失敗	○	○	○	○	○	
1000300	失敗	○	○	○	○	○	
1000400	成功	○	○	○	○	○	
1000500	失敗	○	○	○	○	○	
1000600	失敗	○	○	○	○	○	
1000700	成功	○	○	○	○	○	
1000800	失敗	○	○	○	○	○	
1000900	失敗	○	○	○	○	○	
1001000	失敗	○	○	○	○	○	
1001100	失敗	○	○	○	○	○	
1001200	成功	○	○	○	○	○	

1001300	失敗	○	x	○	○	○
1001400	失敗	○	x	○	○	○
1001500	成功	○	○	○	○	○
1001600	成功	○	○	x	○	○
1001700	成功	○	○	x	○	○
1001800	成功	○	○	x	○	○
1001900	失敗	○	○	○	○	○
1002000	失敗	○	x	x	○	○
1002100	失敗	○	○	○	○	○
1002200	失敗	○	x	○	○	○
1002300	失敗	○	x	○	○	○
1002400	成功	○	○	○	○	○
1002500	失敗	○	x	○	○	○
1002600	成功	○	○	○	○	○
1002700	成功	○	○	○	○	○
1002800	失敗	○	x	○	○	○
1002900	失敗	○	x	○	○	○
1003000	成功	○	○	○	○	○
1003100	失敗	x	x	x	○	○
1003200	失敗	x	x	x	○	○
1003300	成功	○	○	○	○	○
1003400	成功	○	○	○	○	○
1003401	成功	○	○	○	○	○
1003402	成功	○	○	○	○	○
1003403	成功	○	○	○	○	○
1003404	失敗	○	x	x	○	○
1003405	成功	○	○	○	○	○
1003500	成功	○	○	○	○	○
1003501	失敗	○	x	x	○	○
1003600	成功	○	○	○	○	○
1003601	失敗	○	x	x	○	○
1003700	成功	○	○	○	○	○
1003701	失敗	○	x	x	○	○
1003800	成功	○	○	○	○	○
1003801	失敗	○	x	x	○	○
1003900	成功	○	○	○	○	○
1003901	成功	○	○	○	○	○

1003902	成功	○	○	○	○	○
1003903	成功	○	○	○	○	○
1003904	失敗	○	x	x	○	○
1003905	成功	○	○	○	○	○
1004000	成功	○	○	○	○	○
1004001	失敗	○	x	x	○	○
1004100	成功	○	○	○	○	○
1004101	失敗	○	x		○	○
1004200	成功	○	○	○	○	○
1004201	失敗	○	x	x	○	○
1004300	成功	○	○	○	○	○
1004301	失敗	○	x	x	○	○
1004400	成功	○	○	○	○	○
1004401	失敗	○	x	x	○	○
1004500	失敗	x	x	x	○	○
1004501	失敗	○	x	x	○	○
1004600	成功	○	○	○	○	○
1004601	成功	○	○	○	○	○
1004602	成功	○	○	○	○	○
1004603	成功	○	○	○	○	○
1004604	失敗	○	x	○	○	○
1004605	失敗	○	x	○	○	○
1004700	失敗	○	x	○	○	○
1004701	失敗	○	x	○	○	○
1004800	成功	○	○	○	○	○
1004801	成功	○	○	○	○	○
1004802	成功	○	○	○	○	○
1004803	成功	○	○	○	○	○
1004804	成功	○	○	○	○	○
1004805	失敗	○	x	x	○	○
1004900	成功	○	○	○	○	○
1004901	成功	○	○	○	○	○
1004902	成功	○	○	○	○	○
1004903	失敗	○	x	x	○	○
1004904	成功	○	○	○	○	○
1004905	成功	○	○	○	○	○
1005000	成功	○	○	○	○	○

1005001	成功	x	○	○	○	○
1005002	成功	○	○	○	○	○
1005003	成功	x	○	○	○	○
1005004	成功	○	○	○	○	○
1005005	成功	x	○	○	○	○
1005100	成功	○	○	○	○	○
1005101	失敗	○	x	x	○	○
1005200	成功	○	○	○	○	○
1005201	失敗	○	x	x	○	○
1005202	成功	○	○	○	○	○
1005203	失敗	○	x	x	○	○
1005204	成功	○	○	○	○	○
1005205	成功	○	○	○	○	○
1005300	成功	○	○	○	○	○
1005301	成功	○	○	○	○	○
1005302	成功	○	○	○	○	○
1005303	成功	○	○	○	○	○
1005304	成功	○	○	○	○	○
1005305	失敗	○	x	x	○	○
1005306	成功	○	○	○	○	○
1005307	成功	○	○	○	○	○
1005400	失敗	○	x	○	○	○
1005500	失敗	○	x	○	○	○
1005600	成功	○	○	○	○	○
1005700	成功	○	○	○	○	○
1005800	失敗	○	x	○	○	○
1005900	失敗	○	x	○	○	○
1006000	失敗	○	x	○	○	○
1006100	失敗	○	x	○	○	○
1006200	成功	○	○	○	○	○
1006300	成功	○	○	○	○	○
1006400	失敗	○	○	○	○	○
1006500	失敗	○	○	○	○	○
1006600	失敗	○	○	○	○	○
1006700	成功	○	○	○	○	○
1006800	失敗	○	x	x	○	○
1006900	失敗	○	○	○	○	○

1007000	失敗	○	x	x	○	○	
1007100	失敗	○	○	○	○	○	
1007200	失敗	○	○	○	○	○	
1007300	失敗	○	○	○	○	○	
1007400	成功	○	○	○	○	○	
1007500	失敗	○	○	○	○	○	Delta CRL dose not exists.
1007600	失敗	○	○	○	○	○	

(2) Windows 標準 CryptoAPI を利用した場合の考察

CryptoAPI の「Certificate Verification Functions」を利用した場合の期待値と異なる場合について考察する。

(a) crlSign との関連について

本実装にて利用しているライブラリでは、検証対象の証明書の発行元の署名鍵と同一の署名鍵で CRL が署名されていた場合、その発行元証明書が鍵使用目的に crlSign が無い場合も有効としている。次の 2 つが該当(tc1003100、tc1003200)

(b) requiredExplictPolicy との関連について

ポリシ検証初期パラメータで、init-explicit-policy が FALSE の場合、パス中の証明書に requiredExplictPolicy に関わらず、ポリシを要求しないため、有効となる。tc1004500 が該当。

(c) anyPolicy との関連について

ポリシ検証初期パラメータで、init-explicit-policy が TRUE の場合、パス中の証明書に anyPolicy があった場合、検証処理が例外となり、無効となる。次の 3 つが該当(tc1005001、tc1005003、tc1005005)

(3) 独自パス構築・検証ルーチンを CAPI の Revocation Provider とした考察

独自のパス構築・検証ルーチンを CryptoAPI の Revocation Provider として場合、パス構築に関して、ネットワーク上のリポジトリにある証明書を動的に採取する機能を持たないため、中間 CA の証明書をネットワーク上のリポジトリから採取することが前提である、GPKI カテゴリ、オリジナルカテゴリの検証を行っていない。

(a) Windows 2000

Windows 2000 では相互認証したパスを構築することができないため、NIST カテゴリでは、相互認証を前提としたテストでは期待値と異なる。また、ポリシ関連の処理が適切に行われない。

(b) Windows XP

Windows XP では、相互認証したパスを構築することが可能なために、Windows2000 の検証結果に比べて、期待値と合致するものが増えている。ただし、Windows XP でも、ポリシ検証の初期パラメータを指定することができないため、検証結果が期待値と異なるものがある。

7.4.3 オリジナルテストケースの結果と考察

(1) サンプルプログラム実行結果

CryptoAPI サンプルプログラム、Java サンプルプログラム、JDK 1.4 オリジナルのテスト実行結果を示す。

テストケース	期待値	CAPI sample	Java sample	Sun JDK 1.4	Java サンプル実装備考
3000101	成功	○	○	○	
3000102	成功	x	○	×	Can not process KEY-UPDATE
3000103	成功	○	○	×	Can not process KEY-UPDATE
3000104	成功	○	○	○	
3000201	成功	○	○	○	
3000202	成功	○	○	○	
3000203	成功	○	○	○	*1
3000204	成功	○	○	○	*1
3000301	成功	○	○	○	
3000302	成功	x	○	×	UNKNOWN
3000303	成功	○	○	×	UNKNOWN
3000304	成功	○	○	○	
3000401	成功	○	○	○	
3000402	成功	○	○	○	

3000501	失敗	○	○	○	
3000502	失敗	○	×	○	
3000503	失敗	○	×	○	
3000601	成功	○	○	○	
3000602	失敗	○	○	○	
3000701	成功	○	○	○	
3000702	失敗	○	○	○	
3000801	失敗	○	○	○	
3000901	失敗	x	○	○	
3001001	失敗	○	○	○	
3001101	失敗	○	○	○	
3001201	失敗	○	×	○	
3001202	成功	x	○	×	UNKNOWN
3001301	失敗	○	×	○	
3001302	成功	x	○	×	UNKNOWN
3001401	成功	x	○	×	UNKNOWN
3001402	成功	x	○	×	UNKNOWN
3001501	成功	○	○	○	
3001502	成功	○	○	○	
3001503	成功	○	○	○	
3001504	成功	○	○	○	
3001601	成功	○	○	○	
3001602	成功	○	○	○	
3001701	失敗	○	○	○	1st CA certificate has a certificate policy "0.2.392.200117.1.9.2002.3.103999.1.10" and policyConstraints.required=0. 2nd CA is name rollover certificate. Certificate has no policy extension. So it is failed to build a policy tree(For GPKI) .Can not verify CRL signature, Because NOT care KEYUPDATE schema(For JDK)
3001702	失敗	○	×	○	1st CA certificate has a certificate policy "0.2.392.200117.1.9.2002.3.103999.1.10" and no policy-constraints extension. Look like permitted to failing that ca not buildi policy tree.(For GPKI) Can not verify CRL signature, Because NOT care KEYUPDATE schema(For JDK)
3001703	成功	x	○	×	1st CA certificate has a certificate policy

					"0.2.392.200117.1.9.2002.3.103999.1.10" and policyConstraints.required=2. 2nd CA is name rollover certificate. Certificate has no policy extension. So it is failed to build a policy tree.(For GPKI) Can not verify CRL signature, Because NOT care KEYUPDATE schema(For JDK)
3001801	失敗	○	○	○	
3001802	失敗	○	○	○	
3001803	成功	x	○	×	UNKNOWN
3001901	失敗	○	○	○	
3001902	失敗	○	○	○	

*1 Directory String match & name rollover issues.

(2) Printable-UTF8 Name Rollover の問題

テストケース 3000203、3000204 は、UTF8 と PrintableString の場合に、NameRollover ができるかというテストであるが、どちらのサンプル実装においても検証成功となり期待値と合致している。これは、X.520 を参照した結果、において PrintableString である文字セットの場合は、エンコードと関係なく(UTF8 であろうと、PrintableString であろうと)PrintableString と同じ処理をすべきと判断したためである。

(3) Windows 標準 CryptoAPI を利用したプログラムの考察

CryptoAPI の「Certificate Verification Functions」を利用した場合の期待値と異なる場合について考察する。

トラストポイントの CA が鍵更新を行った場合、CRL を発行している、署名鍵と異なる署名鍵の証明書をトラストポイントとしてパラメータを与えた場合、CRL の署名検証処理が例外となり、無効となる。期待値と異なる検証結果全てがこれに当てはまる。

8 GPKI テストケース

本章では、前章の相互運用テストスイートで利用するためにサンプルとして用意した GPKI テストケースについて解説する。

GPKI テストケースは、4 章で解説した主なテストクライテリアなどをもとに、以下の３つのテストカテゴリに分けて定義した。

- GPKI 模擬テストケース
- NIST テストケース
- オリジナルテストケース

8.1 GPKI 模擬テストケース

GPKI 模擬テストケースでは、GPKI が提供する枠組みである民間からの電子申請(B2G), 官職による公文書の発行(G2B), 府省間の連絡(G2G)について、以下のような認証パス構築・パス検証ができることを要件としている。

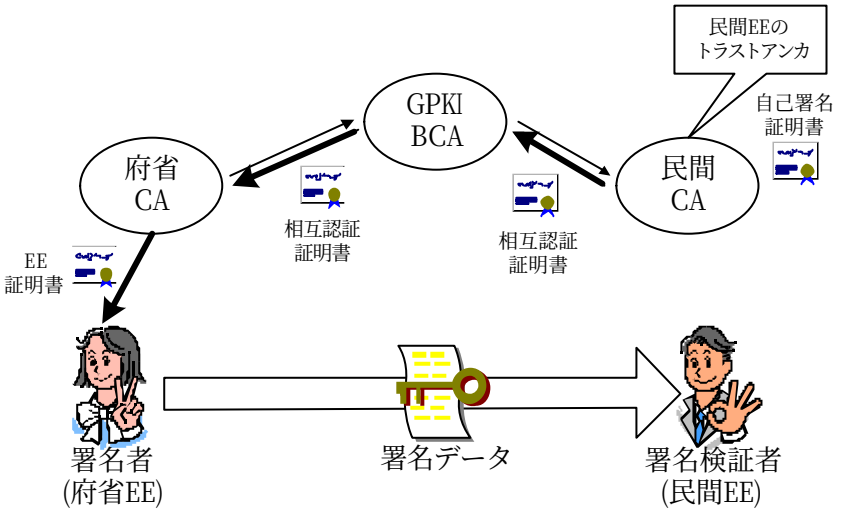


図 8-1 Ｇ２Ｂの認証パス

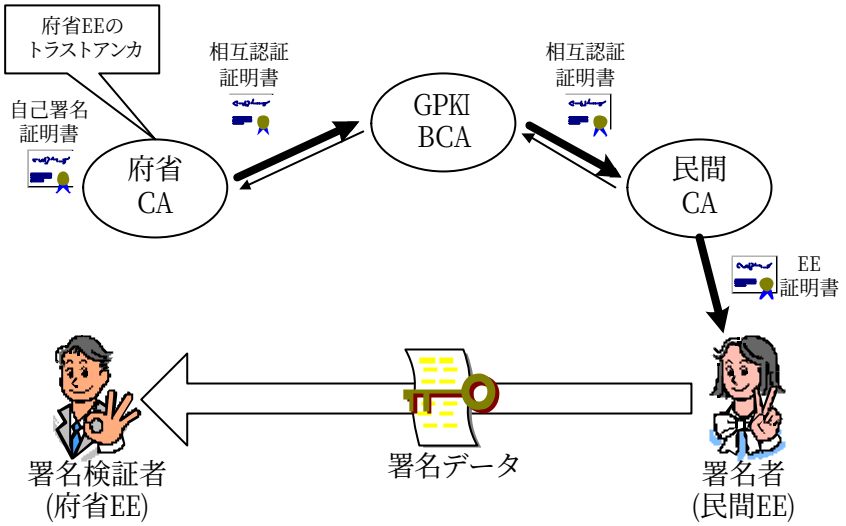


図 8-2　B２Gの認証パス

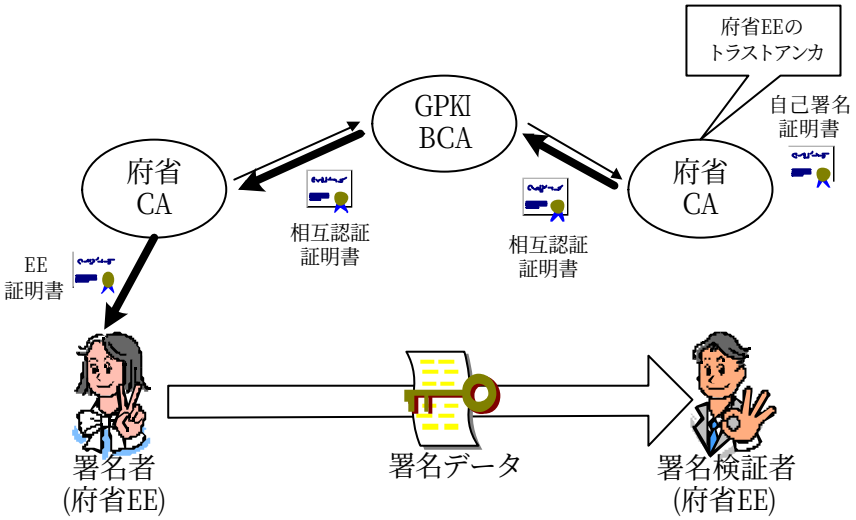


図 8-3　G２Gの認証パス

本テストケースでは、これらの各認証パスにおいて、署名者証明書の状態が以下のような場合について検証する。

- 有効な署名者証明書
- 失効している署名者証明書
- 有効期限の切れた署名者証明書

またこれらの認証パスには、GPKI で既に運用されている以下の各認証局のデータを模擬して用いる。

- 府省認証局

- 経済産業省認証局(およびその下位認証局)
- 国土交通省認証局
- 総務省認証局

- ブリッジ認証局
  - GPKI ブリッジ認証局
- 民間認証局
  - 日本認証サービス(株)の A-Sign2 認証局
  - セコムトラストネット(株)の セコムパスポート for G-ID 認証局
  - 商業登記認証局

GPKI模擬テストケースでは、上記の三種類の認証パス、三種類の署名者証明書状態以上のデータ(三種類の認証パス、三種類の署名者証明書状態、七つの認証局、

8.2 NIST テストケース

NIST テストケースでは、米国 NIST が DoD BCA における相互運用性テストの際に用いたテストケース（「X.509 Path Validation Test Suite, Version 1.07」）を再現している。

NIST テストケースは、証明書の検証と失効状態の検証とに大別される。

各テストケースにはテストレベルが設定されており、以下のように定義されている。

Level 0: 実行する必要がないテスト

Level 1: 相互運用に必要な最低限のテスト

Level 2: 保障を高めるテスト。予算がある限り検証すべき。

Level 3: デバグ用テスト。Level 2 までのテストが失敗した場合に、必要に応じてテストする。

各テストケースは、それぞれの視点から設計されているが、実際にはいくつか重複するテストケースが存在する。このようなテストケースについては、各項目でテスト要件のみを定義し、実際のテストケースはある一つの項目でのみ定義されている。

8.2.1 証明書検証

(1) 証明書処理テスト(CP)

X.509 や RFC3280 で必要としている、各証明書の基本的な検証内容を確認すべきテストケースを定義している。

- ✧ 証明書の署名検証(CP.01)

- ✧ 証明書の有効期限検証(CP.02, CP.03)
- ✧ 証明書の DN 比較(CP.04)
- ✧ 証明書の失効検証(CP.05, CP.06)

(2) 中間証明書処理テスト(IC)

中間証明書について更に確認すべきテストケースである。

- ✧ cA フラグの検証(IC.01, IC.02)
- ✧ パス長制約の検証(IC.03 → PL.01 にて定義)
- ✧ keyUsage(keyCertSign)の検証(IC.04, IC.05)
- ✧ keyUsage(cRLSign)の検証(IC.06)

(3) ポリシ処理テスト(PP)

X.509 の authority-constrained-policy-set と user-constrained-policy-set という二種類の制約方法に従って、テストケースを定義している。また、テスト結果の表示方法についてもテスト要件を定義している。

- ✧ authority-constrained-policy-set の検証(PP.01)
- ✧ user-constrained-policy-set の検証(PP.08)
- ✧ ポリシ修飾子の検証(PP.02: 未定義)
- ✧ ポリシマッピングを禁止しない場合の検証(PP.03: 未定義)
- ✧ ポリシマッピングを禁止する場合の検証(PP.04: 未定義)
- ✧ requireExplicitPolicy の検証(PP.06)
- ✧ アプリケーションでの検証結果表示に関する検証(PP.05, PP.07, PP.09→PP.06 にて定義, PP.10)

(4) パス長テスト(PL)

- ➢ パス長の検証(PL.01)

8.2.2 失効状態の検証

(1) fullCRL 関連テスト

CRL を用いて検証する場合について、CRL の基本的な検証と、issuingDistributionPoint や deltaCRL を用いた場合の検証について定義している。

- ➢ CRL による失効検証(RL.01)

- CRL の署名検証(RL.02)
- CRL と証明書の発行者比較(RL.03)
- CRL で失効されていた時の検証(RL.04)
- 未知の critical な crlEntryExtension の扱い(RL.05)
- 未知の critical な crlExtension の扱い(RL.06)
- nextUpdate の検証(RL.07)
- deltaCRL の検証(RL.08)
- issuingDP の検証(RL.09)

8.3 オリジナルテストケース

平成 13 年度に「PKI 関連相互運用性の調査」において実施した相互接続実験「Challenge PKI 2001」プロジェクトの成果には、アプリケーションの認証パス検証に関する問題点の指摘が含まれている。

本節では、これらの指摘事項を踏まえて、上記の「GPKI 模擬テストケース」および「NIST パス検証テストと同等のテストケース」に含まれないオリジナルテストケースを定義した。

8.3.1 CA の鍵更新

自己署名証明書を持つ CA が鍵更新する際には、旧鍵と新鍵を関係づけるための自己発行証明書も合わせて発行する必要がある。本テスト項目では、これら自己発行証明書を用いた認証パス構築・パス検証が正しく行えるかどうかを検証する。

- OldWithOld を用いたケース
- NewWithOld を用いたケース
- OldWithNew を用いたケース
- NewWithNew を用いたケース

8.3.2 PrintableString と UTF8String の混在

X.509 では、現時点では証明書の DN に PrintableString をはじめいくつかのエンコード方法の使用を認めているが、一方で 2003 年 12 月 31 日以降に発行する証明書は、UTF8String でエンコードしなければいけないことも明記している。

本テスト項目では、この前後の時期に想定され得る PrintableString でエンコードされた証明書と UTF8String でエンコードされた証明書が混在する状況で、EE が適切に認証パスの構築・検証を行い、署名データを正しく検証できることを確認

する。

- PrintableStringのCAとUTF8StringのCAが相互認証するケース
- PrintableStringのCAがUTF8Stringへ鍵更新するケース

8.3.3 UTCTimeとGeneralizedTimeの混在

X.509では現時点では証明書の有効期間にUTCTimeとGeneralizedTimeの２種類のエンコード方法の使用を認めているが、2050年以降にわたって有効な証明書を発行する際には、GeneralizedTimeでエンコードしなければいけないことも明記している。

本テスト項目では、この前後の時期に想定され得るUTCTimeで有効期間をエンコードした証明書とGeneralizedTimeで有効期間をエンコードした証明書が混在する状況で、EEが適切に認証パスの構築・検証を行い、署名データを正しく検証できることを確認する。

- UTCTimeのCA-XとGeneralizedTimeのCA-Yが相互認証するケース

8.3.4 OCSPとCRLの混在

PKIでは主に証明書の失効情報を提供する方法として、リポジトリで失効リストを公開する方法と、OCSPレスポンダによって証明書の失効情報を個々に提供する方法との２種類がある。GPKIでは府省認証局は統合リポジトリでCRL/ARLを公開するが、商業登記認証局はOCSPレスポンダによって失効情報を個々に提供している。

本テスト項目では、従ってOCSPモデルとCRLモデルが混在するGPKI環境下で、EEが適切に認証パスの構築・検証を行い、署名データを正しく検証できることを確認する。

- 認証パス中に商業登記OCSPレスポンダが存在するケース
- 認証パス中に一般のOCSPレスポンダが存在するケース
- OCSPレスポンスを検証できないケース

8.3.5 ポリシ制御

GPKIの各府省・民間認証局はブリッジ認証局と相互認証する際に、各認証局がGPKIで利用することを表す証明書ポリシを決め、これらをマッピングすることによって、適切なセキュリティレベルを維持している。

本テスト項目では、EEが、証明書ポリシやポリシマッピングを含んだ認証パスの構築・検証を行い、署名データを正しく検証できることを確認する。

- 認証パス中に、要求した証明書ポリシーを含んでいるケース
- 認証パス中に、要求した証明書ポリシーを含んでいないケース

- 認証パス中の証明書ポリシーが正しくマッピングされているケース
- 認証パス中の証明書ポリシーが正しくマッピングされてないケース

8.3.6 各種制約

GPKI では、異なる運営主体を持つ各認証局間のセキュリティレベルを維持するために、ポリシ制約によって証明書ポリシを用いた認証パスの検証を必須としている。また、ブリッジ認証局を介した B2B を防ぐために、民間認証局からブリッジ認証局へと発行する相互認証証明書に名前制約を含めることを必須としている。

また、GPKI では特に必須としていないが、認証パスの長さを制限するために、パス長制約が X.509 で定義されている。

本テスト項目では、EE がこれら各種制約を含めた認証パスの構築・検証を行い、署名データを正しく検証できることを確認する。

- 認証パスの長さを制約したケース
- 認証パス中の証明書ポリシーを制約したケース
- 認証パス中でのポリシーマッピングを制約したケース
- 認証パス中での名前空間を制約したケース

8.3.7 DN のエンコードに関連するテストケース

2003 年 12 月 31 日以降、DN のエンコーディングには UTF8 が使用される。これにより、認証パス検証におけるいくつかの検証処理で不具合が発生する可能性がある。本テスト項目では、これらについて以下の２種類に大別して検証を行う。

(1) 各種制約の評価

DN エンコーディングの変更に伴い、自己署名証明書が正しく自己署名証明書として認識されなくなる可能性がある。認証パス検証の中では、自己署名証明書か否かで処理が変わるものがいくつかある。

本テスト項目では、EE がこれら各種制約を含めた認証パスの構築・検証を行い、署名データを正しく検証できることを確認する。

- BasicConstraints
- NameConstraints
- PolicyConstraints

(2) 失効情報の取得・検証

通常、失効情報は署名者の名前（DN）とシリアル番号をもって検索する。DN の

エンコーディングが変更された場合、署名者の名前が一致しなくなる可能性がある。

本テスト項目では、EE がこれら失効情報の取得・検証を含めた認証パスの構築・検証を行い、署名データを正しく検証できることを確認する。

- CRL と証明書
- OCSP レスポンスと証明書

9 まとめ

Challenge PKI 2002 は、PKI/GPKI の主にリライングパーティ(署名検証者)側における相互運用性の問題を解決することが大きなテーマである。

相互運用性の問題は他にも多々あるが、PKI の標準技術という観点から見た場合、このリライングパーティ(署名検証者)側における相互運用性、すなわち認証パス検証に関連した相互運用性の問題が一番難しい。そして、これらは、リライングパティ(署名検証者)側にとっては必須の機能であり、電子政府アプリケーションにおいても必ず実装されるものであると考えられる。

認証パス検証は、これまでの PKI アプリケーションにおいても、X.509、RFC3280 といった標準のサブセットに準拠した形で実装されてきた。しかし、それはサブセットであり、どんな状況でも対応できるというものではない。

特に、電子政府のように広い PKI ドメインにおける認証、そして、更に用途に応じた認証をしようとした場合、認証パス検証は難易度を増す。

日本に限らず、各国の電子政府は、これまでにない広い範囲での認証が要求されている。これまでにない要求ということは、標準に対しても、これまでにない要求が必要になるか、もしくは、これまで使用されていなかった部分が使われる面がある。従って、標準に沿って実装といっても単純ではない。

一方、標準化自体にも問題点がある。IETF では、RFC3365（IETF 標準プロトコルについての強いセキュリティ要件）において、IETF 標準プロトコルが適切な強いセキュリティメカニズムを利用しなければならない（MUST）としている。インターネットのプロトコルの多くは、シンプルさを特長として成長してきたが、ここに来てセキュリティは、もはや無視できなくなってきており、単にシンプルなだけでは通用しなくなってきている。そうした中で、シンプルな標準、シンプルな実装でプロトタイプの開発を繰り返すといった手法だけでは、新たなプロトコルの開発などは、困難になっているように思われる。

以上のような背景に中、Challenge PKI 2002 では、PKI/GPKI の相互運用性の問題を、将来に渡って解決する手がかりをさぐることも大きなテーマであった。そのため、これまでにも多く見受けられてきた、すでに実装された製品間での相互運用実験を行うというのではなく、もう少し根本的に、相互運用を達成するために何が必要かという観点から作業が進められた。そして、テスト環境/テストクライテリアの提供、サンプル実装の提供、本報告書にあるような標準から実装に至るまでの解説などを目標に掲げてきた。

実際に相互運用テストスイート上で、設計したテストケースを使用し、ふたつの

サンプル実装を動作させた。実際に動作させてみて感じたことは、RFC3280 のような複雑な標準を策定するためには、標準自体を検証するための体系が最初から必要なのではないかということである。

例えば、標準の作成と同時にテストケースの設計も進めるべきなのではないか、また、テストケースは、広く共有されるべきではないのかといったことが上げられる。このような手法を採用すると、標準の曖昧さなどの問題をより早くフィードバックし、できればドラフト段階で、標準の曖昧さの最小化などが行う事が容易になるのではないだろうか。

また、標準の実装段階においても、個々の実装者がテストケースを設計することは、余りにも負担が大きく独自のテストケースの氾濫と標準に対するカバレッジを下げる方向に働く。テストケースの共有化を行なえば、開発も促進されると考えられる。

こうしたことは、IETF などの標準化団体に、提案していくことも必要である。世界レベルでテストケースの共有ができれば、そのまま、世界レベルでの相互運用性の解決の一助にはなる。

日本の電子政府の仕様を策定する上でも同様のことが言える。GPKI、LGPKI、公的個人認証基盤、更に、日本だけでなく海外との相互認証などを行うといったことを目的とした相互運用性仕様の作成を想定した場合、この作業は、RFC3280 などの標準化作業と基本的には同じだと考えられる。そして、設計を行なう上で、相互運用テストスイートを使って、シミュレーションを行うといったことも重要となるかもしれない。

電子政府を成功させるためには、国内での仕様を取りまとめていくだけでなく、この仕様を基に、世界での標準の策定に積極的に関与して行くといったことも重要である。

Challenge PKI 2002 プロジェクトの成果が、PKI/GPKI のアプリケーションの開発の促進、そして、世界レベルでの標準化の策定の促進に寄与できれば、プロジェクトの意義が確認されることになる。

10 付録

10.1 X.509 4th Edition 証明書プロファイル

X.509 4th Edition[X509.4]7 章および 8 章において証明書プロファイルが記述されている。これを表にまとめた。(表 10-1、表 10-2)

10.1.1 基本領域

表 10-1 X.509 4th で規定される証明書基本領域

項目名	説明
Version	X509 証明書のバージョン。拡張領域がある場合は v3 となる。
serialNumber	証明書を発行した CA によって割り振られる、証明書毎にユニークな整数。
signature	CA が証明書に署名するときに使われるアルゴリズムの Algorithm Identifier。
Validity	この証明書の有効期間
notBefore	この時刻より証明書を有効とする。
notAfter	この時刻より証明書を有効期限切れとする。
Issuer	この証明書を発行/署名した CA の識別名。
Subject	公開鍵フィールドの公開鍵と関連付けられた主体者の識別名。
subjectPublicKeyInfo	公開鍵値とアルゴリズム名。
issuerUniqueID	Issuer 名が再利用される時、Issuer を一意に識別するための識別値。
subjectUniqueID	Subject 名が再利用される時、Subject を一意に識別するための識別値。

10.1.2 拡張領域

表 10-2 X.509 4th で規定される証明書拡張領域

項目名	説明	クリチカルフラグ
authorityKeyIdentifier	この証明書の署名検証に使われ	Non-critical

	る公開鍵を示す識別子。	
keyIdentifier	秘密鍵の識別子。	-
authorityCertIssuer	機関証明書発行者名。	-
authorityCertSerialNumber	機関証明書シリアルナンバー。	-
subjectKeyIdentifier	主体者鍵識別子。	Non-critical
keyUsage	鍵使用目的。	どちらでもよい
digitalSignature	電子署名の検証。	-
nonRepudiation	否認防止に使われる電子署名の検証。	-
keyEncipherment	秘密鍵等の機密情報の暗号化。	-
dataEncipherment	鍵、機密情報以外の、利用者のデータの暗号化。	-
keyAgreement	鍵共有のための公開鍵	-
keyCertSign	証明書上の CA 署名を検証。	-
cRLSign	CRL 上の機関署名を検証。	-
encipherOnly	keyAgreement ビットがセットされている時、鍵共有公開鍵をデータ暗号化にのみ使用。	-
decipherOnly	keyAgreement ビットがセットされている時、鍵共有公開鍵をデータ複合にのみ使用。	-
extKeyUsage	公開鍵の使用目的。keyUsage フィールドと矛盾がないよう追加・置き換えされる。	どちらでもよい
privateKeyUsagePeriod	証明書に含まれる公開鍵に対応する、秘密鍵の使用期限。署名鍵にのみ適用される。[RFC3280]4.2.1.2 節において推奨しないと明記	Non-critical
notBefore	この時刻より鍵を有効とする。	-
notAfter	この時刻より鍵を有効期限切れとする。	-
certificatePolicies	発行者によって承認された、証明書の用途に基づく証明書ポリシが設定される。	どちらでもよい

PolicyInformation	ポリシ識別子とポリシ修飾子で構成され、critical の場合、その規則に従って使用することが強制される。	4th Edition より “anyPolicy” 追加
CertPolicyId	ポリシ識別子	-
PolicyQualifierInfo	ポリシ修飾子	-
SupportedPolicyQualifiers	受け入れポリシ限定子	-
anyPolicy	全てのポリシを受け入れるときにセットされる。	4th Edition より追加。
policyMappings	issuerDomainPolicy と subjectDomainPolicy のペアで、２つのフィールドに設定された値が等しいことを定義する。	どちらでもよい
issuerDomainPolicy	証明書発行者側ポリシ	-
subjectDomainPolicy	主体者側ポリシ	-
subjectAltName	基本領域の subject とは別に、証明書の所有者を識別するための複数の名前形式。	どちらでもよい
otherName	OTHER-NAME 情報オブジェクトクラスで定義された形式の名前。	-
rfc822Name	RFC822 において定義されたインターネット電子メールアドレス。	-
dNSName	RFC1035 において定義されたインターネットドメイン名。	-
x400Address	ITU-T Tec.X.411|ISO/IEC 10021-4 において定義された O/R アドレス。	-
directoryName	ITU-T Tec.X.501|ISO/IEC9495-2 において定義されたディレクトリ名。	-
ediPartyName	EDI パートナーとの間で合意している形式の名前。	-

	uniformResourceIdentifier	RFC1630 において定義された WWW のためのユニークリソース識別子。	-
	iPAddress	二進数で表現される RFC791 において定義された IP アドレス。	-
	registeredID	CCITT Rec.X.660 | ISO/IEC 9834-1 に従って割り当てられている、登録されたオブジェクトのための識別子。	-
issuerAltName		基本領域の issuer とは別または追加の形式で、発行者名を設定。	どちらでもよい
subjectDirectoryAttributes		証明書所有者のために必要とされるディレクトリ属性値を格納。RFC2459 4.2.1.9 節ではでは使用しないよう勧告されていたが、[RFC3280]4.2.1.9 節ではその記述が無くなった。	Non-critical
basicConstraints		証明書所有者が、証明書の署名検証のために使用される公開鍵をもつ CA であるか否かを設定。	どちらでもよい
	cA	この証明書の公開鍵が証明書の署名検証に使用されるか否かを示す。	-
	pathLenConstraint	cA が True にセットされている時、認証パス中でこの証明書が相互認証した CA 証明書のパスの最大値を設定。	-
nameConstraints		CA 証明書でのみ使用される拡張子で、発行者が発行する証明書の名前を制限する。	どちらでもよい
	permittedSubtrees	許可サブツリー	-
	excludedSubtrees	除外サブツリー	-
policyConstraints		CA 証明書で使用し、認証パス中のポリシの関係を制限する。	どちらでもよい
	requireExplicitPolicy	受け入れ可能なポリシ識別子	-

	inhibitPolicyMapping	ポリシマッピングを禁止	4th Edition より追加。
inhibitAnyPolicy		ポリシ識別子として、*any-policy* の使用を禁止。	どちらでもよい
cRLDistributionPoints		CA 証明書、EE 証明書、属性証明書において、失効リストの配布点を指定。	どちらでもよい
freshestCRL		CA 証明書、EE 証明書において、最新の失効情報を取得するための情報を指定。	どちらでも良い 4th Edition より追加。

10.2 X.509 4th Edition CRL プロファイル

X.509 4th Edition[X509.4]8 章において CRL プロファイルが記述されている。これを表にまとめた。(表 10-3、表 10-4、表 10-5)

10.2.1 基本領域

表 10-3 X.509 4th で規定される CRL の基本領域

項目名	説明
Version	失効リストのバージョン（v1 or v2）
Signature	失効リストの署名アルゴリズム
Issuer	失効リストの発行者名
thisUpdate	失効リストの発行日
nextUpdate	失効リストの次回発行日
revokedCertificate	失効証明書リスト
userCertificate	失効証明書のシリアル番号
revocationDate	CA が失効処理した日時
crlEntryExtensions	失効証明書に関する追加情報

10.2.2 エントリ拡張領域

表 10-4 X.509 4th で規定される CRL エントリ拡張領域

項目名	説明	クリチカルフラグ

reasonCode	失効理由	証明書が失効した理由を指定。	Non-critical
Unspecified	未定義	失効理由を定義しない。 ※この場合、reasonCodeを省略すべき。	
keyCompromise	鍵危殆	EE の秘密鍵の信用性に問題	
cACompromise	CA 危殆	CA の秘密鍵の信用性に問題	
affiliationChanged	内容変更	鍵の信用失墜以外の理由で、証明書の内容等に変更があった。	
Superseded	取替	鍵の信用失墜以外の理由で、証明書が取り替えられた。	
cessationOfOperation	運用停止	鍵の信用失墜の疑いがなく、証明書が発行された目的で使用される必要がなくなった。	
certificateHold	証明書保留	証明書が保留されていることを示す。	
removeFromCRL	復旧	差分失効リストでのみ使用される。証明書失効情報が失効リストから削除される。	
priviledgeWithdrawn	特権取消	証明書中に記されていた特権が取り消されたために、証明書が失効された。	
aACompromise	属性危殆	属性証明書中の情報が危殆化した。	
holdInstractionCode	保留指示コード	reasonCode が保留(certificateHold)の証明書に対する行動を指示します。	Non-critical
invalidityDate	無効日	実際に証明書が無効となった日（またはその可能性のある日）	Non-critical

certificateIssuer	失効リスト中の証明書の発行社名。	失効リストの発行者と、証明書の発行者が異なる場合のみ使用。	Critical

10.2.3 拡張領域

表 10-5 X.509 4th で規定される CRL の拡張領域

項目名	説明	クリチカルフラグ
authorityKeyIdentifier	この証明書の署名検証に使われる公開鍵を示す識別子。	Non-critical
issuerAltName	基本領域の issuer とは別、または追加の形式で発行者名を設定。	どちらでもよい
cRLNumber	CRL 発行者によって割り当てられるシーケンスナンバー。	Non-critical
crlScope	CRL が管理する証明書の範囲に関する情報	Critical **4th Edition より追加。**
statusReferrals	適切な失効情報が使用されていることを保証する。	Critical **4th Edition より追加。**
cRLStreamIdentifier	ユニークな CRL 番号同士の前後関係を識別する。	Non-critical **4th Edition より追加。**
orderedList	失効証明書がシリアルナンバーでソートされている。	Non-critical **4th Edition より追加。**
deltaInfo	この CRL の差分失効リストが利用可能であることを示す。	Critical **4th Edition より追加。**
issuingDistributionPoint	このＣＲＬの配布点。	Critical
deltaCRLIndicator	参照されているベース失効リストに対して差分失効リストであることを示す。	Critical
baseUpdateTime	差分失効情報が提供された日時。	Non-critical

		4th Edition より追加。

10.3 RFC3280 証明書プロファイル

X.509 4th Edition([X509.4])の 7 章および 8 章で述べられている証明書プロファイルと、RFC3280 の 4 章で述べられている証明書プロファイルの差分のみ述べる

X.509 4th Edition との違いは拡張領域の以下の 2 拡張のみである。

表 10-6 X.509 4th/RFC3280 の拡張領域の差異

項目名	説明	クリテチカルフラグ
AuthorityInformationAccess	この拡張を含む証明書の発行者に対する，CA の情報とサービスにアクセスする方法。	Non-critical
SubjectInformationAccess	この拡張を含む証明書のサブジェクトに対する，情報とサービスにアクセスする方法。	Non-critical

10.4 RFC3280 CRL プロファイル

X.509 4th Edition([X509.4])の 8 章で述べられている CRL/ARL プロファイルと、RFC3280 の 5 章で述べられている CRL プロファイルの差分のみ述べる。

下記の拡張は、**X5094th edition でのみ記述**されており、RFC3280 には記述されていない。

※RFC3280 は、X.5094thEdition のドラフト段階から平行して作成された時間的経緯があり、X.5094thEdition の完成版に記述されている拡張群を全て反映していない。

表 10-7 X.509 4th/RFC3280 の CRL の差異

項目名	説明	クリティカルフラグ
crlScope	CRL が管理する証明書の範囲に関す	Critical

	る情報	
statusReferrals	ステータスリファラル	Critical
cRLStreamIdentifier	The CRL stream identifier field is used to identify the context within which the CRL number is unique.	Non-critical
orderedList	証明書の順序がシリアルナンバーでソートされている。	Non-critical
deltaInfo	差分失効リストの取得情報。	Critical
baseUpdateTime	差分失効情報が提供された日時。	Non-critical

10.5 X.509 4th Edition における CRL 種別のまとめ

証明書利用者にとって、CRL は表 10-8 に示す７つの種類に分類できる。図 10-1 に、Full and complete CRL、Full and complete EPRL、Full and complete CARL 及び Delta CRL、 EPRL、 CARL が持つ情報の違いを簡単に示す。

表 10-8 CRL の分類

名称	説明
Full and complete CRL	認証局によって発行された、失効した全てのエンドエンティティ及び CA 証明書の一覧。
Full and complete EPRL	認証局によって発行された、失効したエンドエンティティ及び証明書の一覧。 ※EPRL は、エンドエンティティ CRL ともいう
Full and complete CARL	認証局によって発行された、失効した CA 証明書の一覧。 ※CARL は、Certification Authority Revocation List の略。
Distribution Point CRL、EPRL、CARL	認証局によって発行された証明書の一部または全部を包括するもの。
Indirect CRL、 EPRL、CARL	失効した証明書の一覧を含む CRL。ただしその一覧に、その CRL を署名・発行した認証局によって発行されていない証明書を含むもの。
Delta CRL、EPRL、CARL	変更点だけを含む CRL（デルタ CRL)。このデルタ CRL の中に示された CRL を参照することで、与えられたスコープにおいて、初めて完全となる。デルタ CRL は、証明書の失効状態の全体像を形成

	するために、結合した CRL（同一のスコープについて完全な CRL）と共に使われる。
Indirect dCRL、 EPRL、 CARL	１つ以上の CRL に対する変更点だけを含む CRL。ただしその一覧に、その CRL を署名・発行した認証局によって発行されていない証明書を含む。 Indirect dCRL、 EPRL、 CARL は、与えられたスコープについて完全である。 ※dCRL は Delta CRL の略

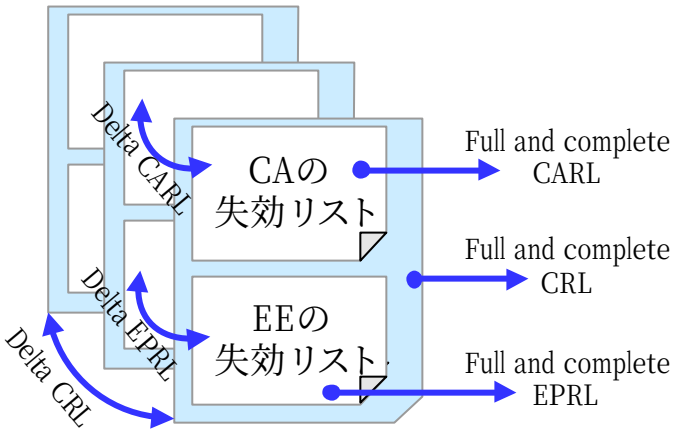


図 10-1 CRL の分類

証明書利用者は、これらの CRL を、CRL DP（CRL Distribution Point）や、証明書発行者のディレクトリエントリによって示された場所から取り出す。このエントリは、例えば以下の属性から読み込むことができる。

- Certificate Revocation List;
- Authority Revocation List;
- Delta Revocation List;

証明書等から、どの CRL が必要かを判断するためのパラメータを決定し、適切な CRL を取得すれば、証明書利用者は CRL を処理し、失効検証を行うことができる。

11 図一覧

12 表一覧

13 参考文献

[X509.4]
ITU-T Recommendation X.509 (2000) ISO/IEC 9594-8:2001,
"Information Systems - Open Systems Interconnection - The Directory: Public key and attribute certificate frameworks.",
March 2000

[RFC3280]
W.Polk, W.Ford, D.Solo,
IETF PKIX-WG RFC3280: " Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile",
April 2002
http://www.ietf.org/rfc/rfc3280.txt

[RFC2459]
R.Housley, W.Ford, W.Polk, D.Solo,
IETF PKIX-WG RFC2459: " Internet X.509 Public Key Infrastructure Certificate and CRL Profile",
January 1999
http://www.ietf.org/rfc/rfc2459.txt

[RFC2587]
S.Boeyen,T.Howes,P.Richard,
IETF PKIX-WG RFC2587: " Internet X.509 Public Key Infrastructure LDAPv2 Schema",
June 1999
http://www.ietf.org/rfc/rfc2587.txt

[RFC2256]
M.Wahl,
IETF NETWORK-WG RFC2256: "A Summary of the X.509(96) User Schema for use with LDAPv3",
December 1997
http://www.ietf.org/rfc/rfc2256.txt

[pkix-ldap3-draft]
D.W.Chadwick, S.Legg,
IETF PKIX-WG DRAFT: " Internet X.509 Public Key Infrastructure LDAP Schema and Syntaxes for PKIs",
June 2002
http://www.ietf.org/internet-drafts/draft-ietf-pkix-ldap-pki-schema-00.txt

[pkix-rmap]
A.Arsenault, Diversinet S. Turner,
IETF PKIX-WG: "Internet X.509 Public Key Infrastructure: Roadmap”,
January 2003
http://www.ietf.org/internet-drafts/draft-ietf-pkix-roadmap-09.txt

[uniformed search]
知識なしの探索 － 人工知能の探索アルゴリズム(2) 先を読んで知的な行動を選択するエージェント
http://aibm4.main.eng.hokudai.ac.jp/~kurihara/classes/AI/uninformed.ppt