IW2002 PKI応用編

富士ゼロックス株式会社
稲田 龍
<Ryu.Inada@fujixerox.co.jp>

PKIアプリケーション

アプリケーションの動向

- HTTPS
- S/MIME
- 電子署名アプリケーション

HTTPS

- SSLを使い通信路の暗号化と接続先(元)の認証を行う
- Internet Explorer/Netscape Navigatorなどが標準でサポート
- 多くの場合、サイト側の認証と暗号化のみが行われているが、クライアント側の認証も使われつつある
  - うまく使えば、Single Sign Onとなる
  - パスワードベースの認証に比べユーザの負担小

S/MIME

- MIMEを使い電子メールの暗号化と電子署名
  - メジャーではないが、そろそろ使われ始めている

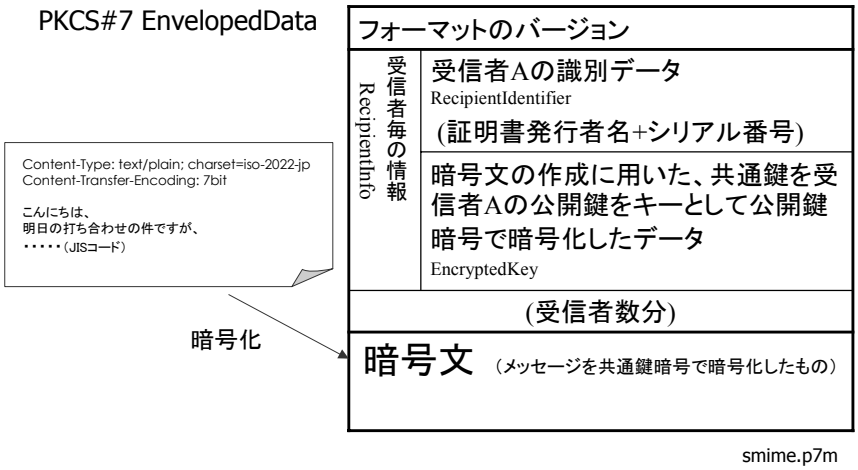
- Microsoft Outlook/Outlook Express
- Netscape Navigator
- OrangeSoft Winbiff
- Becky!
- MEW(電子署名のみ)

S/MIMEのメールメッセージ

- 本文や添付ファイルをメールの形式(MIME)に
- MIME化したメッセージを入力として署名データまたは暗号文を作成
- PKCS#7形式にエンコード
  - RFC2315
    PKCS #7 : Cryptographic Message Syntax Version 1.5
  - フォーマットはASN.1で表記される。
  - エンコードのルールはBERまたはDERに従う。
- BASE64変換して、規定のMIMEヘッダを付ける
  - RFC2311
    S/MIME Version 2 Message Specification
  - RFC2633
    S/MIME Version 3 Message Specification

暗号化データの形式

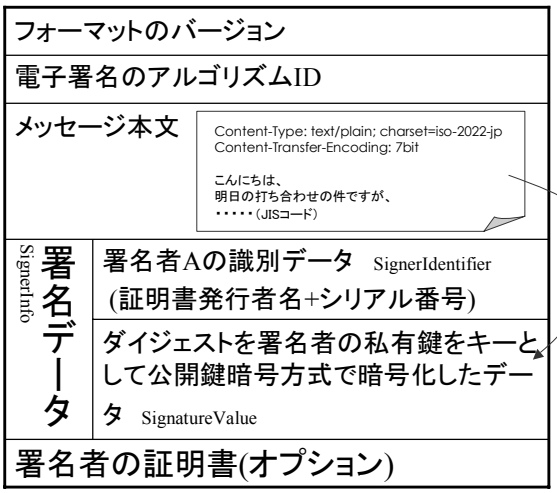
PKCS#7 EnvelopedData



smime.p7m

電子署名データの形式

PKCS#7 SignedData

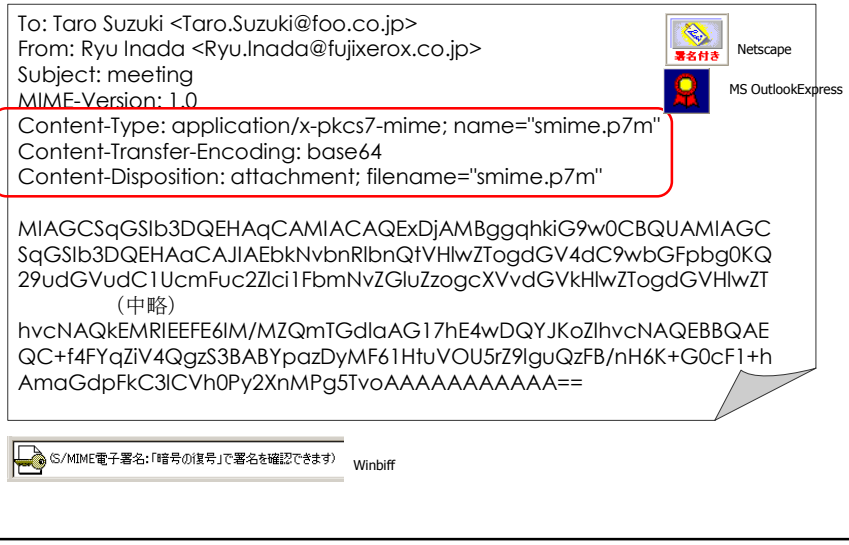


- メッセージ本文を含むタイプと、分離しているタイプがある。

smime.p7s
または、
smime.p7m

電子署名をつけたメールの例 1

電子署名をつけたメールの例 2



電子署名アプリケーション

- ファイル/データ等に対して電子署名

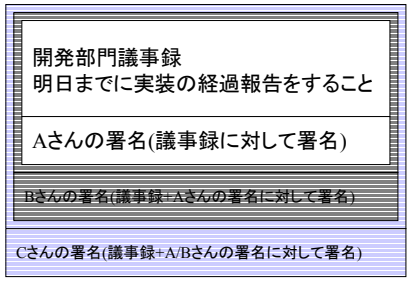
- 電子署名法の施行/電子政府での採用
- 専用アプリケーション
  - 電子申請など
- 汎用アプリケーション
  - Acrobat
  - Microsoft Office XP
  - DocuWorks

シリアル署名とパラレル署名

- 署名をどの部分に行うかの違い
- 用途により使い分けが必要
- Acrobat/Office XP/DocuWorksともにシリアル署名を実装

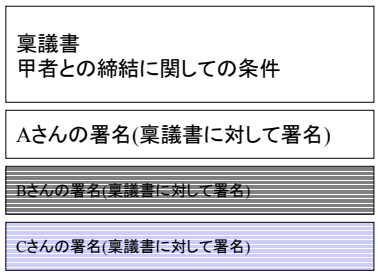
シリアル署名

- 署名を「追加」していくイメージ
- 長所
  - 署名の順番がわかる
- 短所
  - オリジナルの文書に対しての署名



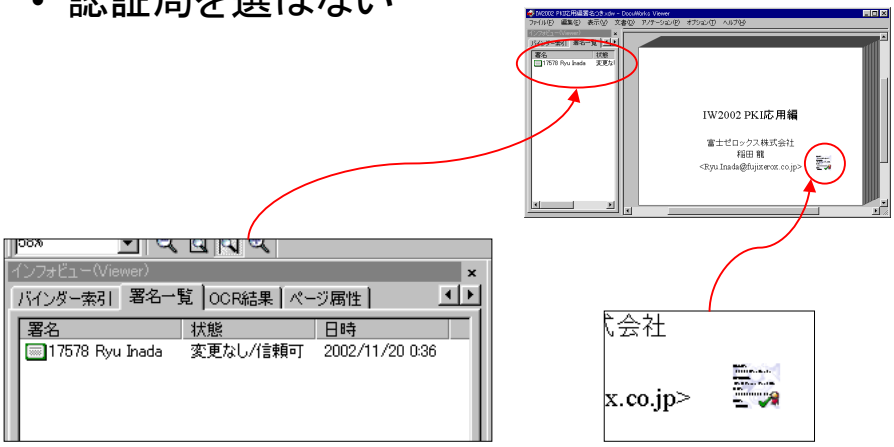
パラレル署名

- 署名対象に対してのみ署名を行う
- 順番に関係なく署名検証可能



DocuWorks V5電子証明書署名機能

- Windows Crypto API上で作成
- 認証局を選ばない



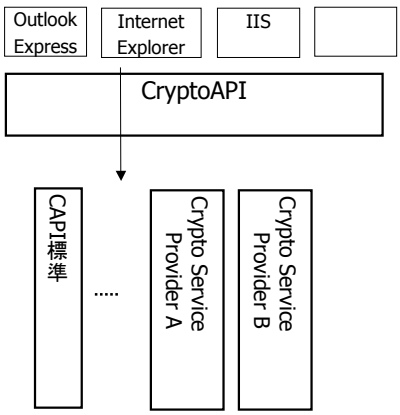
PKI実装

PKI実装面

- Windows系
  - Crypto API(Microsoft)
  - CertWorker(NEC)
  - OpenSSL
- JAVA系
  - JDK/JCE
- UNIX系
  - OpenSSL

Windows Crypto API

- CSP(Crypto Service Provider)モデル
- IE 4.0以降から提供
- 暗号エンジンをモジュール化
- 複数の暗号エンジンを保持
- Third Party提供のCPSを利用可能
- 証明書の検証に関しても良く考えられている
- 暗号エンジンを作る場合、Microsoftにコード署名をしてもらう必要あり



OpenSSL

- もとはUNIX系の実装でSSLeayと呼ばれていた。
- 多くのUNIX系プラットフォームのデファクト実装
  - Linux/*BSD*に採用
- Windowsプラットフォームでも動作
- ApacheのSSL/TLSのエンジンとして広く使われている
  - Apache 1.X+mod_ssl+OpenSSL
  - Apache 1.X+Apache_SSL+OpenSSL
  - Apache 2.X(標準でSSL/TLSをサポート)
- 0.9.7beta4が出ており、OCSPのサポート/楕円関数のサポートが入った

JDK 1.4のフレームワーク

- java.security.cert以下に実装されている。
- クライアントとして使う面では十二分な実装
  - JDK本体で証明書の基本的なハンドリングが可能
  - JCE(Java Cryptographic Extensions)で暗号周りの機能を提供
    - Windows同様Third PartyのJCEに差し替えることが可能
    - Sunより証明書を発行してもらい、その証明書でコード署名を行う必要あり
  - JSSE(Java Secure Socket Extensions)でSSL/TLSを提供

- RFC3280の証明書検証アルゴリズム相当のメカニズムを実装
- CertPathBulder/CertPathValidator/CertStoreの3つに仮想化
  - CertPathBulder
  - CertPathValidator
  - CertStore

JDK 1.4の証明書検証機能(その1)

- 必要となる部品
  - Java.security.cert.X509Extension
  - Java.security.cert.X509Certificate
  - Java.security.cert.X509CRL
  - Java.security.cert.PolicyQualifierInfo
- 問題点
  - 扱える拡張が少ない（get… で取れるものが少ない）
    - CRLDPをえることさえ出来ない
  - Java.security.cert.X509Extension .getExtensionValue(oid)で
    - 指定したOIDのデータがDERで取得できる
    - DERを解釈するAPIは無いので自分でやること
  - PolicyQualifierInfoはあるが、中途半端である
    - getExtensionValueでとってきて、解釈をして処理を行い適切な値をPolicyQaulierInfoに入れる必要がある。

JDK 1.4の証明書検証機能(その2)

- 必要となる部品
  - Java.security.cert.CertPathValidator(API)
  - Java.security.cert.PKIXCertPathChecker
  - Java.security.cert.CertStore
  - Java.security.cert.PKIXParameters(主な入力)
  - Java.security.cert.PKIXBuilderParameters (入力)
  - Java.security.cert.PKIXCertPathValidatorResult(結果）
- 問題点
  - Java.security.cert.X509Certificate取得出来ない物に関しての処理が甘い
  - 例外的にPolicy関連の処理はしているようである(RFC3280に準拠するため?)
  - 証明書検証処理の追加が一応可能である
    - Java.security.cert.PKIXCertPathChecker経由で行う
    - 一方でCRLCheckerは無いので…
    - でもCRLの一部のExtensionの処理はしているようである
      - どのExtensionを処理しているかが外からはわからない。
    - CRLChecker/RevocationCheckerのようなPKIXCertPathChecker同様に処理の拡張ルーチンを追加できる枠組みが必要ではないか?

JDK 1.4の証明書検証機能(その3)

```
importjava.io.IOException;
importjava.security.InvalidAlgorithmParameterException;
importjava.security.NoSuchAlgorithmException;
importjava.security.cert.CertPath;
importjava.security.cert.CertPathValidator;
importjava.security.cert.CertPathValidatorResult;
importjava.security.cert.CertPathValidatorException;
importjava.security.cert.PKIXBuilderParameters;
importjava.security.cert.PKIXCertPathValidatorResult;
importjava.util.Set;
importjava.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);
        }
}
```

検証用のパラメータを設定

Validatorを実行

最小限のRFC3280準拠
証明書検証のサンプルコード

JDK 1.4の証明書検証機能(その4)

多少、凝った証明書検証サンプルコード

検証用パラメータの設定

```
// 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
);
```

JDK 1.4の証明書検証機能(その5)

パス構築を実行

```
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();
}
```

JDK 1.4の証明書検証機能(その6)

パス検証を実行

```
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();
}
```

SUNの実装についてまとめ(その1)

- 良いところ
  - RFC3280に比較的準拠しているように一見、見える
  - とりあえずそこそこ動く
    - フレームワークがわりと良い
  - なんと言ってもリファレンス実装!
  - ツリーモデル/相互認証モデル/ブリッジモデルのパス検証は一応出来る
    - ポリシー関連は意外とまとも
      - GPKI程度ポリシーの処理はOK
    - それ以外は不明?

SUNの実装についてまとめ(その2)

- 悪いところ
  - 半端な実装
    - そのうち良いのが出てくるんでしょう????
  - RFC3280からの不足分がある
  - 検証プロセスに対して介入できるポイントが不足している
  - 検証プロセスと失効確認プロセスの考察が不足
    - 例えばOCSPのチェックを行うためにはCertPathChecker経由で行うしかない。本来は同様な機能を持つRevocationCheckerが必要ではないか?
  - CRLの検証が不十分
    - IDPの処理をしていないため、GPKIは出来ない

今後の方向性

今後

- 長期署名の話
- 証明書の実効性をあげる
- 楕円関数
- マルチドメイン
- 証明書検証
  - DPD/DPV

長期署名

長期署名

- PKIの電子署名の検証では…
  - トップCAからEEまでの有効期間のANDの期間のみ検証可能
- 実社会の契約では…
  - 契約時から3ヶ月以内に発行された印鑑証明があれば有効
- 定期的に署名のしなおしをするなどの対処が必要

証明書の実効性/有効性

証明書の実効性/有効性

- 昨年の4月にいわゆる「電子署名法」が施行
  - 特定認証局が発行した電子証明書に実印と同様の権限を与えた
- 商業登記法の改正
  - 商業登記局が会社代表者に対して証明書を発行
    - 会社代表者に対しての印鑑証明に相当する
- 欧米では、バイオメトリックス情報を証明書に入れる動きもある
  - 身分証明書の代わりに使える証明書
  - 署名のイメージを入れる動きもある

楕円関数

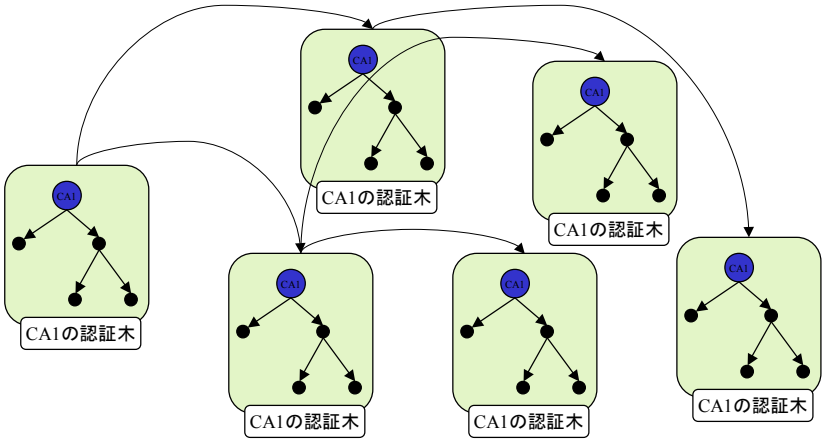
楕円関数

- 楕円曲線の一部を利用して公開鍵として使う
- 証明書の電子署名のアルゴリズムとして
  RSA/DSAのかわりに楕円関数を使う動きもある
  - 検証がRSAに比べ計算量が少なく、PDA/携帯電話などCPUパワーに限りがある場合に有利
    - とはいえ、このごろのPDA/携帯電話のCPUパワーは馬鹿にできない
  - RSAの1024bit相当の強度を300bit程度で実現
    - 証明書を小さくできる
- OpenSSLにSunが実装を提供

マルチドメイン/BCA

マルチドメイン

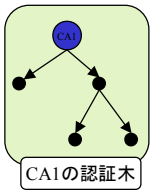
- 複数のPKIドメインが互いに連携して存在
- PKIの認証パス形成が複雑になる



証明書検証

証明書の検証

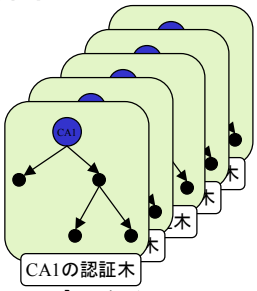
- 木構造の検証でさえ…
  - 証明書の検証プロセス
    - 署名の連鎖の電子署名のチェック
    - 有効期限のチェック
    - CRLのチェック
    - ポリシーのチェック
    - …



- 複雑で煩雑な処理が必要

証明書の検証

- マルチドメイン環境では…
  - パスが長くなるため検証に時間がかかる
  - 木構造での検証に加え
  - ポリシーマップ
  - 相互認証証明書の検証
  - 複数のパスの可能性



- 果たしてすべてのEEが検証できるか?

証明書の検証

- 証明書検証を委任
  - EEでは荷が重い
  - サーバに証明書検証を依頼
  - サーバには十分な資源とネットワークコネクティビティを提供

- RFC 3379

  Delegated Path Validation and Delegated Path Discovery Protocol Requirements
  - DPD(Delegated Path Discovery)
    - パス構築を依頼
  - DPV(Delegated Path Verification)
    - パス検証を依頼

証明書の検証

- CVS(Certificate Validation Protocol)
  - http://www.ietf.org/internet-drafts/draft-ietf-pkix-cvp-01.txt
- SCVP
  - http://www.ietf.org/internet-drafts/draft-ietf-pkix-scvp-10.txt
- DPD/DPV using OCSP with extensions
- DVCS(Data Validation and Certification Server Protocols)
  - RFC 3029

- GPKI 証明書検証サーバ
  - OCSP v1の独自拡張