3.　総合セキュリティ機能の実装

3.1.　概要

昨今のＩＴ関連インフラの普及に伴い、インターネットを利用した企業間電子商取引（Ｂ２Ｂ）および企業と消費者間の電子商取引（Ｂ２Ｃ）の市場が加速度的に拡大している。

インターネットは周知の通りオープンなネットワークであり、このオープンさが爆発的に世界中に普及した主な理由であることは明らかであるが、オープンであるがゆえに大きな弱点も存在する。それは、データの伝播経路上において、第３者がその内容を比較的容易に閲覧、改変が可能であること、あるいは、データを送信した側の本人証明が極めて困難であるということである。こうした事実は、電子商取引を行う際に致命的な問題を招くことは明らかである。

したがって、こうした問題を排除するためのセキュリティ技術の実装が電子商取引には必須の条件であるということができるが、Ｗ３Ｃで標準化が進められている SOAP 仕様では、セキュリティに関する規定が存在しない。

そこで、今回の OpenSOAP 開発において新たにセキュリティ技術に関する実装を行い、OpenSOAP をＢ２Ｂ、Ｂ２Ｃに安全に適用可能とする様な環境を整備した。

3.2.　インターネット上の電子商取引での取引リスク

インターネット上での電子商取引において、セキュリティ技術の実装を行わない場合に発生し得る問題点（取引リスク）について、以下に具体的に説明する。

（１）盗聴

取引内容を、望まない第３者に知られること。前述の様に、経路上のコンピュータ上では、第３者によって参照される可能性がある。

（２）改ざん

取引内容を、伝達途中で悪意の第３者が改変すること。取引内容を示すデータを、通信経路上で書き換えることで発生する。

（３）なりすまし

他人の名前を騙（かた）って、データを送信すること。例えば、出した覚えの無い注文 がいつの間にか出されていたいう問題がこれに相当する。データ送信時、そのデータ自体に確かに本人であるという客観的証明が無い限り、こうした問題は解決不可能である。

（4）否認

データを送信しておきながら、後で送信を否定すること。例えば、注文書を送付しておきながら、注文書を送った覚えが無いので代金は支払えないと主張する、などの行為である。これもなりすましと同様、送信されたデータ自体に本人証明が存在しなければ排除することはできない。

以上挙げた問題点はいずれも、商取引上致命的な問題となる項目ばかりであり、これらのリスク全てを排除することがセキュリティ技術の実装の必要条件となる。

3.3. 基本方式の選定

OpenSOAP におけるセキュリティ技術の実装にあたっては、解読の困難性や他環境との親和性を考慮して、現在最もポピュラーな方式となっているＰＫＩ(Public Key Infrastructure)を採用する。具体的には、公開鍵暗号方式であるＲＳＡ暗号技術を利用し、セキュリティ技術の実装を行う。

公開鍵暗号方式は、自由に配布して良い公開鍵と、個別に管理する秘密鍵の組み合わせにより暗号化、復号化を実施する方式である。この方式は、片方の鍵で暗号化したものはもう一方の鍵以外では復号化できない特性をもっている。この特性は、セキュリティ技術の実装にあたっては非常に重要かつ欠かせないものであるため、今回の開発に利用する。

図 3.3.1 に、ＰＫＩによる暗号化と復号化の方式を示す。ここで例えば、ある通常読解可能な平文に対し、秘密鍵によって暗号化し読解不可能な暗号化文を作成したとする。この場合、この暗号化文を平文に戻すことが可能なのは暗号化に利用した秘密鍵に対応する公開鍵のみとなる。逆に、暗号化の際に公開鍵を利用した場合は、復号化可能なのは対応する秘密鍵のみとなる。

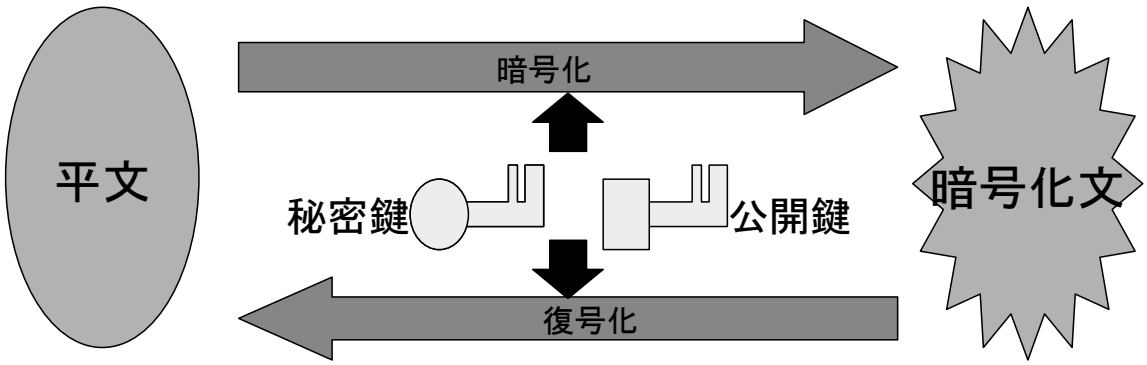

図 3.3.1 ＰＫＩによる暗号化／復号化の基本方式

3.4.　取引リスクへの排除

ここでは、今回採用することとしたＲＳＡ暗号技術を利用することによって、取引リスクをどの様に排除することが可能となるかについて説明する。

図 3.4.1 に、取引リスク排除の基本方式を示す。

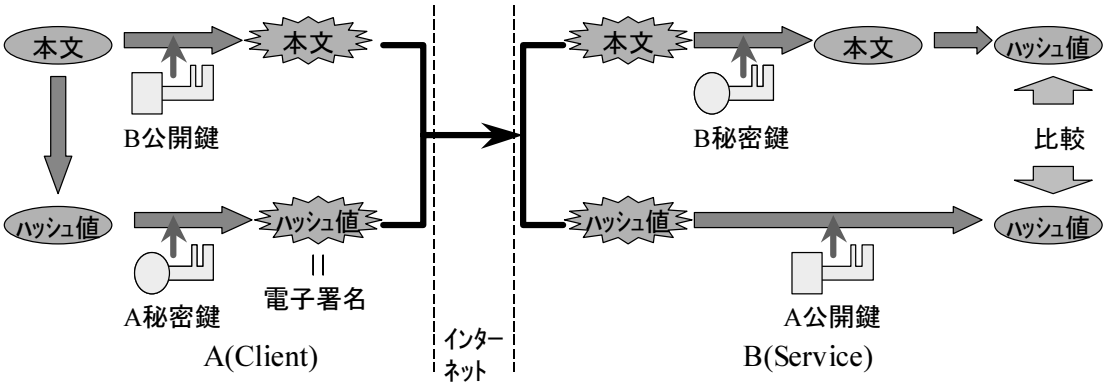

図 3.4.1 取引リスク排除の基本方式

この例は、A(Client)から、B(Service)にインターネットを介してデータを転送する際の方式を示している。取引リスクの排除は、Ａ、Ｂそれぞれの秘密鍵、公開鍵ペアを利用することによって実現する。

以下に、考えられる取引リスク毎にその方式を説明する。

（１）盗聴

盗聴の防止は、発信したデータ内容を送り先だけが解読可能な様に暗号化することで実 現することができる。具体的には、発信側で受信側の公開鍵を利用して必要な部分を暗号化して送信するれば良い。受信側の公開鍵を利用して暗号化すると、そのデータは受信側の秘密鍵以外では解読不可能となるため、(受信側の秘密鍵を持たない）第３者は解読が不可能となり、盗聴を防止することができる。

（２）改ざん

改ざんの防止は、改ざんの検出を可能にすることによって実現する。以下の手順で改ざんを検出する。

【送信側】

A) データ部分をハッシュ化（注１）する。

B) ハッシュ値を送信側の秘密鍵で暗号化し、その結果（電子署名と呼

ぶ）をデータに添付して送信する。

【受信側】

A) 受信したデータ部分をハッシュ化する。
B) データに添付されてきた電子署名を送信側の公開鍵で復号化する。
C) A)と B)のデータを比較し、等しければ改ざんされていないものと判断する。

もし、データ部分に改ざんが行われていれば、受信側において A)の結果が変化するはずであり、その結果 A)と B)が異なることとなって改ざんの検出がされるはずである。
また、改ざん結果を元に第３者が改めて電子署名を行ったとしても、その署名は（送信側の秘密鍵で暗号化されない限り）送信側の公開鍵で複号化することは不可能なので、改ざんの検出を抑止することはできない。
以上により、第３者による改ざんが行われた場合は、必ず検出が可能となる。

注１）ハッシュ化
任意の長さを持つデータを、別のデータ（ハッシュ値と呼ばれ、一般には元の長さにかかわらず、１００～２００ビット程度のある一定の長さとなる）に変換すること。現在、数種類のアルゴリズムが提案されている。これらのアルゴリズムは、ハッシュ値が異なれば、元データも異なる様に設計されている。逆に、元データが異なっていてもハッシュ値が同一となることは理論的にはあり得るが、ハッシュ値のデータ長をある程度長くすることで、こうしたケースを確率的に極めて低くすることが可能である。
以上により、元データの比較結果は、ハッシュ値の比較結果と同一であるとみなすことができる。
なお、今回の実装で利用したアルゴリズムは以下の通りである。

SHA1　　：ハッシュ値長は１６０ビット
MD5　　：ハッシュ値長は１２８ビット
RIPEMD　：ハッシュ値長は１６０ビット

（3）　　なりすまし

なりすましは、改ざんと全く同様の手順によって検出する。
この検出は、電子署名の復号化の時点で検出することができる。
例えば、送信者Ｘが、Ａの名を騙ってデータを送信したとする。この場合、受信側ではＡの公開鍵を利用して電子署名を複号化しようとするが、この署名は（Ａの秘密鍵で暗号化された場合以外は復号化することができないため）復号化に失敗する。したがって、電子署名の復号化に失敗した時点で、このデータの送信者がＡではないということを検出することが可能となる。

（４）否認
否認の防止は、上記なりすましの防止手段（電子署名の利用）によって、同時に実現することが可能である。
すなわち、例えば送信者Ａが電子署名を付加してデータを送信したとする。受信側でこの電子署名をＡの公開鍵で復号化した時点で、この署名が確かにＡの秘密鍵で署名されたことが証明される。したがって、Ａは、自身の秘密鍵が盗まれていない限りは送信の事実を否認することはできない。（秘密鍵を盗まれているので、盗んだ誰かの行為である、と主張することも可能であるが、少なくともＡに対して秘密鍵の管理責任を問うことは可能である）

以上により、公開鍵暗号方式であるＲＳＡ暗号技術を利用し、暗号化／復号化および電子署名の付加／検証を行うことで、取引リスクを完全に排除することが可能となることがわかる。

3.5.　セキュリティ技術の実装方式

インターネットでのセキュリティ技術の実装方式として、ＳＳＬなどトランスポート層の暗号化による方式が一般的である。しかし、OpenSOAP ではサーバ間のメッセージ転送等の機能提供のために、メッセージ自体の指定部分を暗号化、認証する機能が必要となる。したがって、今回の実装では、前項で説明した方式に基づいて、ＳＯＡＰメッセージに対する暗号化、認証の機能をＡＰＩとして提供する形態を取るものとする。

前章で検討した取引リスクを回避するため、以下に示す種類のＡＰＩを準備した。

3.5.1.　鍵ペアの作成

【入力】

（なし）

【出力】

・ 秘密鍵と公開鍵のペア

セキュリティ機能の中で利用する鍵のペアを作成する。

図 3.5.1 と図 3.5.2 に秘密鍵と公開鍵の例を示す。

```
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDCthTeENTEyTvPcB0OeozNqGm+lrtC/dAajFR7Wd+94umH0pc8
TKd5+MGWymrgLZa1xBZoB9rp7x5dAwyJAHX41D8sS75y6pFl+6TAEXJ9fEUo16h1
aLTNgaz67b/Rhvl1BA+9A2kXNzMvSExSjPYWgRqKYtW5w3y1dNntfc6hwQIBAwKB
gQCBzriUCziDMNKKSr4JpwiJGvEpudIsqTVnCDhSO+p+l0ZajGTS3cT7+yu53EdA
HmR5LWRFWpHxShQ+AghbVaP6DgQ3FwMXhW28jBuHG3MmtlY/XkArdLpG1nq0WRiS
h49aPwZM8khKt/o1zwSG/f6vY/LLFdElefFxDwT752rVAwJBAOwX9GXOBUPlkJ7P
ACMTCWZ4FGny83m+pvttLsamzI7h0gF9Cmn3nWaZRh+iAUPswsiKHz3KrEWK1PPO
ZwFK4Z8CQQDTIOVDa8pehzqKrHVFMroES7VgVUC/3rxEh72hdCkswJukCT8uBQm8
ophyI4bMC0yipDqEUVVHB7bqhAyhY4CfAkEAnWVNmTQDgpkLFIoAF2IGRFANm/dM
+9RvUkjJ2cSIX0E2q6ixm/po7xDZamwA1/MshbFqKTHILlyN997vVjHrvwJBAIzA
mNedMZRaJwcdo4N3Jq2HzkA41dU/KC2v08D4G3MrEm1bf3QDW9MXEEwXrzKyMxcY
JwLg44Svz0cCsxZCVb8CQQCorwq57vMyikpWB9ijl599G0YfRyXbI+qh7cliOFoV
JzLkSdDaqc+of0qZLp7tKgHhLR2WjZO/K/p8OYCGHFzW
-----END RSA PRIVATE KEY-----
```

図 3.5.1 秘密鍵の例

```
-----BEGIN RSA PUBLIC KEY-----
MIGHAoGBAMK2FN4Q1MTJO89wHQ56jM2oab6Wu0L90BqMVHtZ373i6YfSlzxMp3n4
wZbKauAtlrXEFmgH2unvHl0DDIkAdfjUPyxLvnLqkWX7pMARcn18RSjXqHVotM2B
rPrtv9GG+XUED70DaRc3My9ITFKM9haBGopi1bnDfLV02e19zqHBAgED
-----END RSA PUBLIC KEY-----
```

図 3.5.2 公開鍵の例

ここで作成した鍵を、以降示す処理で利用する。

3.5.2.　暗号化処理

【入力】

・ ＳＯＡＰメッセージ

・ メッセージ受信者の公開鍵

【出力】

・ ＳＯＡＰメッセージ

入力された SOAP メッセージに対し、(あらかじめ暗号化を指示する様にマーキ

ングされている）必要部分を公開鍵で暗号化する。

ＳＯＡＰメッセージ中の暗号化を指示する部位を図 3.5.3 に示す。

```
<A xmlns:s="http://security.opensoap.jp/1.0/" s:encrypt="True">
5.000000
</A>
```

図 3.5.3 暗号化指示する部位の定義

s:encrypt="True" によって、この内容（5.000000）を暗号化することが指示される。暗号化しない場合は、ここに“False”を与える。暗号化によって得られた（変更された）ＳＯＡＰメッセージの該当部を図 3.5.4 に示す。

```
<A xmlns:s="http://security.opensoap.jp/1.0/" s:encrypt="True">
eU1Ol+kaoguufSmmoS3Blg0QcRgwvajt/fmHKYI49CLSIUupVXSe+CYnTJn65F13C84EWABT
YwjORuuAuNd4ogtKcdoZu/cBkfh8x8G0s6nYBmsV4QasOcLfPTlh282L/J95+t18mON+PS7b
8LQ77AXPUZpyiBfwpLCcYAfIddc=
</A>
```

図 3.5.4 暗号化結果

この様に内容は暗号化され、第３者には解釈不能な形とすることができる。この結果からもわかる通り、暗号化処理によってデータ量は増大するため、必要部位を最小限暗号化できる様な仕様としている。

3.5.3.　復号化処理

【入力】

・ ＳＯＡＰメッセージ
・ 自身の秘密鍵

【出力】

・ ＳＯＡＰメッセージ

入力されたＳＯＡＰメッセージに対し、暗号化された部分を得られた公開鍵で復号化する。

図 3.5.4 で示される暗号化された部位が、図 3.5.3 で示される平文に変換される。

3.5.4.　電子署名付加処理

【入力】

・ ＳＯＡＰメッセージ
・ 自身の秘密鍵

【出力】

・ ＳＯＡＰメッセージ

入力されたＳＯＡＰメッセージについて、Ｂｏｄｙ部（ヘッダ部を除く実データ部）をハッシュ化し、自身の秘密鍵で暗号化して得られたデータ（電子署名）をＳＯＡＰメッセージのヘッダ部に付加する。

図 3.5.5 に、付加される署名の例を示す。

```
-  <SOAP-SEC:Signature xmlns:SOAP-SEC=
-  "http://schemas.xmlsoap.org/soap/security/2000-12"   actor="some-URI"
   mustUnderstand="1">
- <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
- <SignedInfo>
  <CanonicalizationMethod Algorithm=
"http://www.w3.org/TR/2000/CR-xml-c14n-20001026" />
  <SignatureMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
- <Reference URI="#Body">
- <Transforms>
  <Transform   Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"
/>
  </Transforms>
  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
  <DigestValue>jo85ZsXLkw/344p3Tbuo0lrIMWw=</DigestValue>
  </Reference>
  </SignedInfo>
  <SignatureValue>
HRtZWoGPrm3W5EjXOScLbWy7yiQ0pIKxtyCZTcqONhdw277+YkLGcBSjrBfCumjI
RLWv8kBGYBJjB8qoota+s7afRY/iN25wS2u8OeHCpk1j1JJVzTy/zYsYPFMyQbcBlqk
AYJIRyGaKhCB5rFnHw3rEqBtFWOd1aa6Uiz2eVd8=
  </SignatureValue>
  </ds:Signature>
  </SOAP-SEC:Signature>
```

図 3.5.5 ＳＯＡＰメッセージに付加される電子署名

実際にの電子署名は、<SignatureValue>にて示される項目内に設定される。

3.5.5. 電子署名検証処理

【入力】

・ ＳＯＡＰメッセージ

・ メッセージ送信者の公開鍵

【出力】

・ ＳＯＡＰメッセージ

入力されたＳＯＡＰメッセージについて、電子署名をメッセージ送信者の公開鍵で復号化する。これを、Ｂｏｄｙ部をハッシュ化した結果と比較する。一致していれば検証に成功したこととなる。

3.5.6.　電子署名付加／検証に関する留意事項

電子署名の付加、検証関連処理は、複数の署名を取り扱える様にする。これは、送信者だけでなく OpenSOAP サーバも電子署名を付加する可能性があるからである。

OpenSOAP サーバが電子署名を付加することで、サーバから返送されて来たＳＯＡＰメッセージについても、その正当性を検証することが可能となる。

アプリケーション側では、以上示したＡＰＩを利用してセキュリティ機能の実現を図るが、電子署名の付加／検証は、完全に同一なデータでなければ成功しないため、以下に示す点に注意が必要である。

- 処理の順序に注意が必要である。すなわち、メッセージ送信側では、暗号化処理→電子署名付加処理の順で実行し、メッセージ受信側では、電子署名検証処理→復号化処理の順で実行する必要がある。
- 署名付加／検証処理時点の文字列の文字コード種別を同一にする必要がある。

3.6.　認証局の提供

3.6.1.　認証局の必要性と概要

今まで述べてきた様に、秘密鍵と公開鍵のペアを準備することで、ＳＯＡＰメッセージに対するセキュリティ確保が可能となった。しかし、これらの鍵の配布、管理に不備があれば、セキュリティ確保の前提が崩れることとなってしまう。

例えば公開鍵の配布に関する最も確実な方法として、利用者が提供者から直接（ＦＤなどのメディアに格納する等して）公開鍵を受領する等の方法は最も確実ではあるが、利用者の増大時に対応困難であることや、配布の即時性欠如といった問題は避けられない。そこで、これらの媒介を行うサイトとして認証局を置くことが有用となる。

個々のクライアントやサービス提供者は、公開鍵を直接やり取りするのではなく、この認証局を介して登録、取得を行う。このことによって、公開鍵の多数配布の問題や即時性の問題を回避することが可能となる。

公開鍵の配布は、電子証明書と呼ばれる形態にて配布を行う。これには、公開鍵の他、その公開鍵の有効期限等の情報を含んだものである。

図 3.6.1 に、認証局の概要を示す。

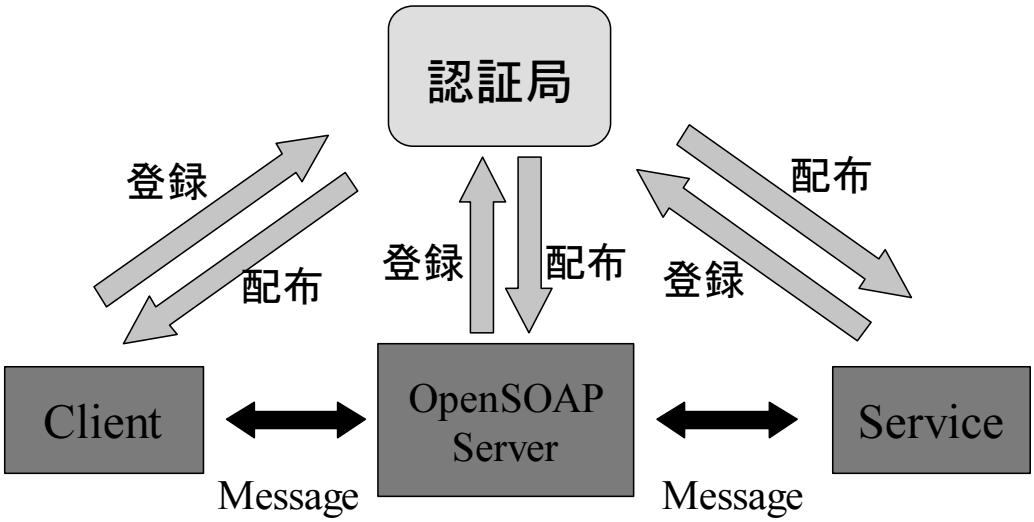

図 3.6.1 認証局の概要

3.6.2. 認証局の機能

認証局は、主に以下に示す機能を有する。

（１）セキュリティ機能で利用する公開鍵の登録
（２）上記公開鍵（電子証明書）の配布
（３）公開鍵の失効処理（秘密鍵の漏洩に伴う）
（４）これらの機能実現のためのＤＢ管理／運用

電子証明書には以下の情報が含まれる。

（１）発行者（認証局）名と（次以降の項目に対する）署名
（２）証明書のシリアル No。と有効期限
（３）所有者情報（名称）
（４）所有者の公開鍵

また、認証局のＤＢに含まれる項目は以下の通りである。

（１）所有者情報（名称）
（２）シリアル No.（DB 内でユニークな値）
（３）有効期限
（４）失効かどうかを示す値

（５）公開鍵

認証局のＤＢでは所有者情報(名称)は複数存在することが可能となっている。その理由は、以下の通りである。

A) 公開鍵失効させ、新たな公開鍵を登録するため（失効した公開鍵も参照用に残す
B) 期限の異なる公開鍵を同時に登録可能とするため

3.6.3.　認証局関連ＡＰＩ（認証局ＤＢアクセス用）

認証局関連機能の実装は、具体的にはＡＰＩの提供という形態で行っている。このＡＰＩは、認証局のＤＢアクセス用ＡＰＩと公開鍵アクセスＡＰＩの２種類に大別される。

ここでは、まず認証局ＤＢアクセス関連のＡＰＩについて概要を示す。

（１）公開鍵の登録

【入力】

- 鍵の所有者名称
- 有効期限
- 公開鍵

【出力】

- シリアル番号

指定公開鍵を認証局ＤＢに登録する。

（２）公開鍵の失効設定

【入力】

- 鍵の所有者名称
- シリアル番号

【出力】

（なし）

認証局ＤＢに対し、指定シリアル番号の公開鍵を失効登録する。

（３）指定所有者のレコードの検索

【入力】
- 鍵の所有者名称

【出力】
- ＤＢレコード

指定された所有者のＤＢレコードを検索取得する。

（４）レコードの削除

【入力】
- シリアル番号

【出力】
（なし）

指定されたシリアル番号のＤＢレコードを削除する。

（５）認証局ＤＢのレコードから電子証明書を作成

【入力】
- 発行者名
- 発行者の秘密鍵
- ＤＢレコード

【出力】
- 電子証明書

認証局ＤＢのれこーどから電子証明書を作成する。作成する際は、発行者（＝認証局）の署名を証明書内に付加する。

3.6.4. 認証局関連ＡＰＩ（電子証明書アクセス用）

ここでは、電子証明書から情報を取得するＡＰＩについて概要を示す。

（１）発行者の署名検証

【入力】
- 電子証明書
- 発行者の公開鍵

【出力】
（なし）

発行者の公開鍵を利用して、電子証明書の署名を検証する。

（２）各データ項目の取得

【入力】

・ 電子証明書

【出力】

・ 各データ項目（発行者名、シリアル番号、所有者名称、有効期限、公開鍵）

電子証明書内の各データ項目を取得する。
本ＡＰＩは、実際にはデータ項目ごとにメソッドを準備している。

3.6.5.　認証局サービス機能構築支援

認証局によるオンラインサービスは、前記ＡＰＩを利用し、OpenSOAP のサービスとして実装することが可能である。今回の開発成果の中にも認証局サービスと、それにアクセスするクライアントの例をサンプルとして提供した。

認証局サービスでは、基本的には上記認証局アクセス用ＡＰＩの機能を実装することになるが、公開鍵の登録処理はサービスに含めるべきではない。その理由は、登録機能をサービスとして外部にオープンにすることで、無制限、無差別な登録処理により認証局ＤＢが攻撃され、信頼性が低下する可能性があるからである。登録機能自体は、人為的手段が介在する様オフライン処理にて実施する様にすべきである。(登録処理を認証局でローカルに実施するサンプルも提供している)

公開鍵の失効処理についても、悪意の第３者が勝手に失効処理を行ってしまうという懸念がある。しかし、失効処理を要求する側で、失効する公開鍵に対応した秘密鍵にて署名し、認証局サービス側ではその署名を検証する形とすることでそのリスクを回避することができる。(すなわち、対応する秘密鍵を持つ者だけが公開鍵を失効させることが可能になる)

3.6.6.　認証局利用側で検討すべき事項

常に最新の状態を反映するために、公開鍵利用に際しては、常に認証局サービスに公開鍵を問い合わせるのが正確だが､応答性の低下を招いてしまう｡そこで、ある程度時間が経過した時点だけ認証局サービスに問い合わせる様にし、その他はローカルにキャッシュした公開鍵を利用する等の対応が必要と考えられる。

3.6.7. 認証局の課題

（１）認証局ＤＢ

現在のＡＰＩでは、独自の形式のバイナリファイルにてデータベースを管理している。しかし、レコード数、トランザクション数の増大に対応するため、将来的にはＤＢのアプリケーションを利用することも検討する必要があると思われる。

（２）証明書書式

現在の電子証明書は独自の書式で実装しているが、標準的な証明書書式（X.509）との互換性について検討が必要である。実際には、証明書作成ＡＰＩ自体を上記標準書式へ対応する様にすることや、証明書書式変換ＡＰＩの準備等が考えられる。