HITACHI
Inspire the Next

Cosminexus V11 アプリケーションサーバ Cosminexus XML Security - Core ユーザーズガイド

解説・手引書

3021-3-J26-30

前書き

■ 対象製品

マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」の前書きの対象製品の説明を参照してください。

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

HITACHI，Cosminexus，uCosminexus は，株式会社 日立製作所の商標または登録商標です。

AIX は，世界の多くの国で登録された International Business Machines Corporation の商標です。

AMD は，Advanced Micro Devices, Inc.の商標です。

Itanium は，Intel Corporation またはその子会社の商標です。

Linux は，Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。

Microsoft は，マイクロソフト 企業グループの商標です。

Oracle および Java は，オラクルおよびその関連会社の登録商標です。

Red Hat is a registered trademark of Red Hat, Inc. in the United States and other countries.

Red Hat は，米国およびその他の国における Red Hat, Inc.の登録商標です。

Red Hat Enterprise Linux is a registered trademark of Red Hat, Inc. in the United States and other countries.

Red Hat Enterprise Linux は，米国およびその他の国における Red Hat, Inc.の登録商標です。

UNIX は，The Open Group の登録商標です。

Windows は，マイクロソフト 企業グループの商標です。

Windows Server は，マイクロソフト 企業グループの商標です。

すべての SPARC 商標は，米国 SPARC International, Inc. のライセンスを受けて使用している同社の米国およびその他の国における商標または登録商標です。SPARC 商標がついた製品は，米国 Sun Microsystems, Inc. が開発したアーキテクチャに基づくものです。

その他記載の会社名，製品名などは，それぞれの会社の商標もしくは登録商標です。

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

■ 発行

2022 年 3 月　3021-3-J26-30

■ 著作権

All Rights Reserved. Copyright (C) 2020, 2022, Hitachi, Ltd.

変更内容

変更内容(3021-3-J26-30) uCosminexus Application Server 11-20，uCosminexus Client 11-20，uCosminexus Developer 11-20，uCosminexus Service Architect 11-20，uCosminexus Service Platform 11-20

追加・変更内容	変更個所
記載内容は変更なし（リンク情報だけを変更した）。	−

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルをお読みになる際の前提情報については，マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」のはじめにの説明を参照してください。

目次

前書き　2
変更内容　4
はじめに　5

1　XML Security - Core の概要　8
1.1　マニュアルの説明　9
1.2　XML Security - Core とは　10
1.3　XML Security - Core の仕組み　11
1.4　XML Security - Core の利用例　12

2　XML Security - Core の機能　14
2.1　XML 署名とは　15
2.1.1　XML 署名の種類　15
2.1.2　XML 署名の構文　17
2.2　XML 暗号とは　19
2.2.1　暗号化できるデータの種類　19
2.2.2　鍵合意　20
2.2.3　XML 暗号の構文　20
2.2.4　データの直列化および復元処理　22
2.2.5　DataReference および KeyReference　23
2.3　XML 署名および XML 暗号に共通の機能　24
2.3.1　XML 構文の生成および解析処理　24
2.3.2　サポートするアルゴリズム　24
2.3.3　コンテキストの設定　25
2.3.4　鍵情報の設定および取得　26
2.3.5　XML 文書の出力処理　27
2.3.6　デバッグダンプ機能　28

3　環境設定　32
3.1　クラスパスを設定する　33
3.2　セキュリティプロバイダを設定する　34
3.3　鍵と証明書を設定する　36
3.3.1　鍵および自己署名証明書を生成する　36
3.3.2　認証局が発行する証明書を取得する　36
3.4　環境設定するときの注意事項　37

4　アプリケーションの開発　38

4.1　XML 署名アプリケーションを開発する　39
4.1.1　Enveloped 署名を生成する　39
4.1.2　Enveloping 署名を生成する　41
4.1.3　Detached 署名を生成する　43
4.1.4　Enveloped 署名，Enveloping 署名，または Detached 署名を検証する　45
4.2　XML 暗号アプリケーションを開発する　48
4.2.1　XML データの要素またはコンテンツを暗号化する　48
4.2.2　XML データの要素またはコンテンツを復号化する　50
4.2.3　バイナリデータを暗号化する　52
4.2.4　バイナリデータを復号化する　54
4.2.5　鍵データを暗号化する　56
4.2.6　鍵データを復号化する　58
4.2.7　鍵合意で作成した鍵を利用してデータを暗号化する　59
4.2.8　鍵合意で作成した鍵を利用してデータを復号化する　62
4.3　URI 参照解決および ID 解決のカスタマイズ　66
4.3.1　URI 参照解決　66
4.3.2　ID 解決　67
4.4　アプリケーションを開発するときの注意事項　68
4.4.1　XML Security - Core の仕様に関する注意事項　68
4.4.2　セキュリティに関する注意事項　69

付録　73

付録 A　トラブルシュート　74
付録 A.1　トレースの設定項目　74
付録 A.2　トレースの設定方法　76
付録 B　標準仕様への対応　78
付録 C　用語解説　81

索引　82

1 XML Security - Core の概要

XML Security - Core は，XML 署名データを生成または検証したり，データを暗号化または復号化したりするための製品です。この章では，XML Security - Core の特長や処理の仕組み，および利用例について説明します。

1.1 マニュアルの説明

このマニュアルで使用している表記について説明します。

このマニュアルで使用している記号

このマニュアルで使用している記号を次のように定義します。

記号	意味
斜体	斜体で記述している部分は，使用している環境に合わせて読み替えたり，変更したりする必要があることを示します。

フォルダとパスの表記

このマニュアルでは，Windows，AIX，および Linux で共通の内容の場合，Windows の「フォルダ」を「ディレクトリ」と表記しています。また，「¥」を「/」と表記しています。

Windows の場合，「ディレクトリ」を「フォルダ」に，「/」を「¥」に置き換えてお読みください。

API のリファレンスについて

このマニュアルには，XML Security - Core が提供する API のリファレンスは記載されていません。API の仕様については，次のフォルダに格納されている Javadoc 形式の API リファレンスを参照してください。

API リファレンス格納場所（Windows の場合だけ）

Application Server のインストール先ディレクトリ/XMLSEC/docs/xsecapi

1.2 XML Security - Core とは

XML Security - Core※は，XML 署名データを生成または検証したり，データを暗号化または復号化したりするための製品です。この節では，XML Security - Core が必要とされる背景を説明します。

インターネットが普及し，企業間での電子商取引が日常的に行われるようになった現在，データの完全性を保証する XML 署名やデータの秘匿性を保証する XML 暗号の需要が高まっています。

XML 署名は，電子署名の一種です。XML 署名をデータに付与しておくことで，XML データに対する改ざんを防止し，データの完全性を保証できます。また，XML 署名を使用すると，身分を偽ってデータをやり取りする成り済ましも防止できます。さらに，カード番号など第三者に知られたくないデータをやり取りする場合は，データの秘匿性を保証する必要があります。XML Security - Core は，XML 暗号によってデータを暗号化し，データの秘匿性を保証できます。

注※

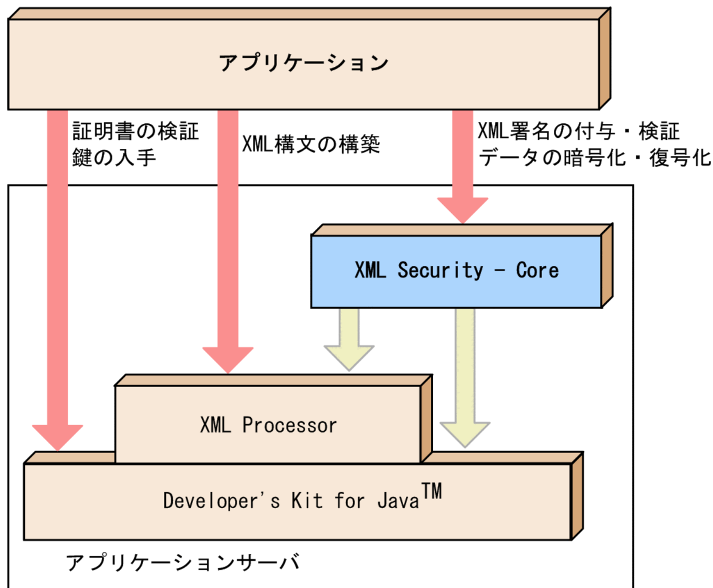
アプリケーションサーバが提供する XML 署名・暗号処理機能のことです。このマニュアルでは，XML 署名・暗号処理機能のことを XML Security - Core と表記しています。

1.3　XML Security - Core の仕組み

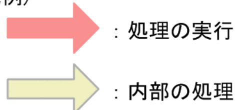
XML Security - Core は，Developer's Kit for Java と XML Processor（JAXP をサポートした XML プロセッサ）を前提として動作します。XML Security - Core では，Developer's Kit for Java が提供する Java 標準クラスライブラリを使用します。

Developer's Kit for Java が提供する Java 標準クラスライブラリを利用する場合の XML Security - Core の仕組みを次の図に示します。

図 1-1　XML Security - Core の仕組み



（凡例）



XML Processor は，署名構文または暗号構文の構築に必要となります。Developer's Kit for Java は，鍵管理や証明書を検証する場合に必要になります。Developer's Kit for Java は，XML 署名を付与したり，検証したりする場合の署名エンジンの役割も果たします。また，データを暗号化または復号化する場合は，Developer's Kit for Java は暗号エンジンとなります。

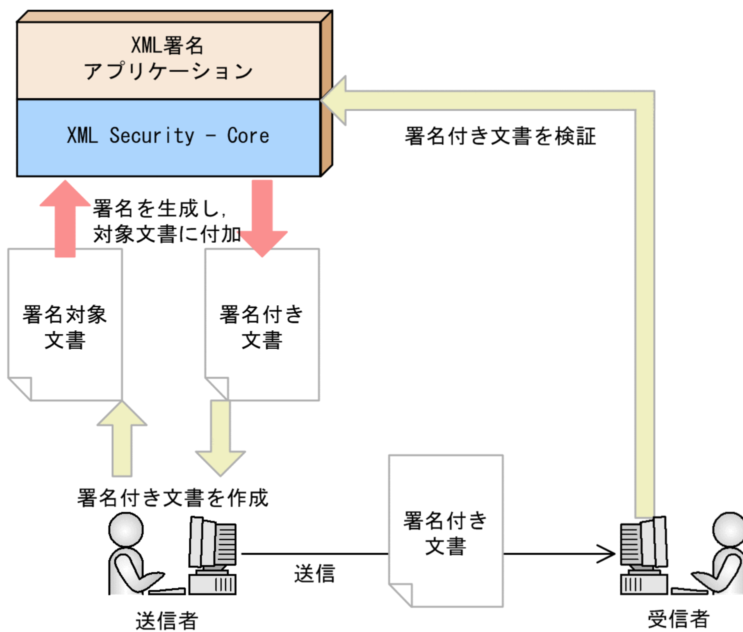
1.4　XML Security - Core の利用例

この節では，XML Security - Core を使用して，どのようなことができるのかについて，利用例を示して説明します。

XML Security - Core は，XML 署名データを生成または検証したり，データを暗号化または復号化したりするアプリケーションを開発するために必要な API を提供します。さらに，開発したアプリケーションを実行する際の実行エンジンの役割も果たします。

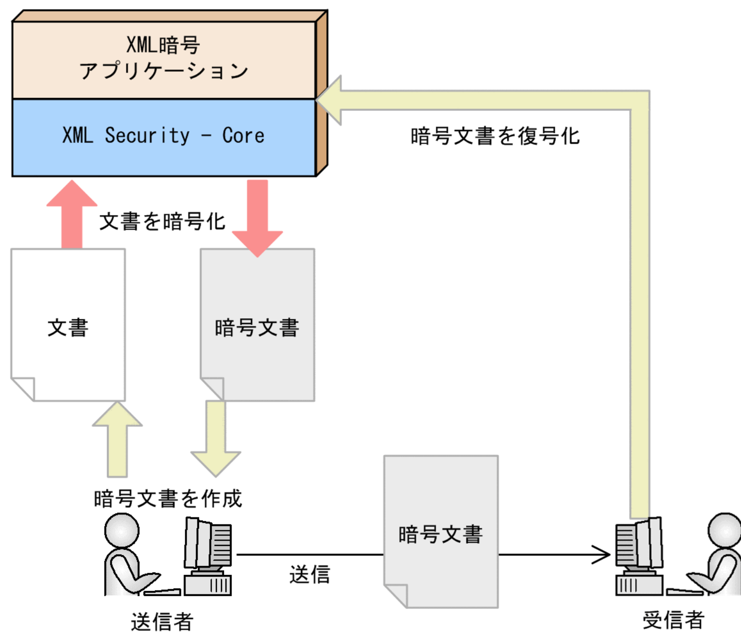
XML Security - Core で開発した XML 署名アプリケーションを利用する場合の例を次の図に示します。

図 1-2　XML Security - Core の利用例（XML 署名アプリケーションを利用する場合）



XML Security - Core で開発した XML 暗号アプリケーションを利用する場合の例を次の図に示します。

図 1-3　XML Security - Core の利用例（XML 暗号アプリケーションを利用する場合）



2 XML Security - Core の機能

この章では，XML Security - Core の XML 署名および XML 暗号に関連する機能の概要を説明します。XML 署名および XML 暗号の機能について説明したあとに，XML 署名と XML 暗号の両方に共通の機能について説明します。

2.1 XML 署名とは

この節では，XML 署名とは何かを説明し，XML 署名の種類および構文を説明します。

XML 署名とは，XML 文書をはじめとするさまざまな電子データに対して付けられる電子署名の一種です。ネットワークを介して電子データをやり取りすることが多くなった現在，電子データを安全にやり取りするために，セキュリティ問題に対する関心が高まっています。電子データに XML 署名を付ければ，その電子データの完全性が保証できます。また，XML 署名は，成り済まし防止および否認防止にも有効です。

この節では，XML 署名によってどのように電子データの完全性を保証するのか，また，どのように成り済ましや否認を防止するのかについて説明します。

電子データの完全性を保証する

Web サイトやインターネットなど，ネットワークを介してデータを送受信する場合，送受信の途中，第三者によってデータが改ざんされてしまうおそれがあります。例えば，受注や発注の金額が改ざんされた場合など，重大な損失が発生するおそれがあります。XML 署名は，次のような仕組みで改ざんによる被害を防止し，データの完全性を保証します。

送信前の電子データのダイジェスト値が，XML 署名に含まれます。そのため，データに XML 署名を付けておけば，送信から受信の間でデータが改ざんされているかどうかを，受信したデータのダイジェスト値と XML 署名に含まれるダイジェスト値を比較することで調べられます。XML 署名付きのデータを受信した人は，専用のアプリケーションで XML 署名を検証します。

成り済ましまたは否認を防止する

A さんではない人が，A さん本人であると身分を偽って電子データを送信することを，**成り済まし**と呼びます。反対に，A さん本人が送信した電子データに対して，A さんが「自分が送信したのではない」と主張することを，**否認**と呼びます。電子データをやり取りする場合，こうした成り済ましや否認を防止する必要があります。XML 署名は，次のような仕組みで成り済ましおよび否認を防止します。

電子データの送信者がだれなのかを特定するために，XML 署名には署名生成時に使用した秘密鍵と対になる公開鍵を含む証明書の情報を含むことができます。証明書とは，信頼できる第三者機関である認証局（CA）によって発行される，電子的な身分証のようなものです。また，秘密鍵は，その持ち主が安全に管理しているので，A さんの秘密鍵は A さんだけが使用できます。この場合，対になる公開鍵で検証に成功するのは，A さんが管理している秘密鍵で生成した XML 署名だけです。そのため，電子データに付いている XML 署名の検証が成功すれば，証明書から識別される A さんが，その電子データを確かに送信したことが証明されます。

このように，XML 署名では，秘密鍵，公開鍵，および証明書を利用して電子データの送信者を認証するので，成り済ましや否認を防止できます。

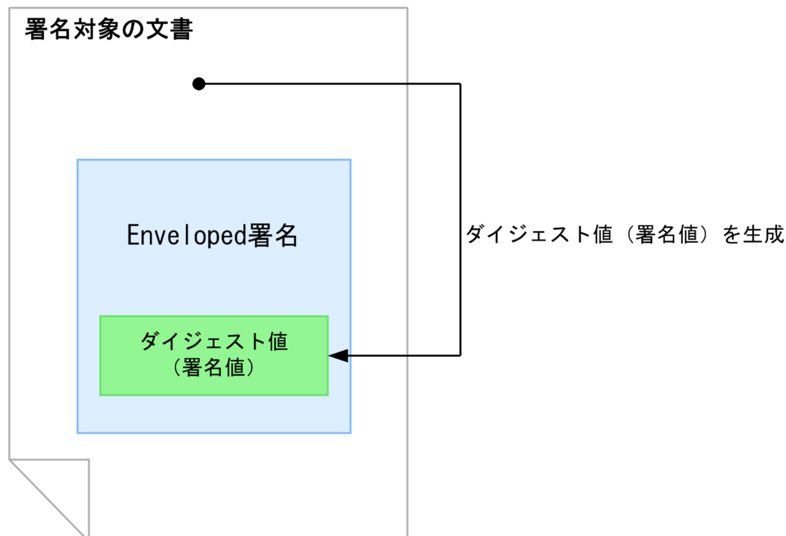
2.1.1 XML 署名の種類

XML 署名には，Enveloped 署名・Enveloping 署名・Detached 署名の三つがあります。それぞれの XML 署名について説明します。

(1)　Enveloped 署名

Enveloped 署名は，署名の対象となる文書の内部に置かれます。Enveloped 署名と署名対象の文書の関係を次の図に示します。

図 2-1　Enveloped 署名と署名対象の文書の関係

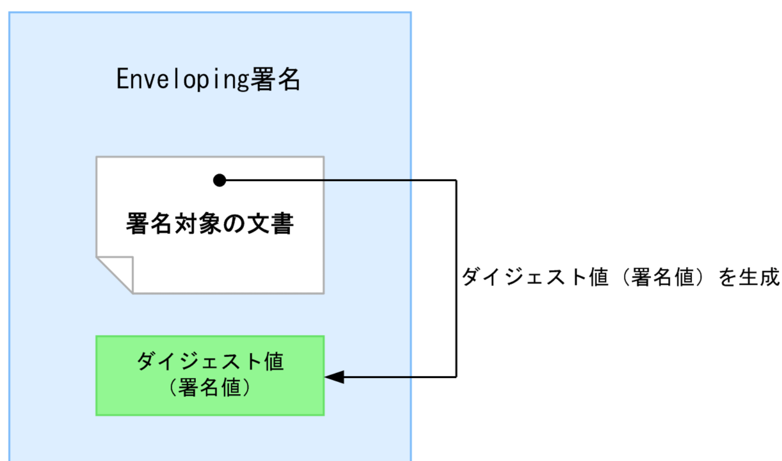


Enveloped 署名は署名対象の文書の内部に置かれますが，Enveloped 署名そのものは署名の対象にはなりません。

(2)　Enveloping 署名

Enveloping 署名は，Enveloping 署名の内部に署名の対象となる文書が置かれます。Enveloping 署名と署名対象の文書の関係を次の図に示します。

図 2-2　Enveloping 署名と署名対象の文書の関係

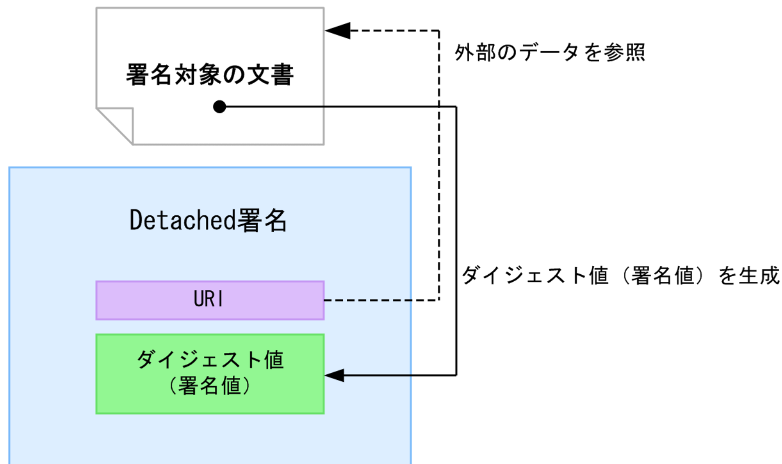


Enveloping 署名は，署名対象の文書を署名要素で包括する形式の XML 署名です。

(3) Detached 署名

Detached 署名は，署名の対象と署名が独立した形式の XML 署名です。Detached 署名と署名対象の文書の関係を次の図に示します。

図 2-3 Detached 署名と署名対象の文書の関係



Detached 署名は，署名対象には任意の形式の電子データを指定できます。また，XML 文書内に署名要素と署名対象要素を並列に配置させることもできます。

2.1.2 XML 署名の構文

XML 署名には，署名対象となる文書のダイジェスト値のほかに，アルゴリズムや鍵情報などの要素が含まれています。この節では，XML 署名にどのような要素が含まれているのか，XML 署名の構文について説明します。

XML 署名の構文を次の図に示します。なお，次の図では「?」は 0 回または 1 回，「+」は 1 回以上，「*」は 0 回以上の繰り返しを表します。

図 2-4 XML 署名の構文の概要

```
<Signature ID?>                        <!-- 署名要素 -->  ･･･1.
  <SignedInfo>                         <!-- 署名情報要素 --> ･･･2.
    <CanonicalizationMethod/>          <!-- 正規化アルゴリズム要素 --> ･･･3.
    <SignatureMethod/>                 <!-- 署名アルゴリズム要素 --> ･･･4.
    (<Reference URI?>                  <!-- 参照要素 --> ･･･5.
      (<Transforms>)?                  <!-- 変換処理要素 -->
      <DigestMethod>                   <!-- ダイジェストアルゴリズム要素 --> ･･･6.
      <DigestValue>                    <!-- ダイジェスト値要素 --> ･･･7.
    </Reference>)+
  </SignedInfo>
  <SignatureValue>                     <!-- 署名値要素 --> ･･･8.
  (<KeyInfo>)?                         <!-- 鍵情報要素 -->
  (<Object ID?>)*                      <!-- オブジェクト要素 -->
</Signature>
```

（網掛け）で示した要素は，必須の要素です。

まず，必須の要素について説明します。

1. 署名要素

   XML 署名のルート要素です。

2. 署名情報要素

   実際に署名の対象となる情報です。署名値要素を求める場合に使用されるアルゴリズムが含まれています。

3. 正規化アルゴリズム要素

   署名値を計算する前に，署名情報要素を正規化するアルゴリズムです。内容が同じ XML 文書でも，表現方法が異なる場合がありますが，正規化を実施することによって，表現方法が統一されます。

4. 署名アルゴリズム要素

   正規化された署名情報要素から署名値要素を求めるアルゴリズムです。署名情報要素の署名値を計算します。

5. 参照要素

   ダイジェストアルゴリズム要素およびダイジェスト値要素を含みます。また，変換処理要素を含む場合もあります。

6. ダイジェストアルゴリズム要素

   署名対象（参照要素から参照するデータを変換処理した結果）のダイジェスト値を計算するアルゴリズムです。変換が完了した署名対象のダイジェスト値を計算します。

7. ダイジェスト値要素

   署名対象（参照要素から参照するデータを変換処理した結果）のダイジェスト値の情報です。署名対象の文書に対する署名は，ダイジェスト値と，そのダイジェスト値を含む署名情報要素を計算することで生成されます。

8. 署名値要素

   署名情報要素の値から計算された値です。署名値要素は，必ず署名情報要素の外側に置きます。

次に，任意の要素について説明します。

- 変換処理要素

  ダイジェスト計算の入力データを生成します。例えば，署名要素自身を署名対象から外す場合は，Enveloped Signature 変換を使用します。

- 鍵情報要素

  署名の検証に必要な鍵を示す要素です。証明書や鍵の名称などが含まれます。これらの情報を第三者に公開したくない場合は，鍵情報要素は省略できます。

- オブジェクト要素

  任意のデータを格納してよい要素です。例えば，Enveloping 署名で署名対象のデータを格納する場合に使用します。

2.2　XML 暗号とは

この節では，XML 暗号について説明してから，XML Security - Core で暗号化できるデータ，XML Security - Core がサポートする暗号化処理の機能，および XML 暗号の構文について説明します。

XML 暗号とは，第三者に知られたくないデータを送信する場合に，データを暗号化するための仕組みです。電子商取引やオンラインショッピングなどが普及してきたことによって，ネットワーク上で個人情報をやり取りする機会が増えました。個人情報を保護するために，XML 暗号を始めとする暗号技術は，電子署名と並んで重要な技術となっています。XML 暗号には，次のような特長があります。

データの秘匿性を保証する

ネットワークを介してデータを送受信する場合，カード番号など第三者に知られたくない情報が漏洩しないように，データを保護する必要があります。XML 暗号は，次のような仕組みで情報漏洩による被害を防止し，データの秘匿性を保証します。

SSL など従来のセキュリティ技術では，送受信のときの通信経路だけを暗号化するため，複数のサーバを経由してデータをやり取りする場合，中継のサーバ上ではデータの秘匿性は保証されません。XML 暗号を使用する場合は，ユーザーは，XML 暗号アプリケーションで送信するデータ全体または一部を暗号化してからデータを送信します。データそのものが暗号化されているので，通信経路でも中継サーバ上でも，データの秘匿性が保証されます。

鍵合意によって共通鍵を安全に作成できる

共通鍵暗号技術では，特定の二者間だけで共有する共通鍵のデータをどのようにして安全にやり取りするのかが重要になります。鍵合意を利用すると，共通鍵のデータをネットワークなどでやり取りする必要がありません。そのため，共通鍵を作成するときに，不特定の第三者に共通鍵のデータを盗聴されるおそれが少なくなります。鍵合意の仕組みの詳細については，「2.2.2　鍵合意」を参照してください。

2.2.1　暗号化できるデータの種類

XML Security - Core で暗号化できるデータの種類を説明します。

(1)　XML データの要素またはコンテンツ

XML Security - Core では，XML データの要素やコンテンツを暗号化できます。XML データの要素またはコンテンツを暗号化する場合は，暗号化の前にデータの直列化という処理が必要になります。また，復号化のあとには，復元という処理が必要になります。

XML データの要素またはコンテンツの暗号化については，「4.2.1　XML データの要素またはコンテンツを暗号化する」を参照してください。復号化については，「4.2.2　XML データの要素またはコンテンツを復号化する」を参照してください。データの直列化または復元については，「2.2.4　データの直列化および復元処理」を参照してください。

(2) バイナリデータ

XML Security - Core では，画像などのバイナリデータを暗号化できます。バイナリデータの暗号化については，「4.2.3　バイナリデータを暗号化する」を参照してください。復号化については，「4.2.4　バイナリデータを復号化する」を参照してください。

(3) 鍵データ

XML Security - Core では，データを暗号化するときに使用する鍵データも暗号化できます。鍵データの暗号化については，「4.2.5　鍵データを暗号化する」を参照してください。復号化については，「4.2.6　鍵データを復号化する」を参照してください。

2.2.2 鍵合意

XML Security - Core は，Diffie-Hellman 鍵合意アルゴリズムを利用した鍵合意の機能を提供しています。鍵合意とは，XML データやバイナリデータなど，データを暗号化する場合に使用する鍵，またはデータを暗号化する場合に使用する鍵そのものを暗号化するときに使用する鍵を作成する仕組みです。Diffie-Hellman 鍵合意によって共通鍵を生成する例を説明します。

例えば，A さんと B さんという二人のユーザーがいる場合，二人はそれぞれ自分の秘密鍵と相手の公開鍵の鍵ペアを用意します。A さんは，A さんの秘密鍵と B さんの公開鍵を使用して，新しい値「K」を生成します。B さんは，B さんの秘密鍵と A さんの公開鍵を使用して，A さんと同じ値「K」を生成できます。したがって，A さんと B さんは，値「K」を共通鍵として使用できます。

2.2.3 XML 暗号の構文

この節では，XML 暗号の構文について説明します。

XML 暗号の構文を次の図に示します。なお，次の図では「?」は 0 回または 1 回，「*」は 0 回以上の繰り返しを表します。

図 2-5　XML 暗号の構文の概要

```
<EncryptedData Id? Type? MimeType? Encoding?>  <!-- 暗号データ要素 -->  ･･･1.
  <EncryptionMethod/>?                         <!-- 暗号アルゴリズム要素 -->
  (<ds:KeyInfo>                                <!-- 鍵情報要素 -->
    <EncryptedKey>?                            <!-- 暗号鍵要素 -->
    <AgreementMethod>?                         <!-- 鍵合意要素 -->
    <ds:KeyName>?                              <!-- 鍵名称要素 -->
    <ds:RetrievalMethod>?                      <!-- 暗号鍵取得アルゴリズム要素 -->
    <ds:*>?
  </ds:KeyInfo>)?
  <CipherData>                                 <!-- 暗号化データ要素 -->  ･･･2.
    <CipherValue>?                             <!-- 暗号化値要素 -->
    <CipherReference URI?>?                    <!-- 暗号化参照要素 -->
  </CipherData>
  <EncryptionProperties>?                      <!-- 暗号プロパティ要素 -->
</EncryptedData>
```

（網掛け）で示した要素は，必須の要素です。

まず，必須の要素について説明します。

1. **暗号データ要素**

   XML 暗号のルート要素です。

2. **暗号化データ要素**

   暗号化データを指定する要素です。暗号化データは，暗号化値要素または暗号化参照要素で指定します。

次に，任意の要素について説明します。

- **暗号アルゴリズム要素**

  データを暗号化する場合の暗号アルゴリズムを指定する要素です。受信者があらかじめ暗号アルゴリズムを知っているときは，暗号アルゴリズム要素を省略できます。

- **鍵情報要素**

  暗号化したデータを復号化する場合に使用する鍵の情報を持つ要素です。

- **暗号鍵要素**

  公開鍵で共通鍵を暗号化した鍵データを指定する要素です。

- **鍵合意要素**

  鍵合意で共通鍵を作成する場合にアルゴリズムやパラメタを指定する要素です。

- **鍵名称要素**

  鍵情報要素で指定されている鍵の所有者の名前を示す要素です。

- **暗号鍵取得アルゴリズム要素**

  暗号データ要素や暗号鍵要素などに結び付けられた暗号化データの復号に必要な鍵を含む，暗号鍵要素へのリンクを示す要素です。

- **暗号化値要素**

  暗号化したオクテットシーケンスを base64 で符号化した要素です。

- 暗号化参照要素
  暗号化したオクテットシーケンスが暗号データ要素の外部にある場合，外部位置への参照情報を指定する要素です。
- 暗号プロパティ要素
  追加情報を指定する要素です。

2.2.4　データの直列化および復元処理

XML Security - Core では，XML データの要素またはコンテンツを暗号化する場合，暗号化の対象となる部分を直列化して UTF-8 文字列として出力してから，暗号化を実施します。そのため，暗号化された XML データの要素またはコンテンツを復号化する場合，復号化結果として出力される UTF-8 文字列のデータを XML データに復元する必要があります。

この節では，XML Security - Core の XML 暗号機能によるデータの直列化および復元処理について説明します。なお，説明の中に出てくる API の仕様については，次のフォルダに格納されている API リファレンスを参照してください。

API リファレンスの格納場所（Windows の場合だけ）

```
Application Serverのインストール先ディレクトリ/XMLSEC/docs/xsecapi
```

(1)　直列化処理

XML データの要素またはコンテンツを暗号化する場合，XMLEncryption クラスの encryptXML メソッドを使用します。encryptXML メソッドは，DOM サブツリーを直列化したあと，暗号化を実行します。encryptXML メソッドは，次の規則に従って XML データを直列化します。

- 直列化した結果は，UTF-8 で出力する。
- DOM サブツリー内のノードは，文書順に出力する。
- DOM サブツリー内のノードは，省略しないですべて出力する。例えば，デフォルト属性が指定されている場合は，デフォルト属性も出力する。
- DOM サブツリー内のノードに対して，ノード外からの情報は導入しない。例えば，C14N のように，名前空間宣言を祖先から導入しない。
- 子ノードを持たない要素ノードも含めて，すべての要素ノードの開始タグおよび終了タグを出力する。例えば，元の XML 文書で<foo/>と記述されている要素は，<foo></foo>として出力する。
- 要素内の属性または名前空間宣言の出力順序は不定。

直列化を実行する場合に名前空間宣言を生成したり，名前空間宣言を祖先ノードから導入したりする必要があるときは，独自の直列化処理を実装してください。その場合は，独自に直列化した結果のオクテット列を encrypt メソッドで暗号化する必要があります。

(2) 復元処理

暗号化された XML データの要素またはコンテンツを復号化する場合，XMLEncryption クラスの decryptXML メソッドを使用します。decryptXML メソッドは，XML データを復号化したあと，復号化結果から DOM サブツリーを復元します。decryptXML メソッドは，次の規則に従って DOM サブツリーを復元します。

- 復号化の結果を UTF-8 で表現された XML データとして，復元処理を実行する。
- 要素または要素コンテントとして well-formed な XML データに対して，復元処理を実行する。
- EncryptedData と置き換えた場合に，すべての名前空間が宣言されている XML データだけに対して復元処理を実行する。

注意事項

XML Security - Core では，エンティティ参照を含む XML データに対しては復元処理を実行できません。ただし，定義済みのエンティティ（amp，lt，gt，apos，quot）を含む XML データに対しては，復元処理を実行できます。

暗号化のときとは異なる名前空間コンテキストで復号化する場合などは，独自の復元処理を実装してください。その場合は，decrypt メソッドで復号した結果を独自に復元する必要があります。

2.2.5 DataReference および KeyReference

DataReference 要素は EncryptedData 要素を，KeyReference 要素は EncryptedKey 要素を参照する要素です。これらの要素は，URI 属性で参照先を識別して，該当する要素を取得します。ここでは，DataReference 要素および KeyReference 要素の URI 参照について説明します。

XML Security - Core は，DataReference 要素および KeyReference 要素の URI 参照解決として，同一文書内の ID 指定だけをサポートしています。DataReference 要素および KeyReference 要素の URI 参照解決では，次のような場合はエラーとなるので，注意してください。

- ID 指定以外の URI 参照解決を使用している場合
- ID 指定によって識別できる要素が EncryptedData 要素または EncryptedKey 要素ではない場合

なお，DataReference 要素および KeyReference 要素の子要素の処理はサポートしていません。

XML Security - Core がサポートしている URI 参照解決の詳細については，「4.3.1　URI 参照解決」を参照してください。

2.3　XML 署名および XML 暗号に共通の機能

この節では，XML 署名と XML 暗号に共通の機能を説明します。XML 署名と XML 暗号の処理に共通する機能とは，XML 構文を生成・解析する機能および XML 文書を出力する機能です。また，この節では XML 構文を生成または解析するときに必要となるアルゴリズム，コンテキストの設定，および鍵情報の設定についても説明します。

2.3.1　XML 構文の生成および解析処理

XML 署名および XML 暗号の構文（要素，属性，および要素コンテンツ）は，W3C によって仕様として規定されています。アプリケーションを開発する場合，XML Security - Core が提供する API を利用して，XML 構文に従って署名文書や暗号文書を作成したり解析したりします。

XML Security - Core は，W3C の仕様で明示的に規定される要素ごとに，対応するクラスまたはインタフェースを提供しています。ただし，署名値（署名値要素）とダイジェスト値（ダイジェスト値要素）については，XML Security - Core が自動的に処理します。XML Security - Core の API の仕様については，次のフォルダに格納されている API リファレンスを参照してください。

API リファレンスの格納場所（Windows の場合だけ）

```
Application Serverのインストール先フォルダ/XMLSEC/docs/xsecapi
```

なお，XML 構文を生成および解析する場合，アルゴリズムを使用したり，W3C の仕様で規定されていない情報を実行時のコンテキストとして API で指定したりする必要があります。XML Security - Core がサポートするアルゴリズムについては，「2.3.2　サポートするアルゴリズム」を参照してください。コンテキストについては，「2.3.3　コンテキストの設定」を参照してください。また，XML 署名に鍵を含める場合は，「2.3.4　鍵情報の設定および取得」を参照してください。

2.3.2　サポートするアルゴリズム

XML 署名構文では，署名データの生成および検証で実行するアルゴリズムが規定されています。また，XML 暗号構文では，データを暗号化および復号化する場合に実行するアルゴリズムが規定されています。この項では，XML Security - Core がサポートしているアルゴリズムについて説明します。

XML Security - Core がサポートしている，XML 署名構文および XML 暗号構文で使用するアルゴリズムの要素名，名称，および識別子を次の表に示します。なお，要素名の後ろの括弧は，その要素の役割を説明しています。

表 2-1　アルゴリズム一覧

要素名（役割）	名称	識別子
Transform 要素	base64	http://www.w3.org/2000/09/xmldsig#base64

要素名（役割）	名称	識別子
（変換処理）	Enveloped Signature	http://www.w3.org/2000/09/xmldsig#enveloped-signature
	正規化	CanonicalizationMethod 要素の項を参照のこと。
	XPath	http://www.w3.org/TR/1999/REC-xpath-19991116
	XPath Filter 2	http://www.w3.org/2002/06/xmldsig-filter2
CanonicalizationMethod 要素 （正規化処理）	Canonical XML Version 1.0 (omit comments)	http://www.w3.org/TR/2001/REC-xml-c14n-20010315
	Canonical XML Version 1.0 with comments	http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
	Exclusive XML Canonicalization Version 1.0 (omit comments)	http://www.w3.org/2001/10/xml-exc-c14n#
	Exclusive XML Canonicalization Version 1.0 with comments	http://www.w3.org/2001/10/xml-exc-c14n#WithComments
DigestMethod 要素 （ダイジェスト値計算）	SHA1	http://www.w3.org/2000/09/xmldsig#sha1
	SHA256	http://www.w3.org/2001/04/xmlenc#sha256
	SHA512	http://www.w3.org/2001/04/xmlenc#sha512
SignatureMethod 要素 （署名値計算）	HMAC-SHA1	http://www.w3.org/2000/09/xmldsig#hmac-sha1
	DSAwithSHA1(DSS)	http://www.w3.org/2000/09/xmldsig#dsa-sha1
	RSAwithSHA1	http://www.w3.org/2000/09/xmldsig#rsa-sha1
EncryptionMethod 要素 （暗号化処理，鍵ラッピング）	TRIPLEDES	http://www.w3.org/2001/04/xmlenc#tripledes-cbc
	AES-128	http://www.w3.org/2001/04/xmlenc#aes128-cbc
	TRIPLEDES KeyWrap	http://www.w3.org/2001/04/xmlenc#kw-tripledes
	AES-128 KeyWrap	http://www.w3.org/2001/04/xmlenc#kw-aes128
AgreementMethod 要素 （鍵合意処理）	Diffie-Hellman	http://www.w3.org/2001/04/xmlenc#dh

2.3.3 コンテキストの設定

XML Security - Core では，実行環境に固有の情報のうち，XML 構文に明示的に現れない情報をコンテキストとして処理します。コンテキストとして処理される情報を次に示します。

- **処理モード**

  実行中の処理のモードを表します。署名に関する処理モードには，署名生成モードと署名検証モードの2 種類があります。また，暗号に関する処理モードには，暗号化モードと復号化モードの 2 種類があります。

- **文書オブジェクト**

  署名文書（Signature 要素が置かれる XML 文書），または暗号文書（EncryptedData 要素またはEncryptedKey 要素が置かれる XML 文書）を表す Document オブジェクトです。

- **ベース URI**

  相対 URI で外部リソースを取得する場合に使用する URI です。署名対象または暗号化対象を取得する場合に，署名文書または暗号文書から外部ファイルを相対パスで参照するときに必要になります。

- **URI 参照の解決手段**

  署名対象または暗号化対象を指定する場合の URI 参照を，ファイルなどの具体的なデータに結び付けるためのコールバックです。利用環境に応じてカスタマイズできます。URI 参照解決のカスタマイズについては，「4.3.1　URI 参照解決」を参照してください。

- **XML 要素の ID 解決手段**

  URI 参照のうち，署名文書または暗号文書内の要素を識別する場合に，ID と XML 要素をマッピングするためのコールバックです。利用環境に応じてカスタマイズできます。ID 解決のカスタマイズについては，「4.3.2　ID 解決」を参照してください。

- **鍵の解決手段**

  署名の生成・検証，またはデータの暗号化・復号化に必要な鍵を取得するためのコールバックです。鍵情報に基づいて，鍵を取得します。

- **XML 名前空間プレフィックス**

  署名文書または暗号文書を生成するときに，XML 署名，XML 暗号，または各種変換アルゴリズムなど，仕様で決められた名前空間を識別する名前空間プレフィックスを指定します。

- **擬似乱数生成器**

  暗号文書を作成するときに，暗号の安全性を高めるための乱数を生成するオブジェクトです。

2.3.4　鍵情報の設定および取得

XML 署名または XML 暗号では，鍵情報（KeyInfo 要素）のスキーマが規定されています。XML Security - Core でアプリケーションを開発する場合，鍵情報を基に，XML 署名データの生成・検証，およびデータの暗号化・復号化に必要な鍵を特定して，その鍵を取得することもできます。

XML Security - Core では，KeyInfo 要素の下位にある要素に対応するクラスで鍵情報を取得する手段を提供しています。KeyInfo 要素から実際に鍵を取得する処理は，利用環境に応じてアプリケーションの開発者が実装する必要があります。

KeyInfo 要素の下位要素のうち，XML Security - Core がサポートしている要素とその要素の名前空間を次の表に示します。

表 2-2　KeyInfo 要素の下位要素の一覧

項番	要素名	名前空間
1	KeyName	http://www.w3.org/2000/09/xmldsig#
2	KeyValue	
3	DSAKeyValue	
4	RSAKeyValue	
5	RetrievalMethod	
6	X509Data	
7	X509Certificate	
8	X509CRL	
9	X509IssuerSerial	
10	X509SKI	
11	X509SubjectName	
12	EncryptedKey	http://www.w3.org/2001/04/xmlenc#
13	AgreementMethod	
14	DHKeyValue	

2.3.5　XML 文書の出力処理

XML Security - Core では，XML 署名処理または XML 暗号を実施した Document オブジェクトを XML 文書としてファイルなどに出力できます。XML 文書を出力する機能として，次の処理をサポートしています。

- 出力先（ファイルを含めたストリーム）
- エンコーディング
    - UTF-8／16（推奨※1）
    - Shift_JIS
    - Windows-31J
    - EUC-JP
    - ISO-2022-JP
    - US-ASCII

    - ISO-8859-1
- 改行文字（LF，CR，CRLF）
- 空タグの出力
- XML 宣言，DOCTYPE 宣言出力の有無
- コメントの省略
- 冗長な名前空間宣言※2 の省略

注※1

エンコーディングは，UTF-8／16 を推奨します。

UTF-8／16 以外のエンコーディングによる出力では，適切に対応する文字が出力されない可能性があります。

各エンコーディングの出力結果は，JDK のエンコーダに依存します。

注※2

祖先要素で同じ宣言をしている場合に，その宣言がなくても名前空間の状態が変わらない宣言を指しています。

2.3.6 デバッグダンプ機能

意図した対象を正しく署名/署名検証もしくは暗号/復号しているかを確認するために署名/署名検証および暗号/復号処理時に使用される中間データを出力することができます。ここでは，デバッグダンプ機能について説明します。

注意事項

本機能を使用すると，サイズの大きなファイル もしくは，多量のファイルを出力する可能性があります。本機能をデバッグ目的以外に使用しないでください。運用時に中間データを取得したい場合については，API リファレンスの XMLSecurityContext クラス，SignedInfo インタフェース，および Reference インタフェースを参照してください。

API リファレンスの格納場所（Windows の場合だけ）

Application Server のインストール先ディレクトリ/XMLSEC/docs/xsecapi

(1) 出力内容

ヘッダ情報

ダンプされるデータには，中間データに関する情報を含むヘッダが付加されて出力されます。ヘッダ情報には，ダンプ対象を表す文字列，時刻，およびスレッド ID を含みます。ヘッダ Decription に出力される内容の詳細については表 2-4 を参照してください。

表 2-3　ヘッダの種類

#	ヘッダ名	出力内容	備考
1	Description	ダンプ対象を表す文字列	中間データによって出力される内容が異なる。
2	Timestamp	処理時刻	yyyy/mm/dd hh:mm:ss.sss 形式
3	Thread-Id	スレッド ID	−

（凡例）

−：該当しません

ヘッダの出力例（Transform への出力情報時）

```
------------------------------------------------
Description: dereferenced data: URI=” ”
Timestamp: 2004/08/31 12:04:06.255
Thread-Id: 000c62c8
```

中間データの情報

出力される中間データで得られる情報について次の表に示します。

表 2-4　出力される中間データの情報

#	出力される情報の種類	出力される処理	ヘッダ Description の出力内容	出力される中間データ
1	Transform への入力情報	署名，署名検証，復号※	dereferenced data: URI=<URI>	変換処理対象のデータ
2	Digest への入力情報	署名，署名検証	digest input: URI=<URI>	ダイジェスト値計算対象のデータ
3	署名/検証計算への入力情報	署名，署名検証	canonicalized SignedInfo	署名/検証計算対象データ
4	暗号化処理への入力情報	暗号	encrypt input	暗号化処理対象データ
5	復号化処理への入力情報	復号	decrypt input	復号化処理対象データ

注※

暗号データが CipherReference に指定されている場合だけ

注意事項

デバッグダンプ機能によって出力される中間データは，すべてバイナリデータです。中間データに漢字コードや改行コードなどが含まれる場合，テキストエディタでは正しく表示されない場合があります。ダンプされるデータがノードセットである場合，Inclusive XML Canonicalization 処理された結果が出力されます。ただし，コメントは含まれません。ノードセットおよび Inclusive XML Canonicalization 処理の詳細については，W3C のホームページを参照してください。

(2) 出力単位

デバックダンプ出力の単位は，一処理につき一出力とします。

例 1：一つの XML 文章に一つの XML 署名を行う場合→ダンプ出力は 1 回。

例 2：一つの XML 文章に二つの XML 署名を行う場合→ダンプ出力は 2 回。

(3) デバックダンプ出力設定方法と出力先

システムプロパティの設定内容

- デバックダンプ出力設定

  デバッグダンプ出力を行うかどうかの設定を行います。true に設定された場合，デバッグダンプ出力を行います。

- 出力先フォルダ

  デバッグダンプで出力されるファイルの出力先を設定します。出力先が設定されていない場合，および設定された出力先が存在しなかった場合は標準出力にダンプ出力されます。出力先フォルダが設定されていても，デバッグダンプ出力設定がされていない場合にはダンプ出力は行われません。

推奨するデバッグダンプの出力先（Windows の場合）

Application Server のインストール先ディレクトリ/XMLSEC/logs

推奨するデバッグダンプの出力先（UNIX の場合）

/opt/Cosminexus/XMLSEC/logs

設定値

デバックダンプ機能の設定項目とデフォルト値を次の表に示します。

表 2-5 デバックダンプの設定項目に設定する値

設定項目	プロパティ名	デフォルト値
デバックダンプ出力設定	com.cosminexus.xml.security.debug.dump	設定なし
出力先フォルダ	com.cosminexus.xml.security.debug.dumpdir	設定なし

システムプロパティの設定方法

システムプロパティの設定は，java コマンドの-D オプションを利用します。次に`C:¥Program Files¥HITACHI¥Cosminexus¥XMLSEC¥logs` にダンプ出力を行う例を以下に示します。

システムプロパティの設定例（-D オプションを利用する場合）

```
java -Dcom.cosminexus.xml.security.debug.dump=true -Dcom.cosminexus.xml.security.debug.du
mpdir=C:¥Program Files¥HITACHI¥Cosminexus¥XMLSEC¥logs YourApplicationClass
```

J2EE サーバまたは Web コンテナサーバの Component Container で XML Security - Core を使用する場合は，Component Container のユーザー定義ファイル（usrconf.properties）にシステムプロパティを記述します。ユーザー定義ファイルは java.util.Properties 形式で記述する必要があります。そのため，

パスに「¥」記号や空白文字が含まれる場合は，「¥¥」としたり，空白の前に「¥」記号を追加したりして，エスケープする必要があります。また，java.util.Properties 形式の場合は日本語を直接指定できないので，ディレクトリ名などで日本語を使用している場合は，注意してください。ユーザー定義ファイルにシステムプロパティを記述する場合の例を次に示します。

システムプロパティの設定例（ユーザー定義ファイルに記述する場合）

```
com.cosminexus.xml.security.debug.dump=true
com.cosminexus.xml.security.debug.dumpdir=C:¥¥Program¥ Files¥¥Hitachi¥¥Cosminexus¥¥XMLSEC
¥¥logs
```

Component Container のユーザー定義ファイルの設定方法については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

(4) ファイル名

ダンプ出力時にディレクトリを指定された際に出力されるファイルの名前は，プレフィックス csmxsec に実行された処理，スレッド ID，および時刻を合わせたものになります。

```
csmxsec_{処理名}_スレッドID_yyyymmddhhmmsssss.dmp
```

表 2-6 ファイル名に付加される処理名

#	処理	処理名
1	署名	sig
2	署名検証	ver
3	暗号	enc
4	復号	dec

例：実行された処理が署名，スレッド ID が 01b3f8f6，および実行された時間が 2004 年 10 月 1 日 12:00:12345 の場合のファイル名。

csmxsec_sig_01b3f8f6_20041001120012345.dmp

3 環境設定

この章では，XML Security - Core を使用してアプリケーションを開発したり，実行したりする場合に必要な環境設定について説明します。「4.　アプリケーションの開発」の作業を実行する前に必ずお読みください。

3.1 クラスパスを設定する

XML Security - Core でアプリケーションを開発したり，実行したりするためには，jar ファイルをクラスパスに設定する必要があります。クラスパスに設定する必要がある jar ファイルの名称と格納場所を次の表に示します。

表 3-1　クラスパスに設定する jar ファイル一覧

jar ファイル名	格納されている場所	
	Windows の場合	UNIX の場合
csmxsec.jar	*Application Server インストール先ディレクトリ*/XMLSEC/lib	/opt/Cosminexus/XMLSEC/lib
csmjaxp.jar	*Application Server インストール先ディレクトリ*/jaxp/lib	/opt/Cosminexus/jaxp/lib
hntrlib2j64.jar	*OS インストール先ドライブ*/Program Files/Hitachi/HNTRLib2/classes	/opt/hitachi/HNTRLib2/classes

参考

Component Container 上のアプリケーションから XML Security - Core を利用する場合は，Web コンテナサーバでも J2EE サーバでも，表 3-1 に示した jar ファイルは Component Container によって自動的に読み込まれます。そのため，Component Container 上のアプリケーションから XML Security - Core を利用する場合は，クラスパスの設定は必要ありません。

3.2　セキュリティプロバイダを設定する

XML Security - Core を使用するためには，Java セキュリティのアルゴリズムおよび証明書タイプの標準名が使用できるようにセキュリティのプロバイダを設定する必要があります。この節では，XML Security - Core を使用するためのセキュリティプロバイダの設定について説明します。

アルゴリズムを有効にするための設定を次の表に示します。

表 3-2　アルゴリズムを有効にするための設定

アルゴリズム		Java セキュリティ	
種別	名称	エンジンクラス	アルゴリズムの標準名
DigestMethod	SHA1	MessageDigest	SHA-1
	SHA256		SHA-256
	SHA512		SHA-512
SignatureMethod	HMAC-SHA1	Mac	HmacSHA1
	DSAwithSHA1(DSS)	Signature	SHA1withDSA
	RSAwithSHA1		SHA1withRSA
EncryptionMethod	TRIPLEDES	Cipher	DESede/CBC/NoPadding
	AES-128		AES/CBC/NoPadding
	AES-128 KeyWrap		AES/ECB/NoPadding
	TRIPLEDES KeyWrap		DESede/CBC/NoPadding
		MessageDigest	SHA-1
AgreementMethod	Diffie-Hellman	KeyAgreement	DiffieHellman

鍵情報の要素を有効にするための設定を次の表に示します。

表 3-3　鍵情報の要素を有効にするための設定

鍵情報の要素名	Java セキュリティ	
	エンジンクラス	アルゴリズムまたは証明書の標準名
ds:DSAKeyValue	KeyFactory	DSA
ds:RSAKeyValue		RSA
xenc:DHKeyValue		DiffieHellman
ds:X509Certificate	CertificateFactory	X.509
ds:X509CRL		

なお，セキュリティのプロバイダを設定については，Java^{TM} Cryptography Architecture API Specification & Reference も参照してください。

3.3　鍵と証明書を設定する

XML 署名データを生成および検証するためには，鍵および証明書を生成して，設定する必要があります。この節では，Developer's Kit for Java に付属する keytool を使用して，鍵および証明書を生成する場合について説明します。keytool の詳細については，Java2SDK のドキュメントを参照してください。

3.3.1　鍵および自己署名証明書を生成する

keytool -genkey を使用すると，キーストアの中に公開鍵と秘密鍵のペア，および自己署名証明書を生成できます。

keytool -genkey で生成した公開鍵は，自己署名証明書でラップされます。自己署名証明書は，署名の発行者と署名によって認証される公開鍵を所有するエンティティが同じなので，証明書としての信頼度は低くなります。

通信相手との取り決めなどによって，自己署名証明書で運用できる場合は，keytool -export で指定の公開鍵を含む自己署名証明書を出力します。出力した自己署名証明書を通信相手に配布します。また，keytool -import を使用すれば，出力した自己署名証明書を別のキーストアに読み込むこともできます。

自己署名証明書よりも信頼性の高い証明書が必要な場合は，民間組織が運営する認証局などから証明書を取得する必要があります。

注意事項

キーストアは，鍵および証明書のリポジトリを管理するためのデータベースです。多くの証明書を管理したり，CRL を管理したりするためには使用できません。

3.3.2　認証局が発行する証明書を取得する

自己署名証明書ではなく，信頼できる認証局が発行する証明書が必要な場合は，keytool -certreq を使用して証明書の発行に必要な証明書署名要求（CSR）を生成します。証明書の発行を要求すると，認証局は証明書応答を作成します。ユーザーは，keytool -import を使用して認証局から証明書応答をインポートし，自己証明書を証明書連鎖に置き換えます。

3.4　環境設定するときの注意事項

XML Security - Core の環境設定をするときに注意が必要な項目を説明します。

クラスローダによる jar ファイルの読み込み

XML Security - Core は，csmxsec.jar が各 Java VM 内で 1 回だけ読み込まれるシステム構成で使用する必要があります。一つの Java VM 内で複数のクラスローダによって XML Security - Core の jar ファイルが読み込まれた場合，おのおのが同じ設定でトレースファイルを出力しようとするため，トレースファイルが壊れるおそれがあります。

Component Container 上で XML Security - Core を使用する場合は，Component Container によって XML Security - Core の jar ファイルが適切に読み込まれるように設定されます。ただし，一つのシステムで複数の Web コンテナサーバおよび J2EE サーバを運用する場合は，各サーバの Component Container のユーザー定義ファイル（usrconf.properties）を適切に設定して，XML Security - Core のトレースファイルの設定が重複しないようにする必要があります。

ユーザー定義ファイルの設定方法については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

4 アプリケーションの開発

この章では，XML Security - Core を使用した，XML 署名アプリケーションおよび XML 暗号アプリケーションの開発の概要について，コーディング例を示しながら説明します。アプリケーション開発の概要のあとに，各アプリケーションのカスタマイズについて説明します。また，最後にアプリケーション開発時の注意事項を説明しますので，実際にアプリケーションを開発する前に必ずお読みください。

4.1　XML 署名アプリケーションを開発する

XML Security - Core を使用して，XML 署名アプリケーションを開発するときの流れについて説明します。開発の流れは，コーディング例を示して説明します。XML 署名機能は，署名の生成と署名の検証との二つに大別できます。この節では，XML 署名を生成，および検証するアプリケーションの開発について説明します。

XML 署名には，署名の位置によって次に示す 3 種類の署名があります。

- Enveloped 署名
- Enveloping 署名
- Detached 署名

この節では，Enveloped 署名，Enveloping 署名，および Detached 署名を生成および検証するアプリケーションの開発を説明します。なお，説明の中に示されるコーディング例は，次に格納されているサンプルプログラムに記載されています。必要に応じて参照してください。

サンプルプログラムの格納場所（Windows の場合だけ）

```
Application Serverのインストール先ディレクトリ/XMLSEC/samples/xsec
```

4.1.1　Enveloped 署名を生成する

この項では，Enveloped 署名を生成する場合の処理内容とコーディング例を説明します。

(1)　XML 署名文書の Document オブジェクトの生成

JAXP の DocumentBuilder クラスを使用して，署名対象を読み込んで，Document オブジェクトを生成します。Document オブジェクトを取得するときに，名前空間が有効になるように設定してください。
コーディングの例を次に示します。

```
        DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        Document doc = dbf.newDocumentBuilder().parse(input);
```

(2)　XML 署名構文の構築

XMLSignatureFactory クラスを使用して，各要素に対応するオブジェクトを生成します。署名対象およびアルゴリズムは，XML 署名構文を構築するときに設定しておきます。ここでは，次の設定をする場合の例を説明します。

- 署名対象：URI="" (文書全体)

- ダイジェストアルゴリズム：SHA1
- 変換アルゴリズム：Enveloped Signature 変換
- 正規化アルゴリズム：コメントなし XML-C14N
- 署名アルゴリズム：RSAwithSHA1

コーディングの例を次に示します。

```
        XMLSignatureFactory xsf = XMLSignatureFactory.newInstance();
        Reference ref = xsf.newReference("", xsf.newDigestMethod(
                DigestMethod.URI_SHA1, null),
                        Collections.singletonList(
                xsf.newTransform(
                        Transform.URI_ENVELOPED_SIGNATURE, null)));
        SignedInfo si =
                xsf.newSignedInfo(xsf.newCanonicalizationMethod(
                        CanonicalizationMethod.URI_OMIT_COMMENTS,
                        null),
                        xsf.newSignatureMethod(
                                SignatureMethod.URI_RSA_SHA1, null),
                Collections.singletonList(ref));
        XMLSignature sig = xsf.newXMLSignature(si, null, null);
```

(3) XML 署名データの生成

Enveloped 署名データを生成するときは，コンテキストおよび署名生成位置を指定する必要があります。ここでは，次の三つの処理について，コーディング例を示して説明します。

- コンテキストの設定
- XML 署名を生成する位置の設定
- XML 署名データの生成

(a) コンテキストの設定

Enveloped 署名データの生成に必要なコンテキストを設定します。署名に使用する鍵を使って鍵リゾルバを生成し，コンテキストに鍵リゾルバを設定します。コンテキストの処理モードは，「署名生成」を指定してください。コーディングの例を次に示します。

```
        XMLSecurityContext context = new XMLSecurityContext(
                XMLSecurityContext.Mode.SIGN, doc);
        context.setKeyResolver(
                new AdhocKeyResolver(Utilities.getPrivateKey()));
```

(b) XML 署名を生成する位置の設定

署名対象の Document オブジェクト中の，どの位置に XML 署名を生成するのかを設定します。文書要素の末尾に署名を生成する場合のコーディングの例を次に示します。

```
        DOMPosition pos = new DOMPosition(doc.getDocumentElement(),
                null);
```

(c) XML署名データの生成

XMLSignatureクラスのsignメソッドを使用して，Signature要素の署名値を生成します。Reference要素のダイジェスト値と同時に，Signature要素の署名値も生成されます。コーディングの例を次に示します。

```
        sig.sign(context, pos);
```

(4) XML署名文書の出力

XMLSerializerクラスを使用して，Signature要素を持つDocumentオブジェクトをXML形式で出力します。コーディングの例を次に示します。

```
        XMLOutputFormat format = new XMLOutputFormat();
        format.setOmitRedundantNamespaceDecls(true);
        format.setEncoding("Shift_JIS");
        OutputStream os = new BufferedOutputStream(
                new FileOutputStream(output));
        XMLSerializer xsr = new XMLSerializer(os, format);
        xsr.serialize(doc);
        os.close();
```

4.1.2 Enveloping署名を生成する

この項では，Enveloping署名を生成する場合の処理内容とコーディング例を説明します。

(1) XML署名文書のDocumentオブジェクトの生成

JAXPのDocumentBuilderクラスを使用して，署名対象のXML文書を読み込んで，署名を付与したい要素（署名対象）を取得します。このとき，名前空間が有効になるように設定してください。次に，Signature要素を文書要素とする署名文書を作成する準備として，新たに空のDocumentオブジェクトを生成します。また，Enveloping署名に署名対象を含めるために，生成したDocumentオブジェクトに署名対象をあらかじめインポートしておきます。コーディングの例を次に示します。

```
        DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document doc0 = db.parse(input);
        Element elem0 = doc0.getDocumentElement();
        Document doc = db.newDocument();
        Element elem = (Element) doc.importNode(elem0, true);
```

(2) XML 署名構文の構築

XMLSignatureFactory クラスを使用して，各要素に対応するオブジェクトを生成します。Enveloping 署名では，署名対象を DOMNodeContainer オブジェクトでラップして，XMLObject オブジェクトに設定します。

署名対象および各種アルゴリズムは，XML 署名構文を構築するときに設定しておきます。ここでは，次の設定をする場合の例を説明します。なお，署名対象を指定するための Id は「foo」とします。

- 署名対象：URI="#foo"
- ダイジェストアルゴリズム：SHA1
- 変換アルゴリズム：なし
- 正規化アルゴリズム：コメントなし XML-C14N
- 署名アルゴリズム：RSAwithSHA1

コーディングの例を次に示します。

```
        XMLSignatureFactory xsf = XMLSignatureFactory.newInstance();
        XMLObject obj = xsf.newXMLObject(Collections.singletonList(
                new DOMNodeContainer(elem)));
        obj.setId(”foo”);
        Reference ref = xsf.newReference(”#foo”, xsf.newDigestMethod(
                DigestMethod.URI_SHA1, null), null);
        SignedInfo si =
                xsf.newSignedInfo(xsf.newCanonicalizationMethod(
                CanonicalizationMethod.URI_OMIT_COMMENTS, null),
                xsf.newSignatureMethod(
                         SignatureMethod.URI_RSA_SHA1, null),
                Collections.singletonList(ref));
        XMLSignature sig = xsf.newXMLSignature(si, null,
                Collections.singletonList(obj));
```

(3) XML 署名データの生成

Enveloping 署名データを生成するには，コンテキストおよび署名生成位置を指定する必要があります。ここでは，次の三つの処理について，コーディング例を示して説明します。

- コンテキストの設定
- ML 署名を生成する位置の設定
- XML 署名データの生成

(a) コンテキストの設定

Enveloping 署名データの生成に必要なコンテキストを設定します。署名に使用する鍵を使って鍵リゾルバを生成し，コンテキストに鍵リゾルバを設定します。コンテキストの処理モードは，「署名生成」を指定してください。コーディングの例を次に示します。

```
        XMLSecurityContext context
                = new XMLSecurityContext(
                         XMLSecurityContext.Mode.SIGN, doc);
        context.setKeyResolver(
                new AdhocKeyResolver(Utilities.getPrivateKey()));
```

(b)　XML 署名を生成する位置の設定

署名対象の Document オブジェクト中の，どの位置に XML 署名を生成するのかを設定します。次の例では，Signature 要素が新しい文書要素となります。コーディングの例を次に示します。

```
        DOMPosition pos = new DOMPosition(doc, null);
```

(c)　XML 署名データの生成

XMLSignature クラスの sign メソッドを使用して，Signature 要素の署名値を生成します。Reference 要素のダイジェスト値と同時に，Signature 要素の署名値も生成されます。コーディングの例を次に示します。

```
        sig.sign(context, pos);
```

(4)　XML 署名文書の出力

XMLSerializer クラスを使用して，Signature 要素を持つ Document オブジェクトを XML 形式で出力します。冗長な名前空間を省略し，Shift_JIS で出力する場合のコーディングの例を次に示します。

```
        XMLOutputFormat format = new XMLOutputFormat();
        format.setOmitRedundantNamespaceDecls(true);
        format.setEncoding("Shift_JIS");
        OutputStream os = new BufferedOutputStream(
                new FileOutputStream(output));
        XMLSerializer xsr = new XMLSerializer(os, format);
        xsr.serialize(doc);
        os.close();
```

4.1.3　Detached 署名を生成する

この項では，Detached 署名を生成する場合の処理内容とコーディング例を説明します。

(1)　XML 署名文書の Document オブジェクトの生成

JAXP の DocumentBuilder を使用して空の Document オブジェクトを生成します。ここで生成する Document オブジェクトは，Signature 要素を文書要素とする署名文書を作成するために必要となります。Document オブジェクトを取得するときに，名前空間が有効になるように設定してください。コーディングの例を次に示します。

```
        DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        Document doc = dbf.newDocumentBuilder().newDocument();
```

(2)　XML 署名構文の構築

XMLSignatureFactory クラスを使用して，各要素に対応するオブジェクトを生成します。署名対象および各種アルゴリズムは，XML 署名構文を構築するときに設定しておきます。ここでは，次の設定をする場合の例を説明します。

- 署名対象：input で指定したファイル
- ダイジェストアルゴリズム：SHA1
- 変換アルゴリズム：コメントなし XML-C14N
- 正規化アルゴリズム：コメントなし XML-C14N
- 署名アルゴリズム：RSAwithSHA1

コーディングの例を次に示します。

```
        XMLSignatureFactory xsf = XMLSignatureFactory.newInstance();
        Reference ref = xsf.newReference(
                new File(input).toURI().toString(),
                xsf.newDigestMethod(DigestMethod.URI_SHA1, null),
                Collections.singletonList(xsf.newTransform(
                Transform.URI_C14N_OMIT_COMMENTS, null)));
        SignedInfo si =
                xsf.newSignedInfo(xsf.newCanonicalizationMethod(
                CanonicalizationMethod.URI_OMIT_COMMENTS, null),
                xsf.newSignatureMethod(
                        SignatureMethod.URI_RSA_SHA1, null),
                        Collections.singletonList(ref));
        XMLSignature sig = xsf.newXMLSignature(si, null, null);
```

(3)　XML 署名データの生成

Detached 署名データを生成するには，コンテキストおよび署名生成位置を指定する必要があります。ここでは，次の三つの処理について，コーディング例を示して説明します。

- コンテキストの設定
- XML 署名を生成する位置の設定
- XML 署名データの生成

(a) コンテキストの設定

Detached 署名データの生成に必要なコンテキストを設定します。署名に使用する鍵を使って鍵リゾルバを生成し，コンテキストに鍵リゾルバを設定します。コンテキストの処理モードは，「署名生成」を指定してください。コーディングの例を次に示します。

```
        XMLSecurityContext context = new XMLSecurityContext(
                XMLSecurityContext.Mode.SIGN, doc);
        context.setKeyResolver(
                new AdhocKeyResolver(Utilities.getPrivateKey()));
```

(b) XML 署名を生成する位置の設定

署名対象の Document オブジェクト中の，どの位置に XML 署名を生成するのかを設定します。次の例では，Signature 要素が新しい文書要素となります。コーディングの例を次に示します。

```
        DOMPosition pos = new DOMPosition(doc, null);
```

(c) XML 署名データの生成

XMLSignature クラスの sign メソッドを使用して，Signature 要素の署名値を生成します。Reference 要素のダイジェスト値と同時に，Signature 要素の署名値も生成されます。コーディングの例を次に示します。

```
        sig.sign(context, pos);
```

(4) XML 署名文書の出力

XMLSerializer クラスを使用して，Signature 要素を持つ Document オブジェクトを XML 形式で出力します。冗長な名前空間を省略し，Shift_JIS で出力する場合のコーディングの例を次に示します。

```
        XMLOutputFormat format = new XMLOutputFormat();
        format.setOmitRedundantNamespaceDecls(true);
        format.setEncoding(”Shift_JIS”);
        OutputStream os = new BufferedOutputStream(
                new FileOutputStream(output));
        XMLSerializer xsr = new XMLSerializer(os, format);
        xsr.serialize(doc);
        os.close();
```

4.1.4 Enveloped 署名，Enveloping 署名，または Detached 署名を検証する

この項では，XML 署名（Enveloped 署名，Enveloping 署名，または Detached 署名）を検証する場合の処理内容とコーディング例を説明します。

(1) XML 署名文書の Document オブジェクトの取得

JAXP の DocumentBuilder クラスを使用して，検証の対象となる XML 文書を読み込み，Document オブジェクトを取得します。このとき，XML 署名構文の構築のために，あらかじめ Document オブジェクトから Signature 要素を取得しておく必要があります。また，Document オブジェクトを取得するときに，名前空間が有効になるように設定してください。Enveloped 署名の場合は文書要素の末尾が，Enveloping 署名または Detached 署名の場合は文書要素が Signature 要素となっているときのコーディングの例を次に示します。

Enveloped 署名の場合

```
DocumentBuilderFactory dbf =
    DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
Document doc = dbf.newDocumentBuilder().parse(input);
Node last = doc.getDocumentElement().getLastChild();
if ((last == null) || !(DOMUtils.matchesName(
    last, XMLSignature.XMLNS,"Signature"))) {
    throw new Exception("missing ds:Signature");
}
Element sigelem = (Element) last;
```

Enveloping 署名または Detached 署名の場合

```
DocumentBuilderFactory dbf =
    DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
Document doc = dbf.newDocumentBuilder().parse(input);
Element sigelem = doc.getDocumentElement();
if (!(DOMUtils.matchesName(sigelem, XMLSignature.XMLNS,
    "Signature"))) {
    throw new Exception("missing ds:Signature");
}
```

(2) XML 署名構文の構築

XMLSignatureFactory クラスを使用して，XMLSignature オブジェクトを生成します。コンテキストおよび Document オブジェクト中の Signature 要素を指定します。XMLSignature オブジェクトが生成されると同時に，Signature 要素の下位要素に対応するオブジェクトも自動的に生成されます。なお，コンテキストの処理モードは，「署名検証」を指定してください。コーディングの例を次に示します。

```
XMLSignatureFactory xsf = XMLSignatureFactory.newInstance();
XMLSecurityContext context = new XMLSecurityContext(
        XMLSecurityContext.Mode.VERIFY, doc);
XMLSignature sig = xsf.newXMLSignature(context, sigelem);
```

(3) XML 署名データの検証

XML 署名データを検証するときは，コンテキストを指定する必要があります。ここでは，次の処理について，コーディング例を示して説明します。

- コンテキストの設定
- XML 署名データの検証

(a)　コンテキストの設定

XML 署名データの検証に必要なコンテキストを設定します。検証に使用する鍵を使って鍵リゾルバを生成して，コンテキストに鍵リゾルバを設定します。コーディングの例を次に示します。

```
        context.setKeyResolver(
                new AdhocKeyResolver(Utilities.getPublicKey()));
```

(b)　XML 署名データの検証

XMLSignature クラスの verify メソッドで，Signature 要素の署名値を検証します。このとき，Signature 要素の署名値と同時に，Reference 要素のダイジェスト値も検証されます。

なお，署名情報や署名対象データが改ざんされていた場合，検証結果は失敗（verify メソッドの結果が false）となります。検証結果が失敗の場合，署名対象データの改ざんの有無については，ReferenceValidity クラスを使用して調べることができます。ReferenceValidity クラスで調べた結果，署名対象データが改ざんされていないにもかかわらず，検証に失敗する場合は，署名情報が改ざんされているおそれがあります。コーディングの例を次に示します。

```
        boolean result = sig.verify(context);
        if (result == true) {
            System.out.println("signature is valid.");
        } else {
            Iterator itr =
                        sig.getSignedInfo().getReferences().iterator();
            while (itr.hasNext()) {
                Reference ref = (Reference) itr.next();
                ReferenceValidity validity =
                        ref.getReferenceValidity();
                System.out.println("reference: URI=" + ref.getURI()
                        + ", validity=" + validity.getStatus());
            }
            System.out.println("signature is invalid.");
        }
```

4.2　XML 暗号アプリケーションを開発する

この節では，XML Security - Core を使用して XML 暗号アプリケーションを開発するときの流れを説明します。XML 暗号アプリケーションの機能は，データの暗号化とデータの復号化との二つに大別できます。XML Security - Core で暗号化または復号化できるデータの種類を次に示します。

- XML データの要素またはコンテンツ
- バイナリデータ
- 鍵データ

この節では，暗号化機能と復号化機能の開発の流れを，データの種類ごとに説明します。また，XML Security - Core を使用すると，鍵合意によって共通鍵を作成できます。この節では，共通鍵を作成する機能の開発の流れについても説明します。

なお，この節で説明するコーディング例は，次に格納されているサンプルプログラムに記載されています。必要に応じて参照してください。

サンプルプログラムの格納場所（Windows の場合だけ）

```
Application Serverのインストール先ディレクトリ/XMLSEC/samples/xsec
```

4.2.1　XML データの要素またはコンテンツを暗号化する

この項では，XML データの要素またはコンテンツを暗号化するアプリケーションの開発の流れを，コーディング例を示して説明します。

(1)　XML データの Document オブジェクトの生成

JAXP の DocumentBuilder を使用して暗号化したい XML 文書を読み込み，Document オブジェクトを取得します。Document オブジェクトを取得するときに，名前空間が有効になるように設定してください。コーディングの例を次に示します。

```
        DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        Document doc = dbf.newDocumentBuilder().parse(input);
```

(2)　暗号アルゴリズムの指定

XMLEncryptionFactory クラスを使用して XMLEncryption オブジェクトを生成し，暗号化に必要な暗号アルゴリズムを指定します。ここでは，暗号アルゴリズムに TripleDES を指定する場合のコーディングの例を示します。

```
        XMLEncryptionFactory xef =
                XMLEncryptionFactory.newInstance();
        EncryptionMethod em = xef.newEncryptionMethod(
                EncryptionMethod.URI_TRIPLEDES, null);
        XMLEncryption xenc = xef.newXMLEncryption(em, null);
```

(3)　暗号データの生成

暗号データを生成するときは，コンテキストおよび暗号化対象を指定してから，暗号データを生成します。ここでは，次の三つの処理について，コーディング例を示して説明します。

- コンテキストの設定
- 暗号化対象の指定
- 暗号データの生成

(a)　コンテキストの設定

データの暗号化に必要なコンテキストを設定します。暗号化に使用するデータ暗号化鍵で鍵リゾルバを生成し，コンテキストに鍵リゾルバを設定します。コンテキストの処理モードは，「暗号化」を指定してください。コーディングの例を次に示します。

```
        XMLSecurityContext context = new XMLSecurityContext(
                XMLSecurityContext.Mode.ENCRYPT, doc);
        context.setKeyResolver(
                new AdhocKeyResolver(Utilities.getSecretKey()));
```

(b)　暗号化対象の指定

暗号化の対象とする個所を指定します。CreditCard 要素を指定して暗号化する場合のコーディングの例を次に示します。

```
        Element elem = Utilities.getElement(
                doc, ”http://example.org/payment”,”CreditCard”);
        if (elem == null) {
                throw new Exception(”missing p:CreditCard”);
        }
        DOMFragment toBeEncrypted = new DOMFragment(elem);
```

なお，CreditCard 要素のコンテンツを指定して暗号化したい場合は，DOMFragment toBeEncrypted = の部分を次のようにコーディングします。

```
          DOMFragment toBeEncrypted =
                new DOMFragment(elem.getFirstChild(),
                  elem.getLastChild());
```

(c)　暗号データの生成

「(b)　暗号化対象の指定」で指定した個所の暗号値を計算します。コーディングの例を次に示します。

```
        DataContainer encrypted =
                xenc.encryptXML(context, toBeEncrypted);
```

(4)　暗号構文の構築

XMLEncryptionFactory クラスを使用して，EncryptedData 要素以下の構文に対応するオブジェクトを生成します。この場合，CipherValue 要素を使用して「(3)(c)　暗号データの生成」で計算した暗号値を設定します。次に，EncryptedData クラスの replace メソッドを使用して，「(3)(b)　暗号化対象の指定」で指定した部分を EncryptedData 要素に置き換えます。

暗号化するデータが要素なのかコンテンツなのかは，EncryptedData 要素の Type 属性で指定しておきます。また，暗号値を CipherValue ではなく別の場所に格納する場合は，CipherReference 要素を使用して，暗号値を参照する必要があります。

ここでは，CipherValue 要素を使用して，要素を暗号化する場合のコーディングの例を示します。

```
        CipherData cd = xef.newCipherData(xef.newCipherValue(
                encrypted.getAsByteArray()));
        EncryptedData ed = xef.newEncryptedData(em, null, cd, null);
        ed.setType(EncryptedData.URI_TYPE_ELEMENT);
        ed.replace(context, toBeEncrypted);
```

(5)　暗号データの出力

XMLSerializer クラスを使用して，暗号化したい部分を EncryptedData 要素に置き換えた Document オブジェクトを XML 形式で出力します。冗長な名前空間を省略し，Shift_JIS で出力する場合のコーディングの例を次に示します。

```
        XMLOutputFormat format = new XMLOutputFormat();
        format.setOmitRedundantNamespaceDecls(true);
        format.setEncoding("Shift_JIS");
        OutputStream os = new BufferedOutputStream(
                new FileOutputStream(output));
        XMLSerializer xsr = new XMLSerializer(os, format);
        xsr.serialize(doc);
        os.close();
```

4.2.2　XML データの要素またはコンテンツを復号化する

この項では，XML データの要素またはコンテンツを復号化するアプリケーションの開発の流れを，コーディング例を示して説明します。

(1)　XML データの Document オブジェクトの取得

JAXP の DocumentBuilder を使用して暗号化されたデータを含む XML 文書を読み込み，Document オブジェクトを取得します。Document オブジェクトを取得するときに，名前空間が有効になるように設定してください。コーディングの例を次に示します。

```
        DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        Document doc = dbf.newDocumentBuilder().parse(input);
```

(2)　暗号データの取得

Document オブジェクトから EncryptedData 要素を取得し，EncryptedData オブジェクトを生成します。コーディングの例を次に示します。

```
        Element encelem =
                Utilities.getElement(doc, XMLEncryption.XMLNS,
                        "EncryptedData");
        if (encelem == null) {
            throw new Exception("missing xenc:EncryptedData");
        }
```

(3)　暗号構文の構築

XMLEncryptionFactory クラスを使用して，EncryptedData 要素以下の構文に対応するオブジェクトを生成します。なお，オブジェクトを生成する前に，復号化に必要なコンテキストを設定しておく必要があります。コンテキストの処理モードは，「復号化」を指定してください。

オブジェクトを生成したら，復号化に必要な暗号アルゴリズムを指定します。このとき，復号化に使用する鍵を取得し，コンテキストに設定します。

コーディングの例を次に示します。

```
        XMLEncryptionFactory xef =
                XMLEncryptionFactory.newInstance();
        XMLSecurityContext context = new XMLSecurityContext(
                XMLSecurityContext.Mode.DECRYPT, doc);
        EncryptedData ed = xef.newEncryptedData(context, encelem);
        XMLEncryption xenc =
                xef.newXMLEncryption(
                        ed.getEncryptionMethod(),ed.getKeyInfo());
        context.setKeyResolver(
                new AdhocKeyResolver(Utilities.getSecretKey()));
```

(4) XML データの復号化

XMLEncryption クラスの decryptXML メソッドを使用して，暗号データを復号化します。次に，EncryptedData クラスの replace メソッドを使用して EncryptedData 要素を復号化したデータに置き換えます。コーディングの例を次に示します。

```
        DOMFragment decrypted = xenc.decryptXML(context, ed);
        ed.replace(context, decrypted);
```

(5) 復号化結果の出力

XMLSerializer クラスを使用して，EncryptedData 要素を復号したデータに置き換えた Document オブジェクトを XML 形式で出力します。冗長な名前空間を省略し，Shift_JIS で出力する場合のコーディングの例を次に示します。

```
        XMLOutputFormat format = new XMLOutputFormat();
        format.setOmitRedundantNamespaceDecls(true);
        format.setEncoding("Shift_JIS");
        OutputStream os = new BufferedOutputStream(
                new FileOutputStream(output));
        XMLSerializer xsr = new XMLSerializer(os, format);
        xsr.serialize(doc);
        os.close();
```

4.2.3 バイナリデータを暗号化する

この項では，バイナリデータを暗号化するアプリケーションの開発の流れを，コーディング例を示して説明します。

(1) バイナリデータの Document オブジェクトの生成

JAXP の DocumentBuilder を使用して空の Document オブジェクトを生成します。ここで生成する Document オブジェクトは，EncryptedData 要素を文書要素とする暗号文書を作成するために必要となります。Document オブジェクトを生成するときには，名前空間が有効になるように設定してください。コーディングの例を次に示します。

```
        DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        Document doc = dbf.newDocumentBuilder().newDocument();
```

(2)　暗号アルゴリズムの指定

XMLEncryptionFactory クラスを使用して XMLEncryption オブジェクトを生成し，暗号化に必要な暗号アルゴリズムを指定します。ここでは，暗号アルゴリズムに TripleDES を指定する場合のコーディングの例を示します。

```
        XMLEncryptionFactory xef =
                XMLEncryptionFactory.newInstance();
        EncryptionMethod em = xef.newEncryptionMethod(
                EncryptionMethod.URI_TRIPLEDES, null);
        XMLEncryption xenc = xef.newXMLEncryption(em, null);
```

(3)　暗号データの生成

暗号データを生成するときは，コンテキストおよび暗号化対象のバイナリデータを指定してから，暗号データを生成します。ここでは，次の三つの処理について，コーディング例を示して説明します。

- コンテキストの設定
- 暗号化対象の指定
- 暗号データの生成

(a)　コンテキストの設定

データの暗号化に必要なコンテキストを設定します。暗号化に使用するデータ暗号化鍵で鍵リゾルバを生成し，コンテキストに鍵リゾルバを設定します。コンテキストの処理モードは，「暗号化」を指定してください。コーディングの例を次に示します。

```
        XMLSecurityContext context = new XMLSecurityContext(
                XMLSecurityContext.Mode.ENCRYPT, doc);
        context.setKeyResolver(
                new AdhocKeyResolver(Utilities.getSecretKey()));
```

(b)　暗号化対象の指定

暗号化の対象とするバイナリデータを指定します。コーディングの例を次に示します。

```
        DataContainer toBeEncrypted = new DataContainer(
                new BufferedInputStream(new FileInputStream(input)));
```

(c)　暗号データの生成

「(b)　暗号化対象の指定」で指定したデータの暗号値を計算します。コーディングの例を次に示します。

```
        DataContainer encrypted =
                xenc.encrypt(context, toBeEncrypted);
```

(4)　暗号構文の構築

XMLEncryptionFactory クラスを使用して，EncryptedData 要素以下の構文に対応するオブジェクトを生成します。この場合，CipherValue 要素を使用して「(3)(c)　暗号データの生成」で計算した暗号値を設定します。次に，EncryptedData クラスの generate メソッドを使用して，EncryptedData 要素を生成します。生成した EncryptedData 要素は，Document オブジェクトに挿入します。コーディングの例を示します。

```
CipherData cd = xef.newCipherData(xef.newCipherValue(
        encrypted.getAsByteArray()));
EncryptedData ed = xef.newEncryptedData(em, null, cd, null);
Element encelem = ed.generate(context);
doc.appendChild(encelem);
```

(5)　暗号データの出力

XMLSerializer クラスを使用して，暗号化したいバイナリデータを EncryptedData 要素として挿入した Document オブジェクトを XML 形式で出力します。冗長な名前空間を省略し，Shift_JIS で出力する場合のコーディングの例を次に示します。

```
XMLOutputFormat format = new XMLOutputFormat();
format.setOmitRedundantNamespaceDecls(true);
format.setEncoding("Shift_JIS");
OutputStream os =
        new BufferedOutputStream(
                new FileOutputStream(output));
XMLSerializer xsr = new XMLSerializer(os, format);
xsr.serialize(doc);
os.close();
```

4.2.4　バイナリデータを復号化する

この項では，バイナリデータを復号化するアプリケーションの開発の流れを，コーディング例を示して説明します。

(1)　バイナリデータの Document オブジェクトの取得

JAXP の DocumentBuilder を使用して暗号化されたバイナリデータを含む XML 文書を読み込み，Document オブジェクトを取得します。Document オブジェクトを取得するときに，名前空間が有効になるように設定してください。コーディングの例を次に示します。

```
DocumentBuilderFactory dbf =
        DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
Document doc = dbf.newDocumentBuilder().parse(input);
```

(2)　暗号データの取得

Document オブジェクトから EncryptedData 要素を取得し，EncryptedData オブジェクトを生成します。Document 要素が EncryptedData 要素の場合のコーディングの例を次に示します。

```
Element encelem = doc.getDocumentElement();
if (!(DOMUtils.matchesName(encelem, XMLEncryption.XMLNS,
        "EncryptedData"))) {
    throw new Exception("missing xenc:EncryptedData");
```

(3)　暗号構文の構築

XMLEncryptionFactory クラスを使用して，EncryptedData 要素以下の構文に対応するオブジェクトを生成します。なお，オブジェクトを生成する前に，復号化に必要なコンテキストを設定しておく必要があります。コンテキストの処理モードは，「復号化」を指定してください。

オブジェクトを生成したら，復号化に必要な暗号アルゴリズムを指定します。このとき，復号化に使用する鍵を取得し，コンテキストに設定します。

コーディングの例を次に示します。

```
XMLEncryptionFactory xef =
        XMLEncryptionFactory.newInstance();
XMLSecurityContext context = new XMLSecurityContext(
        XMLSecurityContext.Mode.DECRYPT, doc);
EncryptedData ed = xef.newEncryptedData(context, encelem);
XMLEncryption xenc =
        xef.newXMLEncryption(ed.getEncryptionMethod(),
                ed.getKeyInfo());
context.setKeyResolver(
        new AdhocKeyResolver(Utilities.getSecretKey()));
```

(4)　バイナリデータの復号化

XMLEncryption クラスの decrypt メソッドを使用して，暗号化したバイナリデータを復号化します。コーディングの例を次に示します。

```
DataContainer decrypted = xenc.decrypt(context, ed);
```

(5)　復号化結果の出力

「(4)　バイナリデータの復号化」で生成した DataContainer オブジェクトから復号化結果をオクテットストリームとして取得し，ファイルに出力します。コーディングの例を次に示します。

```
OutputStream os = new FileOutputStream(output);
Utilities.copy(decrypted.getAsOctetStream(), os);
os.close();
```

4.2.5　鍵データを暗号化する

この項では，鍵データを暗号化するアプリケーションの開発の流れを，コーディング例を示して説明します。

(1)　鍵データの Document オブジェクトの生成

JAXP の DocumentBuilder を使用して空の Document オブジェクトを生成します。ここで生成する Document オブジェクトは，EncryptedKey 要素を文書要素とする暗号文書を作成するために必要となります。Document オブジェクトを生成するときには，名前空間が有効になるように設定してください。
コーディングの例を次に示します。

```
        DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        Document doc = dbf.newDocumentBuilder().newDocument();
```

(2)　暗号アルゴリズムの指定

XMLEncryptionFactory クラスを使用して XMLEncryption インタフェースを生成し，暗号化に必要な暗号アルゴリズムを指定します。ここでは，暗号アルゴリズムに TripleDES 鍵ラッピングを指定する場合のコーディングの例を示します。

```
        XMLEncryptionFactory xef =
                XMLEncryptionFactory.newInstance();
        EncryptionMethod em = xef.newEncryptionMethod(
                EncryptionMethod.URI_KEYWRAP_TRIPLEDES, null);
        XMLEncryption xenc = xef.newXMLEncryption(em, null);
```

(3)　暗号データの生成

暗号データを生成するときは，コンテキストおよび暗号化対象の鍵データを指定してから，暗号データを生成します。ここでは，次の三つの処理について，コーディング例を示して説明します。

- コンテキストの設定
- 暗号化対象の指定
- 暗号データの生成

(a)　コンテキストの設定

鍵データの暗号化に必要なコンテキストを設定します。暗号化に使用する鍵暗号化鍵で鍵リゾルバを生成し，コンテキストに鍵リゾルバを設定します。コンテキストの処理モードは，「暗号化」を指定してください。コーディングの例を次に示します。

```
        XMLSecurityContext context = new XMLSecurityContext(
                XMLSecurityContext.Mode.ENCRYPT, doc);
```

```
        context.setKeyResolver(
                new AdhocKeyResolver(Utilities.getSecretKey()));
```

(b) 暗号化対象の指定

暗号化の対象とする鍵データを生成して，指定します。TripleDES 鍵を暗号化の対象とする場合のコーディングの例を次に示します。

```
        SecretKey wk = Utilities.generateSecretKey(”DESede”, 168);
        System.out.println(”wrapped key=”
                + Utilities.toHexString(wk.getEncoded()));
        DataContainer toBeEncrypted =
                new DataContainer(wk.getEncoded());
```

(c) 暗号データの生成

「(b) 暗号化対象の指定」で指定した鍵データの暗号値を計算します。コーディングの例を次に示します。

```
        DataContainer encrypted =
                xenc.encrypt(context, toBeEncrypted);
```

(4) 暗号構文の構築

XMLEncryptionFactory クラスを使用して，EncryptedKey 要素以下の構文に対応するオブジェクトを生成します。この場合，CipherValue 要素を使用して「(3)(C) 暗号データの生成」で計算した暗号値を設定します。次に，EncryptedData クラスの generate メソッドを使用して，EncryptedKey 要素を生成します。生成した EncryptedKey 要素は，Document オブジェクトに挿入します。コーディングの例を示します。

```
        CipherData cd = xef.newCipherData(xef.newCipherValue(
                encrypted.getAsByteArray()));
        EncryptedKey ek =
                xef.newEncryptedKey(em, null, cd, null, null, null);
        Element encelem = ek.generate(context);
        doc.appendChild(encelem);
```

(5) 暗号化した鍵の出力

XMLSerializer クラスを使用して，暗号化したい鍵データを EncryptedKey 要素として挿入した Document オブジェクトを XML 形式で出力します。冗長な名前空間を省略し，Shift_JIS で出力する場合のコーディングの例を次に示します。

```
        XMLOutputFormat format = new XMLOutputFormat();
        format.setOmitRedundantNamespaceDecls(true);
        format.setEncoding(”Shift_JIS”);
        OutputStream os = new BufferedOutputStream(
                new FileOutputStream(output));
        XMLSerializer xsr = new XMLSerializer(os, format);
```

```
        xsr.serialize(doc);
        os.close();
```

4.2.6 鍵データを復号化する

この項では，鍵データを復号化するアプリケーションの開発の流れを，コーディング例を示して説明します。

(1) 鍵データの Document オブジェクトの取得

JAXP の DocumentBuilder を使用して暗号化された鍵データを含む XML 文書を読み込み，Document オブジェクトを取得します。Document オブジェクトを取得するときに，名前空間が有効になるように設定してください。コーディングの例を次に示します。

```
        DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        Document doc = dbf.newDocumentBuilder().parse(input);
```

(2) 暗号データの取得

Document オブジェクトから EncryptedKey 要素を取得し，EncryptedKey オブジェクトを生成します。Document 要素が EncryptedKey 要素の場合のコーディングの例を次に示します。

```
        Element encelem = doc.getDocumentElement();
        if (!(DOMUtils.matchesName(encelem, XMLEncryption.XMLNS,
                "EncryptedKey"))) {
            throw new Exception("missing xenc:EncryptedKey");
```

(3) 暗号構文の構築

XMLEncryptionFactory クラスを使用して，EncryptedKey 要素以下の構文に対応するオブジェクトを生成します。なお，オブジェクトを生成する前に，復号化に必要なコンテキストを設定しておく必要があります。コンテキストの処理モードは，「復号化」を指定してください。

オブジェクトを生成したら，復号化に必要な暗号アルゴリズムを指定します。このとき，復号化に使用する鍵を取得し，コンテキストに設定します。

コーディングの例を次に示します。

```
        XMLEncryptionFactory xef =
                XMLEncryptionFactory.newInstance();
        XMLSecurityContext context = new XMLSecurityContext(
                XMLSecurityContext.Mode.DECRYPT, doc);
        EncryptedKey ek = xef.newEncryptedKey(context, encelem);
        XMLEncryption xenc =
                xef.newXMLEncryption(ek.getEncryptionMethod(),
```

```
                ek.getKeyInfo());
        context.setKeyResolver(
                new AdhocKeyResolver(Utilities.getSecretKey()));
```

(4)　鍵データの復号化

XMLEncryption クラスの decrypt メソッドを使用して，暗号化した鍵データを復号化します。コーディングの例を次に示します。

```
        DataContainer decrypted = xenc.decrypt(context, ek);
```

(5)　復号化結果の取得

「(4)　鍵データの復号化」で生成した DataContainer オブジェクトから復号化結果をバイト配列として取得し，鍵オブジェクトを生成します。コーディングの例を次に示します。

```
        byte[] bytes = decrypted.getAsByteArray();
        SecretKey wk = new SecretKeySpec(bytes, "DESede");
        System.out.println("wrapped key=" + Utilities.toHexString(
                wk.getEncoded()));
```

4.2.7　鍵合意で作成した鍵を利用してデータを暗号化する

XML データの要素やコンテンツ，またはバイナリデータを暗号化する場合に使用する鍵や，鍵そのものを暗号化する場合に使用する鍵は，鍵合意で生成できます。鍵合意で鍵を生成するには，AgreementMethod クラスを使用します。この項では，Diffie-Hellman 鍵合意アルゴリズムで生成した共通鍵を利用してデータを暗号化するアプリケーションの開発の流れを，コーディング例を示して説明します。

(1)　Document オブジェクトの生成

JAXP の DocumentBuilder を使用して暗号化したい XML 文書を読み込み，Document オブジェクトを取得します。Document オブジェクトを取得するときに，名前空間が有効になるように設定してください。コーディングの例を次に示します。

```
        DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        Document doc = dbf.newDocumentBuilder().parse(input);
```

(2)　暗号アルゴリズムの指定

XMLEncryptionFactory クラスを使用して XMLEncryption オブジェクトを生成し，暗号化に必要な暗号アルゴリズムを指定します。ここでは，暗号アルゴリズムに TripleDES を指定する場合のコーディングの例を示します。

```
        XMLEncryptionFactory xef =
                XMLEncryptionFactory.newInstance();
        EncryptionMethod em = xef.newEncryptionMethod(
                EncryptionMethod.URI_TRIPLEDES, null);
        XMLEncryption xenc = xef.newXMLEncryption(em, null);
```

(3)　鍵合意による共通鍵の生成

XML Security - Core では，Diffie-Hellman 鍵合意アルゴリズムを利用して暗号化に必要な共通鍵を生成します。ここでは，次の三つの処理について，コーディング例を示して説明します。

- AgreementMethod オブジェクトの生成
- 鍵合意のコンテキストの設定
- 共通鍵の作成

> **参考**
>
> ここでは，KA-Nonce 要素，OriginatorKeyInfo 要素，または RecipientKeyInfo 要素は指定しない場合の例を説明しますが，必要に応じてこれらの要素を指定することもできます。

(a)　AgreementMethod オブジェクトの生成

Diffie-Hellman 鍵合意アルゴリズムで鍵を生成するには，まず DHKeyAgreementParams オブジェクトを生成し，生成した DHKeyAgreementParams オブジェクトを引数にして KeyInfoFactory クラスで AgreementMethod オブジェクトを生成します。コーディングの例を次に示します。

```
        KeyInfoFactory kif = xef.getKeyInfoFactory();
        DHKeyAgreementParams kap = new DHKeyAgreementParams(null,
               xef.newDigestMethod(DigestMethod.URI_SHA1, null),
                        null, null);
        AgreementMethod am =
                kif.newAgreementMethod(AgreementMethod.URI_DH,kap);
```

(b)　鍵合意のコンテキストの設定

鍵合意に必要な情報を DHKeyAgreementContext に設定します。鍵合意によって作成される共通鍵を使用するアルゴリズム，送信者の秘密鍵，および受信者の公開鍵を鍵合意のコンテキストとして設定する必要があります。コーディングの例を次に示します。

```
          DHKeyAgreementContext kac = new DHKeyAgreementContext(em,
                  Utilities.getAliceDHPrivateKey(),
                  Utilities.getBobDHPublicKey());
```

(c)　共通鍵の作成

「(b)　鍵合意のコンテキストの設定」で設定した内容に従って，共通鍵を生成します。コーディングの例を次に示します。

```
        SecretKey key = am.generateSecretKey(kac);
        System.out.println("agreed key=" + Utilities.toHexString(
                key.getEncoded()));
```

(4)　暗号データの生成

暗号データを生成する前に，コンテキストおよび暗号化対象を指定します。ここでは，次の三つの処理について，コーディング例を示して説明します。

- コンテキストの設定
- 暗号化対象の指定
- 暗号データの生成

(a)　コンテキストの設定

データの暗号化に必要なコンテキストを設定します。鍵リゾルバを使用して，鍵合意で生成した共通鍵をコンテキストに設定します。コンテキストの処理モードは，「暗号化」を指定してください。コーディングの例を次に示します。

```
        XMLSecurityContext context = new XMLSecurityContext(
                XMLSecurityContext.Mode.ENCRYPT, doc);
        context.setKeyResolver(new AdhocKeyResolver(key));
```

(b)　暗号化対象の指定

暗号化の対象とする個所を指定します。CreditCard 要素を指定して暗号化する場合のコーディングの例を次に示します。

```
        Element elem = Utilities.getElement(
                doc, "http://example.org/payment","CreditCard");
        if (elem == null) {
            throw new Exception("missing p:CreditCard");
        }
        DOMFragment toBeEncrypted = new DOMFragment(elem);
```

(c)　暗号データの生成

「(b)　暗号化対象の指定」で指定した個所の暗号値を計算します。コーディングの例を次に示します。

```
        DataContainer encrypted =
                xenc.encryptXML(context, toBeEncrypted);
```

(5)　暗号構文の構築

XMLEncryptionFactory クラスを使用して，EncryptedData 要素以下の構文に対応するオブジェクトを生成します。この場合，KeyInfo 要素には AgreementMethod クラスを指定します。また，CipherValue 要素を使用して「(4)(c)　暗号データの生成」で計算した暗号値を設定します。最後に，EncryptedData クラスの replace メソッドを使用して，「(4)(b)　暗号化対象の指定」で指定した部分を EncryptedData 要素に置き換えます。コーディングの例を次に示します。

```
CipherData cd = xef.newCipherData(xef.newCipherValue(
        encrypted.getAsByteArray()));
KeyInfo ki = kif.newKeyInfo(Collections.singletonList(am));
EncryptedData ed = xef.newEncryptedData(em, ki, cd, null);
ed.setType(EncryptedData.URI_TYPE_ELEMENT);
ed.replace(context, toBeEncrypted);
```

(6)　暗号データの出力

XMLSerializer クラスを使用して，暗号化したい部分を EncryptedData 要素に置き換えた Document オブジェクトを XML 形式で出力します。冗長な名前空間を省略し，Shift_JIS で出力する場合のコーディングの例を次に示します。

```
XMLOutputFormat format = new XMLOutputFormat();
format.setOmitRedundantNamespaceDecls(true);
format.setEncoding("Shift_JIS");
OutputStream os = new BufferedOutputStream(
        new FileOutputStream(output));
XMLSerializer xsr = new XMLSerializer(os, format);
xsr.serialize(doc);
os.close();
```

4.2.8　鍵合意で作成した鍵を利用してデータを復号化する

暗号化された XML データの要素やコンテンツ，バイナリデータ，または鍵を復号化する場合に使用する鍵は，鍵合意で生成できます。鍵合意で鍵を生成するには，AgreementMethod クラスを使用します。この項では，Diffie-Hellman 鍵合意アルゴリズムで生成した共通鍵を利用してデータを復号化するアプリケーションの開発の流れを，コーディング例を示して説明します。

(1)　Document オブジェクトの取得

JAXP の DocumentBuilder を使用して暗号化されたデータを含む XML 文書を読み込み，Document オブジェクトを取得します。Document オブジェクトを取得するときに，名前空間が有効になるように設定してください。コーディングの例を次に示します。

```
DocumentBuilderFactory dbf =
        DocumentBuilderFactory.newInstance();
```

```
        dbf.setNamespaceAware(true);
        Document doc = dbf.newDocumentBuilder().parse(input);
```

(2) 暗号データの取得

Document オブジェクトから EncryptedData 要素を取得し，EncryptedData オブジェクトを生成します。コーディングの例を次に示します。

```
        Element encelem = Utilities.getElement(
                doc, XMLEncryption.XMLNS,"EncryptedData");
        if (encelem == null) {
            System.err.println("missing xenc:EncryptedData");
        }
```

(3) 暗号構文の構築

XMLEncryptionFactory クラスを使用して，EncryptedData 要素以下の構文に対応するオブジェクトを生成します。コンテキストの処理モードは，「復号化」を指定してください。オブジェクトを生成したら，復号化に必要な暗号アルゴリズムを指定します。このとき，復号化に使用する鍵も取得します。コーディングの例を次に示します。

```
        XMLEncryptionFactory xef =
                XMLEncryptionFactory.newInstance();
        XMLSecurityContext context = new XMLSecurityContext(
                XMLSecurityContext.Mode.DECRYPT, doc);
        EncryptedData ed = xef.newEncryptedData(context, encelem);
        EncryptionMethod em = ed.getEncryptionMethod();
        KeyInfo ki = ed.getKeyInfo();
        XMLEncryption xenc = xef.newXMLEncryption(em, ki);
```

(4) 鍵合意による共通鍵の生成

XML Security - Core では，Diffie-Hellman 鍵合意アルゴリズムを利用して，復号化に必要な共通鍵を生成します。ここでは，次の三つの処理について，コーディング例を示して説明します。

- AgreementMethod オブジェクトの取得
- 鍵合意のコンテキストの設定
- 共通鍵の作成

(a) AgreementMethod オブジェクトの取得

Diffie-Hellman 鍵合意アルゴリズムで生成した鍵を使用して暗号化されたデータを復号化するには，まず KeyInfo 要素から AgreementMethod オブジェクトを取得します。コーディングの例を次に示します。

```
        AgreementMethod am = (AgreementMethod) ki.getSingleContent(
                XMLEncryption.XMLNS, "AgreementMethod");
        if (am == null) {
```

```
        throw new Exception("missing xenc:AgreementMethod");
    }
```

(b) 鍵合意のコンテキストの設定

鍵合意に必要な情報を DHKeyAgreementContext に設定します。コンテキストの設定では，鍵合意によって作成される共通鍵を使用するアルゴリズム，受信者の秘密鍵，および送信者の公開鍵が必要です。コーディングの例を次に示します。

```
    DHKeyAgreementContext kac = new DHKeyAgreementContext(em,
            Utilities.getBobDHPrivateKey(),
            Utilities.getAliceDHPublicKey());
```

参考

受信者の秘密鍵および送信者の公開鍵は，RecipientKeyInfo 要素または OriginatorKeyInfo 要素から取得することもできます。

(c) 共通鍵の作成

「(b) 鍵合意のコンテキストの設定」で設定した内容に従って，共通鍵を生成します。コーディングの例を次に示します。

```
    SecretKey key = am.generateSecretKey(kac);
    System.out.println("agreed key=" + Utilities.toHexString(
            key.getEncoded()));
```

(5) データの復号化

データを復号化する前に，復号化に必要な鍵をコンテキストに設定しておきます。次に，XMLEncryption クラスの decryptXML メソッドを使用して，暗号データを復号化します。最後に，EncryptedData クラスの replace メソッドを使用して EncryptedData 要素を復号化したデータに置き換えます。コーディングの例を次に示します。

```
    context.setKeyResolver(new AdhocKeyResolver(key));
    DOMFragment decrypted = xenc.decryptXML(context, ed);
    ed.replace(context, decrypted);
```

(6) 復号化結果の出力

XMLSerializer クラスを使用して，EncryptedData 要素を復号したデータに置き換えた Document オブジェクトを XML 形式で出力します。冗長な名前空間を省略し，Shift_JIS で出力する場合のコーディングの例を次に示します。

```
    XMLOutputFormat format = new XMLOutputFormat();
    format.setOmitRedundantNamespaceDecls(true);
```

```
        format.setEncoding("Shift_JIS");
        OutputStream os = new BufferedOutputStream(
                new FileOutputStream(output));
        XMLSerializer xsr = new XMLSerializer(os, format);
        xsr.serialize(doc);
        os.close();
```

4.3 URI 参照解決および ID 解決のカスタマイズ

同じ XML 文書内の別の要素や外部ファイルを参照したり，ID 値に対応する要素を取得したりしたい場合に，URI 参照解決および ID 解決をユーザーごとにカスタマイズできます。この節では，URI 参照解決および ID 参照解決のカスタマイズについて説明します。

4.3.1 URI 参照解決

XML 署名構文の Reference 要素などにある URI 属性には，署名対象となるデータの所在を記述することで，そのデータを参照できるようになります。**URI 参照解決**とは，この URI に基づいてデータを取得することを指しています。

URI 参照には，同一文書内での参照と外部リソースへの参照の 2 種類があります。同一文書内での URI 参照では，同じ XML 文書内の別の要素を参照します。外部リソースへの URI 参照では，署名文書または暗号文書とは別の，外部にあるファイルを参照します。XML Security - Core がデフォルトでサポートしている URI 参照解決と URI 属性値の関係を次の表に示します。

表 4-1　デフォルトでサポートしている URI 参照解決の一覧

種別	URI 属性値	URI 参照解決
同一文書内	""(空文字列)	URI 属性を含む XML 文書全体。コメントは含まない。
	"#*ID*"	URI 属性を含む XML 文書の ID 属性値が *ID* である要素以下のサブツリー。コメントは含まない。
	"#xpointer(/)"	URI 属性を含む XML 文書全体。コメントを含む。
	"#xpointer(id('*ID*'))"	URI 属性を含む XML 文書の ID 属性値が ID である要素以下のサブツリー。コメントを含む。
外部リソース	"http:"で始まる絶対 URI	指定された URI を HTTP スキームによって開いた結果。
	"file:"で始まる絶対 URI	指定された URI でローカルファイルを開いた結果。
	相対 URI	コンテキストとして与えるベース URI を用いて絶対 URI に変換して参照解決を実行する。

XML Security - Core は，表 4-1 の URI 参照解決を提供していますが，次のような場合は，URI 参照解決をカスタマイズする必要があります。

- FTP など，異なるスキームを使用する場合
- ネットワーク上のデータをローカルにキャッシュして使用する場合
- XPointer 仕様のフルセットや拡張機能など，特殊な URI 参照を使用する場合
- アプリケーション固有の解決方法を使用する場合

URI 参照解決のカスタマイズの詳細については，次に格納されている API リファレンスの ResourceResolver インタフェースを参照してください。

API リファレンスの格納場所（Windows の場合だけ）

Application Server のインストール先ディレクトリ/XMLSEC/docs/xsecapi

4.3.2 ID 解決

URI 参照の中には，URI="#ID"などのように XML 要素の ID を含むものがあります。通常，こうした XML 要素の ID は，DTD や XMLSchema などのスキーマを使用して XML プロセッサが識別し，DOM の Document.getElementById メソッドによって，ID 値に対応する要素を取得できる仕組みになっています。このような仕組みを **ID 解決**と呼びます。

しかし，アプリケーションによっては，実行時に DTD や XMLSchema などのスキーマを利用できないこともあり，その場合には ID 値に対応する要素を取得できません。ID 解決に失敗すると，URI 参照解決にも失敗するので，XML 署名データの生成および検証ができなくなってしまいます。

XML Security - Core では，開発者がアプリケーションに適した ID 解決を実装できるような機構を提供しています。ID 参照解決のカスタマイズの詳細については，次に格納されている API リファレンスの IdResolver インタフェースを参照してください。

API リファレンスの格納場所（Windows の場合だけ）

Application Server のインストール先ディレクトリ/XMLSEC/docs/xsecapi

4.4　アプリケーションを開発するときの注意事項

XML Security - Core を使用してアプリケーションを開発する前に知っておいていただきたい注意事項について説明します。注意事項には，XML Security - Core の仕様に関するものと，セキュリティに関するものとの二つがあります。

4.4.1　XML Security - Core の仕様に関する注意事項

(1)　DOM の名前空間サポート

XML Security - Core は，内部処理で DOM の名前空間がサポートされている必要があります。XML 文書を JAXP の DocumentBuilder クラスで処理する場合は，DocumentBuilderFactory クラスの isNamespaceAware メソッドの戻り値が true となる設定で作成した DocumentBuilder クラスを使用してください。また，名前空間がサポートされている DOM の要素や属性の作成は，Document クラスの createElementNS メソッドや createAttributeNS メソッドを使用します。

注意事項

デフォルトの設定では，DocumentBuilderFactory クラスの isNamespaceAware メソッドの戻り値は false になります。XML 文書を JAXP の DocumentBuilder クラスで処理する場合は，デフォルトの設定を変更してください。

(2)　同一文書内への参照

URI 参照解決のカスタマイズでは，同一文書内への参照を解決できません。XML Security - Core がデフォルトでサポートしている URI 参照解決を使用してください。XML Security - Core がデフォルトでサポートしている URI 参照解決については，「4.3.1　URI 参照解決」を参照してください。

(3)　Shift_JIS エンコードの XML 文書の処理制約

XML Security - Core は，内部的な XML 文書の処理では，XML Processor の Shift_JIS 切り替え機能を利用できません。そのため，次の場合は encoding="Shift_JIS"の指定に対して Windows-31J コンバータを使用できません。

- URI 参照によって取得する外部の XML 文書を処理する場合
- バイナリ形式の XML 文書を XMLCanonicalizer クラスで処理する場合

なお，Shift_JIS 切り替え機能の詳細については，マニュアル「XML Processor ユーザーズガイド」を参照してください。

(4)　Transform アルゴリズムを指定する順序

Enveloped Signature 変換や XPath 関数 here( )など，Transform 要素の DOM ツリー上の位置を必要とする変換の前では，変換結果がオクテットシーケンスとなる変換を実行しないでください。Transform 要素の位置情報が失われてしまうため，変換は失敗します。

(5)　外部エンティティの処理

XML Security - Core は，内部的な XML 文書の処理では，外部ファイルを参照するエンティティ参照や外部 DTD を含む XML 文書をサポートしていません。そのため，次の場合は XML 文書を処理できません。

- URI 参照によって取得する外部の XML 文書が，外部エンティティ参照や外部 DTD を含む場合
- 外部エンティティ参照や外部 DTD を含む XML 文書を XMLCanonicalizer クラスで処理する場合

(6)　エンティティ参照の制約

XML Security - Core で処理する DOM ツリーのエンティティ参照は，展開されている必要があります。XML 文書を JAXP の DocumentBuilder クラスで処理する場合は，DocumentBuilderFactory クラスの isExpandEntityReferences メソッドの戻り値が true となる設定（デフォルト設定）で作成した DocumentBuilder クラスを使用してください。

なお，XML データの要素またはコンテンツを暗号化した，EncryptedData を復号化する場合，decryptXML メソッドの内部では DOM サブツリーの復元処理が実行されます。このとき，復号化された XML データにエンティティ参照が含まれていると，エラーが発生します。ただし，定義済みのエンティティ（amp，lt，gt，apos，quot）の場合は，エラーは発生しません。

4.4.2　セキュリティに関する注意事項

(1)　XML 署名構文に記述する情報

XML 署名では，XML 署名構文の中に記述される情報を基に，セキュリティを確保します。XML 署名アプリケーションは，XML 署名構文に記述された情報の内容が正しいことを確認する必要があります。そのため，XML 署名アプリケーションを開発する場合，署名対象と指定するアルゴリズムを意識する必要があります。署名対象と指定するアルゴリズムについての注意事項を説明します。

(a)　署名対象

署名対象は，Reference 要素の URI 属性と変換処理に基づいて決定します。XML 署名アプリケーションを開発する場合，署名対象としたいデータを正しく取得できるように Reference 要素の URI 属性と変換処理を指定してください。署名対象に関する注意事項を次に示します。

- Reference 要素に指定されている情報についての注意事項

検証側が意図しない個所に署名が付与されている場合，検証が成功しても，データの完全性が保証されないことがあります。署名を検証するときは，指定された署名対象が，検証に使用する XML 署名アプリケーションにとって有効な指定であることを必ず確認してください。

例えば，明細書データのクレジットカード番号の部分に付与された署名を検証する場合は，クレジットカード番号が署名対象となっていることを確認する必要があります。クレジットカード番号以外の部分が署名対象となっていた場合，クレジットカード番号が改ざんされていたとしても，署名検証によって改ざんを検知することはできません。

- 署名対象データをキャッシュする場合の注意事項

  XML 署名アプリケーションは，署名対象とするデータをキャッシュする場合がありますが，XML 署名エンジンは，署名対象データからダイジェスト値を計算するだけです。キャッシュされた署名対象データが最新の署名対象データと同じかどうかは確認しないので，注意してください。

- 変換処理時に失われる情報についての注意事項

  XML 署名の署名対象となるデータは，変換処理の結果から得られるデータだけです。変換処理で失われた情報は保証されないので，注意してください。例えば，コメントを除く正規化の変換処理を指定した場合，コメントの情報は保証されません。

- XML 署名データがほかのデータを参照する場合の注意事項

  XML 署名は参照先のデータまでは保証しないため，署名対象データの中で，ほかのデータを参照する場合は注意する必要があります。参照先のデータを XML 署名によって保証したいときは，参照先のデータも明示的に署名対象とする必要があります。例えば，HTML データで<img>タグによって参照している画像データを保証したい場合は，HTML データだけでなく画像データも署名対象とします。

(b) アルゴリズム

XML 署名を検証する場合，検証する XML 署名文書内で指定されているアルゴリズムの有効性を確認する必要があります。署名アルゴリズムや正規化アルゴリズムなどが，検証に使用する XML 署名アプリケーションにとって有効な指定であることを確認してください。

(2) セキュリティモデルの選択

XML 署名仕様および XML 暗号仕様では，公開鍵署名などの公開鍵ベースのセキュリティや，鍵ハッシュ認証コードまたは共通鍵暗号などの共通鍵ベースのセキュリティなど，さまざまなセキュリティモデルを適用できます。ただし，XML 署名仕様および XML 暗号仕様ではセキュリティモデルの信頼性や利便性については言及していません。そのため，XML 署名アプリケーションまたは XML 暗号アプリケーションを開発する場合は，アプリケーションの用途に応じて適切なセキュリティモデルを選択する必要があります。

(3) セキュリティの強度

XML 署名または XML 暗号のセキュリティの強度は，署名アルゴリズム，暗号アルゴリズム，ダイジェストアルゴリズム，鍵生成の強度，鍵のサイズ，鍵または証明書認証のセキュリティ，鍵配布の仕組みなどに依存します。XML 署名アプリケーションまたは XML 暗号アプリケーションを開発する場合は，開発に必要な XML 署名ライブラリの機能，および認証基盤となる PKI や暗号エンジンのライブラリの信頼性を考慮して，開発に使用するライブラリを選択するようにしてください。

なお，操作手順やシステムの運用に関係するユーザーの完全性，またはシステム管理を実行する上での強制力なども，システム全体のセキュリティ強度に影響します。システムを設計する場合は，これらの要素も考慮してください。

(4) XML 署名と XML 暗号を併用する場合

XML 署名と XML 暗号を併用する場合は，アプリケーションを開発するときに次のことを考慮する必要があります。

(a) XML 署名と XML 暗号の順序性

XML 署名と XML 暗号を併用する場合，受信者は送信者がデータに署名を付与してから暗号化したのか，データを暗号化してから署名を付与したのかを知る必要があります。受信者は，送信者の処理と逆の順序で署名の検証およびデータの復号化を実行する必要があるためです。例えば，送信者が署名を付与してからデータを暗号化していた場合，受信者はまずデータを復号化してから署名を検証します。受信者が送信者と同じ順序で署名の検証およびデータの復号化を実行してしまった場合，署名の検証に失敗します。

(b) XML 署名によって保証されるデータ

XML 署名が保証するのは，その署名が付与されている個所のデータだけです。暗号化されたデータに XML 署名が含まれていても，そのデータ全体の完全性または送信者の本人性が保証されるわけではありません。

(c) 暗号化の対象

データを暗号化する場合，より高度なセキュリティを確保するためには，そのデータのダイジェストまたは署名も暗号化する必要があります。ダイジェストまたは署名を暗号化しないで平文のままにしておくと，悪意のある第三者から攻撃されやすくなるおそれがあります。

(5) 共通鍵を 3 人以上で共有する場合

3 人以上で共有している共通鍵は，その共通鍵を共有しているすべてのメンバに対して送信するデータを暗号化するときにだけ使用します。その共通鍵を共有しているメンバのうち，一部のメンバに対してだけデータを送信する場合は，使用しないでください。一部のメンバに対してだけデータを送信した場合でも，共通鍵を共有するほかのメンバがデータを盗聴すると，データを復号できてしまうため，情報が漏洩するおそれがあります。

(6) 暗号データの改ざんへの対策

一般的な暗号技術では，同じ鍵を使用して同じデータを暗号化した場合に暗号化結果が同じにならないように，IV とよばれるランダムな値を暗号化対象とともに暗号化することがあります。IV は，暗号化されたデータとともに送信されます。暗号化したデータに IV を付加することで，送信中に第三者によってデータが解読されるおそれは少なくなりますが，IV 自体または暗号化されたデータの改ざんは防止できません。IV や暗号データが改ざんされると，復号化結果も異なってしまいます。IV または暗号データの改ざんを防止するためには，平文に対して署名するなどの対策が必要です。

(7) DoS 攻撃への対策

XML 署名または XML 暗号の相互参照などを悪用して無限再帰となる処理を要求したり，容量の大きいデータや常にリダイレクトを必要とするデータへの参照が必要となるデータを送信したりするなどの DoS 攻撃への対策として，アプリケーションを開発する場合は次の機能を組み込んでおく必要があります。

- 再帰処理を制限する機能
- 1 回の処理で使用できるリソースの容量を制限する機能

(8) 暗号データの安全性

XML 暗号によって暗号化したデータは，コンピュータウィルスなどの有害なデータであっても，ファイアウォールやウィルスを検知するソフトウェアなどで検出できない場合があります。そのため，暗号化されたデータの安全性は保証されていないという前提で，ファイアウォールやウィルスを検知するソフトウェアには次の対処のどれかを実施することをお勧めします。

- 暗号データを拒否する。
- 安全なデータかどうかを確認するため，復号化したデータへのアクセスを要求する。
- ローカルファイルへのアクセスを制限するなどして，データを受信したアプリケーションが任意のコンテンツを処理してもシステム全体のセキュリティを確保する。

付録

付録 A　トラブルシュート

XML Security - Core でアプリケーションを実行してトラブルが発生した場合，トラブルの原因を究明するために，製品のサポートサービスの障害窓口にトレース情報を提出する場合があります。トレースには，メソッドの呼び出しなど，トラブルの発生個所を特定するための情報が出力されます。

デフォルトでは，トレース情報は出力しないよう設定されています。そのため，トラブルが発生した場合は，トレース情報を出力するように設定をしてから再度処理を実行して，トラブルを再現してトレース情報を収集します。

ここでは，トレース情報を出力するための設定方法，およびトレースの収集方法について説明します。なお，ここで説明するトレース情報は，アプリケーションのデバッグには利用できないので，注意してください。

参考

XML Security - Core のトレースは，HNTRLib の共通形式（HNTR 形式）で出力されるため，Manager のログエージェント機能を利用して，一括して管理できます。この一括管理を利用するには，ログエージェント設定ファイルの設定プロパティで，HNTR 形式ログ収集プラグインを設定する必要があります。

ログエージェント設定ファイルの設定方法については，マニュアル「アプリケーションサーバ システム構築・運用ガイド」を参照してください。ログエージェント設定ファイルの形式や設定プロパティなどの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

付録 A.1　トレースの設定項目

トレース情報を出力するためには，システムプロパティまたは設定ファイルでトレースの情報を設定する必要があります。システムプロパティと設定ファイルとの両方でトレースの情報を設定した場合は，システムプロパティの設定が優先されます。ここでは，トレースに設定できる内容，および設定値について説明します。

(1)　設定内容

トレースに設定できるトレースの情報は，次の五つです。

- 出力レベル
  トレースファイルに出力する情報のレベルを指定します。出力レベルには，次に示す 6 段階のレベルがあります。推奨する出力レベルは，WARN です。
  1. OFF
     トレースは出力されません。

2. ERROR

致命的なエラーを出力します。

3. WARN

警告レベルまでのエラーを出力します。

4. CAUTION

主要な処理情報を出力します。

5. INFO

CAUTION レベルよりも詳細な処理情報を出力します。

6. DEBUG

INFO レベルよりも詳細な処理情報を出力します。

- 出力先

トレースファイルを出力するディレクトリを絶対パスで指定します。推奨する出力先を次に示します。

推奨するトレースファイルの出力先（Windows の場合）

Application Server インストール先ディレクトリ/XMLSEC/logs

推奨するトレースファイルの出力先（UNIX の場合）

/opt/Cosminexus/XMLSEC/logs

- 出力先ファイル名のプレフィックス

トレースファイル名のプレフィックスを指定します。プレフィックスと面番号の組み合わせたものが，トレースファイル名となります。プレフィックスが「csmxsec_trace」で面番号が「3」の場合のトレースファイル名の例を次に示します。

csmxsec_trace3.log

- 出力サイズ

トレースファイル 1 面当たりのファイルサイズをバイト単位で指定します。4,096～2,147,483,647 の間の値を指定してください。

- 出力面数

トレースファイルを何面で循環させるかを指定します。1～16 の間の値を指定してください。

注意事項

CAUTION レベル，INFO レベル，または DEBUG レベルでトレースを出力した場合，トレースに個人情報が出力されることがあります。CAUTION レベル，INFO レベル，または DEBUG レベルは，障害が発生した場合にトレースを採取するときだけ使用します。

出力レベルを CAUTION，INFO，または DEBUG に設定すると，多くの情報が出力されるため，性能が低下するおそれがあります。トレース採取後は，必ず出力レベルを OFF，ERROR，または WARN に変更してください。

(2) 設定値

トレースファイルの設定項目とデフォルト値を次の表に示します。なお，システムプロパティや設定ファイルに記述するときのプロパティ名は，「com.cosminexus.xml.security.logging.*設定名*」とします。

表 A–1 トレースの設定項目に設定する値

設定項目	設定名	デフォルト値
出力レベル	trace_level	OFF
出力先	trace_dir	${user.dir}※1
出力先ファイル名のプレフィックス	trace_fileprefix	csmxsec_trace※2
出力サイズ	trace_filesize	1048576
出力面数	trace_filenum	4

注※1

システムプロパティで設定されるカレントディレクトリです。

注※2

同じプロセスの中，または同時に実行される可能性があるプロセスの間で同じ名称を使用した場合，動作は保証されません。

> **注意事項**
>
> XML Security - Core を使用してアプリケーションを実行する場合は，トレースファイルおよびトレースファイルが出力されるディレクトリに対するアクセス権が必要です。トレースファイルまたはトレースファイルが出力されるディレクトリに対してアクセス権がない場合，エラーが発生してアプリケーションを実行できません。

付録 A.2 トレースの設定方法

トレースを収集するためのシステムプロパティの設定および設定ファイルの設定方法について説明します。システムプロパティにも設定ファイルにも設定されていない設定項目については，「表 A-1 トレースの設定項目に設定する値」に示したデフォルト値に従ってトレースファイルが出力されます。

(1) システムプロパティの設定

システムプロパティの設定は，java コマンドの-D オプションを利用します。出力レベルを「WARN」とする場合の設定例を次に示します。

システムプロパティの設定例（-D オプションを利用する場合）

```
java -Dcom.cosminexus.xml.security.logging.trace_level=WARN YourApplicationClass
```

J2EE サーバまたは Web コンテナサーバの Component Container で XML Security - Core を使用する場合は，Component Container のユーザー定義ファイル（usrconf.properties）にシステムプロパティを記述します。ユーザー定義ファイルは java.util.Properties 形式で記述する必要があります。そのため，パスに「¥」記号や空白文字が含まれる場合は，「¥¥」としたり，空白の前に「¥」記号を追加したりして，エスケープする必要があります。また，java.util.Properties 形式の場合は日本語を直接指定できないので，ディレクトリ名などで日本語を使用している場合は，注意してください。ユーザー定義ファイルにシステムプロパティを記述する場合の例を次に示します。

システムプロパティの設定例（ユーザー定義ファイルに記述する場合）

```
com.cosminexus.xml.security.logging.trace_level=WARN
com.cosminexus.xml.security.logging.trace_dir=C:¥¥Program¥ Files¥¥Hitachi¥¥Cosminexus¥¥XM
LSEC¥¥logs
com.cosminexus.xml.security.logging.trace_fileprefix=MyEJBServerA_trace
```

Component Container のユーザー定義ファイルの設定方法については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

(2) 設定ファイルの設定

設定ファイルは，java.util.Properties 形式のファイルで作成します。設定項目については，「表 A-1 トレースの設定項目に設定する値」を参照してください。設定ファイルは，次の順序で検索されます。

1. システムプロパティ com.cosminexus.xml.security.logging.properties の設定値
2. ${user.dir}（＝システムプロパティに設定されるカレントディレクトリ）の com.cosminexus.xml.security.logging.properties ファイル
3. ${user.home}（＝システムプロパティに設定されるユーザーのホームディレクトリ）の com.cosminexus.xml.security.logging.properties ファイル

注意事項

- 設定ファイルに設定した内容よりも，システムプロパティの設定内容が優先されます。
- システムプロパティや設定ファイルの設定が間違っていた場合，エラーメッセージが表示されます。この場合，アプリケーションは起動しません。
- 設定ファイルの読み込みが失敗した場合，エラーメッセージが表示されます。この場合，アプリケーションは起動しません。

付録 B　標準仕様への対応

W3C の XML 署名仕様および XML 暗号仕様のうち，XML Security - Core がサポートする仕様について説明します。

XML Security - Core がサポートする XML 署名仕様（2002/2/12 勧告）を次の表に示します。なお，XML 署名仕様（2002/2/12 勧告）の詳細については，W3C のホームページを参照してください。

表 B-1　XML 署名標準仕様のサポート範囲一覧

区分	項目	推奨レベル	サポートの有無
署名形式	Enveloped 署名	必須	○
	Enveloping 署名	必須	○
	Detached 署名	必須	○
アルゴリズム	SHA1	必須	○
	base64	必須	○
	HMAC-SHA1	必須	○
	DSAwithSHA1	必須	○
	RSAwithSHA1	推奨	○
	Canonical XML（コメント付き）	推奨	○
	Canonical XML（コメントなし）	必須	○
	Exclusive CanonicalXML（コメント付き）	任意	○
	Exclusive CanonicalXML（コメントなし）	任意	○
	XSLT	任意	×
	XPath	推奨	○
	XPath filter 2.0	任意	○
その他	SignatureValue　生成／検証	必須	○
	Manifest DigestValue　生成／検証	任意	○
	laxly schema valid な Signature 要素の生成	必須	○
	XPointer 式　'#xpointer(/)'	推奨	○
	XPointer 式　'#xpointer(id("ID"))'	推奨	○
	XPath 関数　here( )	推奨	○
	RetrievalMethod	推奨	○

（凡例）

○：サポートしている

×：サポートしていない

XML Security - Core がサポートする XML 暗号仕様（2002/12/10 勧告）を次の表に示します。なお，XML 暗号仕様（2002/12/10 勧告）の詳細については，W3C のホームページを参照してください。

表 B-2　XML 暗号標準仕様のサポート範囲一覧

区分	項目	推奨レベル	サポートの有無
アルゴリズム	TRIPLEDES	必須	○
	AES-128	必須	○
	AES-192	任意	×
	AES-256	必須	×
	RSA-OAEP（AES 128bit 鍵）	必須	×
	RSA-OAEP（AES 256bit 鍵）	必須	×
	RSA-v1.5（AES/DES 192bit 鍵）	必須	×
	Diffie-Hellman 鍵合意	任意	○
	TRIPLEDES 鍵ラッピング	必須	○
	AES-128 鍵ラッピング（128bit 鍵）	必須	○
	AES-192 鍵ラッピング	任意	×
	AES-256 鍵ラッピング（256bit 鍵）	必須	×
	SHA1	必須	○
	SHA256	必須	○
	SHA512	任意	○
	RIPEMD-160	任意	×
	XML 署名	推奨	○
	XML 署名復号変換（Decryption Transform）	任意	×
	Canonical XML（コメントなし）	任意	○
	Canonical XML（コメント付き）	任意	○
	Exclusive XML Canonicalization（コメントなし）	任意	○
	Exclusive XML Canonicalization（コメント付き）	任意	○
	Base64	必須	○
その他	Element タイプおよび Content タイプのサポート	必須	○
	XML 要素および XML 要素のコンテンツの直列化	必須	○
	EncryptedData への置き換え（暗号化する場合）	推奨	○

区分	項目	推奨レベル	サポートの有無
	EncryptedData の復号結果への置き換え	推奨	○
	laxly schema valid な EncryptedData 要素および EncryptedKey 要素の生成	必須	○
	Type, MimeType, Encoding 属性	必須	○
	CipherReference の URI 解決	必須	○
	CipherReference の Transform	推奨	○
	xenc:DHKeyValue	任意	○
	ds:KeyName	推奨	○
	ds:RetrievalMethod	必須	○
	ReferenceList	任意	○
	EncryptionProperties	任意	○

（凡例）

○：サポートしている

×：サポートしていない

付録 C　用語解説

マニュアルで使用する用語について

マニュアル「アプリケーションサーバ & BPM/ESB 基盤 用語解説」を参照してください。

索引

A

AgreementMethod　27
AgreementMethod オブジェクトの取得　63
AgreementMethod オブジェクトの生成　60
AgreementMethod 要素　25
API リファレンス（Windows の場合）　22

C

CanonicalizationMethod 要素　25
CAUTION〔トレースの出力レベル〕　75
csmjaxp.jar　33
csmxsec.jar　33

D

DataReference　23
DEBUG〔トレースの出力レベル〕　75
Detached 署名　17
Detached 署名を生成する　43
DHKeyValue　27
Diffie-Hellman 鍵合意　20
DigestMethod 要素　25
DOCTYPE 宣言　28
Document オブジェクトの取得　46
　XML データを復号化する場合　51
　鍵合意で作成した鍵で復号化する場合　62
　鍵データを復号化する場合　58
　バイナリデータを復号化する場合　54
Document オブジェクトの生成
　Detached 署名の場合　43
　XML データを暗号化する場合　48
　鍵合意で作成した鍵で暗号化する場合　59
　鍵データを暗号化する場合　56
　バイナリデータを暗号化する場合　52
DOM の名前空間サポート〔注意事項〕　68
DoS 攻撃への対策〔注意事項〕　72
DSAKeyValue　27

E

EncryptedKey　27
EncryptionMethod 要素　25
Enveloped 署名　16
Enveloped 署名を生成する　39
Enveloping 署名　16
ERROR〔トレースの出力レベル〕　75

H

hntrlib2j64.jar　33

I

ID 解決　67
ID 解決手段　26
INFO〔トレースの出力レベル〕　75
IV　71

J

jar ファイル　33

K

KeyInfo 要素　26
KeyInfo 要素の下位要素　27
KeyName　27
KeyReference　23
keytool　36
　-certreq　36
　-export　36
　-genkey　36
　-import　36
KeyValue　27

O

OFF〔トレースの出力レベル〕　74

R

Reference 要素に指定されている情報〔注意事項〕　69

RetrievalMethod　27
RSAKeyValue　27

S

Shift_JIS エンコードの XML 文書の処理制約〔注意事項〕　68
SignatureMethod 要素　25

T

trace_dir　76
trace_filenum　76
trace_fileprefix　76
trace_filesize　76
trace_level　76
Transform アルゴリズムを指定する順序〔注意事項〕　69
Transform 要素　24

U

URI 参照解決　66
URI 参照解決の一覧　66
URI 参照の解決手段　26

W

WARN〔トレースの出力レベル〕　75

X

X509Certificate　27
X509CRL　27
X509Data　27
X509IssuerSerial　27
X509SKI　27
X509SubjectName　27
XML Security - Core とは　10
XML Security - Core の仕組み　11
XML 暗号アプリケーションを開発する　48
XML 暗号とは　19
XML 暗号の構文の概要　21
XML 構文の解析　24
XML 構文の生成　24
XML 署名アプリケーションを開発する　39
XML 署名構文に記述する情報〔注意事項〕　69
　アルゴリズム　70
　署名対象　69
XML 署名構文の構築
　Detached 署名の場合　44
　Enveloped 署名の場合　39
　Enveloping 署名の場合　42
　XML 署名を検証する場合　46
XML 署名データがほかのデータを参照する〔注意事項〕　70
XML 署名データの検証　46
XML 署名データの生成
　Detached 署名の場合　44, 45
　Enveloped 署名の場合　40, 41
　Enveloping 署名の場合　42, 43
XML 署名と XML 暗号の順序性〔注意事項〕　71
XML 署名と XML 暗号を併用する　71
XML 署名とは　15
XML 署名によって保証されるデータ〔注意事項〕　71
XML 署名の構文の概要　17
XML 署名の種類　15
XML 署名文書の Document オブジェクトの生成
　Enveloped 署名の場合　39
　Enveloping 署名の場合　41
XML 署名文書の出力
　Detached 署名の場合　45
　Enveloped 署名の場合　41
　Enveloping 署名の場合　43
XML 署名を検証する　45
XML 署名を生成する位置の設定
　Detached 署名の場合　45
　Enveloped 署名の場合　40
　Enveloping 署名の場合　43
XML 宣言　28
XML データの復号化　52
XML データの要素またはコンテンツを復号化する　50
XML 名前空間プレフィックス　26
XML 文書の出力処理　27

あ

アルゴリズム一覧　24
暗号アルゴリズムの指定
　XML データを暗号化する場合　48
　鍵合意で作成した鍵で暗号化する場合　59
　鍵データを暗号化する場合　56
　バイナリデータを暗号化する場合　53
暗号アルゴリズム要素　21
暗号化値要素　21
暗号鍵取得アルゴリズム要素　21
暗号鍵要素　21
暗号化参照要素　22
暗号化した鍵の出力　57
暗号化対象の指定
　XML データを暗号化する場合　49
　鍵合意で作成した鍵で暗号化する場合　61
　鍵データを暗号化する場合　57
　バイナリデータを暗号化する場合　53
暗号化データ要素　21
暗号化できるデータの種類　19
暗号化の対象〔注意事項〕　71
暗号化モード　26
暗号構文の構築
　XML データを暗号化する場合　50
　XML データを復号化する場合　51
　鍵合意で作成した鍵で暗号化する場合　62
　鍵合意で作成した鍵で復号化する場合　63
　鍵データを暗号化する場合　57
　鍵データを復号化する場合　58
　バイナリデータを暗号化する場合　54
　バイナリデータを復号化する場合　55
暗号データの安全性　72
暗号データの改ざんへの対策〔注意事項〕　71
暗号データの出力
　XML データを暗号化する場合　50
　鍵合意で作成した鍵で暗号化する場合　62
　バイナリデータを暗号化する場合　54
暗号データの取得
　XML データを復号化する場合　51
　鍵合意で作成した鍵で復号化する場合　63
　鍵データを復号化する場合　58
　バイナリデータを復号化する場合　55
暗号データの生成
　XML データを暗号化する場合　49
　鍵合意で作成した鍵で暗号化する場合　61
　鍵データを暗号化する場合　56, 57
　バイナリデータを暗号化する場合　53
暗号データ要素　21
暗号プロパティ要素　22

え

エンコーディング　27
エンティティ参照の制約〔注意事項〕　69

お

オブジェクト要素　18

か

改行文字　28
外部エンティティの処理〔注意事項〕　69
鍵合意　20
鍵合意で作成した鍵を利用してデータを暗号化する 59
鍵合意で作成した鍵を利用してデータを復号化する 62
鍵合意による共通鍵の生成　60
鍵合意のコンテキストの設定
　暗号化の場合　60
　復号化の場合　64
鍵合意要素　21
鍵情報の取得　26
鍵情報の設定　26
鍵情報要素　18, 21
鍵データの復号化　59
鍵データを暗号化する　56
鍵データを復号化する　58
鍵の解決手段　26
鍵名称要素　21
鍵を設定する　36
格納場所

API リファレンス（Windows の場合） 22
XML 暗号アプリケーションのサンプルプログラム（Windows の場合） 48
XML 署名アプリケーションのサンプルプログラム（Windows の場合だけ） 39
カスタマイズ 66
空タグの出力 28
環境設定 32

き

擬似乱数生成器 26
共通鍵の作成
暗号化の場合 61
復号化の場合 64
共通鍵の生成 63
共通鍵を 3 人以上で共有する場合〔注意事項〕 71
共通の機能 24

く

クラスパスに設定する jar ファイル一覧 33
クラスパスを設定する 33
クラスローダによる jar ファイルの読み込み 37

こ

コーディングの例
AgreementMethod オブジェクトの生成（送信時） 60
AgreementMethod の取得（受信時） 63
Document オブジェクトの生成（鍵合意で作成した鍵で暗号化する場合） 59
Document オブジェクトの取得 46
Document オブジェクトの取得（XML データを復号化する場合） 51
Document オブジェクトの取得（鍵データを復号化する場合） 58
Document オブジェクトの生成（Detached 署名の場合） 43
Document オブジェクトの生成（Enveloped 署名を生成する場合） 39
Document オブジェクトの生成（Enveloping 署名の場合） 41
Document オブジェクトの生成（XML データを暗号化する場合） 48
Document オブジェクトの生成（鍵合意で作成した鍵で復号化する場合） 62
Document オブジェクトの生成（鍵データを暗号化する場合） 56
Document オブジェクトの生成（バイナリデータを暗号化する場合） 52
Document オブジェクトの生成（バイナリデータを復号化する場合） 54
XML 署名構文の構築（Detached 署名の場合） 44
XML 署名構文の構築（Enveloped 署名を生成する場合） 40
XML 署名構文の構築（Enveloping 署名の場合） 42
XML 署名構文の構築（XML 署名を検証する場合） 46
XML 署名データの検証 47
XML 署名データの生成（Detached 署名の場合） 45
XML 署名データの生成（Enveloped 署名の場合） 41
XML 署名データの生成（Enveloping 署名の場合） 43
XML 署名文書の出力（Detached 署名の場合） 45
XML 署名文書の出力（Enveloped 署名の場合） 41
XML 署名文書の出力（Enveloping 署名の場合） 43
XML 署名を生成する位置の設定（Detached 署名の場合） 45
XML 署名を生成する位置の設定（Enveloped 署名の場合） 40
XML 署名を生成する位置の設定（Enveloping 署名の場合） 43
XML データの復号化 52
暗号アルゴリズムの指定（XML データを暗号化する場合） 48
暗号アルゴリズムの指定（鍵合意で作成した鍵で暗号化する場合） 59
暗号アルゴリズムの指定（鍵データを暗号化する場合） 56
暗号アルゴリズムの指定（バイナリデータを暗号化する場合） 53
暗号化した鍵の出力 57

暗号化対象の指定（XML データを暗号化する場合） 49
暗号化対象の指定（鍵合意で作成した鍵で暗号化する場合） 61
暗号化対象の指定（鍵データを暗号化する場合） 57
暗号化対象の指定（バイナリデータを暗号化する場合） 53
暗号構文の構築（XML データを暗号化する場合） 50
暗号構文の構築（XML データを復号化する場合） 51
暗号構文の構築（鍵合意で作成した鍵で暗号化する場合） 62
暗号構文の構築（鍵合意で作成した鍵で復号化する場合） 63
暗号構文の構築（鍵データを暗号化する場合） 57
暗号構文の構築（鍵データを復号化する場合） 58
暗号構文の構築（バイナリデータを暗号化する場合） 54
暗号構文の構築（バイナリデータを復号化する場合） 55
暗号データの出力（XML データを暗号化する場合） 50
暗号データの出力（鍵合意で作成した鍵で暗号化する場合） 62
暗号データの出力（バイナリデータを暗号化する場合） 54
暗号データの取得（XML データを復号化する場合） 51
暗号データの取得（鍵合意で作成した鍵で復号化する場合） 63
暗号データの取得（鍵データを復号化する場合） 58
暗号データの取得（バイナリデータを復号化する場合） 55
暗号データの生成（XML データを暗号化する場合） 50
暗号データの生成（鍵データを暗号化する場合） 57
暗号データの生成（バイナリデータを暗号化する場合） 53
鍵合意で作成した鍵で暗号化する場合 61
鍵データの復号化 59
共通鍵の作成（送信時） 61
コンテキストの設定（Detached 署名の場合） 45
コンテキストの設定（Enveloped 署名の場合） 40
コンテキストの設定（Enveloping 署名の場合） 42
コンテキストの設定（XML 署名を検証する場合） 47
コンテキストの設定（XML データを暗号化する場合） 49
コンテキストの設定（鍵合意で作成した鍵で暗号化する場合） 61
コンテキストの設定（鍵合意の送信時） 60
コンテキストの設定（鍵データを暗号化する場合） 56
コンテキストの設定（バイナリデータを暗号化する場合） 53
バイナリデータの復号化 55
復号化結果の出力（XML データを復号化する場合） 52
復号化結果の出力（バイナリデータを復号化する場合） 55
復号化結果の取得（鍵データを復号化する場合） 59
コメントの省略 28
コンテキストの設定 25
Detached 署名の場合 45
Enveloped 署名の場合 40
Enveloping 署名の場合 42
XML 署名を検証する場合 47
XML データを暗号化する場合 49
鍵合意で作成した鍵で暗号化する場合 61
鍵データを暗号化する場合 56
バイナリデータを暗号化する場合 53

さ

サポートするアルゴリズム 24
サポート範囲一覧
XML 暗号標準仕様 79
XML 署名標準仕様 78
参照要素 18
サンプルプログラム 39, 48

し

自己署名証明書 36
システムプロパティの設定 76
出力 27

出力サイズ　75
出力先　75
出力先ファイル名のプレフィックス　75
出力面数　75
出力レベル　74
冗長な名前空間宣言　28
証明書を設定する　36
署名アルゴリズム要素　18
署名検証モード　26
署名情報要素　18
署名生成モード　26
署名対象データをキャッシュする〔注意事項〕　70
署名値　24
署名値要素　18
署名要素　18
処理モード　26

せ

正規化アルゴリズム要素　18
セキュリティの強度〔注意事項〕　70
セキュリティプロバイダを設定する　34
セキュリティモデルの選択〔注意事項〕　70
設定ファイルの設定　77

た

ダイジェストアルゴリズム要素　18
ダイジェスト値　24
ダイジェスト値要素　18

ち

注意事項
　XML Security - Core の仕様について　68
　アプリケーションを開発する場合　68
　環境設定するとき　37
　セキュリティについて　69
直列化処理　22

て

データの復号化（鍵合意で作成した鍵で復号化する場合）　64
電子データの完全性を保証　15

と

同一文書内への参照〔注意事項〕　68
トレースの情報　74
　出力サイズ　76
　出力先　76
　出力先ファイル名のプレフィックス　76
　出力面数　76
　出力レベル　76
トレースの設定項目　74
トレースの設定方法　76

な

成り済まし　15
成り済ましを防止　15

に

認証局が発行する証明書　36

は

バイナリデータの復号化　55
バイナリデータを暗号化する　52
バイナリデータを復号化する　54

ひ

否認　15
否認を防止　15
秘密鍵　15
標準仕様への対応　78

ふ

復元処理　23
復号化結果の出力
　XML データを復号化する場合　52
　鍵合意で作成した鍵で復号化する場合　64

バイナリデータを復号化する場合　55
復号化結果の取得
鍵データを復号化する場合　59
復号化モード　26
文書オブジェクト　26

へ

ベース URI　26
変換処理時に失われる情報〔注意事項〕　70
変換処理要素　18

ま

マニュアルの説明　9

り

利用例　12
XML 暗号アプリケーションを利用する場合　13
XML 署名アプリケーションを利用する場合　12