BEA Tuxedo

BEA Tuxedo の
セキュリティ機能

BEA Tuxedo リリース 8.0 J
8.0 版
2001 年 10 月

BEA Tuxedo のセキュリティ機能

Document Edition	Date	Software Version
8.0J	2001 年 10 月	BEA Tuxedo リリース 8.0J

目次

このマニュアルについて

このマニュアルでは、BEA Tuxedo 製品のアプリケーション・トランザクション・モニタ・インターフェイス (ATMI) のセキュリティ機能を紹介し、この機能を使用して ATMI アプリケーションのセキュリティを確保する方法について説明します。

このマニュアルでは、以下の内容について説明します。

- 「第 1 章 ATMI のセキュリティについて」では、BEA Tuxedo 製品の ATMI のセキュリティ機能の概要を示します。
- 「第 2 章 セキュリティの管理」では、`UBBCONFIG` ファイルのパラメータを設定して ATMI アプリケーションのセキュリティを有効にする方法を説明します。
- 「第 3 章 セキュリティのプログラミング」では、セキュリティ保護された環境で ATMI アプリケーションとのインタラクションを実現するために、クライアント側で使用する ATMI 関数について説明します。

対象読者

このマニュアルは、ATMI アプリケーションのセキュリティ機能の設定を担当するアプリケーション開発者を対象にしています。また、ATMI プログラミング環境の知識があることを前提としています。

e-docs Web サイト

BEA Tuxedo 製品のマニュアルは BEA 社の Web サイト上で参照することができます。BEA ホーム・ページの [製品のドキュメント] をクリックするか、または http://edocs.beasys.co.jp/e-docs/index.html に直接アクセスしてください。

マニュアルの印刷方法

このマニュアルは、ご使用の Web ブラウザで一度に 1 ファイルずつ印刷できます。Web ブラウザの [ファイル] メニューにある [印刷] オプションを使用してください。

このマニュアルの PDF 版は、Web サイト上にあります。また、マニュアルの CD-ROM にも収められています。この PDF を Adobe Acrobat Reader で開くと、マニュアル全体または一部をブック形式で印刷できます。PDF 形式を利用するには、BEA Tuxedo の Documents ページの [PDF 版] ボタンをクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader をお持ちでない場合は、Adobe Web サイト http://www.adobe.co.jp/ から無償で入手できます。

関連情報

BEA Tuxedo、分散オブジェクト・コンピューティング、およびトランザクション処理の詳細については、BEA Tuxedo オンライン・マニュアルにある 『CORBA Bibliography』を参照してください。

サポート情報

皆様の BEA Tuxedo マニュアルに対するフィードバックをお待ちしています。ご意見やご質問がありましたら、電子メールで **docsupport-jp@bea.com** までお送りください。お寄せいただきましたご意見は、BEA Tuxedo マニュアルの作成および改訂を担当する BEA 社のスタッフが直接検討いたします。

電子メール メッセージには、8.0 リリースのマニュアルを使用していることを明記してください。

BEA Tuxedo に関するご質問、または BEA Tuxedo のインストールや使用に際して問題が発生した場合は、www.bea.com の BEA WebSUPPORT を通して BEA カスタマ・サポートにお問い合わせください。カスタマ・サポートへの問い合わせ方法は、製品パッケージに同梱されているカスタマ・サポート・カードにも記載されています。

カスタマ・サポートへお問い合わせの際には、以下の情報をご用意ください。

- お客様のお名前、電子メール・アドレス、電話番号、Fax 番号
- お客様の会社名と会社の住所
- ご使用のマシンの機種と認証コード
- ご使用の製品名とバージョン
- 問題の説明と関連するエラー・メッセージの内容

表記上の規則

このマニュアルでは、以下の表記規則が使用されています。

規則	項目
太字	用語集に定義されている用語を示します。
Ctrl + Tab	2 つ以上のキーを同時に押す操作を示します。
イタリック体	強調またはマニュアルのタイトルを示します。

等幅テキスト	コード・サンプル、コマンドとオプション、データ構造とメンバ、データ型、ディレクトリ、およびファイル名と拡張子を示します。また、キーボードから入力する文字も示します。
	例 :
	`#include <iostream.h> void main ( )`
	`the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

等幅太字	コード内の重要な単語を示します。 例 : `void **commit** ( )`

規則	項目
等幅イタリック体	コード内の変数を示します。 例 : `String *expr*`
大文字	デバイス名、環境変数、および論理演算子を示します。 例 : LPT1 SIGNON OR
{ }	構文の行で選択肢を示します。かっこは入力しません。
[ ]	構文の行で省略可能な項目を示します。かっこは入力しません。 例 : `buildobjclient [-v] [-o name ] [-f *file-list*]...[-l *file-list*]...`
|	構文の行で、相互に排他的な選択肢を分離します。記号は入力しません。
...	コマンド行で次のいずれかを意味します。 ■ コマンド行で同じ引数を繰り返し指定できること ■ 省略可能な引数が文で省略されていること ■ 追加のパラメータ、値、その他の情報を入力できること 省略符号は入力しません。 例 : `buildobjclient [-v] [-o name ] [-f *file-list*]...[-l *file-list*]...`
. . .	コード例または構文の行で、項目が省略されていることを示します。省略符号は入力しません。

1 ATMI のセキュリティについて

ここでは、次の内容について説明します。

- セキュリティとは
- セキュリティ・プラグイン
- ATMI のセキュリティ機能
- デフォルトの認証と認可
- セキュリティの相互運用性

注記 BEA Tuxedo リリース 8.0 には、ATMI (アプリケーション・トランザクション・モニタ・インターフェイス) アプリケーションと CORBA アプリケーションを作成するための 2 種類の環境が用意されています。この章では、ATMI アプリケーションに対するセキュリティ機能の実装方法を説明します。CORBA アプリケーションに対するセキュリティ機能の実装方法については、『BEA Tuxedo CORBA アプリケーションのセキュリティ機能』を参照してください。

セキュリティとは

セキュリティとは、コンピュータ内のデータまたはコンピュータ間で送受信されるデータが損なわれないことを保証する技術のことです。ほとんどのセキュリティ機能では、パスワードおよびデータの暗号化を使用してセキュリティを実現します。パスワードとは、秘密の文字列であり、ユーザは、これを入力することにより特定のプログラムやシステムにアクセスできます。データの暗号化とは、データを理解できない形式に変換することであり、変換されたデータは復号化のメカニズムがないと解読できません。

電子商取引などで使用される分散アプリケーションには、悪質なユーザがデータを傍受したり、操作を中断したり、不正な情報を入力できるアクセス・ポイントが多数あります。ビジネスがより広い範囲に分散されるほど、こうした悪質なユーザによる攻撃を受けやすくなります。したがって、このようなアプリケーションの基盤となる分散型のコンピューティング・ソフトウェア、つまりミドルウェアは、セキュリティ機能を備えている必要があります。

BEA Tuxedo 製品には、ATMI アプリケーション用のセキュリティ機能がいくつか用意されており、その大部分は、特定のニーズに合わせてカスタマイズできます。

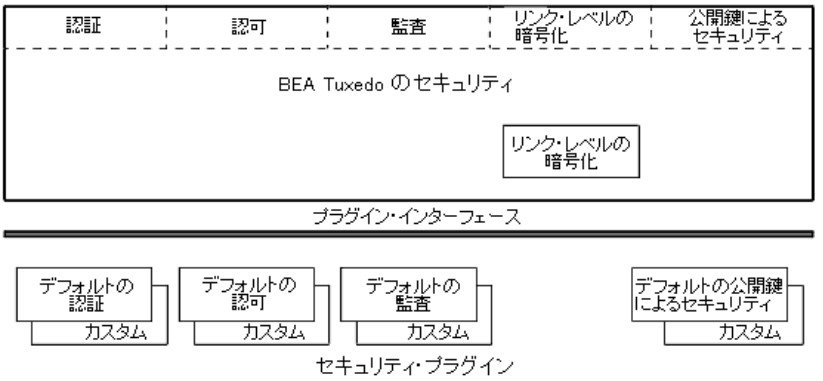
関連項目

- 第 1 章の 2 ページ「セキュリティ・プラグイン」
- 第 1 章の 3 ページ「ATMI のセキュリティ機能」
- 第 2 章の 1 ページ「セキュリティの管理とは」
- 第 3 章の 1 ページ「セキュリティのプログラミングとは」

セキュリティ・プラグイン

次の図に示すように、BEA Tuxedo 製品の ATMI 環境で利用できるセキュリティ機能は、プラグイン・インターフェイスによってインプリメントされています (ただし、例外が 1 つあります)。プラグイン・インターフェイスにより、BEA Tuxedo のユーザは、独自のセキュリティ・プラグインを定義し、動的に追加することができます。セキュリティ・プラグインは、特定のセキュリティ機能をインプリメントするコード・モジュールです。

図 1-1 BEA Tuxedo ATMI プラグイン・セキュリティのアーキテクチャ



セキュリティ・プラグインのインターフェイスの仕様は、一般には公開されていませんが、BEA 社と特別な契約を結んだサード・パーティのセキュリティ・ベンダには公開されています。セキュリティ機能をカスタマイズする場合は、これらのセキュリティ・ベンダにお問い合わせください。たとえば、公開鍵によるセキュリティ機能をカスタマイズする場合は、適切なセキュリティ・プラグインを提供するサード・パーティのセキュリティ・ベンダに問い合わせる必要があります。

セキュリティ・プラグインの詳細 (インストール手順およびコンフィギュレーション手順を含む) については、日本 BEA システムズ (株) の営業担当者にお問い合わせください。

関連項目

- 第 1 章の 3 ページ「ATMI のセキュリティ機能」

ATMI のセキュリティ機能

BEA Tuxedo システムでは、さまざまな方法でセキュリティを実現できます。たとえば、ホスト・オペレーティング・システムのセキュリティ機能を使用して、ファイル、ディレクトリ、およびシステム・リソースへのアクセスを制御することができます。次の表では、BEA Tuxedo 製品の ATMI 環境で利用できるセキュリティ機能を説明します。

表 1-1 ATMI のセキュリティ機能

セキュリティ機能	説明	プラグイン・インターフェイス	デフォルトのインプリメンテーション
オペレーティング・システムのセキュリティ	ファイル、ディレクトリ、およびシステム・リソースへのアクセスを制御します。	該当なし	該当なし

表 1-1 ATMI のセキュリティ機能 (続き)

セキュリティ機能	説明	プラグイン・インターフェイス	デフォルトのインプリメンテーション
認証	ユーザまたはシステム・プロセスに対して指定されている ID を証明し、ID 情報を安全に記録および転送し、必要に応じて ID 情報を利用可能にします。	1 つのインターフェイスとしてインプリメントされます。	デフォルトの認証プラグインには、認証なし、アプリケーション・パスワードを使用、ユーザ・レベルの認証を実行、という 3 つのセキュリティ・レベルが用意されています。このデフォルトのプラグインは、BEA Tuxedo システムで最初に実現された、BEA Tuxedo の従来の認証のインプリメンテーションと同じように機能します。
認可	ID およびその他の情報に基づいて、リソースへのアクセスを制御します。	1 つのインターフェイスとしてインプリメントされます。	デフォルトの認可プラグイン には、2 つのセキュリティ・レベル、つまり、オプションまたは必須のアクセス制御リストが用意されています。このデフォルトのプラグインは、BEA Tuxedo システムで最初に実現された、BEA Tuxedo の従来の認可のインプリメンテーションと同じように機能します。
監査	操作要求とその結果に関する情報を安全に収集、格納、および配布します。	1 つのインターフェイスとしてインプリメントされます。	デフォルトの監査機能は、BEA Tuxedo のイベント・ブローカおよびユーザ・ログ (ULOG) 機能によってインプリメントされます。

表 1-1 ATMI のセキュリティ機能 (続き)

セキュリティ機能	説明	プラグイン・インターフェイス	デフォルトのインプリメンテーション
リンク・レベルの暗号化	対称鍵による暗号化を使用して、ATMI アプリケーション内のマシン間をつなぐネットワーク・リンク上で送受信されるメッセージの秘密性を保護します。	該当なし	RC4 形式の対称鍵による暗号化
公開鍵によるセキュリティ機能	公開鍵 (非対称鍵) による暗号化を使用して、ATMI アプリケーションのクライアントとサーバ間でエンド・ツー・エンドのデジタル署名とデータの秘密性を実現します。この機能は、PKCS-7 標準に準拠しています。	6 つのインターフェイスとしてインプリメントされます。	デフォルトの公開鍵セキュリティでは、次のアルゴリズムがサポートされます。 ■ RSA (公開鍵のアルゴリズム) ■ RSA および DSA (デジタル署名のアルゴリズム) ■ DES-CBC、2 つの鍵による Triple DES、および RC2 (対称鍵のアルゴリズム) ■ MD5 および SHA-1 (メッセージ・ダイジェストのアルゴリズム)

関連項目

- 第 1 章の 6 ページ「オペレーティング・システム (OS) のセキュリティ」
- 第 1 章の 7 ページ「認証」
- 第 1 章の 13 ページ「認可」
- 第 1 章の 19 ページ「監査」
- 第 1 章の 23 ページ「リンク・レベルの暗号化」

■ 第 1 章の 29 ページ「公開鍵によるセキュリティ機能」

オペレーティング・システム (OS) のセキュリティ

ファイルに対するパーミッション設定などのセキュリティ機能を備えた、ホスト側のオペレーティング・システムでは、オペレーティング・システム・レベルのセキュリティが、第一レベルのセキュリティ機能になります。アプリケーション管理者は、ファイルに対してパーミッションを設定することにより、特定のユーザまたはユーザ・グループに対して、アクセス権を付与したり、拒否することができます。

ほとんどの場合、ATMI アプリケーションは、アプリケーション管理者によって管理されます。アプリケーション管理者は、アプリケーションをコンフィギュレーションし、起動し、実行中のアプリケーションを動的に監視し、必要に応じて変更します。ATMI アプリケーションは、管理者が起動し、実行するため、サーバ・プログラムは、管理者のパーミッションで実行されます。これで、サーバ・プログラムは安全であり、「信頼できる」と見なされます。この方法は、基盤となるオペレーティング・システムのログイン・メカニズムとパーミッション設定 (ファイル、ディレクトリ、およびシステム・リソースに対する読み取り権と書き込み権の設定) に基づいています。

クライアント・プログラムは、自分のパーミッションを持つユーザによって直接実行されます。さらに、ネイティブ・クライアント (サーバ・プログラムと同じマシンで実行中のクライアント) を実行するユーザは、`UBBCONFIG` コンフィギュレーション・ファイルにアクセスしたり、掲示板 (ATMI アプリケーションを制御するパラメータおよびアプリケーションの統計情報を格納するために確保されている共用メモリの一部) などのプロセス間通信 (IPC) のメカニズムにアクセスできます。

より高度なセキュリティ機能を備えたプラットフォーム上の ATMI アプリケーションでは、ファイルおよび IPC メカニズムに対するアクセスをアプリケーション管理者だけに限定し、管理者のパーミッション (UNIX ホスト・マシンの `setuid` コマンドまたは別のプラットフォームでの同等のコマンド) を使用して「信頼できる」クライアント・プログラムを実行すると、より安全です。最も高いレベルのオペレーティング・システムのセキュリティを実現するには、ワークステーション・クライアントだけがアプリケーションにアクセスできるようにし、アプリケーション・サーバおよび管理プログラムを実行するマシンではクライアント・プログラムを実行できないようにします。

関連項目

- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 第 2 章の 8 ページ「オペレーティング・システム (OS) のセキュリティ管理」
- 『BEA Tuxedo アプリケーションの設定』の 第 2 章の 1 ページ「コンフィギュレーション・ファイルについて」および 第 3 章の 1 ページ「コンフィギュレーション・ファイルの作成」
- 『ファイル形式、データ記述方法、MIB、およびシステム・プロセスのリファレンス』の `UBBCONFIG(5)`

認証

認証機能とは、通信するプロセスどうしがお互いの ID を証明し合うことです。BEA Tuxedo 製品の ATMI 環境での認証プラグイン・インターフェイスは、共有シークレット・パスワード、ワンタイム・パスワード、CHALLENGE 応答、Kerberos などの認証技術を使用して、セキュリティ・プロバイダによるさまざまな認証プラグインに対応できます。インターフェイスは、必要に応じて GSSAPI (Generic Security Service Application Programming Interface) の仕様に厳密に従います。GSSAPI は、公開されている IETF (Internet Engineering Task Force) の標準です。認証プラグイン・インターフェイスは、サードパーティ・ベンダのセキュリティ製品をできるだけ簡単に BEA Tuxedo システムに統合できるように設計されています。ただし、セキュリティ製品は GSSAPI を使用して記述しておく必要があります。

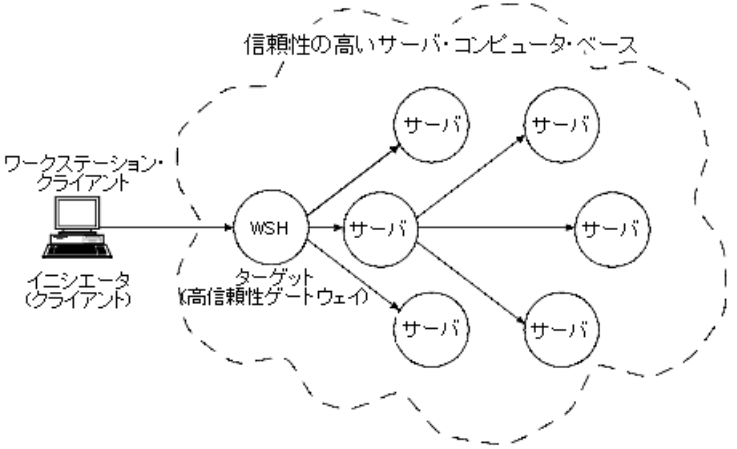
認証プラグインのアーキテクチャ

認証機能を使用するためのプラグイン・インターフェイスは、1 つのプラグインとしてインプリメントされ、デフォルトの認証プラグインを使用することも、カスタマイズした認証プラグインを使用することもできます。

委任された信用認証について

BEA Tuxedo システムなどの分散型のエンタープライズ・ミドルウェア環境で、エンド・ツー・エンドの相互認証を直接行う場合、特に、長時間の接続用に最適化されたセキュリティ・メカニズムでは、大幅にコストがかかります。クライアントから各サーバ・プロセスに対して直接ネットワーク接続を確立したり、サービス要求の処理時に複数の認証メッセージを交換および検証するのは、非効率的です。代わりに、ATMI アプリケーションは、次の図のような、高信頼性委託型認証モデルを使用しています。

図 1-2 ATMI の高信頼性委託型認証モデル



ワークステーション・クライアントは、起動時に、信頼性のあるシステム・ゲートウェイ・プロセスであるワークステーション・ハンドラ (WSH: Workstation Handler) に対して認証を行います。ネイティブ・クライアントの場合は、自身に対して認証を行います (後で説明)。認証が成功すると、認証ソフトウェアは、クライアントにセキュリティ・トークンを割り当てます。トークンとは、プロセス間の転送に適したオペークなデータ構造体です。WSH は、認証済みのワークステーション・クライアントのトークンを安全に格納します。ネイティブ・クライアントの場合は、認証済みのネイティブ・クライアント自身のトークンを安全に格納します。

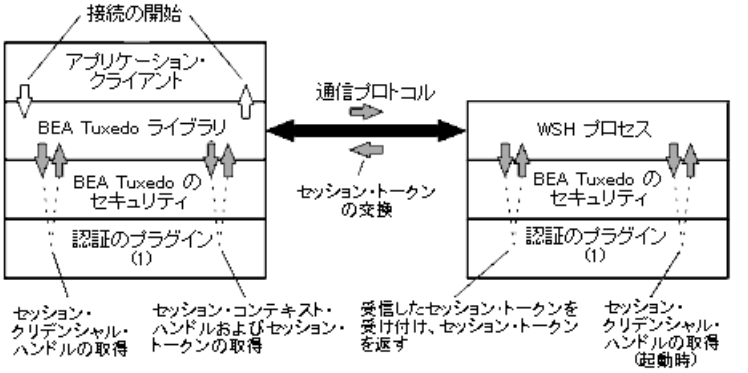
信頼性のあるゲートウェイは、通過するクライアント要求にセキュリティ・トークンを添付します。添付されたセキュリティ・トークンは、クライアント要求のメッセージと共に送信され、送信先のサーバ・プロセスでの認可と監査で使用されます。

このモデルのゲートウェイでは、認証ソフトウェアがクライアントの ID を検証し、適切なトークンを生成することが期待されています。一方、サーバ側では、ゲートウェイ・プロセスで適切なセキュリティ・トークンが添付され、そのセキュリティ・トークンが、クライアント要求を処理するその他のサーバに安全に格納されることが期待されています。

セッションの確立

次の図は、ワークステーション・クライアントと WSH との間でセッションが確立されているときの、BEA Tuxedo システムの ATMI 環境内での制御フローを示しています。ここでは、ワークステーション・クライアントと WSH が、メッセージを交換して相互に認証し合い、長期間にわたる接続を確立する様子を示しています。

図 1-3 クライアントと WSH 間の認証



まず、イニシエータ・プロセス (ミドルウェアのクライアント・プロセスなど) が、BEA Tuxedo の「セキュリティ・コンテキストの開始」関数を呼び出して、戻りコードとして成功または失敗が返されるまで繰り返し、セッション・コンテキストを作成します。セッション・コンテキストは、認証されたユーザと ID 情報を関連付けます。

ワークステーション・クライアントが、ATMI アプリケーションに参加するために C の `tpinit(3c)` または COBOL の `TPINITIALIZE(3cbl)` を呼び出すと、BEA Tuxedo システムは、まず内部の「クリデンシャル取得」関数を呼び出して、セッション・クリデンシャル・ハンドルを取得し、応答を返します。次に、内部の「セキュリティ・コンテキストの開始」関数を呼び出して、セッション・コンテキストを取得します。「セキュリティ・コンテキストの開始」関数は、呼び出し時に入力セッション・トークンを取り (セッション・トークンがある場合)、出力セッション・トークンを返します。セッション・トークンでは、ユーザの ID を確認するためのプロトコルを使用します。イニシエータ・プロセスが出力セッション・トークンをセッションのターゲット・プロセス (WSH) に渡すと、出力セッション・トークンは、別の入力トークンと交換されます。トークンの交換は、双方のプロセスが相互認証を完了するまで繰り返されます。

セキュリティ・プロバイダの認証プラグインでは、セキュリティ・インプリメンテーション用のセッション・トークンおよびセッション・コンテキストの内容が定義されています。したがって、ATMI の認証では、セッション・トークンとセッション・コンテキストをオペークなオブジェクトとして扱う必要があります。交換されるトークンの数は定義されておらず、認証システムのアーキテクチャに応じて異なります。

ネイティブ・クライアントでセッションを開始する場合、イニシエータ・プロセスとターゲット・プロセスは同じです。このプロセスは、ミドルウェア・クライアント・プロセスと見なすことができます。ミドルウェア・クライアント・プロセスは、セキュリティ・プロバイダの認証プラグインを呼び出してネイティブ・クライアントを認証します。

認可トークンおよび監査トークンの取得

認証に成功すると、信頼性のあるゲートウェイは、BEA Tuxedo の内部関数を 2 つ呼び出し、クライアント用の認可トークンおよび監査トークンを取得します。これらのトークンは、ゲートウェイによって格納されます。これらのトークンの組み合わせは、セキュリティ・コンテキストのユーザ ID を表します。セキュリティ・トークンという用語は、集合的に認可トークンと監査トークンを表します。

デフォルトの認証の認可トークンには、次の 2 つの情報が設定されています。

- プリンシパル名—認証されたユーザの名前
- アプリケーション・キー—要求を発行するクライアントを一意に識別する 32 ビット値。詳細については、第 1 章の 48 ページ「アプリケーション・キー」を参照してください。

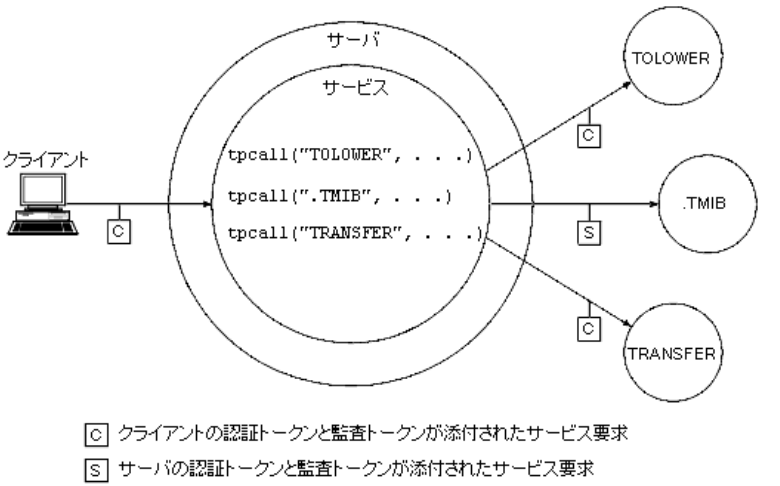
また、デフォルトの認証を使用する場合、監査トークンにもプリンシパル名とアプリケーション・キーが設定されます。

セッション・トークンと同様に、認可トークンと監査トークンもオペークであり、内容は、セキュリティ・プロバイダごとに異なります。認可トークンを使用すると、認可 (パーミッション) をチェックすることができます。監査トークンを使用すると、監査情報を記録することができます。ATMI アプリケーションによっては、認可用と監査用のユーザ ID を別々に設定すると便利です。

クライアント・トークンとサーバ・トークンの交換

クライアントのサービス要求がサーバ側で転送されるとき、サーバの ID が付く場合があります。次の図はその様子を示しています。サーバは、サービス要求に添付されたクライアント・トークンを自らのトークンに置き換えてから、そのサービス要求を適切なサービスに転送します。

図 1-4 サーバ・パーミッションのアップグレード例



注記 サーバが自らの認可トークンと監査トークンを取得する方法、およびこれらのトークンを必要とする理由については、第 2 章の 11 ページ「プリンシパル名の指定」を参照してください。

上の図に示す機能を、「サーバ・パーミッションのアップグレード」と呼びます。この機能では、ドット付きのサービス (「.TMIB」のように名前の先頭にピリオドが付いた、システムに組み込まれているサービス) をサーバが呼び出すたびに、サービス要求にサーバの ID が付き、サーバに対するアクセス権が取得されます。サーバのアクセス権とは、アプリケーション管理者 (システム管理者) のアクセス権のことです。つまり、クライアントからドット付きのサービスを直接呼び出すことはできなくても、まずサーバに要求を送信し、そのサーバからドット付きのサービスに要求を転送することはできます。ドット付きサービスの詳細については、『ファイル形式、データ記述方法、MIB、およびシステム・プロセスのリファレンス』の MIB(5) のリファレンス・ページにある .TMIB サービスの説明を参照してください。

カスタマイズされた認証のインプリメント

ATMI アプリケーションで認証機能を実行するには、デフォルトのプラグインまたはカスタマイズしたプラグインを使用します。どちらのプラグインを使用するかは、BEA Tuxedo のレジストリを設定して決めます。これは、すべてのセキュリティ・プラグインを制御するツールです。

デフォルトの認証プラグインを使用する場合、レジストリを設定する必要はありません。ただし、カスタマイズされた認証プラグインを使用する場合は、プラグインをインストールする前に、プラグインに合わせてレジストリを設定する必要があります。レジストリの詳細については、第 2 章の 3 ページ「BEA Tuxedo レジストリの設定」を参照してください。

関連項目

- 第 1 章の 43 ページ「デフォルトの認証と認可」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 第 2 章の 9 ページ「認証の管理」
- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」
- 第 3 章の 4 ページ「ATMI アプリケーションにクライアント・プログラムを参加させるためのセキュリティ・コードの記述方法」

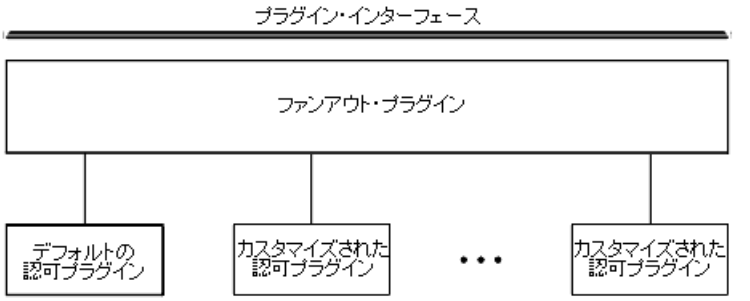
認可

認可の機能により、管理者は ATMI アプリケーションへのアクセスを制御できます。つまり、管理者は、認可の機能を使用して、ATMI アプリケーションのリソースまたは機能に対するアクセス権をプリンシパル (認証されたユーザ) に許可するかどうかを決定します。

認可プラグインのアーキテクチャ

ファンアウトとは、さまざまなプラグインのインプリメンテーションが接続された、アンブレラ (傘) 型のプラグインです。次の図に示すように、認可プラグインのインターフェイスは、ファンアウトの構成でインプリメントされます。

図 1-5 認可プラグインのアーキテクチャ



デフォルトの認可プラグインのインプリメンテーションは、ファンアウト・プラグインとデフォルトの認可プラグインで構成されます。カスタマイズされたプラグインのインプリメンテーションは、ファンアウト・プラグイン、デフォルトの認可プラグイン、および 1 つ以上のカスタム認可プラグインで構成されます。

ファンアウト・プラグイン・モデルでは、まず、呼び出し側がファンアウト・プラグインに要求を送信します。受信された要求は、下位のプラグインに渡され、各プラグインから応答が返されます。最後に、ファンアウト・プラグインは、受け取った複数の応答を 1 つの応答にまとめ (コンポジット応答)、呼び出し側に送信します。

認可要求の目的は、クライアント操作を許可するか、または操作の結果をそのまま残すかを決めることです。各認可プラグインは、*permit*、*deny*、*abstain* のいずれかの応答を返します。*abstain* 応答が返されると、認可プラグインの作成者は、元のプラグインでは対応できないこと、たとえば、プラグインのインストール後にシステムに追加された操作名などに対処できます。

次の表は、認可のファンアウト・プラグインで作成されるコンポジット応答を示しています。デフォルトの認可の場合、コンポジット応答は、デフォルトの認可プラグインによってのみ決まります。

表 1-2 認可のコンポジット応答

プラグインから返される値	コンポジット応答
すべて *permit*、または *permit* と *abstain* の組み合わせの場合	*permit*
少なくとも 1 つが *deny* の場合	*deny*
すべて *abstain* の場合	*deny* ATMI アプリケーションの UBBCONFIG ファイルの SECURITY パラメータに MANDATORY_ACL が設定されている場合 *permit* ATMI アプリケーションの UBBCONFIG ファイルの SECURITY パラメータが設定されていないか、または、このパラメータに MANDATORY_ACL 以外の値が設定されている場合

カスタマイズした認可として、銀行アプリケーションを例に取ります。この例では、ユーザを Customer グループのメンバとし、以下の条件が適用されるとします。

- デフォルトの認可プラグインでは、Customer グループのユーザであれば、誰でも特定の口座から引き出しを行えます。
- カスタマイズした認可プラグインでは、Customer グループのユーザであれば、誰でも特定の口座から引き出しを行えます。ただし、引き出しを行えるのは、月曜日から金曜日の午前 9 時から午後 5 時までです。
- カスタマイズした 2 つ目の認可プラグインでは、Customer グループのユーザであれば、誰でも特定の口座から引き出しを行えます。ただし、引き出せるのは、10,000 ドル未満の金額です。

Customer グループのユーザは、月曜日の午前 10 時に 500 ドルを引き出すことはできます。ただし、同じユーザが、土曜日の午前中に同じ操作を行おうとしてもできません。

このほかにも、さまざまな状況が考えられます。色々な状況を想定し、ビジネス上のニーズに合った最適な状況を定義してください。

認可プラグインのしくみ

認可に関する一部の決定は、認可トークンに格納されているユーザ ID に基づいて行われます。認可トークンは、認証プラグインによって生成されるため、認証プラグインと認可プラグインのプロバイダは、これらのプラグインが協調動作することを保証する必要があります。

BEA Tuxedo システムのプロセスまたはサーバ (/Q サーバの `TMQUEUE(5)` またはイベント・ブローカ・サーバの `TMUSREVT(5)` など) は、クライアントからの要求を受け取ると認可プラグインを呼び出します。これに対し、認可プラグインは、操作前のチェックを行い、操作を許可するかどうかを示す応答を返します。

- 操作が許可された場合、システムはクライアント要求を実行します。
- 操作が許可されない場合、システムはクライアント要求を実行しません。

クライアント操作が許可されると、BEA Tuxedo システムのプロセスまたはサーバは、クライアント操作が完了した後で認可プラグインを呼び出すことができます。これに対し、認可プラグインは、操作後のチェックを行い、操作の結果を許可するかどうかを示す応答を返します。

- 許可される場合、システムは操作の結果を受け付けます。
- 許可されない場合、システムは、操作の結果を変更するか、または操作をロール・バックし (元に戻し) ます。

これらの呼び出しは、アプリケーション・レベルの呼び出しではなく、システム・レベルの呼び出しです。ATMI アプリケーションは、認可プラグインを呼び出せません。

BEA Tuxedo システムに組み込まれているデフォルトの認可プラグインを使用する場合と、カスタマイズした認可プラグインを 1 つ以上使用する場合では、認可のプロセスが多少異なります。デフォルトのプラグインでは、操作後のチェックをサポートしていません。したがって、デフォルトの認可プラグインが、操作後のチェックを行う要求を受け取っても、要求は直ちに返され、何も実行されません。

カスタマイズしたプラグインでは、操作前と操作後の両方のチェックをサポートしています。

デフォルトの認可

クライアントから操作前のチェックの実行が要求され、それに対して ATMI のプロセスによってデフォルトの認可が呼び出されると、認可プラグインは、次のタスクを実行します。

1. 認証プラグインを呼び出して、クライアントの認可トークンから情報を取得します。

   認可トークンは、認証プラグインによって作成されるため、認可プラグインにはトークンの内容が記録されていません。ただし、認可プロセスではこの情報が必要です。

2. 操作前のチェックを実行します。

   認可プラグインは、クライアントの認可トークン、アクセス制御リスト (ACL: Access Control List)、および ATMI アプリケーションのセキュリティ・レベル (ACL の設定 (オプションまたは必須)) を調べて、操作を許可するかどうかを決めます。

3. 操作を実行するかどうかを決定します。

   認可のファンアウト・プラグインは、デフォルトの認可プラグインの決定 (*permit* または *deny*) を受け取ると、その決定に従って動作します。

   - クライアント操作が許可された場合、ファンアウト・プラグインは呼び出し側のプロセスに *permit* を返します。システムはクライアント要求を実行します。
   - クライアント操作が拒否された場合、ファンアウト・プラグインは呼び出し側のプロセスに *deny* を返します。システムはクライアント要求を実行しません。

カスタマイズした認可

カスタマイズした 1 つ以上の認証プラグインを使用するユーザは、BEA Tuxedo 製品の ATMI 環境で提供される別の機能を利用できます。つまり、操作の実行後にさらにチェックを行うことができます。

クライアントから操作前のチェックの実行が要求され、それに対して ATMI のプロセスによってカスタマイズした認可が呼び出されると、認可プラグインは、次のタスクを実行します。

1. 認証プラグインを呼び出して、クライアントの認可トークンから情報を取得します。

2. 操作前のチェックを実行します。

   認可プラグインは、操作、クライアントの認可トークン、および関連するデータを調べて、操作を許可するかどうかを決めます。関連するデータには、ユーザ・データおよび ATMI アプリケーションのセキュリティ・レベルが含まれます。

   認可プラグインは、認可の要件に合うよう、必要に応じてユーザ・データを変更してから操作を実行します。

3. 操作を実行するかどうかを決定します。

   認可のファンアウト・プラグインは、下位にある個々のプラグインの応答 (*permit*、*deny*、または *abstain*) をチェックして、最終的な決定を行います。

   - クライアント操作が許可された場合、ファンアウト・プラグインは呼び出し側のプロセスに *permit* を返します。システムはクライアント要求を実行します。
   - クライアント操作が拒否された場合、ファンアウト・プラグインは呼び出し側のプロセスに *deny* を返します。システムはクライアント要求を実行しません。

クライアント操作が許可されると、ATMI のプロセスは、クライアント操作が完了した後でカスタマイズした認可を呼び出すことができます。その場合、認可プラグインは次のタスクを実行します。

1. 認証プラグインを呼び出して、クライアントの認可トークンから情報を取得します。
2. 操作後のチェックを行います。

   認可プラグインは、操作、クライアントの認可トークン、および関連するデータを調べて、操作の結果を許可するかどうかを決めます。関連するデータには、ユーザ・データおよび ATMI アプリケーションのセキュリティ・レベルが含まれます。

3. 操作の結果を許可するかどうかを決定します。

   認可のファンアウト・プラグインは、下位にある個々のプラグインの応答 (*permit*、*deny*、または *abstain*) をチェックして、最終的な決定を行います。

   - 操作の結果が許可された場合、ファンアウト・プラグインは呼び出し側のプロセスに *permit* を返します。システムは操作の結果を受け付けます。
   - 操作が拒否された場合、ファンアウト・プラグインは呼び出し側のプロセスに *deny* を返します。システムは、操作の結果を変更するか、または操作をロール・バックし (元に戻し) ます。

操作後のチェック機能は、ラベル付けされた文書を扱うセキュリティ・モデルで便利です。たとえば、機密文書へのアクセスを許可されたユーザが、最高機密文書を取り出す操作を実行したとします (通常、文書を識別するラベルの内容は、その文書が取り出されるまでわかりません)。この場合、操作後のチェックにより、操作の拒否または出力データの変更 (機密情報の削除) を効率的に行うことができます。

カスタマイズした認可のインプリメント

ATMI アプリケーションで認可機能を実行するには、デフォルトのプラグインを使用するか、または 1 つ以上のプラグインをカスタマイズします。どちらのプラグインを使用するかは、BEA Tuxedo のレジストリを設定して決めます。これは、すべてのセキュリティ・プラグインを制御するツールです。

デフォルトの認可プラグインを使用する場合、レジストリを設定する必要はありません。ただし、カスタマイズした認可プラグインを使用する場合は、プラグインをインストールする前に、プラグインに合わせてレジストリを設定する必要があります。レジストリの詳細については、第 2 章の 3 ページ「BEA Tuxedo レジストリの設定」を参照してください。

関連項目

- 第 1 章の 43 ページ「デフォルトの認証と認可」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 第 2 章の 33 ページ「認可の管理」
- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」

監査

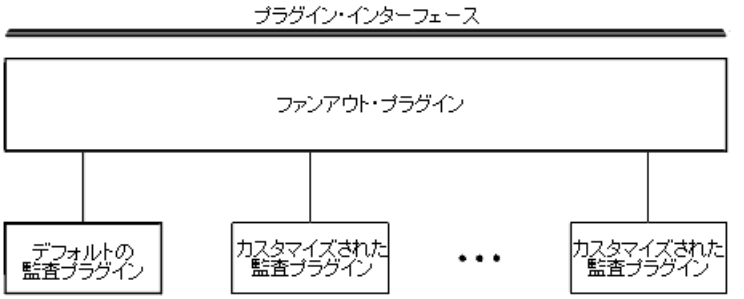
監査とは、操作要求とその結果に関する情報を収集、格納、および配布する方法です。監査証跡の記録からは、ATMI アプリケーションのセキュリティ・レベルに違反するアクションを実行したプリンシパルや、そのようなアクションを実行しようとしたプリンシパルを判別できます。また、これらの記録から、試行された操作、失敗した操作、および成功した操作を判別することもできます。

監査方法 (情報の収集、処理、保護、および配布の方法) は、監査プラグインによって異なります。

監査プラグインのアーキテクチャ

ファンアウトとは、さまざまなプラグインのインプリメンテーションが接続された、アンブレラ (傘) 型のプラグインです。次の図は、ファンアウト型でインプリメントされた、監査プラグインのインターフェイスを示しています。

図 1-6 監査プラグインのアーキテクチャ



デフォルトの監査プラグインのインプリメンテーションは、ファンアウト・プラグインとデフォルトの監査プラグインで構成されます。カスタマイズされたプラグインのインプリメンテーションは、ファンアウト・プラグイン、デフォルトの監査プラグイン、および 1 つ以上のカスタム監査プラグインで構成されます。

ファンアウト・プラグイン・モデルでは、まず、呼び出し側がファンアウト・プラグインに要求を送信します。受信された要求は、下位のプラグインに渡され、各プラグインから応答が返されます。最後に、ファンアウト・プラグインは、受け取った複数の応答を 1 つの応答にまとめ (コンポジット応答)、呼び出し側に送信します。

監査要求の目的は、イベントを記録することです。各監査プラグインは、*success* (監査は成功し、イベントが記録される) または *failure* (監査は失敗し、イベントは記録されない) のどちらかの応答を返します。監査のファンアウト・プラグインでは、下位のすべてのプラグインから応答として *success* が返されると、コンポジット応答は *success* になります。それ以外の場合、コンポジット応答は、*failure* になります。

デフォルトの監査の場合、コンポジット応答は、デフォルトの監査プラグインによってのみ決まります。カスタマイズした監査の場合、コンポジット応答は、ファンアウト・プラグインの下位プラグインの応答によって決まります。ファンアウト・プラグインの動作については、第 1 章の 13 ページ「認可プラグインのアーキテクチャ」を参照してください。

監査プラグインのしくみ

監査に関する一部の決定は、監査トークンに格納されているユーザ ID に基づいて行われます。監査トークンは、認証プラグインによって生成されるため、認証プラグインと監査プラグインのプロバイダは、これらのプラグインが協調動作することを保証する必要があります。

ATMI システムのプロセスまたはサーバ (/Q サーバの `TMQUEUE(5)` またはイベント・ブローカ・サーバの `TMUSREVT(5)` など) は、クライアントからの要求を受け取ると監査プラグインを呼び出します。監査プラグインは、操作が開始する前に呼び出されるため、操作の実行そのものを記録したり、データを格納することができます。データは、操作後の監査で使用できます。これに対し、監査プラグインは、操作前の監査を実行して、監査が成功したかどうかを返します。

ATMI システムのプロセスまたはサーバは、クライアント操作が実行された後で監査プラグインを呼び出すことができます。これに対し、監査プラグインは、操作後の監査を実行して、監査が成功したかどうかを示す応答を返します。

さらに、ATMI システムのプロセスまたはサーバは、セキュリティ違反につながる現象を検出すると、監査プラグインを呼び出す場合があります。たとえば、操作前または操作後の認可のチェックが失敗したり、セキュリティに対する攻撃が検出された場合などです。これに対し、監査プラグインは、操作後の監査を実行して、監査が成功したかどうかを示す応答を返します。

これらの呼び出しは、アプリケーション・レベルの呼び出しではなく、システム・レベルの呼び出しです。ATMI アプリケーションは、監査プラグインを呼び出せません。

BEA Tuxedo システムに組み込まれているデフォルトの監査プラグインを使用する場合と、カスタマイズした監査プラグインを 1 つ以上使用する場合では、監査のプロセスが多少異なります。デフォルトのプラグインでは、操作前の監査をサポートしていません。したがって、デフォルトの監査プラグインが、操作前の監査を行う要求を受け取っても、要求は直ちに返され、何も実行されません。

カスタマイズしたプラグインでは、操作前と操作後の両方の監査をサポートしています。

デフォルトの監査

デフォルトの監査のインプリメンテーションは、BEA Tuxedo のイベント・ブローカとユーザ・ログ (ULOG) で構成されます。これらのユーティリティは、セキュリティ違反だけをレポートします。試行された操作、失敗した操作、および成功した操作についてはレポートしません。

セキュリティ違反の疑いに対して操作後の監査を実行するため、ATMI プロセスによってデフォルトの監査が呼び出されると、監査プラグインは、次のタスクを実行します。

1. 認証プラグインを呼び出して、クライアントの監査トークンから情報を取得します。

   監査トークンは、認証プラグインによって作成されるため、監査プラグインにはトークンの内容が記録されていません。ただし、監査プロセスではこの情報が必要です。
2. 操作後の監査を行います。

   監査プラグインは、クライアントの監査トークン、および操作後の監査要求で配信されたセキュリティ違反を調べます。
3. 操作後の監査が成功したかどうかの結果を発行します。

   監査のファンアウト・プラグインは、デフォルトの監査プラグインからの決定 (*success* または *failure*) を受け取ると、その決定に従って動作します。

   - *success* は、操作後の監査が成功したことを示します。監査のファンアウト・プラグインは、呼び出し側のプロセスに *success* を返し、セキュリティ違反を記録します。
   - *failure* は、操作後の監査が失敗したことを示します。監査のファンアウト・プラグインは、呼び出し側のプロセスに *failure* を返します。

カスタマイズした監査

カスタマイズした 1 つ以上の監査プラグインを使用するユーザは、BEA Tuxedo 製品の ATMI 環境で提供される別の機能を利用できます。つまり、操作の実行前に監査を行うことができます。

クライアントから操作前の監査の実行が要求され、それに対して ATMI のプロセスによってカスタマイズした監査が呼び出されると、監査プラグインは、次のタスクを実行します。

1. 認証プラグインを呼び出して、クライアントの監査トークンから情報を取得します。
2. 操作前の監査を行います。

   監査プラグインはクライアントの監査トークンを調べて、そのデータが後で操作後の監査に必要となる場合は、ユーザ・データを格納することができます。
3. 操作前の監査が成功したかどうかの結果を発行します。

   監査のファンアウト・プラグインは、下位にある個々のプラグインの応答 (*success* または *failure*) をチェックして、最終的な決定を行います。

   - コンポジット応答が *success* の場合は、操作前の監査が成功したことを示します。監査のファンアウト・プラグインは、呼び出し側のプロセスに *success* を返し、クライアントによる操作の試行を記録します。
   - コンポジット応答が *failure* の場合は、操作前の監査が失敗したことを示します。監査のファンアウト・プラグインは、呼び出し側のプロセスに *failure* を返します。

クライアント操作が実行されると、ATMI プロセスは、カスタマイズした監査を呼び出して操作後の監査を実行できます。その場合、監査プラグインは次のタスクを実行します。

1. 認証プラグインを呼び出して、クライアントの監査トークンから情報を取得します。
2. 操作後の監査を行います。

   監査プラグインは、クライアントの監査トークン、操作後の監査要求で配信された完了ステータス、および操作前の監査時に格納されたデータを調べます。
3. 操作後の監査が成功したかどうかの結果を発行します。

   監査のファンアウト・プラグインは、下位にある個々のプラグインの応答 (*success* または *failure*) をチェックして、操作後の監査が成功したか、または失敗したかを決定します。

   - コンポジット応答が *success* の場合は、操作後の監査が成功したことを示します。監査のファンアウト・プラグインは、呼び出し側のプロセスに *success* を返し、操作が完了したことを示すステータスを記録します。

- コンポジット応答が *failure* の場合は、操作後の監査が失敗したことを示します。監査のファンアウト・プラグインは、呼び出し側のプロセスに *failure* を返します。

操作前と操作後の監査を両方通過し、操作自体が成功した場合、その操作は成功したものと見なされます。操作前と操作後の両方の監査データを収集および格納する企業もありますが、これらのデータは大量のディスク領域を消費する可能性があります。

カスタマイズした監査のインプリメント

ATMI アプリケーションで監査機能を実行するには、デフォルトのプラグインを使用するか、または 1 つ以上のプラグインをカスタマイズします。どちらのプラグインを使用するかは、BEA Tuxedo のレジストリを設定して決めます。これは、すべてのセキュリティ・プラグインを制御するツールです。

デフォルトの監査プラグインを使用する場合、レジストリを設定する必要はありません。ただし、カスタマイズした認可プラグインを使用する場合は、プラグインをインストールする前に、プラグインに合わせてレジストリを設定する必要があります。レジストリの詳細については、第 2 章の 3 ページ「BEA Tuxedo レジストリの設定」を参照してください。

リンク・レベルの暗号化

リンク・レベルの暗号化 (LLE: Link-Level Encryption) は、ATMI アプリケーション内のマシン間をつなぐネットワーク・リンク上で送受信されるメッセージの秘密性を保護します。また、対称鍵による暗号化技術である RC4 を使用します。RC4 では、暗号化と復号化の際に同じ鍵を使用します。

LLE を使用する場合、BEA Tuxedo システムは、データを暗号化してからネットワーク・リンク上に送信し、データがネットワーク・リンクを離れると復号化します。システムは、データが通過するすべてのリンクで、この暗号化 / 復号化プロセスを繰り返します。したがって、LLE はポイント・ツー・ポイント機能と呼ばれます。

LLE は、ATMI アプリケーションの次の種類のリンクで使用できます。

- ワークステーション・クライアントからワークステーション・ハンドラ (WSH) へのリンク
- ブリッジ間のリンク
- 管理ユーティリティ (`tmboot` または `tmshutdown`) と `tlisten` とのリンク
- ドメイン・ゲートウェイ間のリンク

LLE によるセキュリティには、0 ビット (暗号化なし)、56 ビット (国際版)、128 ビット (米国およびカナダ版) の 3 つのレベルがあります。国際版の LLE では、0 ビットと 56 ビットの暗号化が可能です。米国およびカナダ版の LLE では、0 ビット、56 ビット、および 128 ビットの暗号化が可能です。

LLE のしくみ

LLE の制御パラメータおよび基本となる通信プロトコルは、リンクの種類によって異なります。ただし、以下の点では基本的に同じです。

- イニシエータ・プロセスが通信セッションを開始します。
- ターゲット・プロセスは、初期接続を受け取ります。
- 上記の両方のプロセスでは、リンク・レベルの暗号化機能が認識され、次の 2 つのコンフィギュレーション・パラメータが設定されます。

  1 つ目は、プロセスで受け付ける暗号化の最低レベルを設定するコンフィギュレーション・パラメータです。これは、0、56、128、のいずれかのキー長で指定します。

  2 つ目は、プロセスでサポートされる暗号化の最高レベルを設定するコンフィギュレーション・パラメータです。これも、0、56、128 のいずれかのキー長で指定します。

以降の説明では、上記の 2 つのパラメータを (*min*, *max*) の形式で表記します。たとえば、あるプロセスの値が (56, 128) の場合は、56 ビットから 128 ビットまでの暗号化がサポートされることを示します。

暗号化キー・サイズの調整

ネットワーク・リンクの両端にある 2 つのプロセスが通信し合うには、まず、これらのプロセスの暗号化キーのサイズが合っていなければなりません。これは、次の 2 段階の手順で確認されます。

1. 各プロセスで、それぞれの *min-max* 値が確認されます。
2. 両方のプロセスでサポートされる、キーの最大サイズが算出されます。

min-max 値の決定

2 つのプロセスのうち、どちらかが起動すると、BEA Tuxedo のローカル・ソフトウェアは、(1) `lic.txt` ファイルの LLE 情報を参照して、インストール済みの LLE のバージョンのビット暗号化機能をチェックし、(2) 両方のプロセスのコンフィギュレーション・ファイルで指定されている、特定の種類のリンクでの LLE の *min-max* 値をチェックします。続いて、ローカル・ソフトウェアは、次の処理を実行します。

- 設定されている *min-max* 値が、インストール済みの LLE のバージョンと対応する場合、ローカル・ソフトウェアは、この値をプロセスの *min-max* 値として割り当てます。
- 設定されている *min-max* 値が、インストール済みの LLE のバージョンと対応しない場合、ローカル・ソフトウェアからはエラーが返されます。たとえば、国際版の LLE がインストールされているときに、*min-max* 値が (0, 128) である場合です。この時点では、リンク・レベルの暗号化は不可能です。
- 特定の種類のリンクのコンフィギュレーションで、*min-max* 値が指定されていない場合、ローカル・ソフトウェアは、最小値として 0 を割り当て、最大値としてインストール済みの LLE のバージョンに対して可能な最高ビットの暗号化レートを割り当てます。つまり、米国およびカナダ版の LLE では (0, 128) を割り当てます。

共通のキー・サイズの検索

2 つのプロセスの *min-max* 値が決まると、キー・サイズの調整が開始します。この調整プロセスを暗号化したり、見えないようにする必要はありません。調整されたキー・サイズは、ネットワーク接続が確立されている間は有効です。

次の表は、2 つのプロセス間で可能な *min-max* 値のすべての組み合わせを調整した結果のキー・サイズを示しています。上段のヘッダ行は、2 つのプロセスのうち、片方のプロセスの *min-max* 値を示しています。左側の列は、もう一方のプロセスの *min-max* 値を示しています。

表 1-3 プロセス間の調整の結果

	(0, 0)	(0, 56)	(0, 128)	(56, 56)	(56, 128)	(128, 128)
(0, 0)	0	0	0	エラー	エラー	エラー
(0, 56)	0	56	56	56	56	エラー
(0, 128)	0	56	128	56	128	128
(56, 56)	エラー	56	56	56	56	エラー
(56, 128)	エラー	56	128	56	128	128
(128, 128)	エラー	エラー	128	エラー	128	128

LLE の旧バージョンとの互換性

BEA Tuxedo 製品の ATMI 環境では、LLE の旧バージョンとの互換性がサポートされています。

BEA Tuxedo リリース 6.5 との相互運用性

次の表は、ATMI アプリケーションの 2 つのプロセスのうち、片方がリリース 6.5 で動作しており、もう一方がリリース 7.1 以上で動作しているときの、調整結果のキー・サイズを示しています。上段のヘッダ行は、リリース 7.1 以上で動作するプロセスの *min-max* 値を示しています。左側の列は、リリース 6.5 で動作するプロセスの *min-max* 値を示しています。

表 1-4 BEA Tuxedo リリース 6.5 と相互運用するときのキー・サイズの調整結果

	(0, 0)	(0, 56)	(0, 128)	(56, 56)	(56, 128)	(128, 128)
(0, 0)	0	0	0	エラー	エラー	エラー
(0, 40)	0	56	56	56	56	エラー
(0, 128)	0	56	128	56	128	128
(40, 40)	エラー	56	56	56	56	エラー
(40, 128)	エラー	56	128	56	128	128

表 1-4 BEA Tuxedo リリース 6.5 と相互運用するときのキー・サイズの調整結果

	(0, 0)	(0, 56)	(0, 128)	(56, 56)	(56, 128)	(128, 128)
(128, 128)	エラー	エラー	128	エラー	128	128

現在インストールされている BEA Tuxedo が (0, 56)、(0, 128)、(56, 56)、または (56, 128) に設定されているときに、LLE の最大レベルが 40 ビットに設定されている ATMI アプリケーションのリリース 6.5 と相互運用させようとすると、調整結果のキー・サイズは、常に 56 に自動アップグレードされます。

この調整結果は、2 つのサイトで共にリリース 6.5 が動作しており、LLE のレベルが 40 ビットに設定されている場合の結果と同じです。どちらの場合も、調整を行うと、56 に自動的にアップグレードされます。

BEA Tuxedo リリース 6.5 より前のバージョンとの相互運用性

次の表は、ATMI アプリケーションの 2 つのプロセスのうち、片方がリリース 6.5 より前のバージョンで動作しており、もう一方がリリース 7.1 以上で動作しているときの、調整結果のキー・サイズを示しています。上段のヘッダ行は、リリース 7.1 以上で動作するプロセスの *min-max* 値を示しています。左側の列は、リリース 6.5 より前のバージョンで動作するプロセスの *min-max* 値を示しています。

表 1-5 BEA Tuxedo リリース 6.5 より前のバージョンと相互運用するときのキー・サイズの調整結果

	(0, 0)	(0, 56)	(0, 128)	(56, 56)	(56, 128)	(128, 128)
(0, 0)	0	0	0	エラー	エラー	エラー
(0, 40)	0	40	40	エラー	エラー	エラー
(0, 128)	0	40	128	エラー	128	128
(40, 40)	エラー	40	40	エラー	エラー	エラー
(40, 128)	エラー	40	128	エラー	128	128
(128, 128)	エラー	エラー	128	エラー	128	128

現在インストールされている BEA Tuxedo が (0, 56) または (0, 128) に設定されているときに、LLE の最大レベルが 40 ビットに設定されている、リリース 6.5 より前のバージョンの ATMI アプリケーションと相互運用させようとすると、調整結果のキー・サイズは、常に 40 になります。

現在インストールされている BEA Tuxedo が (56, 56)、(56, 128)、または (128, 128) に設定されていると、LLE の最大レベルが 40 に設定されている、ATMI アプリケーションのリリース 6.5 より前のバージョンとは相互運用できません。共通のキー・サイズを調整しようとしてもできません。

初期化時の WSL および WSH の接続タイムアウト

ワークステーション・クライアントが初期化に費やせる時間は制限されています。デフォルトでは、LLE を使用しない ATMI アプリケーションでは 30 秒、LLE を使用する ATMI アプリケーションでは 60 秒に制限されています。この 60 秒には、暗号化されたリンクを調整する時間も含まれます。ただし、LLE が設定されている場合は、UBBCONFIG ファイルの MAXINITTIME パラメータでワークステーション・リスナ (WSL) のサーバの値を変更するか、または WS_MIB(5) にある T_WSL クラスの TA_MAXINITTIME 属性の値を変更することによって、制限を変更できます。

LLE のインストールとライセンス供与

LLE ソフトウェアは、BEA Tuxedo システムの一部として、BEA Tuxedo CD-ROM に収められています。米国およびカナダで LLE を使用するための BEA Tuxedo リリース 7.1 のライセンスが供与されている場合、56 ビットまたは 128 ビットの暗号化が可能です。米国およびカナダ以外で LLE を使用するための BEA Tuxedo リリース 7.1 のライセンスが供与されている場合、56 ビットの暗号化が可能です。

BEA Tuxedo のライセンスは、UNIX ホスト・マシンの場合は $TUXDIR/udataobj/lic.txt ファイル、Windows ホスト・マシンの場合は %TUXDIR%¥udataobj¥lic.txt ファイルにすべて格納されています。

次のリストは、米国およびカナダで LLE を実行するためのライセンス・ファイルのサンプルからの抜粋です。

```
[BEA Tuxedo]
VERSION=8.0
LICENSEE=ACME CORPORATION
SERIAL=155566678
ORDERID=
USERS=1000
EXPIRATION=2000-01-31
SIGNATURE=TXmtx+AhQdJgr3sjjznBqRB7SP9Jgr3UzAKctjz+e6RmsFSAhUAhStj
   znBQdL9n=
```

```
[LINK ENCRYPTION]
VERSION=8.0
LICENSEE=ACME CORPORATION
SERIAL=155566678
ORDERID=
USERS=1000
STRENGTH=128
EXPIRATION=2000-01-31
SIGNATURE=TXUAhSPnx2C9kMC0CFG+e6Rgr3UzmsFKRBPdJASAhU7KctjznBqFQsj
   jznBdh0h=
   .
   .
   .
```

関連項目

- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 第 2 章の 33 ページ「リンク・レベルの暗号化の管理」
- 『BEA Tuxedo アプリケーションの設定』の第 7 章の 1 ページ「ネットワークへの ATMI アプリケーションの分散」および 第 8 章の 1 ページ「分散型の ATMI アプリケーション用のコンフィギュレーション・ファイルの作成」

公開鍵によるセキュリティ機能

公開鍵によるセキュリティは、エンド・ツー・エンドのデジタル署名およびデータの暗号化を実現する、次の 2 つの機能を提供します。

- メッセージ・ベースのデジタル署名
- メッセージ・ベースの暗号化

メッセージ・ベースのデジタル署名により、メッセージの受信者は、送信元と送信メッセージの両方を識別し、認証できます。デジタル署名では、送信元と送信メッセージの内容が確実に証明されます。送信メッセージには、送信者の署名が添付されているため、送信者はメッセージを送信した事実を否認することはできません。たとえば、ある人が自分の銀行口座から引き出しを行った後で、その処理が別の人によって行われたと主張することはできません。

さらに、メッセージ・ベースの暗号化では、指定した受信者だけがメッセージを復号化できるため、メッセージの機密性が保たれます。

PKCS-7 への準拠

RSA Laboratories を中心とする主要な通信会社のグループによって規定された、非公式でありながら業界では認知されている公開鍵ソフトウェアの標準があります。これを PKCS (Public-Key Cryptography Standards) と言います。BEA Tuxedo ソフトウェアの ATMI 環境における公開鍵ソフトウェアは、PKCS-7 標準に準拠しています。

PKCS-7 は、ハイブリッド型の暗号化システムのアーキテクチャです。つまり、メッセージを暗号化するときは、ランダムなセッション・キーを用いた対称鍵のアルゴリズムを使用し、ランダムなセッション・キーを暗号化するときは、公開鍵のアルゴリズムを使用します。乱数ジェネレータにより、通信のたびに新しいセッション・キーが作成されます。これで、以前の通信を利用した攻撃を防ぐことができます。

サポートされている公開鍵のアルゴリズム

公開鍵の基となるアルゴリズムは、いずれもよく知られており、一般に市販されています。ATMI アプリケーションに最も適したアルゴリズムを選択するには、処理速度、セキュリティのレベル、およびライセンスに関する制約を考慮してください。たとえば、米国政府は、外国に輸出できるアルゴリズムを制限しています。

公開鍵のアルゴリズム

BEA Tuxedo 製品の ATMI 環境における公開鍵によるセキュリティでは、RSA、ElGamal、Rabin など、基盤となるプラグインでサポートされるすべての公開鍵のアルゴリズムがサポートされます (RSA は、RSA アルゴリズムの発明者の頭文字 (Rivest、Shamir、Adelman) を取ったものです)。これらのアルゴリズムはすべて、デジタル署名と暗号化に使用できます。

RSA など、公開鍵 (または非対称鍵) のアルゴリズムは、次の 2 つの鍵の組み合わせによってインプリメントされます。これらの鍵は異なりますが、数学的には関連性があります。

- 公開鍵。広範囲にわたって公開される鍵であり、デジタル署名を検証したり、データを理解できない形式に変換するために使用します。
- 秘密鍵。秘密に扱われる鍵であり、デジタル署名を作成したり、データを元の形式に戻すために使用します。

デジタル署名のアルゴリズム

BEA Tuxedo 製品の ATMI 環境におけるデジタル署名によるセキュリティでは、RSA、ElGamal、Rabin、Digital Signature Algorithm (DSA) など、基盤となるプラグインでサポートされるすべてのデジタル署名アルゴリズムがサポートされます。DSA 以外のすべてのアルゴリズムは、デジタル署名と暗号化に使用できます。DSA は、デジタル署名には使用できますが、暗号化には使用できません。

デジタル署名のアルゴリズムは、単にデジタル署名を実現するための公開鍵アルゴリズムです。DSA も公開鍵アルゴリズム、つまり、公開鍵と秘密鍵の組み合わせによってインプリメントするアルゴリズムですが、デジタル署名用です。暗号化には適用できません。

対称鍵のアルゴリズム

公開鍵セキュリティでは、以下の 3 つの対称鍵アルゴリズムをサポートしています。

- DES-CBC (Data Encryption Standard for Cipher Block Chaining)

  DES-CBC は、CBC (Cipher Block Chaining) モードで実行する、64 ビット単位のブロック暗号です。56 ビット (64 ビットの暗号化キーから 8 パリティ・ビットを引いたもの) のキーを使用し、米国以外の国でも使用できます。

- 2 つの鍵による Triple-DES (Data Encryption Standard)

  2 つの鍵による Triple DES は、EDE (Encrypt-Decrypt-Encrypt) モードで実行する 128 ビットのブロック暗号です。2 つの鍵による Triple DES では、56 ビット (実際には 112 ビット) のキーを使用します。米国から輸出することは禁止されています。

  最近では、Triple-DES を使用して、DES の暗号鍵を保護し、転送するのが一般的になりました。つまり、入力データ (この場合は単一の DES キー) は、暗号化、復号化、暗号化、の順に繰り返し処理されます (暗号化 / 復号化 / 暗号化のプロセス)。このうち、2 回行われる暗号化では、両方とも同じ暗号鍵が使用されます。

- RC2 (Rivest’s Cipher 2)

  RC2 は、40 ～ 128 ビットの範囲で、暗号鍵のサイズを変更できるブロック暗号です。DES より高速であり、40 ビットの暗号鍵は輸出できます。米国籍の企業の海外子会社および海外支店であれば、56 ビットの暗号鍵を使用することができます。ATMI の公開鍵セキュリティ機能では暗号鍵の長さを 128 ビットに制限していますが、米国では実質的に、RC2 にはどんな長さの鍵を使用することもできます。

BEA Tuxedo をお使いのお客様は、このアルゴリズムのリストを拡張したり変更することはできません。

対称鍵アルゴリズムでは、同じ鍵を使用して、メッセージの暗号化と復号化を行います。公開鍵による暗号化システムでは、通信し合う 2 つのエンティティ間で送信されるメッセージの暗号化に対称鍵暗号化を使用します。対称鍵暗号は、公開鍵暗号より、少なくとも 1000 倍速く実行されます。

ブロック暗号は、対称鍵アルゴリズムの一種であり、固定長の平文 (暗号化されていないテキスト) のブロックを、同じ長さの暗号文 (暗号化されたテキスト) のブロックに変換します。この変換は、ランダムに生成されたセッション・キーの値に基づいて行われます。固定長は、ブロック・サイズと呼ばれます。

メッセージ・ダイジェスト・アルゴリズム

公開鍵によるセキュリティでは、MD5、SHA-1 (Secure Hash Algorithm 1) など、基盤となるプラグインでサポートされるすべてのメッセージ・ダイジェスト・アルゴリズムがサポートされます。MD5 と SHA-1 は、どちらも有名な一方向のハッシュ・アルゴリズムです。一方向のハッシュ・アルゴリズムでは、メッセージが「メッセージ・ダイジェスト」または「ハッシュ値」と呼ばれる固定長の数値文字列に変換されます。

MD5 は、128 ビットのハッシュ値を生成する、高速のアルゴリズムです。32 ビットのマシンでの使用に適しています。SHA-1 は、160 ビットのハッシュ値を生成する、セキュリティ・レベルの高いアルゴリズムですが、処理速度は MD5 よりやや遅くなります。

公開鍵のインストールとライセンス供与

メッセージ・ベースのデジタル署名およびメッセージ・ベースの暗号化のソフトウェアは、BEA Tuxedo システムの一部として、BEA Tuxedo CD-ROM に収められていますが、これを使用するには別個のライセンスが必要です。BEA Tuxedo のライセンスは、UNIX ホスト・マシンの場合は `$TUXDIR/udataobj/lic.txt` ファイル、Windows 2000 ホスト・マシンの場合は `%TUXDIR%¥udataobj¥lic.txt` ファイルにすべて格納されています。

次のリストは、メッセージ・ベースのデジタル署名およびメッセージ・ベースの暗号化用のライセンス・ファイルのサンプルからの抜粋です。

```
[BEA Tuxedo]
VERSION=8.0
LICENSEE=ACME CORPORATION
SERIAL=155566678
ORDERID=
USERS=1000
EXPIRATION=2000-01-31
SIGNATURE=TXmtx+AhQdJgr3sjjznBqRB7SP9Jgr3UzAKctjz+e6RmsFSAhUAhStj
```

```
   znBQdL9n=
   .
   .
   .
[PK ENCRYPTION]
VERSION=8.0
LICENSEE=ACME CORPORATION
SERIAL=155566678
ORDERID=
USERS=1000
STRENGTH=128
EXPIRATION=2000-01-31
SIGNATURE=TX0CFHkaBpKpAlXGEtQqi+/jJvMo1VB9AhUAUAkizwsgYefRwQJDNTF
   0205b1ik=
[PK SIGNATURE]
VERSION=8.0
LICENSEE=ACME CORPORATION
SERIAL=155566678
ORDERID=
USERS=1000
STRENGTH=128
EXPIRATION=2000-01-31
SIGNATURE=TX0CiqA5FCAXJFXUEGvAki+gL+i09eRep9hYdshS/8a70MIJQChUAk9
   zIAhUIH4=
```

関連項目

- 第 1 章の 34 ページ「メッセージ・ベースのデジタル署名」
- 第 1 章の 38 ページ「メッセージ・ベースの暗号化」
- 第 1 章の 40 ページ「公開鍵のインプリメンテーション」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 第 2 章の 40 ページ「公開鍵セキュリティの管理」
- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」
- 第 3 章の 17 ページ「データの完全性と機密性を保護するためのセキュリティ・コードの記述方法」

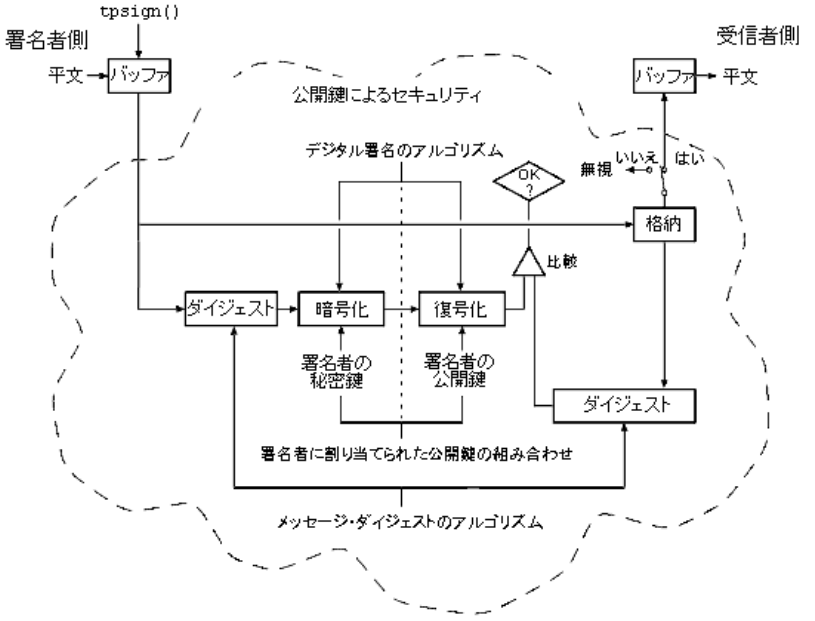
メッセージ・ベースのデジタル署名

メッセージ・ベースのデジタル署名は、メッセージの送信者の ID を証明し、その内容を特定のメッセージ・バッファと結び付けることにより、ATMI のセキュリティ機能を強化します。企業間または企業と一般ユーザの間で、インターネットを介してデータを送受信する ATMI アプリケーションでは、送信側と受信側の両方で認証を行い、送信妨害のない通信を実現することが重要です。これらの条件は、セキュリティ保護されていない社内ネットワークを使用して ATMI アプリケーションを導入する場合にも特に重要です。

メッセージ・ベースのデジタル署名では、エンド・ツー・エンドで内容がセキュリティ保護されます。つまり、メッセージ・バッファは、送信プロセスで発信され、受信プロセスで受け取られるまで保護されます。送信中には、一時的なメッセージ・キュー、ディスク・ベース・キュー、システム・プロセスなどの中継ポイントでも保護され、サーバ間のネットワーク・リンクで転送される間も保護されます。

次の図は、エンド・ツー・エンド型のメッセージ・ベースのデジタル署名のしくみを示しています。

図 1-7 ATMI PKCS-7 のエンド・ツー・エンドのデジタル署名



メッセージ・ベースのデジタル署名では、メッセージのメッセージ・ダイジェストを計算し、次に送信者の秘密鍵でメッセージ・ダイジェストを暗号化して、デジタル署名を生成します。受信者は、暗号化されたメッセージ・ダイジェストを署名者の公開鍵を使って復号化し、復号化したメッセージ・ダイジェストと、別途計算したメッセージ・ダイジェストを比較することによって、署名を検証します。署名者の公開鍵は、署名者の情報に含まれているデジタル証明書から参照するか、または、公開鍵の証明を一意に識別する、発行者識別名および発行者固有のシリアル番号を参照して取得します。

デジタル証明書

デジタル証明書は、インターネットなどのネットワーク経由で、個人およびリソースを識別するために使用される電子ファイルです。デジタル証明書は、信頼性のある第三者機関である「認証局」によって認定された個人またはリソースの ID を特定の公開鍵に安全な方法で結び付けます。公開鍵は重複しないため、公開鍵から所有者を特定できます。

デジタル証明書は、特定の公開鍵が特定のサブスクライバ (所有者) に属することを検証します。証明書の受信者は、証明書に記載されている公開鍵を使用して、その公開鍵に対応する秘密鍵でデジタル証明が作成されたことを検証します。検証が成功すると、証明書で指定されたサブスクライバが、対応する秘密鍵の所有者であること、および、そのサブスクライバによってデジタル署名が作成されたことを、一連の処理で確認できたことになります。

証明書には、次のようなさまざまな情報が含まれています。

- サブスクライバ (所持者、所有者) の名前、およびサブスクライバを一意に識別するためのその他の ID 情報 (証明書を使用する Web サーバの URL、個人の電子メール・アドレスなど)
- サブスクライバの公開鍵
- 証明書を発行した認証局の名前
- シリアル番号
- 証明の有効期間または存続期間 (開始日と終了日を定義)

最も広く知られている証明書の形式は、ITU-T X.509 国際規格によって定義された形式です。したがって、X.509 準拠の ATMI アプリケーションであれば、証明書の読み書きを行えます。BEA Tuxedo 製品の ATMI 環境における公開鍵によるセキュリティ機能では、X.509 バージョン 3 (X.509v3) 準拠の証明書が認識されます。

認証局

証明書は、認証局 (CA: Certification Authority) によって発行されます。証明書と公開鍵の発行対象に対して ID を保証できる、信頼性のある第三者機関または企業は、CA となることができます。CA は、証明書の作成時に秘密鍵を使用して証明書に署名し、デジタル署名を取得します。次に、CA は署名付きの証明書をサブスクライバに返します。証明書と CA の署名の組み合わせにより、有効な証明書が作成されます。

サブスクライバおよびその他の人が、CA のデジタル署名を検証するには、CA の公開鍵を使用します。CA は、そのキーを公開するか、または上位レベルの CA の証明書 (下位レベルの CA の公開鍵の有効性を証明) を発行して、公開鍵を利用可能にします。2 つ目の方法を用いると、CA が階層化されます。

暗号化メッセージの受信者が、信頼する上位 CA によって署名された証明書を持ち、この証明書に CA の公開鍵が含まれている場合は、CA の秘密鍵の信頼性を再帰的に高めることができます。この意味で、証明書は、デジタル署名の信頼性を確認する足がかりとなります。つまり、最終的には、いくつかの上位 CA の公開鍵の信頼性を確認するだけで済みます。一連の証明書を確認することにより、多数のユーザ署名の信頼性を証明できます。

つまり、デジタル署名は通信エンティティの ID を証明しますが、署名の信頼度は、署名を検証するための公開鍵を信頼できる、というレベルと同じです。

BEA が CA となる予定はありません。BEA の公開鍵のプラグイン・インターフェイスにより、BEA Tuxedo のユーザは、CA を自由に選択できます。

証明書のリポジトリ

特定のサブスクライバ用の公開鍵およびその ID を利用可能にし、検証に使用できるようにするため、デジタル証明書をリポジトリに公開したり、その他の方法で公開できます。リポジトリは、証明書およびその他の情報で構成されるデータベースであり、リポジトリ内の情報は、取得したり、デジタル署名を検証するために使用できます。情報を取得するには、検証プログラムを実行し、直接リポジトリ内の証明書を要求することにより、自動的に行います。

PKI (Public-Key Infrastructure)

PKI (Public-Key Infrastructure) は、公開鍵暗号のアプリケーションをサポートするプロトコル、サービス、および標準で構成されます。PKI は比較的新しい技術であるため、定義は曖昧です。たとえば、単に、公開鍵の証明書に基づいた、信頼性を示す階層構造を指す場合があります。また、別のコンテキストでは、エンド・ユーザ用アプリケーションのデジタル署名および暗号化サービスを意味する場合もあります。

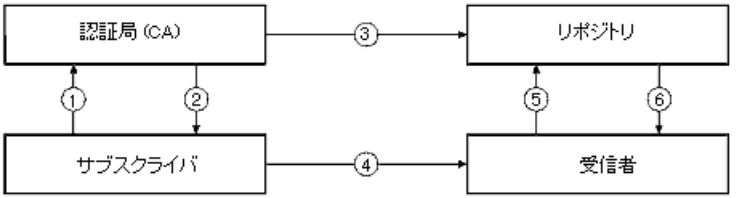
PKI を規定する単一の標準はありませんが、標準の策定を進める動きはあります。現時点では、標準が策定されるかどうか、またはさまざまな相互運用レベルの PKI が複数誕生するかどうかは不明です。この意味で、PKI 技術の現状は、インターネットによる広範囲の接続が可能になるまでの、1980 年代の LAN および WAN 技術に似ていると言えます。

PKI には、次のようなサービスがあります。

- キー登録。公開鍵の新しい証明書を発行します
- 証明書の取り消し。既に発行した証明書をキャンセルします。
- キー選択。パーティの公開鍵を取得します。
- 信頼性の評価。証明書の有効性、および証明書で認可される操作を決定します。

次の図は、PKI の処理の流れを示します。

図 1-8　PKI の処理の流れ



1. サブスクライバは、認証局 (CA) にデジタル証明を申し込みます。
2. CA は、サブスクライバの ID を検証し、デジタル証明書を発行します。
3. CA は、リポジトリに証明書を公開します。
4. サブスクライバは、秘密鍵で電子メッセージにデジタル署名して、送信者が認証済みであること、メッセージが完全であること、およびメッセージを否認できないことを確認します。その後、メッセージを受信者に送信します。
5. 受信者は、メッセージを受信すると、サブスクライバの公開鍵でデジタル署名を検証し、リポジトリでサブスクライバの証明書のステータスと有効性をチェックします。

6. サブスクライバの証明書に関するステータス・チェックの結果が、リポジトリから受信者に返されます。

BEA システムが PKI ベンダとなる予定はありません。BEA システムの公開鍵のプラグイン・インターフェイスにより、BEA Tuxedo のユーザは、PKI ソフトウェアのベンダを自由に選択し、PKI のセキュリティ・ソリューションを使用できます。

関連項目

- 第 1 章の 40 ページ「公開鍵のインプリメンテーション」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 第 2 章の 40 ページ「公開鍵セキュリティの管理」
- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」
- 第 3 章の 17 ページ「データの完全性と機密性を保護するためのセキュリティ・コードの記述方法」

メッセージ・ベースの暗号化

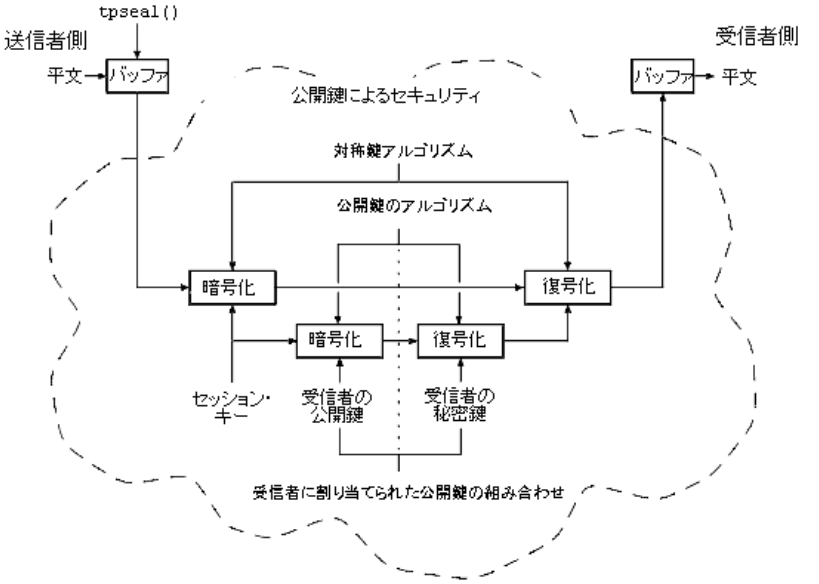
メッセージ・ベースの暗号化では、データが秘密に扱われます。これは、企業間または企業と一般ユーザの間で、インターネットを介してデータを送受信する ATMI アプリケーションでは重要です。データの秘密性は、セキュリティ保護されていない社内ネットワークを使用して ATMI アプリケーションを導入する場合にも特に重要です。

メッセージ・ベースの暗号化では、メッセージが意味のわからない状態に変換され、内容を変更しようとしてもできないため、メッセージの完全性が保たれます。

メッセージ・ベースの暗号化では、エンド・ツー・エンドで内容がセキュリティ保護されます。つまり、メッセージ・バッファは、送信プロセスで発信され、受信プロセスで受け取られるまで保護されます。送信中には、一時的なメッセージ・キュー、ディスク・ベース・キュー、システム・プロセスなどの中継ポイントでも保護され、サーバ間のネットワーク・リンクで転送される間も保護されます。

次の図は、エンド・ツー・エンド型のメッセージ・ベースの暗号化のしくみを示しています。

図 1-9　ATMI PKCS-7 **のエンド・ツー・エンドの暗号化**



メッセージは、対称鍵アルゴリズムとセッション・キーによって暗号化されます。次に、セッション・キーが受信者の公開鍵によって暗号化されます。さらに、受信者は、受信者の秘密鍵を使用して、暗号化されたセッション・キーを復号化します。最後に、受信者は、セッション・キーを使用して暗号化されたメッセージを復号化し、メッセージ内容を取得します。

注記　この図では、(1) メッセージを復号化する直前にデータを圧縮する、および (2) メッセージを復号化した直後にデータを解凍する、という 2 つの手順が説明されていません。

暗号化は、ATMI のメッセージ・バッファ単位で行われます。したがって、メッセージ・ベースの暗号化は、既存の ATMI のプログラミング・インターフェイスおよび通信パラダイムと互換性があります。暗号化のプロセスは、単一のマシン内にある 2 つのプロセス間で送受信されるメッセージに対して行われる場合も、2 つのマシン間でネットワークを介して送受信されるメッセージに対して行われる場合も同じです。

関連項目

- 第 1 章の 40 ページ「公開鍵のインプリメンテーション」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 第 2 章の 54 ページ「デフォルトの認証と認可の管理」
- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」
- 第 3 章の 17 ページ「データの完全性と機密性を保護するためのセキュリティ・コードの記述方法」

公開鍵のインプリメンテーション

公開鍵セキュリティのベースとなるプラグイン・インターフェイスは、6 つのコンポーネント・インターフェイスから成り、それぞれのコンポーネントは、1 つ以上のプラグインを必要とします。これらのインターフェイスを好みのプラグインでインスタンス化すれば、メッセージ・ベースのカスタム・デジタル署名とメッセージ・ベースの暗号化を ATMI アプリケーションに持ち込むことができます。

6 つのコンポーネント・インターフェイスを次に示します。

- 公開鍵の初期化
- 鍵管理
- 証明ルックアップ
- 証明解析
- 証明検査
- 証明資料のマッピング

公開鍵の初期化

公開鍵の初期化インターフェイスによって、公開鍵ソフトウェアは、公開鍵と秘密鍵を開くことができます。たとえば、ゲートウェイ・プロセスでは、メッセージを復号化してから転送するために、特定の秘密鍵へのアクセスが必要なこともあります。このインターフェイスは、ファンアウトとしてインプリメントされます。

鍵の管理

鍵管理インターフェイスによって、公開鍵ソフトウェアは、公開鍵と秘密鍵を管理および使用することができます。なお、メッセージ・ダイジェストとセッション・キーは、このインターフェイスを使用して暗号化および復号化されますが、公開鍵暗号を使用するバルク・データの暗号化は行われません。バルク・データの暗号化は、対称鍵暗号を使用して行われます。

証明書のルックアップ

証明ルックアップ・インターフェイスによって、公開鍵ソフトウェアは、所定のプリンシパルに対する X.509v3 証明を取得できます。プリンシパルは認証されたユーザです。証明データベースは、Lightweight Directory Access Protocol (LDAP)、Microsoft Active Directory、Netware Directory Service (NDS)、またはローカル・ファイルなど、適切なツールを使用して格納することができます。

証明書の解析

証明解析インターフェイスによって、公開鍵ソフトウェアは、簡単なプリンシパル名と X.509v3 証明を関連付けることができます。パーサは、証明を解析して、証明に関連付けるプリンシパル名を生成します。

証明書の検証

証明検査インターフェイスによって、公開鍵ソフトウェアは、特定のビジネス・ロジックに基づいて X.509v3 証明を検証することができます。このインターフェイスはファンアウトとしてインプリメントされているため、BEA Tuxedo のユーザは、自分自身のビジネス・ルールを使用して証明の有効性を判定できます。

証明資料のマッピング

証明資料のマッピング・インターフェイスによって、公開鍵ソフトウェアは、鍵を開くために必要な証明資料へのアクセス、認可トークンの提供、および監査トークンの提供を行うことができます。

カスタマイズした公開鍵のインプリメント

ATMI アプリケーションに公開鍵セキュリティを提供するには、デフォルト・プラグインまたはカスタム・プラグインを使用します。プラグインを選択するには、すべてのセキュリティ・プラグインを制御するツールである、BEA Tuxedo レジストリを設定します。

デフォルト公開鍵プラグインを使用する場合は、レジストリを設定する必要はありません。しかし、カスタム公開鍵プラグインを使用する場合は、公開鍵プラグイン用のレジストリを設定してから、それらのプラグインをインストールする必要があります。レジストリの詳細については、第 2 章の 3 ページ「BEA Tuxedo レジストリの設定」を参照してください。

デフォルトの公開鍵のインプリメンテーション

デフォルト公開鍵のインプリメンテーションでは、以下のアルゴリズムをサポートしています。

- 公開鍵アルゴリズム : RSA
- デジタル署名アルゴリズム : RSA と DSA
- 対称鍵アルゴリズム :
  - DES-CBC
  - 2 つの鍵による Triple - DES
  - RC2
- メッセージ・ダイジェスト・アルゴリズム :
  - MD5
  - SHA-1

関連項目

- 第 1 章の 29 ページ「公開鍵によるセキュリティ機能」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 第 1 章の 29 ページ「公開鍵によるセキュリティ機能」
- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」

- 第 3 章の 17 ページ「データの完全性と機密性を保護するためのセキュリティ・コードの記述方法」

デフォルトの認証と認可

BEA Tuxedo 製品の ATMI 環境に用意されている、デフォルトの認証および認可のプラグインは、BEA Tuxedo で最初に実現された、従来の認証および認可のインプリメンテーションと同じように機能します。

アプリケーション管理者は、デフォルトの認証および認可のプラグインを使用して、5 つのうち、1 つのセキュリティ・レベルを ATMI アプリケーションに設定できます。5 つのセキュリティ・レベルは、以下のとおりです。

- 認証なし
- アプリケーション・パスワードによるセキュリティ
- ユーザ・レベルの認証
- オプションのアクセス制御リスト (ACL)
- 必須のアクセス制御リスト (ACL)

最も低いセキュリティ・レベルでは、認証は行われません。最も高いセキュリティ・レベルでは、アクセス制御リストの機能により、サービスを実行し、イベントをポストし、アプリケーション・キューのメッセージをキューに登録 (または登録解除) するユーザが決定されます。次の表は、セキュリティ・レベルを簡単にまとめたものです。

表 1-6 デフォルトの認証と認可のセキュリティ・レベル

セキュリティ・レベル	説明
認証なし	ATMI アプリケーションに参加する前にクライアントを検証する必要はありません。 このセキュリティ・レベルで ATMI アプリケーションに参加すると、ユーザはすべてのアプリケーション・リソースにアクセスできます。

表 1-6 デフォルトの認証と認可のセキュリティ・レベル (続き)

セキュリティ・レベル	説明
アプリケーション・パスワード	アプリケーション管理者は、ATMI アプリケーション全体に対して 1 つのパスワードを定義します。クライアントがアプリケーションに参加するには、パスワードを提示しなければなりません。 このセキュリティ・レベルで ATMI アプリケーションに参加できると、ユーザはすべてのアプリケーション・リソースにアクセスできます。
ユーザ・レベルの認証	ATMI アプリケーションに参加するには、各クライアントは、アプリケーション・パスワードのほか、有効なユーザ名とユーザ固有のデータ (パスワードなど) を提示しなければなりません。 このセキュリティ・レベルで ATMI アプリケーションに参加できると、ユーザはすべてのアプリケーション・リソースにアクセスできます。
オプションのアクセス制御リスト (ACL)	クライアントは、アプリケーション・パスワード、ユーザ名、およびユーザ固有のデータ (パスワードなど) を提示しなければなりません。 このセキュリティ・レベルで ATMI アプリケーションに参加すると、ユーザによるアプリケーション・リソースへのアクセスは制限されます。つまり、ACL データベースに格納されているアプリケーション・リソースのリストにより、各リソースを使用できるユーザが制限されます。特定のリソースのリストに含まれていないユーザは、オプションの ACL または必須の ACL が使用されていても、そのリソースにはアクセスできません。 ATMI アプリケーションのセキュリティ・レベルが「オプションの ACL」に設定されている場合、ACL データベースにエントリがないリソースには、誰でもアクセスできます。

表 1-6 デフォルトの認証と認可のセキュリティ・レベル (続き)

セキュリティ・レベル	説明
必須のアクセス制御リスト (ACL)	クライアントは、アプリケーション・パスワード、ユーザ名、およびユーザ固有のデータ (パスワードなど) を提示しなければなりません。 このセキュリティ・レベルで ATMI アプリケーションに参加すると、ユーザによるアプリケーション・リソースへのアクセスは制限されます。つまり、ACL データベースに格納されているアプリケーション・リソースのリストにより、各リソースを使用できるユーザが制限されます。特定のリソースのリストに含まれていないユーザは、オプションの ACL または必須の ACL が使用されていても、そのリソースにはアクセスできません。 ATMI アプリケーションのセキュリティ・レベルが「必須の ACL」に設定されている場合、ACL データベースにエントリがないリソースには、どのユーザもアクセスできません。

注記「クライアント」という用語は「クライアント・プロセス」と同義であり、実行中のクライアント・プログラムの特定のインスタンスを意味します。ATMI のクライアント・プログラムは、任意の数の個別インスタンスのアクティブ・メモリ内に存在することができます。

アプリケーション管理者は、UBBCONFIG コンフィギュレーション・ファイルの SECURITY パラメータに適切な値を設定することにより、セキュリティ・レベルを指定できます。

セキュリティ・レベル	SECURITY パラメータに設定する値
認証なし	なし
アプリケーション・パスワードによるセキュリティ	APP_PW
ユーザ・レベルの認証	USER_AUTH
オプションのアクセス制御リスト (ACL)	ACL
必須のアクセス制御リスト (ACL)	MANDATORY_ACL

デフォルト値は NONE です。SECURITY に USER_AUTH、ACL、または MANDATORY_ACL を設定した場合、アプリケーション管理者は、システム側で提供される AUTHSVR という名前の認証サーバを設定しなければなりません。AUTHSVR は、ユーザ単位で認証を行います。

アプリケーション開発者は、AUTHSVR を、ATMI アプリケーションに固有のロジックを持つ認証サーバに置き換えることができます。たとえば、広く使用されている Kerberos のメカニズムを使用して認証を行うため、認証サーバをカスタマイズすることもできます。

クライアントの名前付け

クライアント・プロセスは、ATMI アプリケーションに参加すると、ユーザ・クライアント名 (ユーザとクライアントを組み合わせた名前) およびアプリケーション・キー (一意なクライアント識別名) を持ちます。

- ユーザ・クライアント名は、ユーザ名とクライアント名で構成され、セキュリティ、管理、および通信に使用されます。
- アプリケーション・キーは 32 ビット値であり、クライアントの代わりに呼び出され、アクセス制御のチェック機能で使用されます。

`tpsysadm` および `tpsysop` という 2 つのクライアント名が、特殊なセマンティクス用に用意されています。`tpsysadm` は、アプリケーション管理者として扱われます。`tpsysop` は、アプリケーション・オペレータとして扱われます。

ユーザ / クライアント名

認証されたクライアントは、ATMI アプリケーションに参加すると、ユーザ名とクライアント名を TPINIT バッファの tpinit(3c) に渡すか (アプリケーションが C で記述されている場合)、または、TPINFDEF-REC レコードの TPINITIALIZE(3cbl) に渡します (アプリケーションが COBOL で記述されている場合)。次の表では、ユーザ名とクライアント名、および、TPINIT バッファまたは TPINFDEF-REC レコード内のその他のセキュリティ関連フィールドを説明します。

表 1-7 TPINIT バッファおよび TPINFDEF-REC レコードのセキュリティ関連フィールド

TPINIT	TPINFDEF-REC	説明
usrname	USRNAME	30 文字までの文字列で構成するユーザ名。USER_AUTH、ACL、または MANDATORY_ACL のセキュリティ・レベルには必須です。ユーザ名は呼び出し側を表します。
cltname	CLTNAME	30 文字までの文字列で構成するクライアント名。USER_AUTH、ACL、または MANDATORY_ACL のセキュリティ・レベルには必須です。クライアント名はクライアント・プログラムを表します。
passwd	PASSWD	アプリケーション・パスワード。APP_PW、USER_AUTH、ACL、または MANDATORY_ACL のセキュリティ・レベルには必須です。tpinit() または TPINITIALIZE() は、このパスワードを TUXCONFIG ファイル * に格納された設定済みのアプリケーション・パスワードと比較して検証します。
datalen	DATALEN	後に続くユーザ固有のデータ ** の長さ。

* バイナリ形式の UBBCONFIG ファイル。

** 通常はユーザ・パスワード。

表 1-7 TPINIT バッファおよび TPINFDEF-REC レコードのセキュリティ関連フィールド (続き)

TPINIT	TPINFDEF-REC	説明
data	N/A	ユーザ固有のデータ **。USER_AUTH、ACL、または MANDATORY_ACL のセキュリティ・レベルには必須です。tpinit() または TPINITIALIZE() は、ユーザ固有のデータを認証サーバに転送し、検証します。認証サーバは AUTHSVR です。

* バイナリ形式の UBBCONFIG ファイル。

** 通常はユーザ・パスワード。

認証されたセキュリティ・レベル (USER_AUTH、ACL、または MANDATORY_ACL) では、ユーザ名、クライアント名、およびユーザ固有のデータは、BEA Tuxedo システムで変換されることなく AUTHSVR に転送されます。これらの情報が変換されるとすれば、ワークステーション・クライアントからネットワーク経由で送信されるときに暗号化が行われる場合のみです。

アプリケーション・キー

クライアントが ATMI アプリケーションに参加するたびに、BEA Tuxedo システムはクライアントに 32 ビットのアプリケーション・キーを割り当てます。クライアントは、アプリケーションとの関連付けを解除し、別のユーザとして ATMI アプリケーションに参加しない限り、このアプリケーション・キーをリセットできません。

割り当てられたアプリケーション・キーは、クライアントのセキュリティ・クリデンシャルです。クライアントは、TPSVCINFO 構造体の一部として、すべてのサービス呼び出しに appkey フィールドを指定します。TPSVCINFO の詳細については、『BEA Tuxedo C 言語リファレンス』の tpservice(3c) を参照してください。

次の表は、さまざまなセキュリティ・レベルとクライアントに割当てられるアプリケーション・キーを示します。アプリケーション・キーの割り当ては、最後の項目を除き、すべてハードコーディングされます。

表 1-8 アプリケーション・キーの割り当て

セキュリティ・レベル	メッセージのタイプ	割り当てられるアプリケーション・キー
任意のセキュリティ・レベル	管理者 (tmadmin(1) など) が起動する、ネイティブな ATMI クライアントからのメッセージ	0x80000000 (管理者のアプリケーション・キー)
NONE または APP_PW	tpsysadm というクライアント名で tpinit() または TPINITIALIZE() を呼び出し、管理者によって実行されるネイティブな ATMI クライアントからのメッセージ	0x80000000 (管理者のアプリケーション・キー)
	tpsysop というクライアント名で tpinit() または TPINITIALIZE() を呼び出し、管理者によって実行されるネイティブな ATMI クライアントからのメッセージ	0xC0000000 (オペレータのアプリケーション・キー)
	tpsysadm および tpsysop 以外の任意の ATMI クライアントからのメッセージ	-1

表 1-8 アプリケーション・キーの割り当て (続き)

セキュリティ・レベル	メッセージのタイプ	割り当てられるアプリケーション・キー
USER_AUTH、ACL、または MANDATORY_ACL	tpsysadm というクライアント名で tpinit() または TPINITIALIZE() を呼び出し、管理者とバイパス認証によって実行されるネイティブな ATMI クライアントからのメッセージ	0x80000000 (管理者のアプリケーション・キー)
	tpsysadm というクライアント名で tpinit() または TPINITIALIZE() を呼び出す認証済み ATMI クライアントからのメッセージ	0x80000000 (管理者のアプリケーション・キー)
	tpsysop というクライアント名で tpinit() または TPINITIALIZE() を呼び出す認証済み ATMI クライアントからのメッセージ	0xC0000000 (オペレータのアプリケーション・キー)
	tpsysadm や tpsysop 以外のクライアント名で tpinit() または TPINITIALIZE() を呼び出す認証済み ATMI クライアントからのメッセージ	アプリケーション・キーは、下位 17 ビットのユーザ識別子 (UID) と次の上位 14 ビットのグループ識別子 (GID) です。残りの上位ビットは 0 です。AUTHSVR は、このアプリケーション・キーの値を返します。

さらに、C プログラムの tpsvrinit(3c) または tpsvrdone(3c) (COBOL では TPSVRINIT(3cbl) または TPSVRDONE(3cbl)) から発信されるメッセージには、管理者のアプリケーション・キーである 0x80000000 が割り当てられます。クライアントのアプリケーション・キーは、クライアントからサーバに送信されるメッセージに割り当てられます。この規則の例外については、第 1 章の 11 ページ「クライアント・トークンとサーバ・トークンの交換」を参照してください。

ユーザ識別子 (UID) は、特定のユーザを示すため、アプリケーション側で使用される識別子であり、0 ～ 128K の整数で表されます。グループ識別子 (GID) は、アプリケーション・グループを示すため、アプリケーション側で使用される識別子であり、0 ～ 16K の整数で表されます。

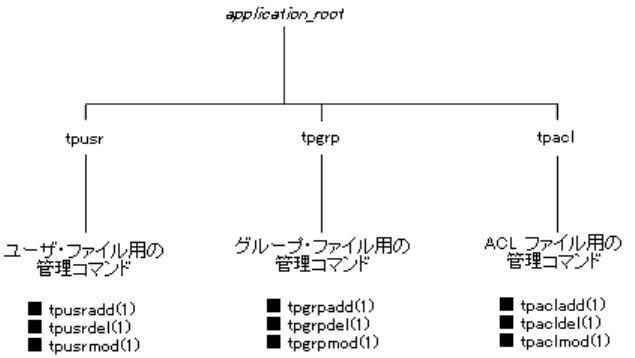
ユーザ、グループ、および ACL のファイル

アクセス制御機能を使用するため、アプリケーション管理者は、(1) ユーザ、(2) グループ、および (3) グループとアプリケーション・エンティティ (サービス、イベント、およびアプリケーション・キューなど) のマッピング・リストを管理する必要があります。3 つ目のアプリケーション・エンティティへのグループのマッピング・リストは、アクセス制御リスト (ACL) と呼ばれます。

クライアントがサービスなどのアプリケーション・リソースに対してアクセスを試みると、システム側では、クライアントのアプリケーション・キーをチェックし、ユーザが属するグループを識別します。次に、システムは、ACL にあるターゲット・リソースをチェックし、そのリソースに対するアクセス権がクライアントのグループに付与されているかどうかを判別します。アプリケーション管理者、アプリケーション・オペレータ、およびアプリケーション管理者またはオペレータの権限で実行中のプロセスやサービス要求は、ACL によるパーミッションのチェックの対象にはなりません。

ユーザ、グループ、および ACL のファイルは、*application_root* ディレクトリに置かれています。*application _root* は、APPDIR 変数に定義された最初のパス名です。次の図は、これらのファイルと、各リストを制御するための管理コマンドを示しています。

図 1-10 デフォルトのユーザ、グループ、および ACL のファイル



注記 Compaq VMS オペレーティング・システムで動作する ATMI アプリケーションでは、ユーザ、グループ、および ACL ファイルの名前に .dat 拡張子が付きます (tpusr.dat、tpgrp.dat、および tpacl.dat など)。

これらのファイルは、コロンで区切られたフラットなテキスト・ファイルであり、アプリケーション管理者 (TUXCONFIG 変数が参照する TUXCONFIG ファイルの所有者) だけが読み書きできます。これらのファイルは一連の専用コマンドで完全に管理されるため、ファイルの形式は無関係です。これらのコマンドを使用できるのは、アプリケーション管理者だけです。

アプリケーション管理者は、tpaclcvt(1) コマンドを使用して、セキュリティに関するデータ・ファイルを ACL チェック用の形式に変換できます。たとえば、UNIX ホスト・マシンで tpaclcvt を実行して /etc/password ファイルを変換し、変換後のファイルを tpusr ファイルに保存できます。この管理者は、tpaclcvt を実行して /etc/group ファイルを変換し、変換後のファイルを tpgrp ファイルに保存することもできます。

AUTHSVR サーバは、tpusr ファイル内のユーザ情報を使用して、ATMI アプリケーションに参加しようとするユーザを認証します。

オプションの ACL と必須の ACL

ACL および MANDATORY_ACL のセキュリティ・レベルは、BEA Tuxedo 製品の ATMI 環境用の、デフォルトの認可インプリメンテーションです。

セキュリティ・レベルが ACL の場合、ターゲット・アプリケーションのエンティティに関連するエントリが tpacl ファイルになくても、クライアントはそのエンティティにアクセスできます。管理者は、このセキュリティ・レベルで、より高度なセキュリティを必要とするリソースに対してだけアクセスを設定できます。つまり、すべてのユーザにアクセスを許可するサービス、イベント、およびアプリケーション・キューについて、tpacl ファイルにエントリを追加する必要はありません。

一方、セキュリティ・レベルが MANDATORY_ACL の場合、ターゲット・アプリケーションのエンティティに関連するエントリが tpacl ファイルにないと、クライアントはそのエンティティにアクセスできません。したがって、このレベルは「必須」と呼ばれます。クライアントがアクセスを必要とするアプリケーション・エンティティごとに、tpacl ファイル内にエントリが必要です。

ACL および MANDATORY_ACL のセキュリティ・レベルで、クライアントが、tpacl ファイルにエントリがあるエンティティに対してアクセスしようとする場合、クライアントに関連するユーザは、そのエンティティへのアクセスを許可されたグループのメンバでなければなりません。そうでない場合、アクセスは拒否されます。

ATMI アプリケーションによっては、システム・レベルとアプリケーション・レベルの両方の認可を使用することが必要な場合もあります。tpacl ファイルのエントリを使用すると、サービスに対するアクセスを許可するユーザを制限できます。また、アプリケーションのロジックを使用して、データ依存型のアクセスを制御することもできます。たとえば、100 万ドル以上のトランザクションを処理するユーザを指定することができます。

先頭にピリオド (.) が付く管理サービス、イベント、およびアプリケーション・キューは、ACL によるパーミッションのチェック対象にはなりません。たとえば、.SysMachineBroadcast、.SysNetworkConfig、および .SysServerCleaning などの管理イベントには、どのクライアントもサブスクライブできます。さらに、アプリケーション管理者、アプリケーション・オペレータ、およびアプリケーション管理者またはオペレータの権限で実行中のプロセスやサービス要求は、ACL によるパーミッションのチェックの対象にはなりません。

関連項目

- 第 2 章の 1 ページ「セキュリティの管理とは」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 第 2 章の 9 ページ「認証の管理」
- 第 2 章の 33 ページ「認可の管理」
- 第 3 章の 1 ページ「セキュリティのプログラミングとは」
- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」
- 第 3 章の 4 ページ「ATMI アプリケーションにクライアント・プログラムを参加させるためのセキュリティ・コードの記述方法」
- 『BEA Tuxedo アプリケーションの設定』の 第 2 章の 1 ページ「コンフィギュレーション・ファイルについて」と 第 3 章の 1 ページ「コンフィギュレーション・ファイルの作成」
- 『ファイル形式、データ記述方法、MIB、およびシステム・プロセスのリファレンス』の UBBCONFIG(5)
- 『ファイル形式、データ記述方法、MIB、およびシステム・プロセスのリファレンス』の AUTHSVR(5)

セキュリティの相互運用性

ATMI アプリケーションを BEA Tuxedo リリース 7.1 より前 (6.5 以前) の BEA Tuxedo ソフトウェアと相互運用させる場合、アプリケーションの開発者および管理者は、いくつかのセキュリティに関する問題を認識しておく必要があります。

この節で説明する「相互運用性」とは、最新版の BEA Tuxedo ソフトウェアが、以前のリリースの BEA Tuxedo ソフトウェアとネットワーク経由で通信する機能のことです。具体的には、「ドメイン間の相互運用性」と「ドメイン内の相互運用性」があり、それぞれ次のことを意味します。

- ドメイン間の相互運用性

  BEA Tuxedo リリース 7.1 以上を実行する ATMI アプリケーションが、BEA Tuxedo リリース 7.1 より前のバージョンのソフトウェアを実行する別の ATMI アプリケーションと相互運用することです。第 1 章の 54 ページ「ドメイン間の相互運用性」を参照してください。

- ドメイン内の相互運用性

  ATMI のアプリケーションが複数のマシンで構成されているときに、そのアプリケーション内で、BEA Tuxedo リリース 7.1 以上を実行するマシンと BEA Tuxedo リリース 7.1 より前のバージョンを実行する別のマシンが相互運用することです。第 1 章の 55 ページ「ドメイン内の相互運用性」を参照してください。

図 1-11 ドメイン間の相互運用性

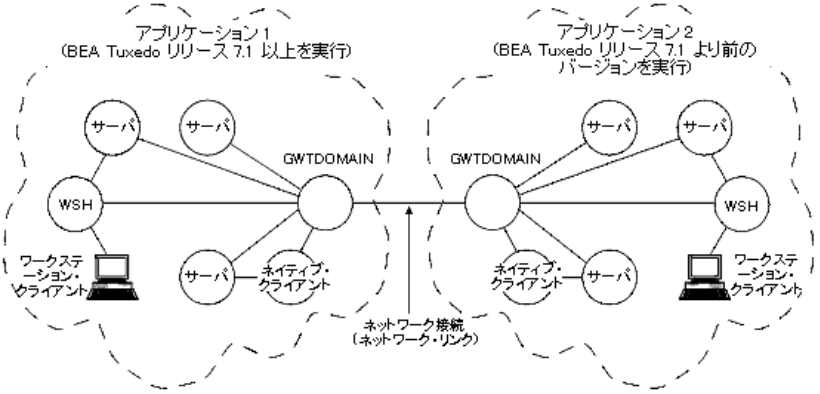
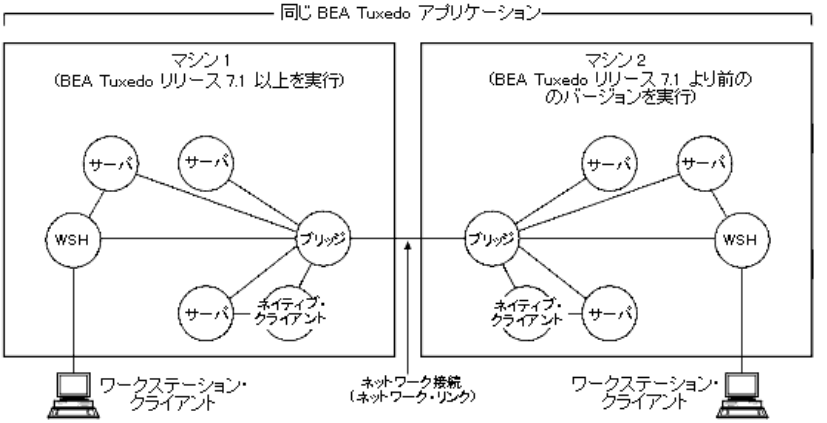


図 1-12　ドメイン内の相互運用性



BEA Tuxedo リリース 7.1 より前のバージョンとの相互運用性

BEA Tuxedo リリース 7.1 より前のバージョンのソフトウェアと相互運用するかどうかは、認証のセキュリティ・レベルによって決まります。BEA Tuxedo リリース 7.1 以上のソフトウェアでインプリメントされる認証機能では、通信プロセスにより相互の ID が証明されます。

デフォルトでは、リリース 7.1 より前の BEA Tuxedo ソフトウェアを実行しているマシンとの相互運用は許可されません。このデフォルト設定を変更するには、アプリケーション管理者が `CLOPT -t` オプションを使用して、リリース 7.1 以上の ATMI アプリケーション内のワークステーション・ハンドラ (WSH)、ドメイン・ゲートウェイ (`GWTDOMAINS`)、およびサーバを、リリース 7.1 より前の BEA Tuxedo ソフトウェアと相互運用できるようにします。`CLOPT -t` オプションの使い方、および `CLOPT -t` を使用するときの認証と認可に関するセキュリティ問題については、第 2 章の 16 ページ「相互運用性の方針の指定」を参照してください。

リンク・レベルの暗号化の相互運用性

BEA Tuxedo ソフトウェアを実行するマシン間でネットワーク・リンクが確立されると、リンク・レベルの暗号化を使用してデータを暗号化してからネットワーク・リンク経由で送信し、データがネットワーク・リンクを離れると復号化することができます。ただし、リンク・レベルの暗号化は、送信側と受信側の両方のマシンの両方に LLE がインストールされている場合にのみ使用できます。

LLE と BEA Tuxedo リリース 7.1 より前のソフトウェアとの相互運用性については、第 1 章の 26 ページ「LLE の旧バージョンとの互換性」を参照してください。

公開鍵によるセキュリティ機能の相互運用性

公開鍵によるセキュリティに関する以下の相互運用性の規則は、BEA Tuxedo 7.1 以上を実行するマシンと、BEA Tuxedo 7.1 より前のバージョンを実行するマシンが相互運用する場合に適用されます。規則の内容を明確にするため、各規則では、BEA Tuxedo 7.1 より前のバージョンを実行するワークステーション・クライアントの例を示しています。

ドメイン間の相互運用性の場合、リリース 7.1 以上のドメイン・ゲートウェイ (GWTDOMAIN) プロセスには、公開鍵によるセキュリティの相互運用性の規則が適用されます。

表 1-9 公開鍵によるセキュリティ機能の相互運用性に関する規則

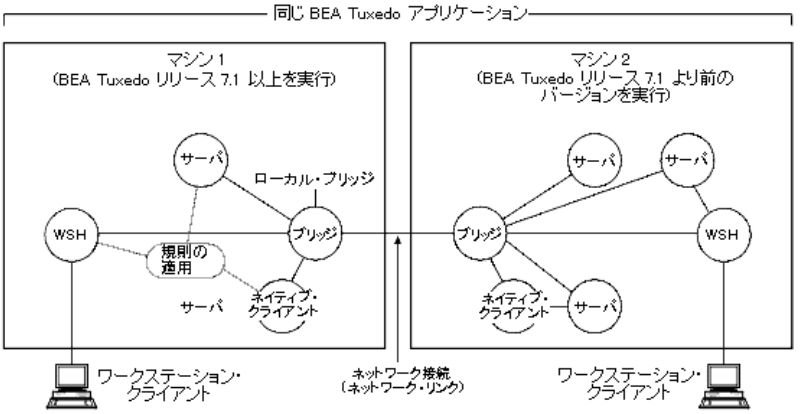
相互運用性の規則	例	備考
BEA Tuxedo リリース 7.1 より前のソフトウェアを実行するマシン宛の暗号化済みのメッセージ・バッファは、マシンに送信されません。	BEA Tuxedo リリース 7.1 より前のワークステーション・クライアント宛の暗号化済みのメッセージ・バッファは、ワークステーション・クライアントに送信されません。	「暗号化済み」とは、リンク・レベルの暗号化ではなく、メッセージ・ベースの公開鍵の暗号化を意味します。

表 1-9 公開鍵によるセキュリティ機能の相互運用性に関する規則 (続き)

相互運用性の規則	例	備考
BEA Tuxedo リリース 7.1 より前のソフトウェアを実行するマシンから受信したメッセージ・バッファは、暗号化を必要とするプロセスに転送されると、受け付けられません。	BEA Tuxedo リリース 7.1 より前のワークステーション・クライアントからの受信したメッセージ・バッファには、暗号化エンベロープが添付されておらず、暗号化を必要とするプロセスに転送された場合は、受け付けられません。	ENCRYPTION_REQUIRED コンフィギュレーション・パラメータについては、第 2 章の 46 ページ「暗号化方針の設定」を参照してください。
BEA Tuxedo リリース 7.1 より前のソフトウェアを実行するマシン宛に送信されたメッセージ・バッファは、デジタル署名が検証および削除されてから、古いマシンに送信されます。	デジタル署名は、検証されると、BEA Tuxedo リリース 7.1 より前のワークステーション・クライアント宛の送信メッセージ・バッファから削除されます。	送信するメッセージ・バッファは、デジタル署名されていても暗号化はされていないと見なされます。送信メッセージ・バッファがデジタル署名されており、さらに暗号化されている場合、そのメッセージは復号化されず、デジタル署名は検証されないため、メッセージは古いマシンに送信されません。
リリース 7.1 より前の BEA Tuxedo ソフトウェアを実行しているマシンからの着信メッセージ・バッファは、デジタル署名を必要とするプロセスに転送された場合は、受け付けられません。	BEA Tuxedo リリース 7.1 より前のワークステーション・クライアントからの受信メッセージ・バッファには、デジタル署名が添付されていないため、デジタル署名を必要とするプロセスに転送された場合は受け付けられません。	SIGNATURE_REQUIRED コンフィギュレーション・パラメータについては、第 2 章の 41 ページ「デジタル署名方針の設定」を参照してください。

ドメイン内の相互運用性の場合、リリース 7.1 以上のネイティブ・クライアント、ワークステーション・ハンドラ (WSH)、またはローカル・ブリッジ・プロセスと通信するサーバ・プロセスには、次の図のような、公開鍵によるセキュリティの相互運用性の規則が適用されます。ブリッジ・プロセスは、パイプ役を果たすだけであり、メッセージ・バッファの内容の復号化やデジタル署名の検証は行いません。

図 1-13 公開鍵によるセキュリティ機能の相互運用性規則の適用



注記 一般に、リリース 7.1 以上の WSH はデジタル署名を検証しません。しかし、リリース 7.1 より前の BEA Tuxedo ソフトウェアを実行しているプロセスにデジタル署名付きのメッセージ・バッファを転送すると、WSH は、デジタル署名を検証してから削除します。

関連項目

- 第 1 章の 58 ページ「セキュリティ機能の互換性」
- 第 2 章の 16 ページ「相互運用性の方針の指定」
- 第 2 章の 41 ページ「デジタル署名方針の設定」
- 第 2 章の 46 ページ「暗号化方針の設定」

セキュリティ機能の互換性

BEA Tuxedo リリース 7.1 以上のソフトウェアを実行する ATMI アプリケーションでは、デフォルトまたはカスタマイズした認証、認可、監査、および公開鍵セキュリティを自由に組み合わせることができます。さらに、これらの 4 つのセキュリティ機能の任意の組み合わせは、リンク・レベルの暗号化とも互換性があります。

デフォルトまたはカスタマイズした認証と認可の組み合わせ

認可セキュリティ・トークンに少なくとも (1) 認証されたユーザ名またはプリンシパル名、および (2) アプリケーション・キーの値 (第 1 章の 48 ページ「アプリケーション・キー」で定義) が必要であることがアプリケーション開発者側で認識されていれば、デフォルトの認証とカスタマイズした認可、またはカスタマイズした認証とデフォルトの認可を組み合わせて使用できます。

認可に関する一部の決定は、認可トークンに格納されているユーザ ID に基づいて行われます。認可トークンは、認証プラグインによって生成されるため、認証プラグインと認可プラグインのプロバイダは、これらのプラグインが協調動作することを保証する必要があります。詳細については、第 1 章の 7 ページ「認証」および 第 1 章の 13 ページ「認可」を参照してください。

デフォルトまたはカスタマイズした認証と監査の組み合わせ

監査セキュリティ・トークンに少なくとも (1) 認証されたユーザ名またはプリンシパル名、および (2) アプリケーション・キーの値 (第 1 章の 48 ページ「アプリケーション・キー」で定義) が必要であることがアプリケーション開発者側で認識されていれば、デフォルトの認証とカスタマイズした監査、またはカスタマイズした認証とデフォルトの監査を組み合わせて使用できます。

監査に関する一部の決定は、監査トークンに格納されているユーザ ID に基づいて行われます。監査トークンは、認証プラグインによって生成されるため、認証プラグインと監査プラグインのプロバイダは、これらのプラグインが協調動作することを保証する必要があります。詳細については、第 1 章の 7 ページ「認証」および 第 1 章の 19 ページ「監査」を参照してください。

公開鍵によるセキュリティ機能の互換性の問題

公開鍵によるセキュリティは、BEA Tuxedo リリース 7.1 以上のソフトウェアでサポートされる、圧縮機能以外のすべての機能やプロセスと互換性があります。暗号化されたメッセージ・バッファは、圧縮機能を使用しても圧縮できません。ただし、公開鍵によるソフトウェアでは、メッセージ・バッファを暗号化する前に内容が圧縮されるため、ある程度サイズを節約できます。

この節では、公開鍵によるセキュリティと、次の ATMI の機能またはプロセスとの互換性および相互運用性について説明します。

- データ依存型ルーティング
- スレッド
- イベント・ブローカ
- /Q
- トランザクション
- ドメイン・ゲートウェイ (GWTDOMAIN)
- その他のベンダのゲートウェイ

データ依存型ルーティングとの互換性および相互運用性

データ依存型ルーティング機能の中心となるのは、プロセスが、受信したメッセージ・バッファの内容を調べることです。受信したメッセージ・バッファが暗号化されている場合、データ依存型ルーティング用に設定されたプロセスでは、受信者の秘密鍵を開き、公開鍵ソフトウェアがそのキーを使用してメッセージ・バッファを復号化できるようにしておく必要があります。データ依存型ルーティングでは、公開鍵ソフトウェアは、デジタル署名を検証しません。

復号化キーを使用できない場合、ルーティング操作は失敗します。システムからは、userlog(3c) エラー・メッセージが返され、操作が失敗したことがレポートされます。

復号化キーを使用できる場合、プロセスは、暗号化されたメッセージ・バッファを復号化したコピーに基づいて、ルーティングに関する決定を行います。次の手順で実行されます。

1. 公開鍵ソフトウェアは、暗号化されたメッセージ・バッファのコピーを作成し、復号化キーを使用してそのバッファを復号化します。
2. プロセスは、これによって得られた平文 (暗号化されていないテキスト) のメッセージ内容を読み取り、ルーティングに関する決定を行います。
3. 公開鍵ソフトウェアは、プライバシーを保護するため、平文のメッセージ内容をゼロ値で上書きします。

その後、システムは、ルーティングに関する決定に基づいて、元の暗号化されたメッセージ・バッファを送信します。

スレッドとの互換性および相互運用性

公開鍵と秘密鍵は、ハンドルを介して表現および操作されます。ハンドルには、それに関連付けられたデータがあります。公開鍵のアプリケーション・プログラミング・インターフェイス (API: Application Programming Interface) では、ハンドルのデータを使用することにより、ハンドルで指定される項目の検索やアクセスを行います。プロセスは、デジタル署名の生成、メッセージの暗号化、およびメッセージの復号化のために、キー・ハンドルを開きます。

キー・ハンドルはプロセスのリソースです。特定のスレッドやコンテキストにバインドされることはありません。キーを開くために必要な通信は、スレッドで現在アクティブなコンテキストの内部で実行されます。その後は、コンテキストが同じ ATMI アプリケーションに関連付けられているかどうかとは無関係に、プロセス内のどのコンテキストでもそのキーを使用できます。

キーの内部データ構造はスレッド・セーフです。つまり、複数のスレッドが 1 つのキーに同時にアクセスできます。

イベント・ブローカとの互換性および相互運用性

一般に、`TMUSREVT(5)` システム・サーバは、暗号化されたメッセージ・バッファを復号化しないで処理します。つまり、メッセージが BEA Tuxedo のイベント・ブローカ・コンポーネントを通過しても、デジタル署名と暗号化エンベロープは何の影響も受けません。ただし、以下の場合、イベント・ブローカ・コンポーネントは、ポストされたメッセージ・バッファを復号化する必要があります。

- メッセージの内容に基づいてサブスクリプションのフィルタ式を評価する場合

  イベント・ブローカが適切な復号化キーにアクセスできない場合、そのサブスクリプションのフィルタ式は false と見なされ、サブスクリプションは不一致と見なされます。

- メッセージの内容へのアクセスが必要な、サブスクリプションの通知アクション (`userlog(3c)` プロセスまたはシステム・コマンド) を実行する場合

  イベント・ブローカが適切な復号化キーにアクセスできない場合、そのサブスクリプションの通知アクションは失敗し、システムからは `userlog(3c)` エラー・メッセージが返されて、処理が失敗したことがレポートされます。

- システム構成に基づき、データ依存型ルーティング用のメッセージ内容にアクセスする、サブスクリプションの通知アクションを実行する場合

  イベント・ブローカが適切な復号化キーにアクセスできない場合、そのサブスクリプションの通知アクションは失敗し、システムからは userlog() エラー・メッセージが返されて、処理が失敗したことがレポートされます。

  トランザクション型のサブスクリプションの場合、トランザクションは「ロールバックのみ」とマーキングされます。

- 暗号化を必要とする管理システムの方針に合わせる場合 (第 2 章の 46 ページ「暗号化方針の設定」を参照)

  イベント・ブローカが適切な復号化キーにアクセスできない場合、tppost(3c) 操作は失敗し、システムからは userlog() エラー・メッセージが返されて、処理が失敗したことがレポートされます。

- デジタル署名を必要とする管理システムの方針に従い、ポストされた暗号化メッセージに有効なデジタル署名が添付されているかどうかを検証する場合 (第 2 章の 41 ページ「デジタル署名方針の設定」を参照)

  イベント・ブローカが適切な復号化キーにアクセスできない場合、tppost(3c) 操作は失敗し、システムからは userlog() エラー・メッセージが返されて、処理が失敗したことがレポートされます。

/Q との互換性および相互運用性

一般に、TMQUEUE(5) または TMQFORWARD(5) のシステム・サーバは、暗号化されたメッセージ・バッファを復号化しないで処理します。つまり、メッセージが BEA Tuxedo の /Q コンポーネントを通過しても、署名と暗号化エンベロープは何の影響も受けません。ただし、以下の場合、/Q コンポーネントは、キューに登録されたメッセージ・バッファを復号化する必要があります。

- システム構成に基づき、データ依存型ルーティング用のメッセージ内容にアクセスする必要のある、TMQFORWARD を実行する場合

  TMQFORWARD が適切な復号化キーにアクセスできない場合、転送操作は失敗します。システムからはメッセージがキューに返され、userlog(3c) エラー・メッセージを生成して失敗をレポートします。

  再試行を周期的に何度か繰り返すと、TMQFORWARD により、読み込み不可能なメッセージがエラー・キューに格納される場合があります。

- 暗号化を必要とする管理システムの方針に合わせる場合 (第 2 章の 46 ページ「暗号化方針の設定」を参照)

  /Q コンポーネントが適切な復号化キーにアクセスできない場合、tpenqueue(3c) 操作は失敗し、システムからは userlog() エラー・メッセージが返されて、処理が失敗したことがレポートされます。

- デジタル署名を必要とする管理システムの方針に従い、キューに登録された暗号化メッセージに有効な署名が添付されているかどうかを検証する場合 (第 2 章の 41 ページ「デジタル署名方針の設定」を参照)

  /Q コンポーネントが適切な復号化キーにアクセスできない場合、tpenqueue(3c) 操作は失敗し、システムからは userlog() エラー・メッセージが返されて、処理が失敗したことがレポートされます。

呼び出し側プロセスに有効な復号化キーがないと、非トランザクション型の tpdequeue(3c) 操作により、キューに登録された暗号化済みメッセージが破壊されます。

無効な署名が添付されたメッセージがキューに登録された場合 (またはそのメッセージがキュー内で破壊または改ざんされた場合) にそのメッセージをキューから取り出そうとすると失敗します。非トランザクション型の tpdequeue() 操作は、このようなメッセージを破壊します。トランザクション型の tpdequeue() 操作を行うと、トランザクションのロールバックが発生し、以降、キューからメッセージを取り出そうとしても失敗します。

トランザクションとの互換性および相互運用性

公開鍵によるセキュリティ操作 (キーの開閉、デジタル署名の要求、暗号化の要求) はトランザクション型ではないため、トランザクションのロールバックによって元に戻すことはできません。ただし、トランザクションは、公開鍵の操作時に発生する以下の障害によってロールバックされる場合があります。

- トランザクション型の要求または応答メッセージを復号化できない場合、関連するトランザクションはロールバックされます。
- デジタル署名が無効か、または欠落していたためにトランザクション型の要求または応答メッセージが廃棄された場合、関連するトランザクションはロールバックされます。
- 暗号化またはデジタル署名を必要とする管理システムの方針に違反したため、トランザクション型の要求または応答メッセージが拒否された場合、関連するトランザクションはロールバックされます。

ドメイン・ゲートウェイとの互換性および相互運用性

BEA Tuxedo リリース 7.1 以上のソフトウェアを実行する 2 つの ATMI アプリケーションを接続するドメイン・ゲートウェイ (GWTDOMAIN) プロセスには、デジタル署名および暗号化エンベロープがあります。さらに、これらのドメイン・ゲートウェイ・プロセスは、デジタル署名を検証し、デジタル署名と暗号化に関する管理システム方針を適用します。

次の図は、ドメイン・ゲートウェイ・プロセスが、ローカルおよびリモートの ATMI アプリケーションと対話する様子を示しています。この図に続く表では、デジタル署名が添付された暗号化メッセージ・バッファが、リリース 7.1 以上のドメイン・ゲートウェイ・プロセスでどのように扱われるかを示しています。

図 1-14 ATMI アプリケーション間の通信

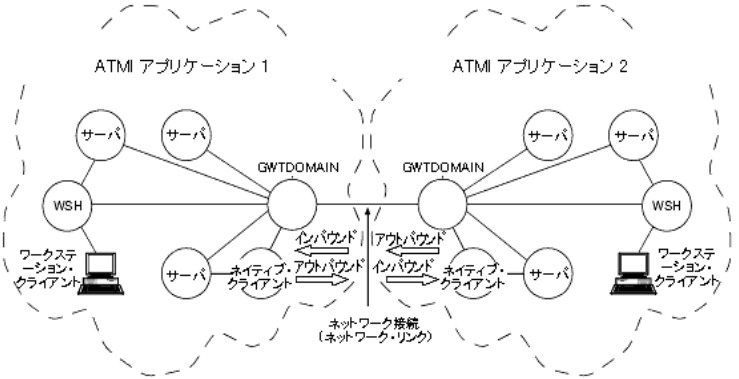


表 1-10 リリース 7.1 以上のドメイン・ゲートウェイ (GWTDOMAIN) プロセスの動作

メッセージのタイプ	条件	結果
インバウンド・メッセージ (リモート・プロセスから発信され、ネットワーク接続を経由して受信されるメッセージ)	暗号化エンベロープあり。デジタル署名はどちらでも可。	ドメイン・ゲートウェイ・プロセスはメッセージを受け付け、暗号化された形式で転送します。データ依存型ルーティング機能が適用され、ドメイン・ゲートウェイ・プロセスが適切な復号化キーを持たない場合、ゲートウェイ・プロセスはメッセージを拒否します。(詳細については、第 1 章の 60 ページ「データ依存型ルーティングとの互換性および相互運用性」を参照)

表 1-10 リリース 7.1 以上のドメイン・ゲートウェイ (GWTDOMAIN) プロセスの動作 (続き)

メッセージのタイプ	条件	結果
インバウンド・メッセージ	暗号化エンベロープもデジタル署名もなし。	ドメイン・ゲートウェイ・プロセスが、暗号化を必要とするグループ、ドメイン、またはマシンの内部で実行されている場合、ゲートウェイ・プロセスはメッセージを拒否します。ドメイン・ゲートウェイによって宣言されたサービスが暗号化を必要とする場合、ゲートウェイ・プロセスはメッセージを拒否します。(詳細については、第 2 章の 41 ページ「デジタル署名方針の設定」を参照) ドメイン・ゲートウェイが暗号化を必要としない場合、ゲートウェイ・プロセスはメッセージを受け付け、転送します。
インバウンド・メッセージ	デジタル署名あり。暗号化なし。	ドメイン・ゲートウェイ・プロセスは、デジタル署名を検証し、メッセージにデジタル署名を添付して転送します。
インバウンド・メッセージ	デジタル署名なし。暗号化なし。	ドメイン・ゲートウェイ・プロセスが、デジタル署名を必要とするグループ、ドメイン、またはマシンの内部で実行されている場合、ゲートウェイ・プロセスはメッセージを拒否します。ドメイン・ゲートウェイによって宣言されたサービスがデジタル署名を必要とする場合、ゲートウェイ・プロセスはメッセージを拒否します。(詳細については、第 2 章の 41 ページ「デジタル署名方針の設定」を参照) ドメイン・ゲートウェイがデジタル署名を必要としない場合、ゲートウェイ・プロセスはメッセージを受け付け、転送します。

表 1-10 リリース 7.1 以上のドメイン・ゲートウェイ (GWTDOMAIN) プロセスの動作 (続き)

メッセージのタイプ	条件	結果
アウトバウンド・メッセージ (ローカル・プロセスから発信され、ネットワーク接続を経由して転送されるメッセージ)	暗号化エンベロープあり。デジタル署名はどちらでも可。	ドメイン・ゲートウェイ・プロセスはメッセージを受け付け、暗号化してネットワーク経由で転送します。 データ依存型ルーティング機能が適用され、ドメイン・ゲートウェイ・プロセスが適切な復号化キーを持たない場合、ゲートウェイ・プロセスはメッセージを拒否します。(詳細については、第 1 章の 60 ページ「データ依存型ルーティングとの互換性および相互運用性」を参照) 暗号化されたメッセージが、BEA Tuxedo リリース 7.1 より前の (6.5 以前) BEA Tuxedo ソフトウェアを実行するプロセスに送信される場合、ドメイン・ゲートウェイ・プロセスはメッセージを拒否します。(詳細については、第 1 章の 55 ページ「BEA Tuxedo リリース 7.1 より前のバージョンとの相互運用性」および 第 1 章の 56 ページ「公開鍵によるセキュリティ機能の相互運用性」を参照)
アウトバウンド・メッセージ	暗号化エンベロープもデジタル署名もなし。	ドメイン・ゲートウェイ・プロセスが、暗号化を必要とするグループ、ドメイン、またはマシンの内部で実行されている場合、ゲートウェイ・プロセスはメッセージを拒否します。ドメイン・ゲートウェイによって宣言されたサービスが暗号化を必要とする場合、ゲートウェイ・プロセスはメッセージを拒否します。(詳細については、第 2 章の 46 ページ「暗号化方針の設定」を参照) ドメイン・ゲートウェイが暗号化を必要としない場合、ゲートウェイ・プロセスはメッセージを受け付け、ネットワーク経由で転送します。

表 1-10 リリース 7.1 以上のドメイン・ゲートウェイ (GWTDOMAIN) プロセスの動作 (続き)

メッセージのタイプ	条件	結果
アウトバウンド・メッセージ	デジタル署名あり。暗号化なし。	ドメイン・ゲートウェイ・プロセスは、デジタル署名を検証し、メッセージにデジタル署名を添付してネットワーク経由で転送します。 メッセージが、BEA Tuxedo リリース 7.1 より前の BEA Tuxedo ソフトウェアを実行するプロセスに送信され、リリース 7.1 より前の BEA Tuxedo ソフトウェアとの相互運用性が可能であると見なされている場合、ドメイン・ゲートウェイ・プロセスは、デジタル署名の検証と削除を行ってから、ネットワーク経由でメッセージを転送します。(詳細については、第 1 章の 55 ページ「BEA Tuxedo リリース 7.1 より前のバージョンとの相互運用性」および 第 1 章の 56 ページ「公開鍵によるセキュリティ機能の相互運用性」を参照)
アウトバウンド・メッセージ	デジタル署名なし。暗号化なし。	ドメイン・ゲートウェイ・プロセスが、デジタル署名を必要とするグループ、ドメイン、またはマシンの内部で実行されている場合、ゲートウェイ・プロセスはメッセージを拒否します。ドメイン・ゲートウェイによって宣言されたサービスがデジタル署名を必要とする場合、ゲートウェイ・プロセスはメッセージを拒否します。(詳細については、第 2 章の 41 ページ「デジタル署名方針の設定」を参照) ドメイン・ゲートウェイがデジタル署名を必要としない場合、ゲートウェイ・プロセスはメッセージを受け付け、ネットワーク経由で転送します。

ほかのベンダのゲートウェイとの互換性および相互運用性

リリース 7.1 以上の ATMI アプリケーションと、別のベンダのゲートウェイ・プロセスを接続するドメイン・ゲートウェイ (GWTDOMAIN) のプロセスは、アウトバウンド・メッセージ・バッファに対して次のように動作します。

1. 暗号化されたメッセージを復号化します。
2. デジタル署名がある場合は、デジタル署名を検証してから削除します。
3. 平文メッセージをネットワーク経由でベンダのゲートウェイ・プロセスに送信します。

さらに、ドメイン・ゲートウェイ・プロセスは、ATMI アプリケーションの暗号化とデジタル署名に関する管理システムの方針を適用します。たとえば、ATMI アプリケーションのドメイン・レベルで、暗号化またはデジタル署名、あるいはその両方が必要な場合、ローカル・ドメイン・ゲートウェイのプロセスは、ほかのベンダのゲートウェイ・プロセスから受信するすべてのメッセージを拒否します。

関連項目

- 第 1 章の 54 ページ「セキュリティの相互運用性」
- 第 2 章の 16 ページ「相互運用性の方針の指定」
- 第 2 章の 41 ページ「デジタル署名方針の設定」
- 第 2 章の 46 ページ「暗号化方針の設定」

2　セキュリティの管理

ここでは、次の内容について説明します。

- セキュリティの管理とは
- セキュリティ管理のタスク
- BEA Tuxedo レジストリの設定
- セキュリティ用の ATMI アプリケーションのコンフィギュレーション
- 管理環境の設定
- デフォルトの認証と認可の管理

セキュリティの管理とは

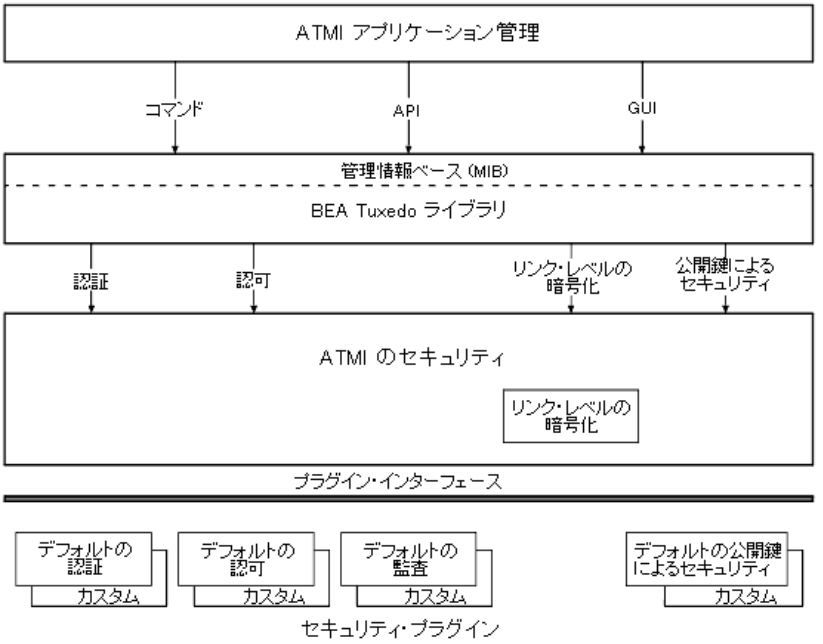
ATMI アプリケーションのセキュリティ管理とは、クライアント、サーバ・マシン、ゲートウェイ・リンクなどのアプリケーションのコンポーネントに対して、セキュリティの方針を設定し、適用することです。ATMI アプリケーションに対するセキュリティ方針はアプリケーション管理者が設定し、これらの方針は、ATMI アプリケーションの基盤となる BEA Tuxedo システムによって実行されます。

BEA Tuxedo システムには、次の ATMI 用のセキュリティ機能が用意されています。

- 認証
- 認可
- 監査
- リンク・レベルの暗号化
- 公開鍵によるセキュリティ機能

アプリケーション管理者は、次の図に示すように、監査以外のすべてのセキュリティ機能を設定できます。

図 2-1 ATMI のセキュリティの管理



関連項目

- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 第 1 章の 1 ページ「セキュリティとは」
- 第 3 章の 1 ページ「セキュリティのプログラミングとは」

セキュリティ管理のタスク

セキュリティ管理には、次のタスクが含まれます。

- BEA Tuxedo レジストリの設定
- セキュリティ用の ATMI アプリケーションのコンフィギュレーション
- 管理環境の設定
- オペレーティング・システム (OS) のセキュリティの管理
- 認証の管理
- 認可の管理
- リンク・レベルの暗号化の管理
- 公開鍵によるセキュリティの管理

関連項目

- 第 2 章の 3 ページ「BEA Tuxedo レジストリの設定」

BEA Tuxedo レジストリの設定

1 つ以上のセキュリティ機能をカスタマイズして ATMI アプリケーションに組み込む場合、アプリケーション管理者は BEA Tuxedo レジストリについて知っておく必要があります。一方、デフォルトのセキュリティ機能だけを ATMI アプリケーションに組み込む場合は、BEA Tuxedo レジストリを変更する必要はありません。

BEA Tuxedo レジストリは、プラグイン・モジュールに関連する情報を格納しておくディスク・ベースのリポジトリです。このレジストリには、デフォルトのセキュリティ・プラグインに関する登録情報が最初に格納されています。

BEA Tuxedo レジストリの目的

BEA のほとんどのミドルウェア製品では、セキュリティなどのコア・サービスのセットで構成される、共通のトランザクション処理のインフラストラクチャ (TP インフラストラクチャ) が使用されています。ATMI アプリケーションでは、適切に定義されたインターフェイスを介して TP インフラストラクチャを使用できます。これらのインターフェイスを使って独自のサービス・コード・モジュールであるプラグイン・モジュール (プラグイン) をロードし、リンクすることにより、アプリケーション管理者は、TP インフラストラクチャのデフォルトの動作を変更できます。

プラグインをロードする最初の手順として、プラグインをホスト・オペレーティング・システムに登録します。プラグインを登録すると、プラグインのエントリが BEA Tuxedo レジストリに追加されます。BEA Tuxedo レジストリは、アクティブなプラグインの情報を格納するバイナリ・ファイルのセットです。レジストリは、BEA Tuxedo システムごとに用意されています。

- UNIX ホスト・マシンの場合、BEA Tuxedo レジストリは `$TUXDIR/udataobj` ディレクトリにあります。
- Windows 2000 ホスト・マシンの場合、BEA Tuxedo レジストリは `%TUXDIR%¥udataobj` ディレクトリにあります。

ATMI アプリケーション内のすべてのワークステーション・クライアントとサーバ・マシンは、同じプラグイン・モジュールのセットを使用する必要があります。

プラグインの登録

カスタマイズしたプラグインを使用する場合、ATMI アプリケーションの管理者はこれらのプラグインを登録し、その他のレジストリ関連のタスクを実行する責任があります。BEA Tuxedo レジストリへのプラグインの登録は、ローカル・マシンからのみ可能です。つまり、管理者は、リモートからホスト・マシンにログオンしている間はプラグインを登録できません。

プラグインの管理では、次の 3 つのコマンドを使用できます。

- `epifreg`―プラグインの登録
- `epifunreg`―プラグインの登録解除
- `epifregedt`―レジストリ情報の編集

これらのコマンドの使用方法については、『Developing Security Services for ATMI and CORBA Environments』を参照してください (このマニュアルには、セキュリティ・プラグインの仕様、およびセキュリティ・プラグインのモジュールを動的にロードし、リンクするためのプラグイン・インターフェイスの説明が掲載されています)。また、カスタマイズしたプラグインをインストールする場合、プラグインの提供元であるサードパーティ・ベンダは、これらのコマンドの使用方法を明記して、カスタマイズしたプラグインにアクセスするための BEA Tuxedo レジストリの設定方法を示す必要があります。

セキュリティ・プラグインの詳細 (インストール手順およびコンフィギュレーション手順を含む) については、日本 BEA システムズ (株) の営業担当者にお問い合わせください。

関連項目

- 第 2 章の 5 ページ「セキュリティ用の ATMI アプリケーションのコンフィギュレーション」

セキュリティ用の ATMI アプリケーションのコンフィギュレーション

アプリケーションが非アクティブの場合、アプリケーション管理者は MASTER マシンで ATMI アプリケーションのセキュリティを設定します。アプリケーションが起動すると、基となる BEA Tuxedo システムにより、ATMI アプリケーション内のほかのマシンにコンフィギュレーション情報が複製転送されます。

管理者は、次のいずれかの方法でアプリケーションのセキュリティを設定できます。

- コンフィギュレーション・ファイル (UBBCONFIG) を編集する
- TM_MIB を変更する
- BEA Administration Console を使用する

セキュリティの種類 (認証、認可、リンク・レベルの暗号化、または公開鍵) およびセキュリティ・ソフトウェアの種類 (デフォルト版またはカスタマイズ版) に応じて、設定するセキュリティ・パラメータは異なります。

コンフィギュレーション・ファイルを編集する

UBBCONFIG コンフィギュレーション・ファイルを編集して、ATMI アプリケーションのセキュリティ方針を設定することができます。UBBCONFIG コンフィギュレーション・ファイルには、どんな名前を付けることもできますが、ファイルの内容は、『ファイル形式、データ記述方法、MIB、およびシステム・プロセスのリファレンス』の UBBCONFIG(5) リファレンス・ページで指定された形式に準拠する必要があります。

UBBCONFIG ファイルおよび TUXCONFIG ファイル (バイナリ形式のコンフィギュレーション・ファイル) の詳細については、『BEA Tuxedo アプリケーションの設定』の 第 2 章の 1 ページ「コンフィギュレーション・ファイルについて」および第 3 章の 1 ページ「コンフィギュレーション・ファイルの作成」を参照してください。

TM_MIB を変更する

TM_MIB でクラスのセットを定義することにより、ATMI アプリケーションの基本的な側面を設定し、管理することができます。クラスは、マシン、サーバ、ネットワークなど、コンポーネントごとに指定されます。管理要求の形式および管理応答の解釈については、管理情報ベース (MIB: Management Information Base) のリファレンス・ページである MIB(5) と TM_MIB(5) リファレンス・ページを両方参照してください。MIB のリファレンス・ページは、『ファイル形式、データ記述方法、MIB、およびシステム・プロセスのリファレンス』で定義されています。

ACL_MIB、DM_MIB、WS_MIB など、ほかの MIB コンポーネントも ATMI アプリケーションのセキュリティ管理に関係があります。ACL_MIB(5) リファレンス・ページでは ACL_MIB、DM_MIB(5) リファレンス・ページでは DM_MIB、WS_MIB(5) リファレンス・ページでは WS_MIB を定義しています。

BEA Tuxedo の MIB の詳細については、まず『ファイル形式、データ記述方法、MIB、およびシステム・プロセスのリファレンス』の MIB(5) を参照してください。『BEA Tuxedo システム入門』も参照してください。

BEA Administration Console を使用する

BEA Administration Console を使用して ATMI アプリケーションのセキュリティ方針を変更することもできます。BEA Administration Console は、アプリケーションをコンフィギュレーションし、監視し、動的に再コンフィギュレーションするための Web ベースのツールです。

BEA Administration Console の詳細については、『BEA Tuxedo システム入門』を参照してください。

関連項目

- 第 2 章の 7 ページ「管理環境の設定」

管理環境の設定

アプリケーション管理者は、アプリケーションのコンフィギュレーション作業の一貫として、ATMI アプリケーション用の環境変数をいくつか定義します。環境変数に定義される値は、BEA Tuxedo の実行可能ファイルおよびデータ・ライブラリを指す絶対パス名です。

ATMI アプリケーションを管理するには、これらのファイルにアクセスできなければなりません。たとえば、アプリケーションのセキュリティ管理に必要なコマンドは、すべて `$TUXDIR/bin` (UNIX ホスト・マシンの場合)、または `%TUXDIR%¥bin` (Windows 2000 ホスト・マシンの場合) に格納されています。

管理環境の設定の詳細については、『BEA Tuxedo アプリケーション実行時の管理』を参照してください。

関連項目

- 第 2 章の 8 ページ「オペレーティング・システム (OS) のセキュリティ管理」
- 第 2 章の 9 ページ「認証の管理」
- 第 2 章の 33 ページ「認可の管理」
- 第 2 章の 33 ページ「リンク・レベルの暗号化の管理」

- 第 2 章の 40 ページ「公開鍵セキュリティの管理」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」

オペレーティング・システム (OS) のセキュリティ管理

アプリケーション管理者は、BEA Tuxedo 製品の ATMI 環境におけるセキュリティ機能のほか、オペレーティング・システムに組み込まれているセキュリティ機能を十分に活用して、ファイル、ディレクトリ、およびシステム・リソースに対するアクセスを制御する必要があります。

ほとんどの場合、ATMI アプリケーションは、アプリケーション管理者によって管理されます。アプリケーション管理者は、アプリケーションをコンフィギュレーションし、起動し、実行中のアプリケーションを監視し、必要に応じて動的に変更します。ATMI アプリケーションを起動し、実行するのは管理者であるため、サーバ・プログラムは、管理者のパーミッションで実行されます。これで、サーバ・プログラムは安全であり、「信頼できる」と見なされます。この方法は、基盤となるオペレーティング・システムのログイン・メカニズムとパーミッション設定 (ファイル、ディレクトリ、およびシステム・リソースに対する読み取り権と書き込み権の設定) に基づいています。

一方、クライアントを起動するのは管理者ではありません。クライアントは、自分自身のパーミッションを持つユーザによって直接実行されます。したがって、クライアントは信頼できるとは言えません。

さらに、ネイティブ・クライアント (つまり、サーバを実行中のマシンと同じマシンで実行中のクライアント) を実行するユーザは、コンフィギュレーション・ファイルにアクセスしたり、共用メモリ内の掲示板などのプロセス間通信 (IPC) のメカニズムにアクセスできます。ATMI システムのセキュリティが追加されても、ネイティブ・クライアントを実行するクライアントは、常にこのようなアクセスを実現できます。

OS のセキュリティで推奨されている事項について

管理者は、次の一般的な方法に従うことにより、オペレーティング・システムのセキュリティ・レベルを高めることができます。

- ファイルおよび IPC 資源に対するアクセスをアプリケーション管理者に制限します。
- `setuid` ユーティリティを使用して、信頼性のあるクライアント・プログラムを必ず管理者のパーミッションで実行します。
- 最も高いレベルのオペレーティング・システムのセキュリティを実現するには、ワークステーション・クライアントだけがアプリケーションにアクセスできるようにし、アプリケーション・サーバおよび管理プログラムを実行するマシンではクライアント・プログラムを実行できないようにします。
- 上記のすべての方法を ATMI のセキュリティ機能と組み合わせて使用し、要求の発行元であるクライアントをアプリケーション側で識別できるようにします。

関連項目

- 第 1 章の 6 ページ「オペレーティング・システム (OS) のセキュリティ」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」

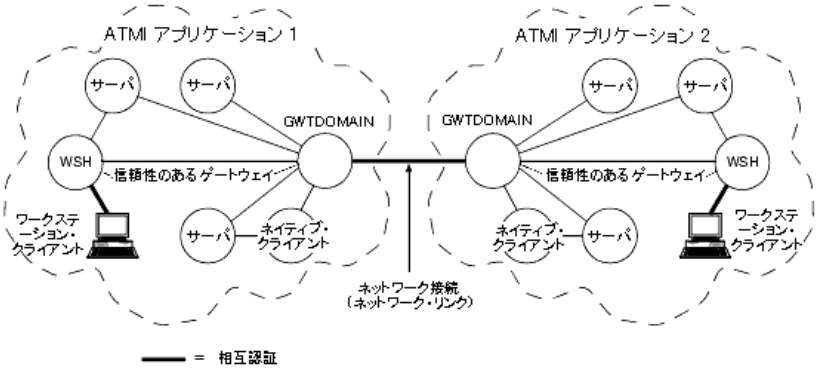
認証の管理

認証とは、通信するプロセスどうしがお互いの ID を証明し合うことです。この機能は、これ以外のほぼすべてのセキュリティ機能の基盤となります。

この節で示す手順以外の認証の管理手順は、基となるアプリケーションの認証システムに依存します。カスタマイズした認証システムを管理する手順については、該当するシステムのマニュアルを参照してください。デフォルトの認証システムを管理する手順については、第 2 章の 54 ページ「デフォルトの認証と認可の管理」を参照してください。

次の図は、BEA Tuxedo リリース 7.1 以上のソフトウェアで使用される、高信頼性委託型認証モデルをしています。高信頼性委託型認証モデルでは、ワークステーション・ハンドラ (WSH) およびドメイン・ゲートウェイ (`GWTDOMAIN`) は、「信頼性のあるシステム・ゲートウェイ・プロセス」と呼ばれます。これについては、第 1 章の 8 ページ「委任された信用認証について」を参照してください。

図 2-2　高信頼性委託型認証モデル



注記　相互認証は、ネイティブ・クライアントには適用されません。ネイティブ・クライアントを認証するのは、ネイティブ・クライアント自身です。

以下は、上の図のコンフィギュレーションの設定に必要な操作の一覧です。すべての操作で、認証と認可のプラグインが必要になります。

- プリンシパル名を指定する
- 相互運用性の方針を指定する
- ドメイン間のリンクを確立する
- ACL 方針を設定する
- クリデンシャル方針の設定

関連項目

- 第 1 章の 7 ページ「認証」
- 第 1 章の 43 ページ「デフォルトの認証と認可」
- 第 2 章の 54 ページ「デフォルトの認証と認可の管理」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 第 1 章の 54 ページ「セキュリティの相互運用性」
- 第 1 章の 58 ページ「セキュリティ機能の互換性」
- 『BEA Tuxedo Domains コンポーネント』の「Domains について」

プリンシパル名の指定

管理者は、次のコンフィギュレーション・パラメータを使用して、BEA Tuxedo リリース 7.1 以上のソフトウェアで作成した ATMI アプリケーションで実行するワークステーション・ハンドラ (WSH)、ドメイン・ゲートウェイ (GWTDOMAIN)、およびサーバ・プロセスのプリンシパル名を指定します。

パラメータ名	説明	設定
UBBCONFIG の SEC_PRINCIPAL_NAME (TM_MIB の TA_SEC_PRINCIPAL_NAME)	アプリケーションの起動時に、ATMI アプリケーション内のそれぞれの WSH、ドメイン・ゲートウェイ、およびサーバ・プロセスは、認証プラグインを呼び出して、SEC_PRINCIPAL_NAME で指定されたセキュリティ・プリンシパル名のセキュリティ・クリデンシャルを取得します。*	1 ～ 511 文字。プリンシパル名がコンフィギュレーション階層のどのレベルでも指定されない場合は、UBBCONFIG ファイルの DOMAINID の文字列がデフォルト値になります。
ローカル・ドメイン・アクセス・ポイントを示す DMCONFIG の CONNECTION_PRINCIPAL_NAME (DM_MIB にある LACCESSPOINT の TA_DMCONNPRINCIPALNAME)**	アプリケーションの起動時に、ATMI アプリケーション内の各ドメイン・ゲートウェイ・プロセスは、認証プラグインをもう一度呼び出して、CONNECTION_PRINCIPAL_NAME で指定された接続プリンシパル名のセキュリティ・クリデンシャルを取得します。*	1 ～ 511 文字。接続プリンシパル名を指定しない場合は、DMCONFIG ファイルで指定されたローカル・ドメイン・アクセス・ポイントを示す DOMAINID の文字列がデフォルト値になります。

* システム・プロセスがクリデンシャルを取得する方法、およびクリデンシャルが必要な理由については、次の節で説明します。

** ローカル・ドメイン・アクセス・ポイントは、「LDOM (エルドム)」または単に「ローカル・ドメイン」とも呼ばれます。

SEC_PRINCIPAL_NAME は、コンフィギュレーションの階層のうち、次の 4 つのレベルのどこでも指定できます。

- UBBCONFIG ファイルの RESOURCES セクション、または TM_MIB の T_DOMAIN クラス
- UBBCONFIG ファイルの MACHINES セクション、または TM_MIB の T_MACHINE クラス
- UBBCONFIG ファイルの GROUPS セクション、または TM_MIB の T_GROUP クラス
- UBBCONFIG ファイルの SERVERS セクション、または TM_MIB の T_SERVER クラス

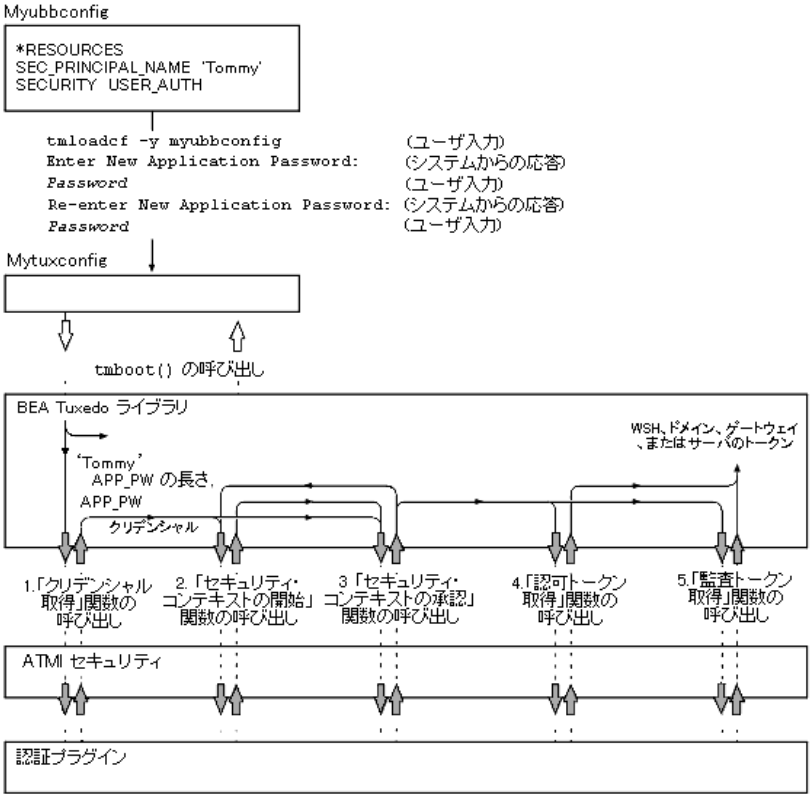
特定のコンフィギュレーション・レベルでのセキュリティ・プリンシパル名は、下位レベルで上書きできます。たとえば、mach1 というマシンに terri というプリンシパル名を指定し、mach1 上で動作する serv1 というサーバに john というプリンシパル名を指定したとします。この場合、mach1 のプロセスは次のように動作します。

- serv1 以外の、mach1 上のすべての WSH、ドメイン・ゲートウェイ、およびサーバ・プロセスは、プリンシパル名として terri を使用します。
- serv1 のすべてのプロセスは、プリンシパル名として john を使用します。

システム・プロセスがクリデンシャルを取得する方法

アプリケーションの起動時に、ATMI アプリケーション内のそれぞれの WSH、ドメイン・ゲートウェイ、およびサーバ・プロセスが認証プラグインを呼び出して (1) セキュリティ・クリデンシャルを取得し、(2) 認可トークンおよび監査トークンを取得するとき、セキュリティ・プリンシパル名を引数として指定します。次の図は、この手順を示しています。

図 2-3 アプリケーションの起動時にクリデンシャルとトークンを取得する



アプリケーション内の各ドメイン・ゲートウェイ・プロセスは、認証プラグインをもう一度呼び出して、割り当てられた接続プリンシパル名のクリデンシャルとトークンを取得します。

システム・プロセスでクリデンシャルが必要な理由

WSH にはクリデンシャルが必要です。クリデンシャルがあれば、ワークステーション・クライアントを認証してアプリケーションに参加させ、認証されたワークステーション・クライアント用の認可トークンと監査トークンを取得することができます。WSH がリリース 7.1 より前のクライアント (BEA Tuxedo 6.5 以前のソフトウェアで動作するクライアント) からの要求を処理する場合、この WSH には、WSH 自身の認可トークンと監査トークンが必要です。これらのトークンがあれば、WSH は認証プラグインを呼び出して、古いバージョンのクライアントの ID を確認できます。この動作については、第 2 章の 16 ページ「相互運用性の方針の指定」を参照してください。

ドメイン・ゲートウェイにも一組のクリデンシャルが必要です。クリデンシャルを取得すると、第 2 章の 23 ページ「ドメイン間のリンクの確立」で説明するように、リモート・ドメイン・ゲートウェイを認証し、ATMI アプリケーション間でリンクを確立することができます。認証されたリモート・ドメイン・ゲートウェイには、認可トークンも監査トークンも割り当てられません。ドメイン・ゲートウェイは、`CONNECTION_PRINCIPAL_NAME` パラメータで指定されたプリンシパル名で、これらのクリデンシャルを取得します。

ドメイン・ゲートウェイがリリース 7.1 より前のクライアントからの要求を処理する場合、つまり、認証プラグインを呼び出して、古いバージョンのクライアントの ID を確認する場合、このドメイン・ゲートウェイにはもう一組のクリデンシャルが必要です。この動作については、第 2 章の 16 ページ「相互運用性の方針の指定」を参照してください。これらのクリデンシャルは、ローカルのアクセス制御リスト (ACL: Access Control List) を適用する場合に ID を確認するときにも必要です (第 2 章の 27 ページ「ACL 方針の設定」を参照)。ドメイン・ゲートウェイは、`SEC_PRINCIPAL_NAME` パラメータで指定されたプリンシパル名で、これらのクリデンシャルを取得します。

システムまたはアプリケーション・サーバがリリース 7.1 より前のクライアントからの要求を処理する場合は、システムまたはアプリケーション・サーバ自身の認可トークンと監査トークンが必要です。これらのトークンがあれば、認証プラグインを呼び出して、古いバージョンのクライアントの ID を確認できます。この動作については、第 2 章の 16 ページ「相互運用性の方針の指定」を参照してください。

サーバは、サーバ・パーミッションのアップグレードを行うときにもサーバ自身のトークンを必要とします。サーバ・パーミッションのアップグレードは、クライアントから発信されサーバを経由して送信されるメッセージに対し、認可トークンおよび監査トークンが割り当てられるときに発生します。サーバのアップグレード機能については、第 1 章の 11 ページ「クライアント・トークンとサーバ・トークンの交換」を参照してください。

注記 アプリケーション・サーバは、認証プラグインを呼び出すことはできません。アプリケーション・サーバの認証プラグインは、基となるシステム・コードによって呼び出されます。

プリンシパル名を指定する UBBCONFIG のエントリ例

次の例は、UBBCONFIG の SEC_PRINCIPAL_NAME パラメータを使用してセキュリティ・プリンシパル名を指定する方法を示しています。CONNECTION_PRINCIPAL_NAME パラメータを使用して、DMCONFIG ファイルの接続プリンシパル名を指定する例については、第 2 章の 26 ページ「リンクを確立するための DMCONFIG のエントリ例」を参照してください。

```
*RESOURCES
SEC_PRINCIPAL_NAME    "Tommy"
      .
      .
      .
*SERVERS
"TMQUEUE"       SRVGRP="QUEGROUP"      SRVID=1
   CLOPT="-t -s secsdb:TMQUEUE"
   SEC_PRINCIPAL_NAME="TOUPPER"
```

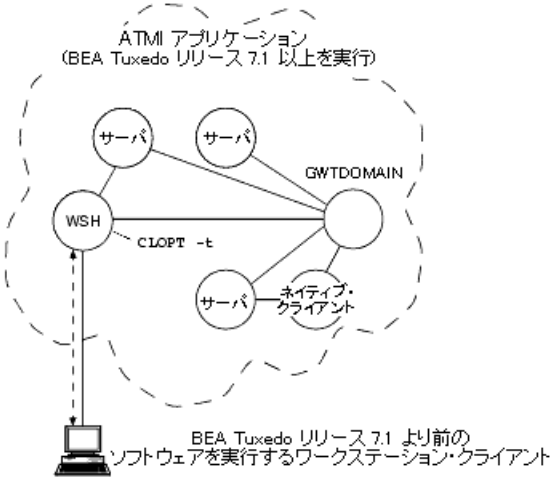
関連項目

- 第 2 章の 16 ページ「相互運用性の方針の指定」
- 第 2 章の 23 ページ「ドメイン間のリンクの確立」
- 第 2 章の 27 ページ「ACL 方針の設定」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」

相互運用性の方針の指定

管理者は、UBBCONFIG ファイルの CLOPT -t オプションを使用して、ATMI アプリケーション内の WSH、ドメイン・ゲートウェイ (GWTDOMAIN) およびサーバ・プロセスが、BEA Tuxedo リリース 7.1 より前 (6.5 以前) のソフトウェアを実行しているマシンと相互運用するように指定できます。また、WSALLOWPRE71 環境変数を使用して、ワークステーション・クライアントが BEA Tuxedo リリース 7.1 より前のソフトウェアを実行するマシンと相互運用するよう指定できます。次に、これらのプロセスの相互運用性を説明する 4 つの図を示します。

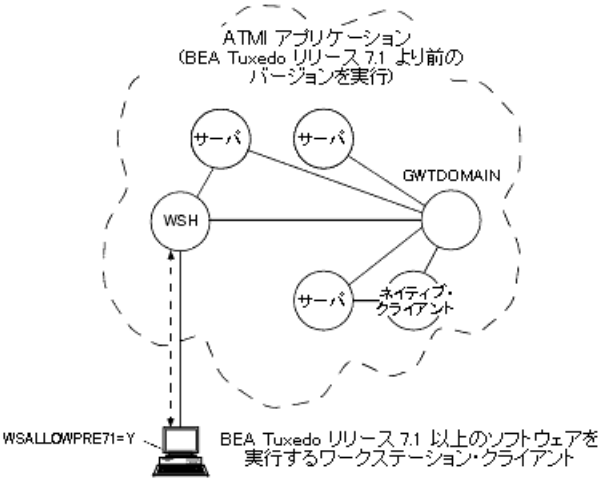
図 2-4 新しい WSH と古いワークステーション・クライアントの相互運用



上の図では、WSH がリリース 7.1 より前の古い認証プロトコルを使用してワークステーション・クライアントを認証し、内部の代替ユーザ関数を呼び出してクライアントの認可トークンと監査トークンを取得し、それらのトークンをクライアント要求に添付しています。WSH を制御するワークステーション・リスナ (WSL) に対して CLOPT -t オプションが指定されないと、新しい WSH と古いワークステーション・クライアントとの間の通信は実現できません。

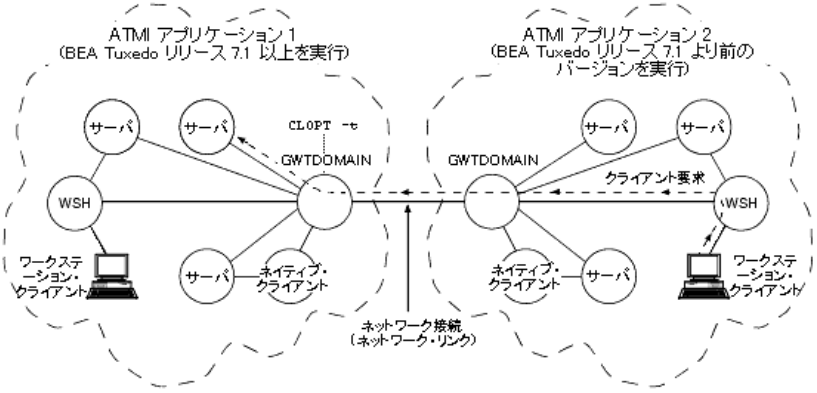
注記 代替ユーザ関数は、認証プラグインを呼び出して、古いクライアントの ID を確認します。詳細については、第 2 章の 19 ページ「古いクライアントの ID の確認」を参照してください。

図 2-5 古い WSH と新しいワークステーション・クライアントの相互運用



上の図では、WSH がリリース 7.1 より前の古い認証プロトコルを使用してワークステーション・クライアントを認証します。クライアント要求は、認可トークンと監査トークンを受け取りません。ワークステーション・クライアントに WSALLOWPRE71 環境変数が設定されていない場合、またはこの環境変数が N に設定されている場合、古い WSH と新しいワークステーション・クライアントとの間の通信は実現できません。

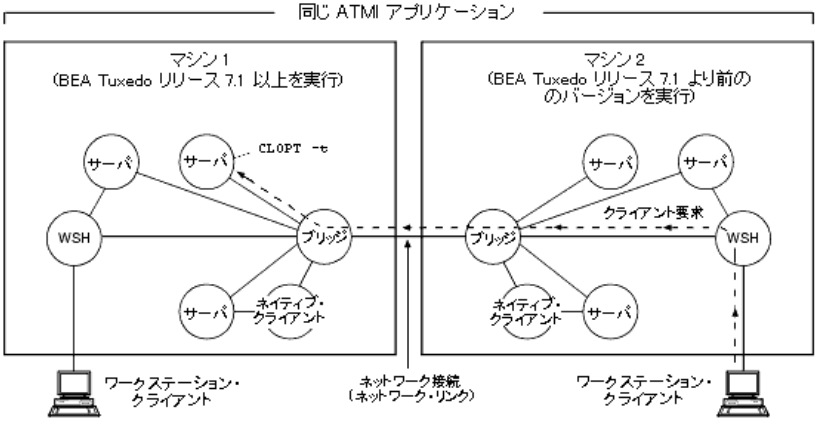
図 2-6 サーバと古い ATMI アプリケーションの相互運用



上の図では、アプリケーション 1 のローカル・ドメイン・ゲートウェイ (GWTDOMAIN) が、リリース 7.1 より前の古い認証プロトコルを使用して、アプリケーション 2 のリモート・ドメイン・ゲートウェイを認証しています。ローカル・ドメイン・ゲートウェイは、リモート・クライアントからの要求を受け取ると、内部の代替ユーザ関数を呼び出してリモート・クライアントの認可トークンと監査トークンを取得し、それらのトークンをクライアント要求に添付します。アウトバウンドのクライアント要求 (アプリケーション 1 からアプリケーション 2 に送信される要求) の場合、要求に添付されたトークンは、ローカル・ドメイン・ゲートウェイで取り除かれ、要求はアプリケーション・キーとともにアプリケーション 2 に送信されます (アプリケーション・キーについては、第 1 章の 48 ページ「アプリケーション・キー」を参照してください)。

ドメイン・ゲートウェイに対して CLOPT -t オプションが指定されない場合、新しい ATMI アプリケーションと古い ATMI アプリケーションとの間の通信は実現できません。

図 2-7 サーバと古い BEA Tuxedo システムの相互運用



上の図では、まずマシン 1 にある送信先のサーバが内部の代替ユーザ関数を呼び出し、マシン 2 にあるリモート・クライアントの認可トークンと監査トークンを取得します。取得したトークンがクライアント要求に添付されると、サーバは、クライアントが認可チェックにパスしたと見なして要求を実行します。サーバに対して CLOPT -t オプションが指定されない場合、新しいサーバと古いクライアントとの間の通信は実現できません。

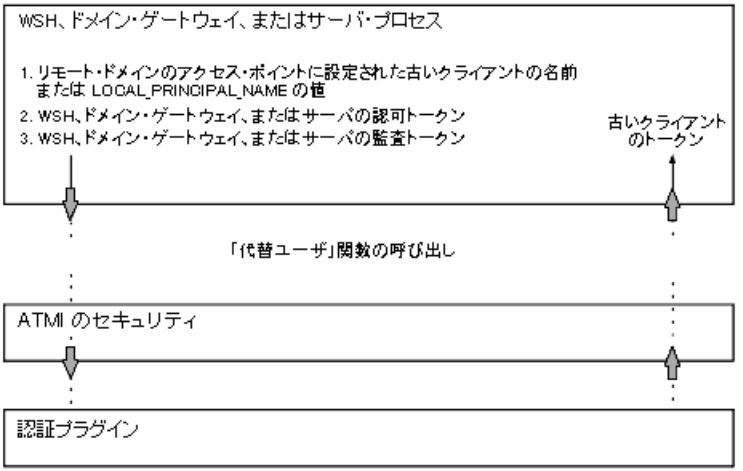
注記 また、上の図で、マシン 1 の WSH がマシン 2 のサーバ宛てのクライアント要求を受け取ると、WSH は要求に添付されたトークンを取り除いてから、その要求をクライアントのアプリケーション・キーとともにマシン 2 のサーバに送信し

ます。同様に、マシン 1 のネイティブ・クライアントがマシン 2 のサーバに要求を送信した場合、ネイティブ・クライアントは、要求に添付されたトークンを取り除いてから、その要求をクライアントのアプリケーション・キーとともにマシン 2 のサーバに送信します。アプリケーション・キーについては、第 1 章の 48 ページ「アプリケーション・キー」を参照してください。

古いクライアントの ID の確認

WSH、ドメイン・ゲートウェイ (GWTDOMAIN)、またはサーバ・プロセスは、内部の代替ユーザ関数を呼び出して、古いクライアントの認可トークンと監査トークンを取得することにより、古いクライアントの ID を確認します。次の図は、この手順を示しています。

図 2-8 古いクライアントの認可トークンと監査トークンの取得



WSH 側で古いクライアントの ID を確認する

CLOPT -t オプションが指定されている場合、WSH は、TPINIT バッファの usrname フィールドを使用するか (C の場合)、または TPINFDEF-REC レコードの USRNAME フィールドを使用して (COBOL の場合) 古いクライアントの ID を確認します。第 3 章の 8 ページ「ATMI アプリケーションへの参加」で説明するとおり、クライアントがアプリケーションに参加しようとすると、WSH はクライアントから TPINIT バッファまたは TPINFDEF-REC レコードを受け取ります。WSH は、代替ユーザ関数を呼び出すときにプリンシパル名としてユーザ名を使用します。

デフォルトの認証プラグインの場合、代替ユーザ関数は、ローカルの tpusr ファイルからユーザ名および関連するアプリケーション・キー (ユーザ識別子とグループ識別子の組み合わせ) を検索し、古いクライアント用に作成された認可トークンと監査トークンに、ユーザ名とアプリケーション・キーを両方とも組み込みます。tpusr ファイルについては、第 2 章の 60 ページ「ユーザ・ファイルとグループ・ファイルの設定」を参照してください。

ドメイン・ゲートウェイ側で古いクライアントの ID を確認する

CLOPT -t オプションが指定されている場合、ドメイン・ゲートウェイは、リモート・ドメイン・アクセス・ポイントに対して設定された LOCAL_PRINCIPAL_NAME 文字列を使用して、古いクライアントの ID を確認します。ドメイン・ゲートウェイは、ローカルの BDMCONFIG ファイル (バイナリ形式の DMCONFIG(5) ファイル) の DM_REMOTE_DOMAINS セクションで、リモート・ドメイン・アクセス・ポイントの LOCAL_PRINCIPAL_NAME 文字列を検索します。このオプションを指定しない場合は、リモート・ドメイン・アクセス・ポイントの DOMAINID 文字列がデフォルト値になります。ドメイン・ゲートウェイは、代替ユーザ関数を呼び出すときにプリンシパル名として LOCAL_PRINCIPAL_NAME 文字列を使用します。

デフォルトの認証プラグインの場合、代替ユーザ関数は、ローカルの tpusr ファイルから LOCAL_PRINCIPAL_NAME 文字列および関連するアプリケーション・キーを検索し、古いクライアント用に作成された認可トークンと監査トークンに、その文字列 (ID) とアプリケーション・キーを組み込みます。

サーバ側で古いクライアントの ID を確認する

CLOPT -t オプションが指定されている場合、サーバは、クライアントに割り当てられたアプリケーション・キーを使用して、古いクライアントの ID を確認します。サーバが受信したクライアント要求には、クライアントに割り当てられたアプリケーション・キーが含まれています。サーバは、ローカルの tpusr ファイルからアプリケーション・キーと関連する名前を検索し、代替ユーザ関数を呼び出すときにプリンシパル名としてその名前を組み込みます。

デフォルトの認証プラグインの場合、代替ユーザ関数は、ローカルの tpusr ファイルから名前および関連するアプリケーション・キーを検索し、古いクライアント用に作成された認可トークンと監査トークンに、名前とアプリケーション・キーを組み込みます。

CLOPT -t オプションの動作のまとめ

次の表は、CLOPT -t オプションを使用して相互運用性を実現できる場合、または実現できない場合の WSH、ドメイン・ゲートウェイ、およびサーバ・プロセスの機能をまとめたものです。

表 2-1 相互運用性を実現できる場合または実現できない場合の WSH、ドメイン・ゲートウェイ、およびサーバ・プロセスの機能

プロセス	相互運用性を実現できる場合 (CLOPT -t)	相互運用性を実現できない場合
ワークステーション・ハンドラ (WSH)	WSH は、アプリケーションに参加するための要求をリリース 7.1 より前のワークステーション・クライアントから受け取ると、リリース 7.1 より前の認証プロトコルを使用してクライアントを認証します。次に、要求内で指定されているユーザ名に基づき、代替ユーザ関数を呼び出してクライアントの認可トークンと監査トークンを取得します。 WSH は、認証されたワークステーション・クライアントからサービス要求を受け取ると、クライアント要求にトークンを添付し、要求先のサーバに転送します。	WSH は、アプリケーションに参加するための要求をリリース 7.1 より前のワークステーション・クライアントから受け取っても拒否します。新しい WSH と古いワークステーション・クライアントとの間の通信は実現できません。

表 2-1 相互運用性を実現できる場合または実現できない場合の WSH、ドメイン・ゲートウェイ、およびサーバ・プロセスの機能 (続き)

プロセス	相互運用性を実現できる場合 (CLOPT -t)	相互運用性を実現できない場合
ドメイン・ゲートウェイ (GWTDOMAIN)	ドメイン・ゲートウェイは、リリース 7.1 より前のリモート・ドメイン・ゲートウェイとの接続を確立すると、リリース 7.1 より前の認証プロトコルを使用してそのリモート・ドメイン・ゲートウェイを認証し、ネットワーク接続を確立します。 ドメイン・ゲートウェイは、古いドメインからのクライアント要求を受け取ると、リモート・ドメイン・アクセス・ポイントに設定された LOCAL_PRINCIPAL_NAME (デフォルトは DOMAINID) の ID に基づき、代替ユーザ関数を呼び出してクライアントの認可トークンおよび監査トークンを取得します。次に、これらのトークンをクライアント要求に添付し、要求先のサーバに転送します。クライアントには、LOCAL_PRINCIPAL_NAME ID と同じアクセス・パーミッションがあります。 アウトバウンドのクライアント要求の場合、ドメイン・ゲートウェイは、要求に添付されたトークンを取り除いてから、その要求をアプリケーション・キーとともに古いドメインに送信します。	ドメイン・ゲートウェイとリリース 7.1 より前のリモート・ドメイン・ゲートウェイとの接続は確立できません。新しいドメインと古いドメインとの間の通信は実現できません。
システムまたはアプリケーション・サーバ	サーバは、リリース 7.1 より前の BEA Tuxedo ソフトウェアを実行するリモート・クライアントから要求を受け取ると、クライアントに割り当てられたアプリケーション・キーに基づき、代替ユーザ関数を呼び出してクライアントの認可トークンと監査トークンを取得します。次に、サーバは、クライアントが認可チェックにパスしたと見なしてクライアント要求を実行します。	サーバは、BEA Tuxedo リリース 7.1 より前のソフトウェアを実行するリモート・クライアントから要求を受け取っても拒否します。新しいサーバと古いクライアントとの間の通信は実現できません。

相互運用性を指定する UBBCONFIG のエントリ例

次の例は、ワークステーション・リスナ (WSL) によって制御されるすべての WSH で相互運用性が実現できることを示しています。

```
*SERVERS
WSL     SRVGRP="group_name" SRVID=server_number ...
        CLOPT="-A -t ..."
```

関連項目

- 第 2 章の 11 ページ「プリンシパル名の指定」
- 第 2 章の 23 ページ「ドメイン間のリンクの確立」
- 第 2 章の 27 ページ「ACL 方針の設定」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 第 1 章の 54 ページ「セキュリティの相互運用性」
- 『BEA Tuxedo Domains コンポーネント』の 第 2 章の 33 ページ「Domains でセキュリティを設定する」および 第 2 章の 47 ページ「ドメイン間の接続を設定する」

ドメイン間のリンクの確立

ドメイン・ゲートウェイ (GWTDOMAIN) が別のドメイン・ゲートウェイとのネットワーク・リンクを確立しようとすると、次のようなイベントが発生します。

1. イニシエータ側とターゲット側のドメイン・ゲートウェイは、リンク・レベルの暗号化 (LLE: Link-Level Encryption) の *min-max* 値を交換します。これらの値は、ゲートウェイ間のリンクに LLE を設定するために使用されます。LLE については、第 1 章の 23 ページ「リンク・レベルの暗号化」を参照してください。
2. イニシエータ側とターゲット側のドメイン・ゲートウェイは、セキュリティ・トークンを交換することにより、認証し合います。このとき、双方で BEA Tuxedo リリース 7.1 以上のソフトウェアを実行していると想定します。

   どちらかまたは両方のドメイン・ゲートウェイが BEA Tuxedo リリース 7.1 より前のソフトウェアを実行している場合、ゲートウェイ・プロセスでは、リリース 7.1 より前の古い認証プロトコルを使って接続が確立されます。

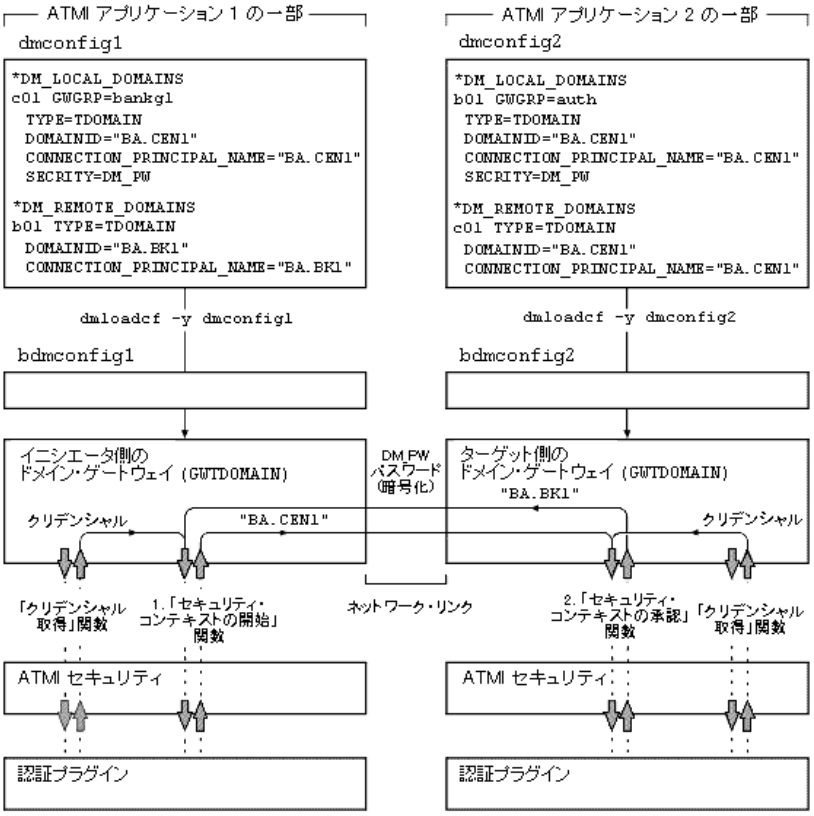
管理者は、次のコンフィギュレーション・パラメータを使用して、BEA Tuxedo リリース 7.1 以上のソフトウェアを実行するドメイン・ゲートウェイ間にリンクを確立します。

パラメータ名	説明	設定
DMCONFIG の CONNECTION_PRINCIPAL_NAME (DM_MIB の TA_DMCONNPRINCIPALNAME)	このパラメータが DMCONFIG ファイルの DM_LOCAL_DOMAINS セクションにある場合、リモート・ドメイン・アクセス・ポイントとの接続を設定する際のこのパラメータの値は、ローカル・ドメイン・アクセス・ポイントのプリンシパル名になります。* デフォルトの認証プラグインで、ローカル・ドメイン・アクセス・ポイントの CONNECTION_PRINCIPAL_NAME に値を割り当てる場合、その値は、ローカル・ドメイン・アクセス・ポイントの DOMAINID パラメータの値と同じでなければなりません。これらの値が一致しないと、ローカル・ドメイン・ゲートウェイ・プロセスは起動せず、システムでは「ERROR: 証明書を取得できません。」という userlog(3c) のメッセージが生成されます。	1 ～ 511 文字。指定しない場合は、ローカル・ドメイン・アクセス・ポイントの DOMAINID 文字列がデフォルト値になります。
DMCONFIG の CONNECTION_PRINCIPAL_NAME (DM_MIB の TA_DMCONNPRINCIPALNAME)	このパラメータが DMCONFIG ファイルの DM_REMOTE_DOMAINS セクションにあり、特定のリモート・ドメイン・アクセス・ポイントを示す場合、ローカル・ドメイン・アクセス・ポイントとの接続を設定する際のこのパラメータの値は、リモート・ドメイン・アクセス・ポイントのプリンシパル名になります。 デフォルトの認証プラグインで、リモート・ドメイン・アクセス・ポイントの CONNECTION_PRINCIPAL_NAME に値を割り当てる場合、その値は、リモート・ドメイン・アクセス・ポイントの DOMAINID パラメータの値と同じでなければなりません。これらの値が一致しないと、ローカル・ドメイン・ゲートウェイとリモート・ドメイン・ゲートウェイとの間の接続は確立できず、システムでは「ERROR: ドメイン *domain_name* の管理用キーを初期化できません。」という userlog(3c) のメッセージが生成されます。	1 ～ 511 文字。指定しない場合は、リモート・ドメイン・アクセス・ポイントの DOMAINID 文字列がデフォルト値になります。

* ローカル・ドメイン・アクセス・ポイントは、「LDOM (エルドム)」または単に「ローカル・ドメイン」とも呼ばれます。リモート・ドメイン・アクセス・ポイントは、「RDOM (アールドム)」または単に「リモート・ドメイン」とも呼ばれます。

次の図は、デフォルトの認証プラグインを使用して、ドメイン間のリンクを確立する方法を示しています。

図 2-9 デフォルトの認証を使用してドメイン間にリンクを確立する



注記 上の図の「クリデンシャル」は、ローカル・ドメイン・アクセス・ポイントに対して設定された CONNECTION_PRINCIPAL_NAME の ID を使用して、アプリケーションの起動時に、各ドメイン・ゲートウェイ・プロセスによって取得されます。

上の図で、イニシエータ側とターゲット側のドメイン・ゲートウェイ間で交換される情報には、ドメイン・ゲートウェイ用に指定された BDMCONFIG ファイルの CONNECTION_PRINCIPAL_NAME 文字列が含まれる点に注目してください。各認証プラグインは、リモート・ドメイン・アクセス・ポイントに割り当てられたパスワード (BDMCONFIG ファイルの DM_PASSWORDS セクションで定義) を使用して文字列を暗号化してから、それをネットワーク経由で送信し、ローカル・ドメイン・アクセス・ポイントに割り当てられたパスワード (BDMCONFIG ファイルの DM_PASSWORDS セクションで定義) を使用して受信した文字列を復号化します。このとき、暗号化アルゴリズムとして、56 ビットの DES (Data Encryption Standard) が使用されます。

暗号化および復号化の操作を成功させるため、ローカルの BDMCONFIG ファイルで指定された、リモート・ドメイン・アクセス・ポイント用のパスワードは、リモートの BDMCONFIG ファイルで指定されたローカル・ドメイン・アクセス・ポイント用のパスワードと同じでなければなりません。同様に、ドメインのセキュリティ・レベルが APP_PW に設定されている場合に暗号化および復号化の操作を成功させるには、各 TUXCONFIG ファイルのアプリケーション・パスワードが同じでなければなりません。認証プロセスを成功させるには、受信した文字列が、送信者の CONNECTION_PRINCIPAL_NAME 文字列と一致しなければなりません。

ドメイン・ゲートウェイがセキュリティ・チェックにパスすると、リンクが確立され、ゲートウェイは、確立されたリンクを介してサービス要求を転送したり、応答を受信することができます。

リンクを確立するための DMCONFIG のエントリ例

次の例は、ローカル・ドメイン・アクセス・ポイント c01 と、リモート・ドメイン・アクセス・ポイント b01 を使って接続を確立するときに、ローカルの DMCONFIG ファイルのコンフィギュレーションが使用されることを示しています。

```
*DM_LOCAL_DOMAINS
# <LDOM name> <Gateway Group name> <domain type>
#       <domain id> [<connection principal name>] [<security>]...
c01     GWGRP=bankg1
        TYPE=TDOMAIN
        DOMAINID="BA.CENTRAL01"
        CONNECTION_PRINCIPAL_NAME="BA.CENTRAL01"
        SECURITY=DM_PW
    .
    .
    .
*DM_REMOTE_DOMAINS
# <RDOM name> <domain type> <domain id>
#       [<connection principal name>]...
```

```
b01     TYPE=TDOMAIN
        DOMAINID="BA.BANK01"
        CONNECTION_PRINCIPAL_NAME="BA.BANK01"
```

関連項目

- 第 2 章の 11 ページ「プリンシパル名の指定」
- 第 2 章の 16 ページ「相互運用性の方針の指定」
- 第 2 章の 27 ページ「ACL 方針の設定」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 『BEA Tuxedo Domains コンポーネント』の 第 2 章の 41 ページ「Domains の認証機能を設定する」

ACL 方針の設定

管理者は、次のコンフィギュレーション・パラメータを使用して、BEA Tuxedo リリース 7.1 以上のソフトウェアを実行する ATMI アプリケーション間で、アクセス制御リスト (ACL: Access Control List) 方針の設定と制御を行います。

パラメータ名	説明	設定
DMCONFIG の ACL_POLICY (DM_MIB の TA_DMACLPOLICY)	リモート・ドメイン・アクセス・ポイントごとに、DMCONFIG ファイルの DM_REMOTE_DOMAINS セクションで指定される場合があります。特定のリモート・ドメイン・アクセス・ポイントに対するこのパラメータの値により、ローカル・ドメイン・ゲートウェイがリモート・ドメインから受信したサービス要求の ID を変更するかどうかが決まります。*	LOCAL または GLOBAL。デフォルト値は LOCAL です。 LOCAL は、サービス要求の ID を変更することを示します。GLOBAL は、サービス要求を変更しないで渡すことを示します。

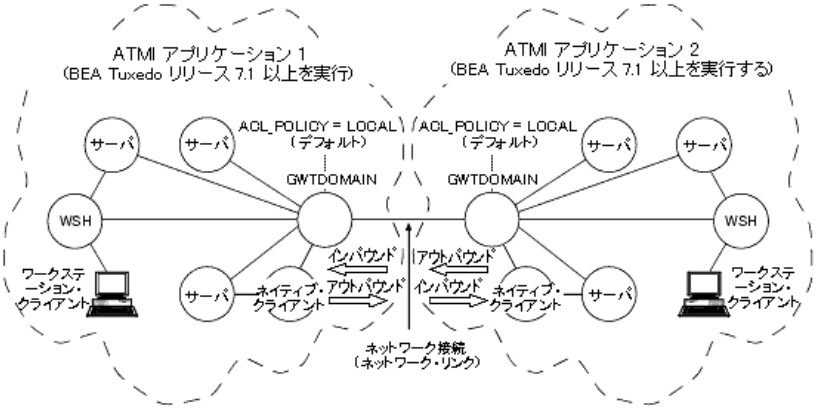
* リモート・ドメイン・アクセス・ポイントは、RDOM (アールドム) または単に「リモート・ドメイン」とも呼ばれます。

パラメータ名	説明	設定
DMCONFIG の LOCAL_PRINCIPAL_NAME (DM_MIB の TA_DMLOCALPRINCIPALNAME)	リモート・ドメイン・アクセス・ポイントごとに、DMCONFIG ファイルの DM_REMOTE_DOMAINS セクションで指定される場合があります。特定のリモート・ドメイン・アクセス・ポイントに対し、ACL_POLICY パラメータに LOCAL (デフォルト値) が設定された場合、ローカル・ドメイン・ゲートウェイは、リモート・ドメインから受け取ったサービス要求の ID を、LOCAL_PRINCIPAL_NAME のプリンシパル名に変更します。	1 ～ 511 文字。指定しない場合、リモート・ドメイン・アクセス・ポイントの DOMAINID 文字列がデフォルト値になります。

* リモート・ドメイン・アクセス・ポイントは、RDOM (アールドム) または単に「リモート・ドメイン」とも呼ばれます。

以下の 3 つの図は、ACL_POLICY の設定が、ローカル・ドメイン・ゲートウェイ (GWTDOMAIN) のプロセスの動作に与える影響を示します。

図 2-10 ローカルな ACL 方針の確立

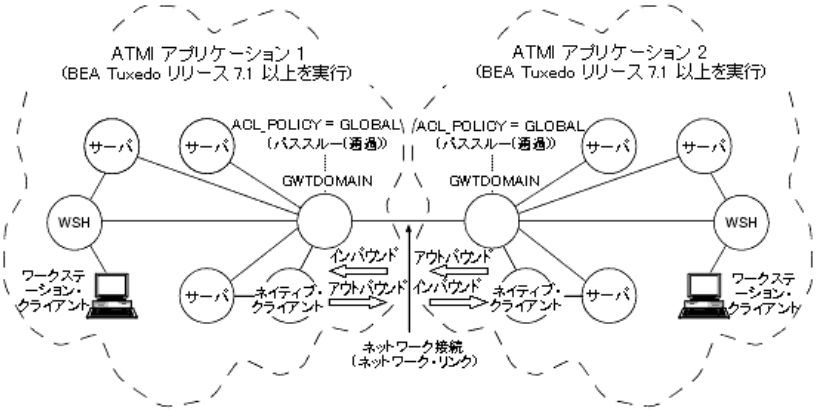


上の図では、各ドメイン・ゲートウェイ (GWTDOMAIN) がインバウンドのクライアント要求 (リモート・アプリケーションからネットワーク経由で受信される要求) を変更しています。変更された要求は、リモート・ドメイン・アクセス・ポイントに設定された LOCAL_PRINCIPAL_NAME の ID を持つため、その ID に設定されたアクセス権も取得することになります。各ドメイン・ゲートウェイは、アウトバウンドのクライアント要求を変更しないで渡します。

このコンフィギュレーションでは、各 ATMI アプリケーションに ACL データベースがあります。このデータベースには、ドメイン内のユーザに関するエントリだけが格納されます。たとえば、リモート・ドメイン・アクセス・ポイントに対して設定された LOCAL_PRINCIPAL_NAME の ID を持つユーザです。

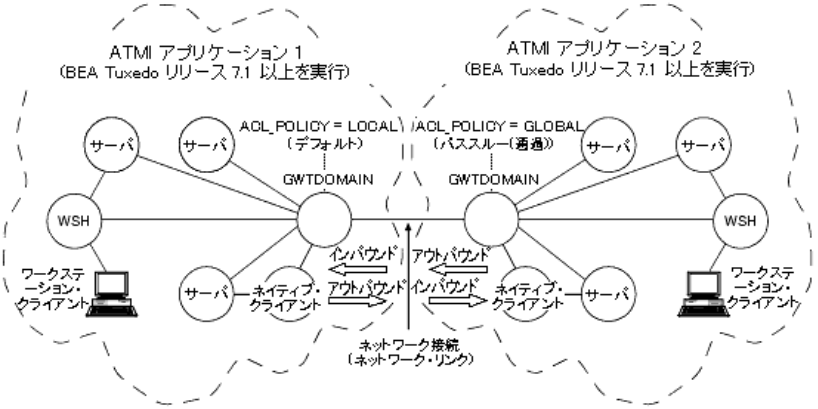
注記 以上の説明は、BEA Tuxedo リリース 7.1 より前のソフトウェアを実行する ATMI アプリケーションにも適用されます。ただし、システムでは、リモート・ドメイン・アクセス・ポイントに設定された DOMAINID の ID が使用されます。基本的に、BEA Tuxedo リリース 6.5 以前のソフトウェアでは、ローカルの ACL 方針はハードコーディングされています。

図 2-11 グローバルな ACL 方針の確立



上の図では、各ドメイン・ゲートウェイ (GWTDOMAIN) は、インバウンドとアウトバウンドのクライアント要求を変更しないで渡します。このコンフィギュレーションでは、各 ATMI アプリケーションに ACL データベースがあります。このデータベースには、ドメイン内のユーザに関するエントリのほか、リモート・ドメインのユーザの情報も格納されます。

図 2-12 一方向ローカルおよび一方向グローバルの ACL 方針の確立



上の図では、ATMI アプリケーション 1 のドメイン・ゲートウェイ (GWTDOMAIN) が、インバウンドのクライアント要求を変更しています。変更された要求は、ATMI アプリケーション 2 のリモート・ドメイン・アクセス・ポイントに設定された LOCAL_PRINCIPAL_NAME ID を持つため、その ID に設定されたアクセス権も取得することになります。アウトバウンドのクライアント要求は、変更されずに渡されます。ATMI アプリケーション 2 のドメイン・ゲートウェイ (GWTDOMAIN) は、インバウンドとアウトバウンドのクライアント要求を変更しないで渡します。

このコンフィギュレーションの ATMI アプリケーション 1 の ACL データベースには、ドメイン内のユーザに関するエントリだけが含まれています。たとえば、アプリケーション 2 のリモート・ドメイン・アクセス・ポイントに設定された LOCAL_PRINCIPAL_NAME の ID を持つユーザです。ATMI アプリケーション 2 の ACL データベースには、ドメイン内のユーザに関するエントリのほか、ATMI アプリケーション 1 のユーザのエントリも格納されています。

リモート・ドメイン・ゲートウェイの偽装化

ドメイン・ゲートウェイは、ローカルの DMCONFIG ファイルの ACL_POLICY パラメータに LOCAL (デフォルト) が設定されたリモート・ドメインからクライアント要求を受け取ると、次のタスクを実行します。

1. 内部の代替ユーザ関数を呼び出して、リモート・ドメイン・アクセス・ポイントに設定された LOCAL_PRINCIPAL_NAME の ID に基づき、クライアントの認可トークンと監査トークンを取得します。
2. 取得したトークンを使用して、既にクライアント要求に添付されているトークンを上書きします。
3. 要求を送信先のサーバに転送します。

代替ユーザ関数の詳細については、第 2 章の 19 ページ「古いクライアントの ID の確認」を参照してください。

ACL 方針を指定する DMCONFIG のエントリ例

次の例では、リモート・ドメイン・アクセス・ポイント b01 を介した接続に対し、ローカルの DMCONFIG ファイルで ACL がグローバルに設定されています。つまり、ドメイン・アクセス・ポイント c01 のドメイン・ゲートウェイ・プロセスは、ドメイン・アクセス・ポイント b01 に対し、クライアント要求を変更しないで受け渡します。ACL がグローバルに設定されている場合、ドメイン・アクセス・ポイント b01 の LOCAL_PRINCIPAL_NAME エントリは無視されます。

```
*DM_LOCAL_DOMAINS
# <LDOM name> <Gateway Group name> <domain type> <domain id>
#       [<connection principal name>] [<security>]...
c01     GWGRP=bankg1
        TYPE=TDOMAIN
        DOMAINID="BA.CENTRAL01"
        CONNECTION_PRINCIPAL_NAME="BA.CENTRAL01"
        SECURITY=DM_PW
    .
    .
    .
*DM_REMOTE_DOMAINS
# <RDOM name> <domain type> <domain id> [<ACL policy>]
#       [<connection principal name>] [<local principal name>]...
b01     TYPE=TDOMAIN
        DOMAINID="BA.BANK01"
        ACL_POLICY=GLOBAL
```

```
CONNECTION_PRINCIPAL_NAME="BA.BANK01"
LOCAL_PRINCIPAL_NAME="BA.BANK01.BOB"
```

関連項目

- 第 2 章の 11 ページ「プリンシパル名の指定」
- 第 2 章の 16 ページ「相互運用性の方針の指定」
- 第 2 章の 23 ページ「ドメイン間のリンクの確立」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」

クリデンシャル方針の設定

管理者は、次のコンフィギュレーション・パラメータを使用して、BEA Tuxedo リリース 8.0 以上を実行する ATMI アプリケーション間でのクリデンシャル方針の設定と制御を行います。

パラメータ名	説明	設定
DMCONFIG の CREDENTIAL_POLICY (DM_MIB の TA_DMCREDENTIALPOLICY)	リモート・ドメイン・アクセス・ポイントごとに、DMCONFIG ファイルの DM_REMOTE_DOMAINS セクションで指定される場合があります。特定のリモート・ドメイン・アクセス・ポイントに対するこの値により、リモート・ドメインの呼び出しに、要求元のユーザのクリデンシャルがアタッチされるかどうかが決まります。* このパラメータは、ユーザのクリデンシャルをリモート・ドメインに送信するかどうかを制御します。ACL_POLICY パラメータも似ていますが、ACL_POLICY パラメータの場合は、着信したクリデンシャルを受け付けるかどうかを制御します。	LOCAL または GLOBAL。デフォルトは LOCAL です。 LOCAL は、要求元ユーザのクリデンシャルがリモート・ドメインの呼び出しにアタッチされないことを意味します。GLOBAL は、要求元ユーザのクリデンシャルをリモート・ドメインの呼び出しにアタッチすることを意味します。

* リモート・ドメイン・アクセス・ポイントは、RDOM (アールドム) または単に「リモート・ドメイン」とも呼ばれます。

認可の管理

認可とは、ATMI アプリケーション内のリソースまたは機能に対するユーザ・アクセスを、アプリケーション固有の規則に従って制限することです。認可は、ユーザが ATMI アプリケーションへの参加を認証された場合にのみ実行されます。

認可の管理手順は、基となる ATMI アプリケーションの認可システムによって異なります。カスタマイズした認可システムを管理する手順については、該当するシステムのマニュアルを参照してください。デフォルトの認可システムを管理する手順については、第 2 章の 54 ページ「デフォルトの認証と認可の管理」を参照してください。

関連項目

- 第 1 章の 13 ページ「認可」
- 第 1 章の 43 ページ「デフォルトの認証と認可」
- 第 2 章の 54 ページ「デフォルトの認証と認可の管理」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 第 1 章の 58 ページ「セキュリティ機能の互換性」

リンク・レベルの暗号化の管理

リンク・レベルの暗号化は、ATMI アプリケーション内のマシン間をつなぐネットワーク・リンク上で送受信されるメッセージの秘密性を保護します。リンク・レベルの暗号化によるセキュリティには、0 ビット (暗号化なし)、56 ビット (国際版)、128 ビット (米国およびカナダ版) の 3 つのレベルがあります。国際版の LLE では、0 ビットと 56 ビットの暗号化が可能です。米国およびカナダ版の LLE では、0 ビット、56 ビット、および 128 ビットの暗号化が可能です。

LLE は、ATMI の次の種類のリンクで使用できます。

- ワークステーション・クライアントからワークステーション・ハンドラ (WSH) へのリンク
- ブリッジ間のリンク
- 管理ユーティリティ (`tmboot` など) または `tlisten` とのリンク
- ドメイン・ゲートウェイ間のリンク

min 値と max 値について

ATMI アプリケーションに LLE を設定する前に、LLE の *min* 値および *max* 値を知っておく必要があります。これらのパラメータのデフォルト値は次のとおりです。

- *min* の場合 : 0
- *max* の場合 : インストール済みの LLE バージョンで可能な暗号化の最高レベルを示すビット数。

たとえば、米国およびカナダ版の LLE のデフォルトの *min* 値と *max* 値は (0, 128) です。デフォルト値を変更するには、アプリケーションの UBBCONFIG ファイルで *min* と *max* に新しい値を割り当てます。

詳細については、第 1 章の 24 ページ「LLE のしくみ」および第 1 章の 25 ページ「暗号化キー・サイズの調整」を参照してください。

インストール済みの LLE バージョンの確認

マシンにインストールされた LLE バージョンを確認するには、冗長モードで tmadmin コマンドを実行します。

```
tmadmin -v
```

ローカルの BEA Tuxedo lic.txt ファイルのキー行は、次のサンプルのようにコンピュータ画面に表示されます。エントリのサンプル STRENGTH=128 は、LLE のバージョンが米国およびカナダ版であることを示しています。

```
[BEA Tuxedo] VERSION=8.0
[LINK ENCRYPTION] VERSION=8.0
STRENGTH=128
   .
   .
   .
```

BEA Tuxedo のライセンスは、UNIX ホスト・マシンの場合は $TUXDIR/udataobj/lic.txt ファイル、Windows 2000 ホスト・マシンの場合は %TUXDIR%¥udataobj¥lic.txt ファイルにすべて格納されています。

ワークステーション・クライアントのリンクに LLE を設定する方法

アプリケーションにワークステーション・クライアントが組み込まれている場合、管理者は、1 つまたは複数のワークステーション・リスナ (WSL) を設定して、ワークステーション・クライアントからの接続要求を監視する必要があります。各 WSL は、関連する 1 つまたは複数のワークステーション・ハンドラ (WSH) を使用して、ワークステーション・クライアントの負荷を管理します。各 WSH は、単一の接続を介して、特定のワークステーション・クライアントで処理されるすべての要求と応答を多重化することにより、複数のワークステーション・クライアントを管理できます。

管理者は、アプリケーションの UBBCONFIG ファイルの SERVERS セクションで WSL サーバを指定することにより、ワークステーション・クライアントに対して ATMI アプリケーションへのアクセスを許可することができます。LLE の *min* および *max* パラメータのデフォルト値を上書きするには、WSL サーバ用のコマンド行オプションである -z および -Z を指定する必要があります (詳細については、第 2 章の 34 ページ「min 値と max 値について」を参照してください)。ただし、リンク・レベルの暗号化は、ローカル・マシンとワークステーション・クライアントの両方に LLE がインストールされている場合にのみ使用できます。

注記 ネットワーク接続の端にあるワークステーション・クライアントでは、TMINENCRYPTBITS および TMAXENCRYPTBITS の 2 つの環境変数を使用して、LLE の *min* および *max* パラメータのデフォルト値を上書きできます。

ワークステーション・クライアントのリンクに LLE を設定するには、次の手順に従います。

1. ATMI アプリケーションの MASTER マシンで作業しており、アプリケーションが非アクティブであることを確認します。
2. テキスト・エディタで UBBCONFIG を開き、SERVERS セクションに次の行を追加します。

```
*SERVERS
WSL     SRVGRP="group_name" SRVID=server_number ...
        CLOPT="-A -- -z min -Z max ..."
```

3. tmloadcf(1) を実行してコンフィギュレーションをロードします。tmloadcf コマンドを実行すると、UBBCONFIG が解析され、TUXCONFIG 変数が指す場所にバイナリ形式の TUXCONFIG ファイルがロードされます。

上の例では、tmboot(1) を実行すると ATMI アプリケーションが起動し、"-A --z *min* -Z *max*" というコマンド行オプションが WSL サーバに渡されます。ワークステーション・クライアントと WSH との間でネットワーク・リンクを確立する場合、ワークステーション・クライアントと WSL は、双方でサポートされるキーの最大サイズが一致するまでキー・サイズの調整を行います。

詳細については、『ファイル形式、データ記述方法、MIB、およびシステム・プロセスのリファレンス』の WSL(5)、WS_MIB(5)、および UBBCONFIG(5) を参照してください。

ブリッジ間のリンクに LLE を設定する方法

BEA Tuxedo システムのアーキテクチャでは、マルチ・マシン・アプリケーション (複数のマシンで構成されたアプリケーション) 内のマシン間に多重化したチャネルを確立して、ネットワーク通信を最適化します。BEA Tuxedo のメッセージは、このチャネルを通じて双方向に流れます。メッセージ・トラフィックは、ブリッジ・サーバと呼ばれる専用の ATMI サーバによって管理されます。

管理者は、ブリッジ・サーバが置かれた ATMI アプリケーション内の各マシンに対して、UBBCONFIG ファイルの NETWORK セクションにエントリを作成します。LLE の *min* および *max* パラメータのデフォルト値を上書きする場合は、ブリッジ・サーバのオプションのランタイム・パラメータである MINENCRYPTBITS および MAXENCRYPTBITS を指定する必要があります (詳細については、第 2 章の 34 ページ「min 値と max 値について」を参照してください)。ただし、ブリッジ間のリンク・レベルの暗号化は、ブリッジ・サーバが置かれたマシンに LLE がインストールされている場合にのみ使用できます。

ブリッジ間のリンクに LLE を設定するには、次の手順に従います。

1. ATMI アプリケーションの MASTER マシンで作業しており、アプリケーションが非アクティブであることを確認します。
2. テキスト・エディタで UBBCONFIG を開き、NETWORK セクションに次の行を追加します。

```
*NETWORK
LMID    NADDR="bridge_network_address" BRIDGE="bridge_device"
        NLSADDR="listen_network_address"
        MINENCRYPTBITS=min
        MAXENCRYPTBITS=max
```

LMID は、ブリッジ・サーバが置かれた論理マシンであり、BRIDGE パラメータで指定されたネットワーク・デバイスに直接アクセスできます。

3. tmloadcf(1) を実行してコンフィギュレーションをロードします。tmloadcf コマンドを実行すると、UBBCONFIG が解析され、TUXCONFIG 変数が指す場所にバイナリ形式の TUXCONFIG ファイルがロードされます。

上の例では、tmboot(1) を実行すると、ATMI アプリケーションが起動し、ブリッジ・サーバは、TUXCONFIG ファイルを読み込んで、MINENCRYPTBITS および MAXENCRYPTBITS などのさまざまなパラメータにアクセスします。リモートのブリッジ・サーバとの間でネットワーク・リンクを確立する場合、ローカルとリモートのブリッジ・サーバは、双方でサポートされるキーの最大サイズが一致するまでキー・サイズの調整を行います。

詳細については、『ファイル形式、データ記述方法、MIB、およびシステム・プロセスのリファレンス』の TM_MIB(5) および UBBCONFIG(5) を参照してください。

tlisten のリンクに LLE を設定する方法

tlisten(1) は、ネットワークに依存しないリスナ・プロセスであり、tmboot(1) などの管理ユーティリティを実行できるマルチ・マシン・アプリケーションのノード間の接続を確立します。アプリケーション管理者は、UBBCONFIG ファイルの NETWORK セクションで定義されたすべてのマシンに tlisten をインストールします。

tlisten リンクに LLE を設定するには、第 2 章の 36 ページ「ブリッジ間のリンクに LLE を設定する方法」で説明した手順に従います。必要に応じ、次のコマンドを入力して、ローカル・マシンで別個の tlisten のインスタンスを起動できます。

```
tlisten -l nlsaddr [-z min -z max]
```

nlsaddr 値は、UBBCONFIG ファイルの NETWORK セクションでこのマシンの NLSADDR パラメータに指定した値と同じにする必要があります。詳細については、『BEA Tuxedo コマンド・リファレンス』の tlisten(1)、および『ファイル形式、データ記述方法、MIB、およびシステム・プロセスのリファレンス』の TM_MIB(5) および UBBCONFIG(5) を参照してください。

ドメイン・ゲートウェイのリンクに LLE を設定する方法

ドメイン・ゲートウェイは GWTDOMAIN プロセスであり、複数の ATMI アプリケーション間でサービス要求とサービス応答を中継します。また、TCP/IP などのネットワーク・トランスポート・プロトコルを介して流れる、特別に設計されたトランザクション処理 (TP) プロトコルを使用して相互運用性を実現します。

ドメイン・ゲートウェイはドメイン・ゲートウェイ・グループに属します。各ドメイン・ゲートウェイ・グループには、別個の Domains コンフィギュレーション・ファイルが必要です。ドメイン・ゲートウェイ・グループは、ローカル・ドメイン・アクセス・ポイント (LDOM) および LDOM の通信先となるリモート・ドメイン・アクセス・ポイント (RDOM) で構成されます。アプリケーションのコンフィギュレーション・ファイルである UBBCONFIG および TUXCONFIG と同様に、Domains のコンフィギュレーション・ファイルもテキスト形式で作成され、バイナリ形式に変換されます。テキスト形式の場合は DMCONFIG ファイル、バイナリ形式の場合は BDMCONFIG ファイルと呼ばれます。DMCONFIG ファイルと BDMCONFIG ファイル、および関連する環境変数については、『ファイル形式、データ記述方法、MIB、およびシステム・プロセスのリファレンス』の DMCONFIG(5) のリファレンス・ページを参照してください。

管理者は、各ローカル・ドメイン・アクセス・ポイント (リモート・ドメイン・アクセス・ポイントからローカル・サービスに対する要求を受け取るポイント) に対して、DMCONFIG ファイルの DM_TDOMAIN セクションにエントリを作成する必要があります。また、定義済みのローカル・ドメイン・アクセス・ポイントからアクセス可能なリモート・ドメイン・アクセス・ポイントに対してエントリを作成する必要もあります。LLE の *min* および *max* パラメータのデフォルト値を上書きする場合は、ドメイン・アクセス・ポイントごとに、オプションのランタイム・パラメータである MINENCRYPTBITS および MAXENCRYPTBITS を指定する必要があります (詳細については、第 2 章の 34 ページ「min 値と max 値について」を参照してください)。ただし、ドメイン間のリンク・レベルの暗号化は、ドメインがあるマシンに LLE がインストールされている場合にのみ使用できます。

ドメイン・ゲートウェイのリンクに LLE を設定するには、次の手順に従います。

1. ATMI アプリケーションの MASTER マシンで作業しており、ATMI アプリケーションが非アクティブであることを確認します。
2. テキスト・エディタで DMCONFIG を開き、DM_TDOMAIN セクションに次の行を追加します。

```
*DM_TDOMAIN
# Local network addresses
LDOM    NWADDR="local_domain_network_address"
        NWDEVICE="local_domain_device"
```

```
        MINENCRYPTBITS=min
        MAXENCRYPTBITS=max
    .
    .
    .
# Remote network addresses
RDOM    NWADDR="remote_domain_network_address"
        NWDEVICE="remote_domain_device"
        MINENCRYPTBITS=min
        MAXENCRYPTBITS=max
    .
    .
    .
```

LDOM はローカル・ドメイン・アクセス・ポイントの識別子であり、*RDOM* はリモート・ドメイン・アクセス・ポイントの識別子です。

3. dmloadcf(1) を実行してコンフィギュレーションをロードします。dmloadcf コマンドを実行すると、DMCONFIG が解析され、BDMCONFIG 変数が指す場所にバイナリ形式の BDMCONFIG ファイルがロードされます。

上の例では、tmboot(1) を実行すると、ATMI アプリケーションが起動します。各ドメイン･ゲートウェイは BDMCONFIG ファイルを読み込んで MINENCRYPTBITS および MAXENCRYPTBITS などのさまざまなパラメータにアクセスし、ローカル・ドメインおよびリモート・ドメインにそれらのパラメータを複製転送します。ローカル・ドメインがリモート・ドメインとのネットワーク・リンクを確立するとき、これらの 2 つのドメインは、双方でサポートされるキーの最大サイズが一致するまでキー・サイズの調整を行います。

詳細については、『ファイル形式、データ記述方法、MIB、およびシステム・プロセスのリファレンス』の DMCONFIG(5) を参照してください。また、『BEA Tuxedo Domains コンポーネント』の 第 2 章の 33 ページ「Domains でセキュリティを設定する」も参照してください。

関連項目

- 第 1 章の 23 ページ「リンク・レベルの暗号化」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 第 1 章の 54 ページ「セキュリティの相互運用性」
- 第 1 章の 58 ページ「セキュリティ機能の互換性」

公開鍵セキュリティの管理

分散型の ATMI アプリケーションの安全性を確保する最も効果的な方法は、リンク・レベルの暗号化と公開鍵による暗号化を組み合わせることです。公開鍵による暗号化は、公開鍵セキュリティの基盤となるフレームワークです。

公開鍵セキュリティにより、メッセージ・ベースのデジタル署名とメッセージ・ベースの暗号化を ATMI アプリケーションに統合することができます。これらの機能を組み合わせると、データの完全性と機密性が保たれます。これは、ATMI アプリケーションが外部の ATMI アプリケーションまたはワークステーション・クライアントと相互運用する場合に特に重要です。

公開鍵セキュリティで推奨されている事項について

- 実行できるセキュリティのレベルは、主に ATMI アプリケーションの動作環境によって決まります。最も高いセキュリティのレベルを実現するには、秘密鍵の情報を保護するハードウェア・デバイスをインストールします。
- 鍵の有効期限と鍵の更新手順についての方針を決定します。認証局が発行する証明書の有効期限は、システムの操作に大きく影響する場合があります。したがって、有効期限の前に、更新した証明書を発行できるようにしておく必要があります。

公開鍵と秘密鍵の組み合わせの割り当て

アプリケーションの管理者と開発者は、公開鍵と秘密鍵の組み合わせ、および関連するデジタル証明書を認証するための認証局を選択する必要があります。次に、鍵の組み合わせを ATMI アプリケーションに割り当てる方法を決定する必要があります。鍵の組み合わせを割り当てる方法はたくさんあります。管理者は、次のうち、1 つまたは複数の方法で鍵の組み合わせを割り当てることができます。

- 1 つの公開鍵を ATMI アプリケーション全体に割り当てる
- ATMI アプリケーション内の各マシンに対して公開鍵と秘密鍵の組み合わせを 1 つ割り当てる
- ATMI アプリケーション内の各サーバに対して公開鍵と秘密鍵の組み合わせを 1 つ割り当てる
- ATMI アプリケーション内の各サービスに対して公開鍵と秘密鍵の組み合わせを 1 つ割り当てる

- 各エンド・ユーザに対して公開鍵と秘密鍵の組み合わせを 1 つ割り当てる

アプリケーションの管理者と開発者は、鍵の組み合わせを割り当てる方法を選択し、実際に割り当てる責任があります。ただし、鍵の組み合わせを割り当てた後の管理作業を行う必要はありません。鍵の配布と管理は、公開鍵セキュリティのプラグインによって行われます。

デジタル署名方針の設定

管理者は、次のコンフィギュレーション・パラメータを使用して、ATMI アプリケーションのデジタル署名に関する方針を設定します。

パラメータ名	説明	設定
UBBCONFIG の SIGNATURE_AHEAD (TM_MIB の TA_SIGNATURE_AHEAD)	デジタル署名付きのメッセージ・バッファに記録されたタイムスタンプ値と、そのメッセージ・バッファが受信された時刻の最大時間差。デジタル署名のタイムスタンプ値が遠い将来の時刻を示す場合、受信側のプロセスではメッセージ・バッファを拒否します。	1 ～ 2147483647 秒。デフォルト値は 3600 秒 (1 時間) です。
UBBCONFIG の SIGNATURE_BEHIND (TM_MIB の TA_SIGNATURE_BEHIND)	デジタル署名付きのメッセージ・バッファが受信された時刻と、そのメッセージ・バッファに添付されたタイムスタンプ値との最大時間差。デジタル署名のタイムスタンプ値が遠い過去の時刻を示す場合、受信側のプロセスではメッセージ・バッファを拒否します。	1 ～ 2147483647 秒。デフォルト値は 604800 秒 (1 週間) です。
UBBCONFIG の SIGNATURE_REQUIRED (TM_MIB の TA_SIGNATURE_REQUIRED)	受信側のプロセスで受け付けるメッセージ・バッファを、デジタル署名付きのものだけに制限するかどうかを決定します。	Y (デジタル署名が必要) または N (デジタル署名は不要) を指定します。デフォルト値は N です。

デジタル署名のタイムスタンプに設定された将来の日付を制限する

SIGNATURE_AHEAD は、コンフィギュレーション階層のうち、ドメイン・レベルで指定されるため、このパラメータに割り当てた値は、ATMI アプリケーションで実行されるすべてのプロセスに適用されます。ドメイン全体にわたるパラメータは、UBBCONFIG ファイルの RESOURCES セクションおよび TM_MIB の T_DOMAIN クラスで設定されます。

SIGNATURE_AHEAD パラメータは、受信したメッセージ・バッファに添付されたタイムスタンプと、検証システムのローカル・クロックに表示される現在時刻との最大時間差を設定します。最小値は 1 秒です。最大値は 2147483647 秒です。デフォルトは 3600 秒 (1 時間) です。

デジタル署名のタイムスタンプが遠い将来の時刻を示す場合、その署名は無効と見なされます。このパラメータを設定すると、将来の日付が指定された署名を拒否できます。ただし、ローカル・クロックの時刻との間に多少ずれが生じていても、その分は無視されます。

将来の日付を制限する UBBCONFIG のエントリ例

```
*RESOURCES
SIGNATURE_AHEAD 2400
```

デジタル署名のタイムスタンプに設定された過去の日付を制限する

SIGNATURE_BEHIND は、コンフィギュレーション階層のうち、ドメイン・レベルで指定されるため、このパラメータに割り当てた値は、ATMI アプリケーションで実行されるすべてのプロセスに適用されます。ドメイン全体にわたるパラメータは、UBBCONFIG ファイルの RESOURCES セクションおよび TM_MIB の T_DOMAIN クラスで設定されます。

SIGNATURE_BEHIND パラメータは、検証システムのローカル・クロックに表示される現在時刻と、受信したメッセージ・バッファに添付されたタイムスタンプとの最大時間差を設定します。最小値は 1 秒です。最大値は 2147483647 秒です。デフォルト値は 604800 秒 (1 週間) です。

デジタル署名のタイムスタンプ値が遠い過去の時刻を示す場合、その署名は無効と見なされます。このパラメータを設定すると、リプレイ攻撃、つまり、署名付きの有効なバッファが再びシステムに送信されるのを阻止することができます。ただし、非同期通信を行うシステム、たとえばディスク・ベースのキューを使用するシステムでは、かなり古い署名が添付されたバッファが有効と見なされる場合があります。したがって、非同期通信を行うシステムでは、SIGNATURE_BEHIND の設定を増やしてください。

過去の日付を制限する UBBCONFIG のエントリ例

```
*RESOURCES
SIGNATURE_BEHIND 300000
```

受信メッセージに対する署名方針の適用

SIGNATURE_REQUIRED は、コンフィギュレーションの階層のうち、次の 4 つのレベルのどこでも指定できます。

- UBBCONFIG ファイルの RESOURCES セクション、または TM_MIB の T_DOMAIN クラス
- UBBCONFIG ファイルの MACHINES セクション、または TM_MIB の T_MACHINE クラス
- UBBCONFIG ファイルの GROUPS セクション、または TM_MIB の T_GROUP クラス
- UBBCONFIG ファイルの SERVICES セクション、または TM_MIB の T_SERVICE クラス

特定のレベルで SIGNATURE_REQUIRED に Y (はい) を設定すると、下位レベルで動作するすべてのプロセスに署名が必要となります。たとえば、mach1 というマシンで SIGNATURE_REQUIRED に Y を設定すると、mach1 上で動作するすべてのプロセスは、デジタル署名が添付されたメッセージだけを受け付けます。

- ドメイン全体にわたるレベル (RESOURCES セクションまたは T_DOMAIN クラス) で設定すると、このパラメータは、ゲートウェイ・プロセスによって宣言されるアプリケーション・サービスを含め、ドメイン内で宣言されるすべてのアプリケーション・サービスに適用されます。デフォルト値は N です。
- マシン・レベル (MACHINES セクションまたは T_MACHINE クラス) で設定すると、このパラメータは、ゲートウェイ・プロセスによって宣言されるアプリケーション・サービスを含め、特定のマシンで宣言されるすべてのアプリケーション・サービスに適用されます。デフォルト値は N です。
- グループ・レベル (GROUPS セクションまたは T_GROUP クラス) で設定すると、このパラメータは、ゲートウェイ・プロセスによって宣言されるアプリケーション・サービスを含め、特定のグループによって宣言されるすべてのアプリケーション・サービスに適用されます。デフォルト値は N です。
- サービス・レベル (SERVICES セクションまたは T_SERVICE クラス) で設定すると、このパラメータは、ゲートウェイ・プロセスによって宣言されるインスタンスを含め、ドメイン内で宣言される特定サービスのすべてのインスタンスに適用されます。デフォルト値は N です。

ドメイン全体にわたるレベル、マシン・レベル、グループ・レベル、またはサービス・レベルでは、SIGNATURE_REQUIRED=Y および ENCRYPTION_REQUIRED=Y を同時に指定することができます。ENCRYPTION_REQUIRED の詳細については、第 2 章の 46 ページ「受信メッセージに対する暗号化方針の適用」を参照してください。

制限

SIGNATURE_REQUIRED を指定するかどうかの方針は、アプリケーション・サービス、アプリケーション・イベント、およびアプリケーション・エンキュー要求に対してのみ適用されます。システム生成されたサービス呼び出しや、システム・イベントのポストには適用されません。

例

mach1 というマシンに SIGNATURE_REQUIRED を設定するには、次の手順に従います。

1. ATMI アプリケーションの MASTER マシンで作業しており、ATMI アプリケーションが非アクティブであることを確認します。
2. テキスト・エディタで UBBCONFIG を開き、MACHINES セクションに次の行を追加します。

```
*MACHINES
mach1  LMID="machine_logical_name"
       TUXCONFIG="absolute_path_name_to_tuxconfig_file"
       TUXDIR="absolute_path_name_to_BEA_Tuxedo_directory"
       APPDIR="absolute_path_name_to_application_directory"
       SIGNATURE_REQUIRED=Y
```

3. tmloadcf(1) を実行してコンフィギュレーションをロードします。tmloadcf コマンドを実行すると、UBBCONFIG が解析され、TUXCONFIG 変数が指す場所にバイナリ形式の TUXCONFIG ファイルがロードされます。

上の例では、tmboot(1) を実行すると、ATMI アプリケーションが起動し、SIGNATURE_REQUIRED=Y パラメータが mach1 というマシンに渡されます。この時点で、mach1 によって宣言されたすべてのアプリケーション・サービスは、ゲートウェイ・プロセスによって宣言されたアプリケーション・サービスも含め、有効なデジタル署名が添付されたメッセージだけを受け付けることができます。mach1 によって制御されるプロセスが、有効なデジタル署名が添付されていないメッセージを受信すると、システム側では次のアクションが実行されます。

- userlog(3c) メッセージを生成します (重大度は WARN (警告))。
- バッファを破棄し、プロセスで受信されなかったように扱います。

注記 NULL バッファ (空のバッファ) にデジタル署名を添付することはできません。したがって、デジタル署名を必要とするプロセスで受信された NULL バッファは、上記のいずれかの方法で、システム側で拒否されます。

イベント・ブローカの署名方針の適用

ポスト済みのメッセージ・バッファにデジタル署名が添付されると、これらの署名は保存され、メッセージ・バッファと共に、適切なイベントのサブスクライバに転送されます。

TMUSREVT(5) システム・サーバが、デジタル署名を必要とするドメイン、マシン、またはサーバ・グループで実行されている場合、このサーバは、コンポジット署名ステータスが TPSIGN_OK に設定されていないイベントを拒否します。詳細については、第 3 章の 59 ページ「コンポジット署名ステータスについて」を参照してください。

TMUSREVT サーバは、サービスの呼び出しやキューへのメッセージの登録などの、サブスクリプションに関する通知アクションを実行できます。有効なデジタル署名を必要とするサービスやキューに対して、有効なデジタル署名が添付されていないメッセージをポストすると、サブスクリプションの通知アクションは失敗します。

システム・イベント (システム側でポストされ、TMSYSEVT サーバで処理されるイベント) には、デジタル署名を添付することができます。デジタル署名に関する管理方針は、TMSYSEVT(5) サーバには適用されません。

/Q の署名方針の適用

キューに登録されたバッファにデジタル署名が添付されると、署名はキュー内に保存され、キューから取り出すプロセスの実行時に転送されます。また、メッセージが TMQFORWARD(5) によって処理され、サービスが呼び出されると、署名は保存されます。

TMQUEUE(5) システム・サーバが、デジタル署名を必要とするドメイン、マシン、またはサーバ・グループで実行されている場合、このサーバは、コンポジット署名ステータスが TPSIGN_OK に設定されていないエンキュー要求を拒否します。詳細については、第 3 章の 59 ページ「コンポジット署名ステータスについて」を参照してください。さらに、キュー空間に関連するサービス名に対してこのような方針が有効な場合、TMQUEUE サーバはデジタル署名を必要とします。

リモート・クライアントの署名方針の適用

ワークステーション・ハンドラ (WSH) が、デジタル署名を必要とするドメイン、マシン、またはサーバ・グループで実行されている場合、WSH は、コンポジット署名ステータスが TPSIGN_OK に設定されていない、アプリケーション・データを含むメッセージ・バッファを拒否します。詳細については、第 3 章の 59 ページ「コンポジット署名ステータスについて」を参照してください。

暗号化方針の設定

管理者は、次のコンフィギュレーション・パラメータを使用して、ATMI アプリケーションの暗号化方針を設定します。

パラメータ名	説明	設定
UBBCONFIG の ENCRYPTION_REQUIRED (TM_MIB の TA_ENCRYPTION_REQUIRED)	受信側のプロセスで受け付けるメッセージ・バッファを、暗号化されたものだけに制限するかどうかを決定します。	Y (暗号化が必要) または N (暗号化は不要) を指定します。デフォルト値は N です。

受信メッセージに対する暗号化方針の適用

ENCRYPTION_REQUIRED は、コンフィギュレーションの階層のうち、次の 4 つのレベルのどこでも指定できます。

- UBBCONFIG ファイルの RESOURCES セクション、または TM_MIB の T_DOMAIN クラス
- UBBCONFIG ファイルの MACHINES セクション、または TM_MIB の T_MACHINE クラス
- UBBCONFIG ファイルの GROUPS セクション、または TM_MIB の T_GROUP クラス
- UBBCONFIG ファイルの SERVICES セクション、または TM_MIB の T_SERVICE クラス

特定のレベルで ENCRYPTION_REQUIRED に Y (はい) を設定すると、下位レベルで動作するすべてのプロセスに暗号化が必要となります。たとえば、mach1 というマシンで ENCRYPTION_REQUIRED に Y を設定すると、mach1 上で動作するすべてのプロセスは、暗号化されたメッセージだけを受け付けます。

- ドメイン全体にわたるレベル (RESOURCES セクションまたは T_DOMAIN クラス) で設定すると、このパラメータは、ゲートウェイ・プロセスによって宣言されるアプリケーション・サービスを含め、ドメイン内で宣言されるすべてのアプリケーション・サービスに適用されます。デフォルト値は N です。
- マシン・レベル (MACHINES セクションまたは T_MACHINE クラス) で設定すると、このパラメータは、ゲートウェイ・プロセスによって宣言されるアプリケーション・サービスを含め、特定のマシンで宣言されるすべてのアプリケーション・サービスに適用されます。デフォルト値は N です。

- グループ・レベル (GROUPS セクションまたは T_GROUP クラス) で設定すると、このパラメータは、ゲートウェイ・プロセスによって宣言されるアプリケーション・サービスを含め、特定のグループによって宣言されるすべてのアプリケーション・サービスに適用されます。デフォルト値は N です。
- サービス・レベル (SERVICES セクションまたは T_SERVICE クラス) で設定すると、このパラメータは、ゲートウェイ・プロセスによって宣言されるインスタンスを含め、ドメイン内で宣言される特定サービスのすべてのインスタンスに適用されます。デフォルト値は N です。

ドメイン全体にわたるレベル、マシン・レベル、グループ・レベル、またはサービス・レベルでは、ENCRYPTION_REQUIRED=Y および SIGNATURE_REQUIRED=Y を同時に指定することができます。SIGNATURE_REQUIRED の詳細については、第 2 章の 43 ページ「受信メッセージに対する署名方針の適用」を参照してください。

制限

ENCRYPTION_REQUIRED を指定するかどうかの方針は、アプリケーション・サービス、アプリケーション・イベント、およびアプリケーション・エンキュー要求に対してのみ適用されます。システム生成されたサービス呼び出しや、システム・イベントのポストには適用されません。

例

STDGRP というサーバ・グループに ENCRYPTION_REQUIRED を設定するには、次の手順に従います。

1. ATMI アプリケーションの MASTER マシンで作業しており、ATMI アプリケーションが非アクティブであることを確認します。
2. テキスト・エディタで UBBCONFIG を開き、GROUPS セクションに次の行を追加します。

```
*GROUPS
STDGRP LMID="machine_logical_name"
       GRPNO="server_group_number"
       ENCRYPTION_REQUIRED=Y
```

3. tmloadcf(1) を実行してコンフィギュレーションをロードします。tmloadcf コマンドを実行すると、UBBCONFIG が解析され、TUXCONFIG 変数が指す場所にバイナリ形式の TUXCONFIG ファイルがロードされます。

上の例では、tmboot(1) を実行すると、ATMI アプリケーションが起動し、ENCRYPTION_REQUIRED=Y パラメータが STDGRP というサーバ・グループに渡されます。この時点で、STDGRP によって宣言されたすべてのアプリケーション・サービスは、ゲートウェイ・プロセスによって宣言されたアプリケーション・サービスも含め、暗号化のエンベロープで保護されたメッセージだけを受け付けることができます。STDGRP によって制御されるプロセスが、暗号化されていないメッセージを受信すると、システム側では次のアクションが実行されます。

- userlog(3c) メッセージを生成します (重大度は ERROR (エラー))。
- バッファを破棄し、プロセスで受信されなかったように扱います。

注記 NULL バッファ (空のバッファ) は暗号化できません。したがって、暗号化を必要とするプロセスで受信された NULL バッファは、上記のいずれかの方法で、システム側で拒否されます。

イベント・ブローカの暗号化方針の適用

ポスト済みのメッセージ・バッファが暗号化されると、これらの署名は保存され、暗号化されたメッセージの内容と共に、適切なイベントのサブスクライバに転送されます。

TMUSREVT(5) システム・サーバが、暗号化を必要とするドメイン、マシン、またはサーバ・グループで実行されている場合、このサーバは、暗号化されていないメッセージを拒否します。

TMUSREVT サーバは、サービスの呼び出しやキューへのメッセージの登録などの、サブスクリプションに関する通知アクションを実行できます。入力する情報の暗号化を必要とするサービスやキューに対して、暗号化されていないメッセージをポストすると、サブスクリプションの通知アクションは失敗します。また、サブスクライバが適切な復号化キーを保持していない場合も、イベント通知アクションは失敗します。

システム・イベント (システム側でポストされ、TMSYSEVT サーバで処理されるイベント) は、暗号化できます。暗号化に関する管理方針は、TMSYSEVT(5) サーバには適用されません。

/Q の暗号化方針の適用

キューに登録されたバッファが暗号化されると、このステータスはキュー内に保存され、バッファは、キューから取り出すプロセスの実行時に暗号化された形式で転送されます。また、メッセージが TMQFORWARD(5) によって処理され、サービスが呼び出されると、暗号化のステータスは保存されます。

TMQUEUE(5) システム・サーバが、暗号化を必要とするドメイン、マシン、またはサーバ・グループで実行されている場合、このサーバは、暗号化されていないエンキュー要求を拒否します。さらに、キュー空間に関連するサービス名に対してこのような方針が有効な場合、TMQUEUE サーバは、暗号化を必要とします。

リモート・クライアントの暗号化方針の適用

ワークステーション・ハンドラ (WSH) が、暗号化を必要とするドメイン、マシン、またはサーバ・グループで実行されている場合、WSH は、暗号化されていないアプリケーション・データ・バッファを含むメッセージ・バッファを拒否します。

プラグインによる復号化キーの初期化

管理者は、次のコンフィギュレーション・パラメータを使用して、アプリケーション内で動作するシステム・プロセスのプリンシパル名と復号化キーを指定します。

パラメータ名	説明	設定
UBBCONFIG の SEC_PRINCIPAL_NAME (TM_MIB の TA_SEC_PRINCIPAL_NAME)	ターゲットのプリンシパル名。これが 1 つまたは複数のシステム・プロセスの ID になります。	1 ～ 511 文字。
UBBCONFIG の SEC_PRINCIPAL_LOCATION (TM_MIB の TA_SEC_PRINCIPAL_LOCATION)	ターゲットのプリンシパルの復号化 (秘密) キーが格納されているファイルまたはデバイスの位置。	1 ～ 511 文字。指定しない場合、NULL 文字列 (長さゼロ) がデフォルト値になります。
UBBCONFIG の SEC_PRINCIPAL_PASSVAR (TM_MIB の SEC_PRINCIPAL_PASSVAR)	ターゲットのプリンシパルのパスワードが格納されている変数。	1 ～ 511 文字。指定しない場合、NULL 文字列 (長さゼロ) がデフォルト値になります。

上記の 3 つのパラメータは、コンフィギュレーションの階層のうち、次の 4 つのレベルのどこでも指定できます。

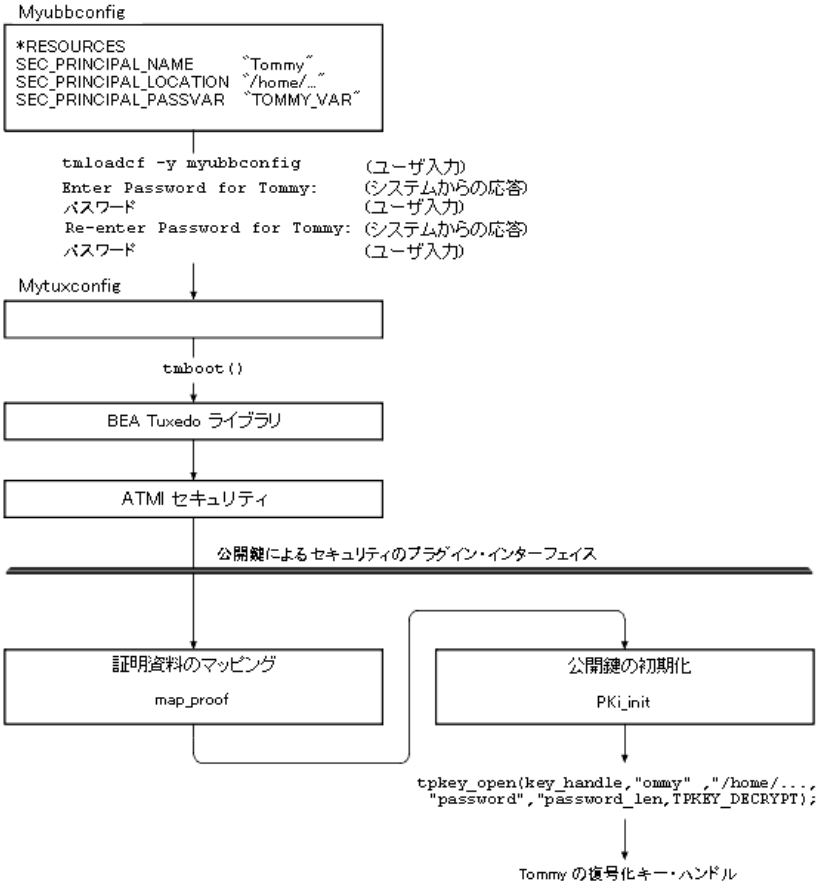
- UBBCONFIG ファイルの RESOURCES セクション、または TM_MIB の T_DOMAIN クラス
- UBBCONFIG ファイルの MACHINES セクション、または TM_MIB の T_MACHINE クラス
- UBBCONFIG ファイルの GROUPS セクション、または TM_MIB の T_GROUP クラス
- UBBCONFIG ファイルの SERVERS セクション、または TM_MIB の T_SERVER クラス

特定のコンフィギュレーション・レベルでのプリンシパル名および復号化キーは、下位レベルで上書きできます。たとえば、mach1 というマシンにプリンシパル名と復号化キーを設定し、mach1 上で動作する serv1 というサーバにプリンシパル名と復号化キーを設定したとします。この場合、mach1 のプロセスは次のように動作します。

- serv1 プロセス以外の mach1 上のすべてのプロセスは、mach1 に割り当てられた復号化キーを使用して、暗号化された受信メッセージ・バッファを復号化します。
- serv1 上のすべてのプロセスは、serv1 に割り当てられた復号化キーを使用して、暗号化された受信メッセージ・バッファを復号化します。

ATMI アプリケーションが起動すると、設定された復号化キーは自動的にオープンします。次の図は、このプロセスのしくみを示しています。

図 2-13 復号化キーの初期化例



次に、この図に示した操作の実行方法を詳しく説明します。

1. 管理者は、ATMI アプリケーションの UBBCONFIG ファイルの特定のレベルで、SEC_PRINCIPAL_NAME、SEC_PRINCIPAL_LOCATION、および SEC_PRINCIPAL_PASSVAR を定義します。
2. 管理者は、tmloadcf(1) を実行してコンフィギュレーションをロードします。tmloadcf コマンドを実行すると、UBBCONFIG が解析され、TUXCONFIG 変数が指す場所にバイナリ形式の TUXCONFIG ファイルがロードされます。
3. 管理者は、プロンプトに従い、ターゲットのプリンシパルのパスワードを入力し、さらにもう一度入力します。

4. 管理者は、tmboot(1) コマンドを実行して ATMI アプリケーションを起動します。
5. 起動時に、map_proof プラグインにより、SEC_PRINCIPAL_NAME、SEC_PRINCIPAL_LOCATION、および SEC_PRINCIPAL_PASSVAR が読み込まれ、各パラメータの値が解析されます。次に、呼び出しプロセスで、要求された復号化キーに対するアクセス権が証明されたかどうかが判別されます。復号化キー、つまり秘密鍵にアクセスできるということは、プリンシパルの ID を所有することと同じです。
6. SEC_PRINCIPAL_PASSVAR に関連するパスワードが、SEC_PRINCIPAL_NAME で指定されたプリンシパルに割り当てられたパスワードと一致する場合、map_proof プラグインは、プリンシパルの名前、位置、およびパスワードを PKi_init プラグインに渡します。
7. PKi_init プラグインは、プリンシパルの名前、位置、およびパスワードを引数に指定し、tpkey_open(3c) を呼び出します。プリンシパルの復号化キー・ハンドルが返されます。

tmloadcf を呼び出してコンフィギュレーションをロードするたびに、SEC_PRINCIPAL_PASSVAR に設定した各復号化キーに対するパスワードの入力を求められます。各パスワードを手動で入力しないで済むようにするには、パスワードを自動的に入力するスクリプトを記述します。このスクリプトには、各パスワードの変数定義を組み込み、次の行で終了する必要があります。

```
tmloadcf -y ubbconfig_name < /dev/null
```

ATMI アプリケーションの起動時にオープンされた復号化キーをクローズするパーミッションは、アプリケーション・プロセスにはありません。復号化キーは、tmshutdown(1) コマンドを実行して ATMI アプリケーションをシャットダウンするまで、オープンしたままの状態になります。

プリンシパル名および復号化キーを指定する UBBCONFIG のエントリ例

```
*RESOURCES
SEC_PRINCIPAL_NAME     "Tommy"
SEC_PRINCIPAL_LOCATION "/home/jhn/secsapp/cert/tommy.pvk"
SEC_PRINCIPAL_PASSVAR  "TOMMY_VAR"
      .
      .
      .
*SERVERS
"TMQUEUE"       SRVGRP="QUEGROUP"      SRVID=1
   CLOPT="-s secsdb:TMQUEUE"
   SEC_PRINCIPAL_NAME=    "TOUPPER"
   SEC_PRINCIPAL_LOCATION="/home/jhn/secsapp/cert/TOUPPER.pvk"
   SEC_PRINCIPAL_PASSVAR= "TOUPPER_VAR"
```

障害のレポートと監査

この節では、デジタル署名とメッセージの暗号化で検出されたエラーがシステム側でどのように処理されるかを説明します。

デジタル署名のエラー処理

メッセージの改ざんが検出された場合、つまり、コンポジット署名ステータスが PSIGN_TAMPERED_MESSAGE または TPSIGN_TAMPERED_CERT の場合 (第 3 章の 59 ページ「コンポジット署名ステータスについて」を参照)、システム側では次のアクションが実行されます。

- userlog(3c) メッセージを生成します (重大度は ERROR (エラー))。
- バッファを破棄し、プロセスで受信されなかったように扱います。

有効期限が切れた証明書、取り消された証明書、有効期限が切れた署名、または古い日付の署名が検出された場合、システム側では次のアクションが実行されます。

- userlog() メッセージを生成します (重大度は WARN (警告))。
- バッファのコンポジット署名ステータスが TPSIGN_OK または TPSIGN_UNKNOWN でない限り、バッファを破棄し、プロセスで受信されなかったように扱います。

SIGNATURE_REQUIRED=Y の設定に基づいて有効なデジタル署名を必要とするプロセスが、コンポジット署名ステータス TPSIGN_UNKNOWN が添付されたメッセージを受信した場合、システム側では次のアクションが実行されます。

- userlog() メッセージを生成します (重大度は WARN (警告))。
- バッファを破棄し、プロセスで受信されなかったように扱います。

暗号化のエラー処理

プロセスが、メッセージの暗号化エンベロープのいずれかと一致するオープンした復号化キーを持たない状態で、暗号化されたメッセージを受信した場合、システムは次のアクションを実行します。

- userlog(3c) メッセージを生成します (重大度は ERROR (エラー))。
- バッファを破棄し、プロセスで受信されなかったように扱います。

ENCRYPTION_REQUIRED=Y の設定に基づいて暗号化された入力を必要とするプロセスが、暗号化されていないメッセージを受信した場合、システムは次のアクションを実行します。

- userlog() メッセージを生成します (重大度は ERROR (エラー))。
- バッファを破棄し、プロセスで受信されなかったように扱います。

関連項目

- 第 1 章の 29 ページ「公開鍵によるセキュリティ機能」
- 第 1 章の 40 ページ「公開鍵のインプリメンテーション」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 第 1 章の 54 ページ「セキュリティの相互運用性」
- 第 1 章の 58 ページ「セキュリティ機能の互換性」

デフォルトの認証と認可の管理

デフォルトの認証および認可は、BEA Tuxedo システムで最初に実現された、従来の認証および認可と同じように機能します。

デフォルトの認証には、認証なし (NONE)、アプリケーション・パスワード (APP_PW)、ユーザ・レベルの認証を実行 (USER_AUTH)、という 3 つのセキュリティ・レベルが用意されています。デフォルトの認可には、2 つのセキュリティ・レベル、つまり、オプションのアクセス制御リスト (ACL) および必須のアクセス制御リスト (MANDATORY_ACL) を指定するセキュリティが用意されています。アクセス制御リストは、ユーザが ATMI アプリケーションへの参加を認証された場合にのみ有効になります。

セキュリティ・レベルの指定

管理者は、UBBCONFIG コンフィギュレーション・ファイルを編集するか、TM_MIB を変更するか、または BEA Administration Console を使用することにより、ATMI アプリケーションのセキュリティ・レベルを指定できます。

コンフィギュレーション・ファイルを編集してセキュリティを指定する

UBBCONFIG ファイルで、SECURITY パラメータに適切な値を設定します。

```
SECURITY {NONE | APP_PW | USER_AUTH | ACL | MANDATORY_ACL}
```

デフォルト値は NONE です。SECURITY に USER_AUTH、ACL、または MANDATORY_ACL を設定すると、システム側で提供される認証サーバ AUTHSVR が起動し、ユーザごとの認証が行われます。

NONE 以外の値を選択した場合は、MASTER サイトで動作する各 ATMI アプリケーションに対して、一意の APPDIR 変数が設定されていることを確認します。セキュリティ機能を使用している場合、複数の ATMI アプリケーションが同じアプリケーション・ディレクトリを共有することはできません。

TM_MIB を変更してセキュリティを指定する

TM_MIB を使用してセキュリティ・レベルを指定するには、T_DOMAIN クラスの TA_SECURITY 属性に値を割り当てなければなりません。ATMI アプリケーションが非アクティブの場合、管理者は、TA_SECURITY に対して UBBCONFIG で有効な任意の値を設定できます。このタスクを完了するには、管理インターフェイス tpadmcall(3c) を実行します。

BEA Administration Console を使用してセキュリティを指定する

BEA Administration Console を使用してセキュリティ・レベルを指定することもできます。BEA Administration Console は、ATMI アプリケーションをコンフィギュレーションし、監視し、動的に再コンフィギュレーションするための Web ベースのツールです。

認証サーバの設定

BEA Tuxedo サーバである AUTHSVR は、認証を実行する 1 つのサービス (AUTHSVC) を備えています。セキュリティ・レベルが ACL または MANDATORY_ACL に設定されている場合、AUTHSVR サーバは、AUTHSVC を ..AUTHSVC として宣言します。

AUTHSVC を ATMI アプリケーションに追加するには、UBBCONFIG ファイルで、認証サービスとして AUTHSVC を定義し、認証サーバとして AUTHSVR を定義する必要があります。たとえば、次のように定義します。

```
*RESOURCES
SECURITY    USER_AUTH
```

```
AUTHSVC    AUTHSVC
   .
   .
   .
*SERVERS
AUTHSVR SRVGRP="group_name" SRVID=1 RESTART=Y GRACE=600 MAXGEN=2
CLOPT="-A"
```

パラメータと値の組み合わせである AUTHSVC　AUTHSVC のエントリを省略すると、デフォルトで AUTHSVC が呼び出されます。

別の例を示します。

```
*RESOURCES
SECURITY   ACL
AUTHSVC    ..AUTHSVC
   .
   .
   .
*SERVERS
AUTHSVR SRVGRP="group_name" SRVID=1 RESTART=Y GRACE=600 MAXGEN=2
CLOPT="-A"
```

パラメータと値の組み合わせである AUTHSVC　..AUTHSVC のエントリを省略すると、デフォルトで ..AUTHSVC が呼び出されます。

AUTHSVR は、ATMI アプリケーション固有のロジックをインプリメントする認証サーバと置き換えることができます。たとえば、広く使用されている Kerberos のメカニズムを使用して認証を行うため、認証サーバをカスタマイズすることもできます。

カスタマイズした認証サービスを ATMI アプリケーションに追加するには、UBBCONFIG ファイルで認証サービスと認証サーバを定義する必要があります。たとえば、次のように定義します。

```
*RESOURCES
SECURITY   USER_AUTH
AUTHSVC    KERBEROS
   .
   .
   .
*SERVERS
KERBEROSSVR SRVGRP="group_name" SRVID=1 RESTART=Y GRACE=600 MAXGEN=2
CLOPT="-A"
```

関連項目

- 第 2 章の 57 ページ「アプリケーション・パスワードによるセキュリティを有効にする方法」
- 第 2 章の 58 ページ「ユーザ・レベルの認証によるセキュリティを有効にする方法」
- 第 2 章の 63 ページ「アクセス制御リストによるセキュリティの有効化」
- 第 1 章の 43 ページ「デフォルトの認証と認可」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」
- 『ファイル形式、データ記述方法、MIB、およびシステム・プロセスのリファレンス』の AUTHSVR(5)

アプリケーション・パスワードによるセキュリティを有効にする方法

デフォルトの認証には、「アプリケーション・パスワード」というセキュリティ・レベルが用意されており、これは、コンフィギュレーション・ファイルの SECURITY APP_PW で指定すると有効になります。このセキュリティ・レベルでは、ATMI アプリケーションに参加するすべてのクライアントに対してアプリケーション・パスワードの入力が求められます。管理者は、ATMI アプリケーション全体に対して 1 つのパスワードを定義し、認可されたユーザだけにパスワードを通知します。

APP_PW のセキュリティ・レベルを有効にするには、次の手順に従います。

1. ATMI アプリケーションの MASTER マシンで作業しており、ATMI アプリケーションが非アクティブであることを確認します。
2. UBBCONFIG ファイルの RESOURCES セクションの SECURITY パラメータに APP_PW を設定します。
3. tmloadcf(1) を実行してコンフィギュレーションをロードします。tmloadcf コマンドを実行すると、UBBCONFIG が解析され、TUXCONFIG 変数が指す場所にバイナリ形式の TUXCONFIG ファイルがロードされます。
4. パスワードの入力を求められます。パスワードは、30 文字以内で構成します。これは ATMI アプリケーションのパスワードとなり、tmadmin の passwd コマンドを使用して変更しない限り有効です。

5. 電話や手紙など、オンライン以外の方法で、ATMI アプリケーションの認可ユーザに対してアプリケーション・パスワードを配布します。

関連項目

- 第 1 章の 43 ページ「デフォルトの認証と認可」
- 第 2 章の 54 ページ「デフォルトの認証と認可の管理」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」

ユーザ・レベルの認証によるセキュリティを有効にする方法

デフォルトの認証には、「ユーザ・レベルの認証」というセキュリティ・レベルが用意されており、これは、コンフィギュレーション・ファイルの SECURITY USER_AUTH で指定すると有効になります。このセキュリティ・レベルでは、ATMI アプリケーションに参加するため、各クライアントは、アプリケーション・パスワードのほか、有効なユーザ名とユーザ固有のデータ (パスワードなど) を提示しなければなりません。ユーザごとのパスワードは、tpusr ファイルに格納されている、ユーザ名とクライアント名の組み合わせに関連付けられたパスワードに一致しなければなりません。ユーザごとのパスワードと、tpusr 内のユーザ名 / クライアント名およびパスワードとの照合は、認証サーバ AUTHSVR の認証サービス AUTHSVC によって実行されます。

USER_AUTH のセキュリティ・レベルを有効にするには、次の手順に従います。

1. UBBCONFIG ファイルを設定します。
2. ユーザ・ファイルとグループ・ファイルを設定します。

これらの手順については、次の 2 つの節で説明します。

UBBCONFIG ファイルを設定する

1. ATMI アプリケーションの MASTER マシンで作業しており、ATMI アプリケーションが非アクティブであることを確認します。
2. テキスト・エディタで UBBCONFIG を開き、RESOURCES セクションと SERVERS セクションに次の行を追加します。

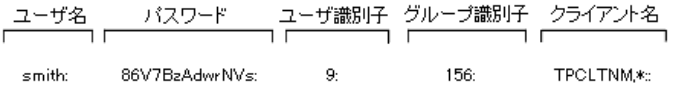
```
*RESOURCES
SECURITY    USER_AUTH
AUTHSVC     AUTHSVC
      .
      .
      .
*SERVERS
AUTHSVR SRVGRP="group_name" SRVID=1 RESTART=Y GRACE=600 MAXGEN=2
CLOPT="-A"
```

   CLOPT="-A" を指定すると、tmboot(1) は、tmboot によって ATMI アプリケーションが起動するときに、"-A" で呼び出されたデフォルトのコマンド行オプションだけを AUTHSVR に渡します。デフォルトでは、AUTHSVR は、tpusr というファイル内のクライアント・ユーザ情報を使用して、ATMI アプリケーションに参加予定のクライアントを認証します。tpusr は、ATMI アプリケーションの APPDIR 変数で定義する最初のパス名によって参照されるディレクトリにあります。
3. tmloadcf(1) を実行してコンフィギュレーションをロードします。tmloadcf コマンドを実行すると、UBBCONFIG が解析され、TUXCONFIG 変数が指す場所にバイナリ形式の TUXCONFIG ファイルがロードされます。
4. パスワードの入力を求められます。パスワードは、30 文字以内で構成します。これは ATMI アプリケーションのパスワードとなり、tmadmin の passwd コマンドを使用して変更しない限り有効です。
5. 電話や手紙など、オンライン以外の方法で、ATMI アプリケーションの認可ユーザに対してアプリケーション・パスワードを配布します。

ユーザ・ファイルとグループ・ファイルの設定

デフォルトの認可システムで使用できる AUTHSVR とアクセス制御チェック機能では、tpusr というユーザ・ファイルが必要です。このファイルには、ATMI アプリケーションへの参加を許可されたクライアント・ユーザのリストが含まれています。アプリケーション管理者は、tpusradd(1)、tpusrdel(1)、および tpusrmod(1) の各コマンドを使用して tpusr を管理します。AUTHSVR サーバは、tpusr ファイルに格納されているクライアント・ユーザ情報を入力として受け取ります。AUTHSVR サーバは、この情報を使用して、ATMI アプリケーションに参加しようとするクライアントを認証します。

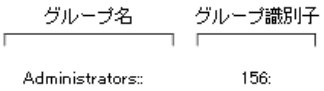
次は、tpusr ファイル内のサンプル・エントリです。



AUTHSVR とアクセス制御チェック機能には、tpgrp というグループ・ファイルも必要です。このファイルには、ATMI アプリケーションへの参加を許可されたクライアント・ユーザに関連付けられたグループのリストが含まれています。アプリケーション管理者は、tpgrpadd(1)、tpgrpdel(1)、および tpgrpmod(1) の各コマンドを使用して tpgrp を管理します。

AUTHSVC は、認証されたクライアント・ユーザにアプリケーション・キーを割り当てます。アプリケーション・キーには、USER_AUTH、ACL、または MANDATORY_ACL のセキュリティ・レベルに対するユーザ識別子および関連するグループ識別子が含まれています。アプリケーション・キーの詳細については、第 1 章の 48 ページ「アプリケーション・キー」を参照してください。

次は、tpgrp ファイル内のサンプル・エントリです。



管理者は、tpusr ファイルおよび tpgrp ファイルで、ユーザとグループのリストを定義しなければなりません。これらのファイルは、ATMI アプリケーションの APPDIR 変数で定義した最初のパス名によって参照されるディレクトリにあります。これらのファイルは、コロンで区切られたフラットなテキスト・ファイルであり、アプリケーション管理者だけが読み書きを行う権限を持ちます。

システムのセキュリティ・データ・ファイルを BEA Tuxedo のユーザ・ファイルとグループ・ファイルに変換する

ホスト・システムには、ユーザとグループのリストを格納したファイルが既に存在する場合があります。これらのファイルを ATMI アプリケーションのユーザ・ファイルおよびグループ・ファイルとして使用するには、BEA Tuxedo システムで受け付けられる形式に変換する必要があります。ファイルを変換するには、次の手順例に示すように、tpaclcvt(1) コマンドを実行します。この手順例は、UNIX ホスト・マシン用に記述されています。

1. ATMI アプリケーションの MASTER マシンで作業しており、ATMI アプリケーションが非アクティブであることを確認します。
2. /etc/password ファイルを BEA Tuxedo システムで受け付けられる形式に変換するには、次のコマンドを入力します。

   ```
   tpaclcvt -u /etc/password
   ```

   このコマンドにより、tpusr ファイルが作成され、変換されたデータが格納されます。tpusr ファイルが既に存在する場合、tpaclcvt により、変換したデータがファイルに追加されます。ただし、重複するユーザ情報が追加されることはありません。

   注記 シャドウ・パスワード・ファイルを使用するシステムでは、ファイル内のユーザごとにパスワードの入力が要求されます。
3. /etc/group ファイルを BEA Tuxedo システムで受け付けられる形式に変換するには、次のコマンドを入力します。

   ```
   tpaclcvt -g /etc/group
   ```

   このコマンドにより、tpgrp ファイルが作成され、変換されたデータが格納されます。tpgrp ファイルが既に存在する場合、tpaclcvt により、変換したデータがファイルに追加されます。ただし、重複するユーザ情報が追加されることはありません。

ユーザおよびグループを追加、変更、または削除する

BEA Tuxedo システムでは、アプリケーション・ユーザのリストを tpusr というファイルで管理し、グループのリストを tpgrp というファイルで管理する必要があります。これらのファイルのエントリを変更するには、コマンドを発行するか、または ACL_MIB 内の適切な属性を変更します。

コマンドを使用してユーザおよびグループのエントリを変更する

以下のいずれかのコマンドを実行すると、tpusr ファイルおよび tpgrp ファイルのエントリをいつでも追加、変更、または削除できます。

コマンド名	目的	エントリが格納されているファイル名
tpusradd(1)	追加	tpusr
tpusrmod(1)	変更	
tpusrdel(1)	削除	
tpgrpadd(1)	追加	tpgrp
tpgrpmod(1)	変更	
tpgrpdel(1)	削除	

これらのコマンドを実行するには、次の手順に従います。

1. 非アクティブな ATMI アプリケーションの場合、アプリケーションの MASTER マシンで作業していることを確認します。アクティブな ATMI アプリケーションの場合は、コンフィギュレーション内のどのマシンからでも作業できます。
2. コマンドの実行方法については、『BEA Tuxedo コマンド・リファレンス』にある各コマンドのエントリを参照してください。

ACL_MIB を使用してユーザおよびグループのエントリを変更する

コマンド行インターフェイス以外の方法として、ACL_MIB(5) の T_ACLPRINCIPAL クラスの該当する属性値を変更して、tpusr のユーザ・エントリを追加、変更、または削除することができます。tpusradd(1) では一度に 1 人のユーザしか追加できないため、同時に複数のユーザ・エントリを追加する場合は、この方法の方がコマンド行インターフェイスより効率的です。

同様に、ACL_MIB(5) の T_ACLGROUP クラスの該当する属性値を変更すると、tpgrp のグループ・エントリを追加、変更、または削除できます。tpgrpadd(1) では一度に 1 つのグループしか追加できないため、同時に複数のグループ・エントリを追加する場合は、この方法の方がコマンド行インターフェイスより効率的です。

BEA Administration Console を使用すると、最も簡単に MIB にアクセスできます。

関連項目

- 第 1 章の 43 ページ「デフォルトの認証と認可」
- 第 2 章の 54 ページ「デフォルトの認証と認可の管理」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」

アクセス制御リストによるセキュリティの有効化

デフォルトの認証には、サービスを実行し、イベントをポストし、アプリケーション・キューのメッセージをキューに登録 (または登録解除) するユーザを決定するアクセス制御リストの機能が備わっています。セキュリティ・レベルには、オプションのアクセス制御リスト (ACL) および必須のアクセス制御リスト (MANDATORY_ACL) があります。アクセス制御リストは、ユーザが ATMI アプリケーションへの参加を認証された場合にのみ有効になります。

アクセス制御リストを使用すると、管理者はユーザを複数のグループにまとめ、それらのグループに対して、メンバ・ユーザがアクセス権を持つオブジェクトを関連付けることができます。アクセス制御は、次の理由により、グループ・レベルで行われます。

- システム管理を簡素化できます。新しいサービスへのアクセス権を付与する場合、1 つのグループに対してアクセス権を付与する方が、個別の複数のユーザに付与するより簡単です。
- パフォーマンスを高めることができます。エンティティを呼び出すたびにアクセス・パーミッションをチェックする必要があるため、パーミッションはすばやく解決できなければなりません。グループの数はユーザの数より少ないため、特権ユーザのリストを検索するよりも、特権グループのリストを検索する方が高速です。

アクセス制御のチェック機能は、アプリケーション管理者によって作成および管理される 3 つのファイルに基づいています。

- `tpusr` にはユーザのリストが格納されています。
- `tpgrp` にはグループのリストが格納されています。
- `tpacl` には、アクセス制御リスト (ACL) と呼ばれる、アプリケーション・エンティティ (サービスなど) に対するグループのマッピングのリストが格納されています。

クライアントのアプリケーション・キー (クライアントを有効なユーザおよび有効なグループ・メンバとして識別する情報を含む) を解析することによって、エンティティ (サービス、イベント、またはアプリケーション・キューなど) は、ユーザが属するグループを識別できます。tpacl ファイルをチェックすることによって、エンティティは、クライアントのグループにアクセス・パーミッションが付与されているかどうかを判定できます。

アプリケーション管理者、アプリケーション・オペレータ、およびアプリケーション管理者またはオペレータの権限で実行中のプロセスやサービス要求は、ACL によるパーミッションのチェックの対象にはなりません。

ユーザ・レベルの ACL エントリが必要な場合は、各ユーザのグループを作成し、そのグループを tpacl ファイルの該当するアプリケーション・エンティティにマッピングします。

オプションの ACL セキュリティを有効にする方法

デフォルトの認証には、「オプションのアクセス制御リスト (ACL)」というセキュリティ・レベルが用意されており、これは、コンフィギュレーション・ファイルの SECURITY ACL で指定すると有効になります。このセキュリティ・レベルでは、各クライアントは、ATMI アプリケーションに参加するため、アプリケーション・パスワード、ユーザ名、およびユーザ固有のデータ (パスワードなど) を提示しなければなりません。ターゲット・アプリケーションのエンティティに関連するエントリが tpacl ファイルになくても、ユーザはそのエンティティにアクセスできます。

管理者は、このセキュリティ・レベルを使用して、セキュリティを強化したいリソースに対してのみアクセス権を設定できます。つまり、すべてのユーザにアクセスを許可するサービス、イベント、およびアプリケーション・キューについて、tpacl ファイルにエントリを追加する必要はありません。ただし、tpacl ファイルにターゲット・アプリケーションのエンティティに関連するエントリがあり、ユーザがこれにアクセスしようとする場合、そのユーザは、そのエンティティへのアクセスを許可されたグループのメンバでなければなりません。そうでない場合、アクセスは拒否されます。

ACL のセキュリティ・レベルを有効にするには、次の手順に従います。

1. UBBCONFIG ファイルを設定します。
2. ACL ファイルを設定します。

これらの手順については、次の 2 つの節で説明します。

UBBCONFIG ファイルを設定する

1. ATMI アプリケーションの MASTER マシンで作業しており、ATMI アプリケーションが非アクティブであることを確認します。
2. テキスト・エディタで UBBCONFIG を開き、RESOURCES セクションと SERVERS セクションに次の行を追加します。

```
*RESOURCES
SECURITY    ACL
AUTHSVC     ..AUTHSVC
      .
      .
      .
*SERVERS
AUTHSVR SRVGRP="group_name" SRVID=1 RESTART=Y GRACE=600 MAXGEN=2
CLOPT="-A"
```

   CLOPT="-A" を指定すると、tmboot(1) は、tmboot によって ATMI アプリケーションが起動するときに、"-A" で呼び出されたデフォルトのコマンド行オプションだけを AUTHSVR に渡します。デフォルトでは、AUTHSVR は、tpusr というファイル内のクライアント・ユーザ情報を使用して、ATMI アプリケーションに参加しようとするクライアントを認証します。tpusr は、ATMI アプリケーションの APPDIR 変数で定義する最初のパス名によって参照されるディレクトリにあります。

3. tmloadcf(1) を実行してコンフィギュレーションをロードします。tmloadcf コマンドを実行すると、UBBCONFIG が解析され、TUXCONFIG 変数が指す場所にバイナリ形式の TUXCONFIG ファイルがロードされます。
4. パスワードの入力を求められます。パスワードは、30 文字以内で構成します。これは ATMI アプリケーションのパスワードとなり、tmadmin の passwd コマンドを使用して変更しない限り有効です。
5. 電話や手紙など、オンライン以外の方法で、ATMI アプリケーションの認可ユーザに対してアプリケーション・パスワードを配布します。

ACL ファイルの設定

アクセス制御のチェック機能では、tpusr というユーザ・ファイル、tpgrp というグループ・ファイル、および tpacl という ACL ファイルが必要です。ACL ファイルには、グループとアプリケーション・エンティティのマッピングが含まれています。エンティティとは、サービス、イベント、またはアプリケーション・キューです。

次は、tpacl ファイル内のサンプル・エントリです。

エンティティ名	エンティティの種類	グループ識別子
TOLOWER:	SERVICE:	156,281,282,305:

管理者は tpacl ファイルでエントリを定義しなければなりません。tpacl ファイルは、ATMI アプリケーションの APPDIR 変数で定義した最初のパス名によって参照されるディレクトリにあります。これらのファイルは、コロンで区切られたフラットなテキスト・ファイルであり、アプリケーション管理者だけが読み書きを行う権限を持ちます。

tpacl ファイルの ACL エントリを変更するには、コマンドを発行するか、または ACL_MIB 内の適切な属性を変更します。

コマンドを使用して ACL エントリを変更する

以下のいずれかのコマンドを実行すると、tpacl ファイルの ACL エントリをいつでも追加、変更、または削除できます。

コマンド名	目的
tpacladd(1)	エントリの追加
tpaclmod(1)	エントリの変更
tpacldel(1)	エントリの削除

これらのコマンドを実行するには、次の手順に従います。

1. 非アクティブな ATMI アプリケーションの場合、アプリケーションの MASTER マシンで作業していることを確認します。アクティブな ATMI アプリケーションの場合は、コンフィギュレーション内のどのマシンからでも作業できます。
2. コマンドの実行方法については、『BEA Tuxedo コマンド・リファレンス』で各コマンドのエントリを参照してください。

ACL_MIB を使用して ACL エントリを変更する

コマンド行インターフェイス以外の方法として、ACL_MIB(5) の T_ACLPERM クラスの該当する属性値を変更して、tpacl の ACL エントリを追加、変更、または削除することができます。tpacladd(1) では一度に 1 つの ACL エントリしか追加できないため、同時に複数の ACL エントリを追加する場合は、この方法の方がコマンド行インターフェイスより効率的です。

BEA Administration Console を使用すると、最も簡単に MIB にアクセスできます。

必須の ACL セキュリティを有効にする方法

デフォルトの認証には、「必須のアクセス制御リスト」というセキュリティ・レベルが用意されており、これは、コンフィギュレーション・ファイルの SECURITY MANDATORY_ACL で指定すると有効になります。このセキュリティ・レベルでは、各クライアントは、ATMI アプリケーションに参加するため、アプリケーション・パスワード、ユーザ名、およびユーザ固有のデータ (パスワードなど) を提示しなければなりません。ターゲット・アプリケーションのエンティティに関連するエントリが tpacl ファイルにない場合、クライアントはそのエンティティにアクセスできません。つまり、アクセスする必要のあるアプリケーション・エンティティのエントリが tpacl ファイルに存在していなければなりません。したがって、このレベルは「必須」と呼ばれます。

ただし、tpacl ファイルにターゲット・アプリケーションのエンティティに関連するエントリがあり、ユーザがこれにアクセスしようとする場合、そのユーザは、そのエンティティへのアクセスを許可されたグループのメンバでなければなりません。そうでない場合、アクセスは拒否されます。

MANDATORY_ACL のセキュリティ・レベルを有効にするには、次の手順に従います。

1. UBBCONFIG ファイルを設定します。
2. ACL ファイルを設定します。

これらの手順については、次の 2 つの節で説明します。

UBBCONFIG ファイルを設定する

1. ATMI アプリケーションの MASTER マシンで作業しており、ATMI アプリケーションが非アクティブであることを確認します。
2. テキスト・エディタで UBBCONFIG を開き、RESOURCES セクションと SERVERS セクションに次の行を追加します。

```
*RESOURCES
SECURITY   MANDATORY_ACL
AUTHSVC    ..AUTHSVC
      .
      .
      .
*SERVERS
AUTHSVR SRVGRP="group_name" SRVID=1 RESTART=Y GRACE=600 MAXGEN=2
CLOPT="-A"
```

CLOPT="-A" を指定すると、tmboot(1) は、tmboot によって ATMI アプリケーションが起動するときに、"-A" で呼び出されたデフォルトのコマンド行オプションだけを AUTHSVR に渡します。デフォルトでは、AUTHSVR は、tpusr というファイル内のクライアント・ユーザ情報を使用して、ATMI アプリケーションに参加しようとするクライアントを認証します。tpusr は、ATMI アプリケーションの APPDIR 変数で定義する最初のパス名によって参照されるディレクトリにあります。

3. tmloadcf(1) を実行してコンフィギュレーションをロードします。tmloadcf コマンドを実行すると、UBBCONFIG が解析され、TUXCONFIG 変数が指す場所にバイナリ形式の TUXCONFIG ファイルがロードされます。
4. パスワードの入力を求められます。パスワードは、30 文字以内で構成します。これは ATMI アプリケーションのパスワードとなり、tmadmin の passwd コマンドを使用して変更しない限り有効です。
5. 電話や手紙など、オンライン以外の方法で、ATMI アプリケーションの認可ユーザに対してアプリケーション・パスワードを配布します。

ACL ファイルの設定

第 2 章の 65 ページ「ACL ファイルの設定」を参照してください。

関連項目

- 第 1 章の 43 ページ「デフォルトの認証と認可」
- 第 2 章の 54 ページ「デフォルトの認証と認可の管理」
- 第 2 章の 3 ページ「セキュリティ管理のタスク」

3　セキュリティのプログラミング

ここでは、次の内容について説明します。

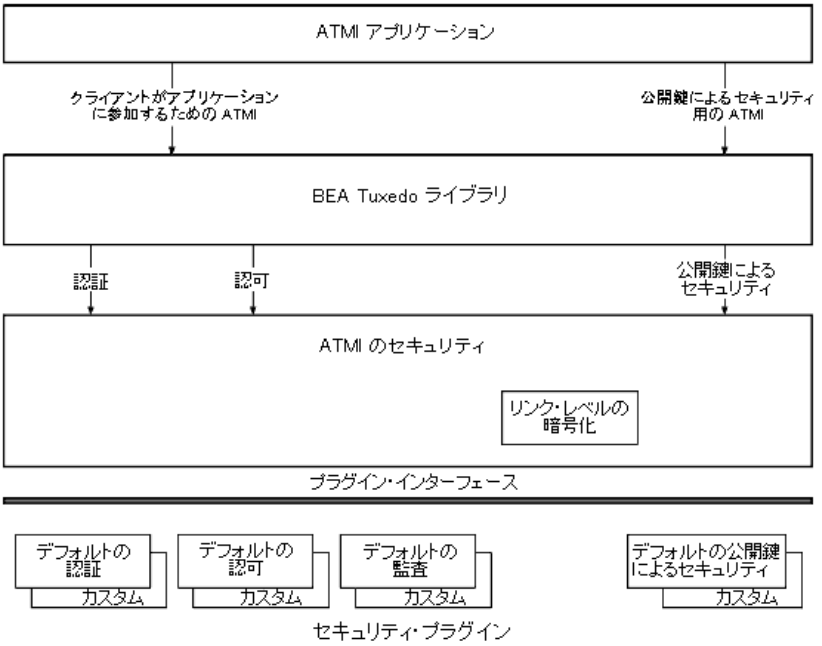
- セキュリティのプログラミングとは
- セキュリティを備えた ATMI アプリケーションのプログラミング
- ATMI アプリケーションにクライアント・プログラムを参加させるためのセキュリティ・コードの記述方法
- データの完全性と機密性を保護するためのセキュリティ・コードの記述方法

セキュリティのプログラミングとは

セキュリティのプログラミングとは、ATMI (Application-to-Transaction Monitor Interface) アプリケーション用のセキュリティ・コードを記述する作業です。アプリケーション・プログラマは、プログラムのロジックを表すコードを記述するほか、ATMI を使用して、アプリケーション・コードを BEA Tuxedo トランザクション・モニタにリンクします。ATMI プログラミング・インターフェイスを使用すると、BEA Tuxedo トランザクション・モニタで制御される、アプリケーション・クライアントとアプリケーション・サーバ間の通信を実現できます。ATMI には、C および COBOL 用のインプリメンテーションがあります。

次の図に示すように、アプリケーション・プログラマは、ATMI 関数を使用して、ユーザを認証したり、ユーザによるアクセスを制御したり、公開鍵による暗号化の技術をアプリケーションに組み込むことができます。ただし、図が示すとおり、監査およびリンク・レベルの暗号化を行うための ATMI 関数は、アプリケーション・レベルでは用意されていません。監査は、BEA Tuxedo のシステム・レベルで実行されます。また、リンク・レベルの暗号化は、アプリケーション管理者側で設定します。

図 3-1 BEA Tuxedo のセキュリティのプログラミング



関連項目

- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」
- 第 1 章の 1 ページ「セキュリティとは」
- 第 2 章の 1 ページ「セキュリティの管理とは」

セキュリティを備えた ATMI アプリケーションのプログラミング

BEA Tuxedo システムには、セキュリティのニーズに応じた、さまざまな ATMI 関数が用意されています。

記述するセキュリティ・コードの種類	使用する ATMI 関数
クライアントが ATMI アプリケーションに参加し、アプリケーション・サービスにアクセスするためのクライアント・プログラム	ATMI アプリケーションに参加するクライアント用の ATMI 関数。認証および認可のプラグインに対するシステム・レベルの呼び出しを行います。
クライアント / サーバ間で送受信されるデータの完全性と機密性を保つためのクライアント・プログラムおよびサーバ・プログラム	公開鍵によるセキュリティ用の ATMI 関数。エンド・ツー・エンドのデジタル署名およびデータの暗号化をサポートします。

関連項目

- 第 3 章の 3 ページ「プログラミング環境の設定」

プログラミング環境の設定

セキュリティ・コードを記述するため、アプリケーション・プログラマには次の権限が必要です。

- BEA Tuxedo のライブラリおよびコマンドに対するアクセス権
- BEA Tuxedo システムのディレクトリ構造内にあるディレクトリとファイルに対する読み取り権および実行権

必要なライブラリおよびコマンドにアクセスするには、環境内で、`TUXCONFIG`、`TUXDIR`、`APPDIR`、およびその他の環境変数を設定する必要があります。詳細については、『BEA Tuxedo アプリケーション実行時の管理』の 第 1 章の 2 ページ「環境変数の設定」を参照してください。

アプリケーション管理者は、ディレクトリおよびファイルのパーミッションを設定します。必要なパーミッションを取得するには、担当の管理者に問い合わせてください。

関連項目

- 第 3 章の 4 ページ「ATMI アプリケーションにクライアント・プログラムを参加させるためのセキュリティ・コードの記述方法」
- 第 3 章の 17 ページ「データの完全性と機密性を保護するためのセキュリティ・コードの記述方法」

ATMI アプリケーションにクライアント・プログラムを参加させるためのセキュリティ・コードの記述方法

クライアント・プログラムには、アプリケーションまたはコンピュータ以外からのデータを収集したり、データをメッセージに組み込んだり、処理対象のメッセージをサーバに転送する、という役割があります。ユーザは、現金自動預入支払機 (ATM)、データ入力端末、およびグラフィックス用デバイスなどを使用することにより、クライアント・プログラムを利用します。

デフォルトの認証および認可では、5 つのセキュリティ・レベルのうち、いずれかをアプリケーションに設定できます。最も低いセキュリティ・レベルでは、認証は行われません。最も高いセキュリティ・レベルでは、アクセス制御リストの機能により、サービスを実行し、イベントをポストし、アプリケーション・キューのメッセージをキューに登録 (または登録解除) するユーザが決定されます。ATMI アプリケーションに対するセキュリティ・レベルの設定は、アプリケーション管理者の役割です。

クライアント・プログラムを ATMI アプリケーションに参加させるため、アプリケーション・プログラマは、次の 2 つのタスクを実行する必要があります。

- 特定のクライアント・プロセスのセキュリティ・データを取得します。
- 取得したデータを BEA Tuxedo システムに渡します。

次の擬似コードは、基本的なクライアント・プログラムの動作を示しています。セキュリティ関連の文は太字で示します。

リスト 3-1 クライアントの擬似コード

```
main()
 {
      tpchkauth() を呼び出してアプリケーションのセキュリティ・レベルをチェック
      usrname および cltname を取得
      アプリケーション・パスワードの入力を要求
      ユーザ固有のパスワードの入力を要求
      TPINIT バッファを割り当て
      最初のクライアント ID を TPINIT バッファに格納
      tpinit() を呼び出してアプリケーションのクライアントとして登録
      バッファを割り当て
      do while true {
            ユーザ入力データをバッファに格納
            サービス要求を送信
            応答を受信
            応答をユーザに渡す }
      アプリケーションを終了
 }
```

上のリスト内の大部分の文は、C または COBOL の ATMI 関数によってインプリメントされます。ただし、ここでは C 言語のインプリメンテーションだけを示しています。

C 言語で記述されたクライアント・プログラムは、tpinit(3c) を使用して、ATMI アプリケーションに設定されたセキュリティ・レベルに準拠し、アプリケーションに参加します。tpinit() の引数は、TPINIT バッファに対するポインタです。COBOL アプリケーションの場合、クライアント・プログラムは TPINITIALIZE(3cbl) を呼び出し、引数として TPINFDEF-REC レコードに対するポインタを取ります。

関連項目

- 第 3 章の 6 ページ「セキュリティ・データの取得」
- 第 3 章の 8 ページ「ATMI アプリケーションへの参加」
- 『C 言語を使用した BEA Tuxedo アプリケーションのプログラミング』および『COBOL を使用した BEA Tuxedo アプリケーションのプログラミング』の 第 4 章の 1 ページ「クライアントのコーディング」
- 『BEA Tuxedo C 言語リファレンス』の `tpinit(3c)`
- 『BEA Tuxedo COBOL リファレンス』の `TPINITIALIZE(3cbl)`
- 第 2 章の 40 ページ「公開鍵セキュリティの管理」
- 第 2 章の 33 ページ「認可の管理」
- 第 1 章の 43 ページ「デフォルトの認証と認可」
- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」

セキュリティ・データの取得

BEA Tuxedo システムには、さまざまなアプリケーションに対応できるように記述された一般的なクライアント・プログラム用の ATMI 関数が用意されています。クライアントは、この ATMI 関数を使用して、参加先の ATMI アプリケーションに必要なセキュリティ・レベルを決定できます。この ATMI 関数は、`tpchkauth(3c)` (C) または `TPCHKAUTH(3cbl)` (COBOL) であり、デフォルトの認証および認可を使用する ATMI アプリケーションで実行されます。`tpchkauth()` 関数および `TPCHKAUTH()` 関数は、カスタマイズした認証または認可を使用する ATMI アプリケーションでも使用できますが、使用方法は、カスタマイズしたセキュリティ機能がどのようにインプリメントされているかに応じて異なります。ここでは、主にデフォルトの認証および認可の場合について説明します。

C を使用するアプリケーション・プログラマは、`tpchkauth()` を使用して ATMI アプリケーションのセキュリティ・レベルをチェックしてから、`tpinit(3c)` を呼び出します。これで、クライアント・プログラムは、`tpinit()` 呼び出しに必要なアプリケーション・パスワードとユーザ認証データの入力を要求できます。`tpchkauth()` は引数なしで呼び出されます。

COBOL を使用するアプリケーション・プログラマは、同じ目的で TPCHKAUTH( ) を使用してから、TPINITIALIZE(3cbl) を呼び出します。TPCHKAUTH(3cbl) と TPINITIALIZE(3cbl) の構文および機能は、tpchkauth(3c) と tpinit(3c) の場合と同じです。

tpchkauth( ) 関数 (TPCHKAUTH( ) ルーチン) を呼び出すと、次のいずれかの値が返されます。

TPNOAUTH
: 通常のオペレーティング・システムへのログイン、およびファイルに対するパーミッションの指定以外は必要ありません。TPNOAUTH は、セキュリティ・レベルが NONE のときに返されます。

TPSYSAUTH
: アプリケーション・パスワードが必要です。クライアント・プログラムは、パスワードの入力をユーザに要求し、入力されたパスワードを TPINIT バッファ (C の場合) または TPINFDEF-REC レコード (COBOL の場合) のパスワード・フィールドに格納します。TPSYSAUTH は、セキュリティ・レベルが APP_PW のときに返されます。

  アプリケーション管理者は、アプリケーション・パスワードをユーザに通知します。アプリケーション・プログラマは、ユーザに対してアプリケーション・パスワードの入力を要求し、入力されたパスワードがテキスト形式で TPINIT バッファまたは TPINFDEF-REC レコードに格納されるようにクライアント・プログラムを記述します。パスワードは、画面に表示されないようにします。

  BEA Tuxedo システムに組み込まれている ud(1)、wud(1) などのクライアント・プログラムは、アプリケーション・パスワードの入力を要求します。ud( ) を使用すると、フィールド化バッファが標準入力から読み取られ、サービスに送信されます。

TPAPPAUTH
: アプリケーション・パスワードが必要です。クライアントは、TPINIT バッファ (C の場合) または TPINFDEF-REC レコード (COBOL の場合) のデータ・フィールド内の認証サービスに渡す値を指定するよう要求されます。TPAPPAUTH は、セキュリティ・レベルが USER_AUTH、ACL、または MANDATORY_ACL のときに返されます。

  アプリケーション・プログラマは、デフォルトの認証および認可で使用される AUTHSVR サーバが提供する、アプリケーションの認証サービスに関するその他の情報を記述して、クライアント・プログラムのコードを作成します。AUTHSVR は、クライアント名やユーザ名などのユーザ固有の認証情報を検証して、クライアント・プログラムが ATMI アプリケーションに参加できるかどうかを判別するサーバであり、管理者が設定します。

関連項目

- 第 3 章の 8 ページ「ATMI アプリケーションへの参加」
- 『C 言語を使用した BEA Tuxedo アプリケーションのプログラミング』と『COBOL を使用した BEA Tuxedo アプリケーションのプログラミング』の 第 4 章の 1 ページ「クライアントのコーディング」
- 『BEA Tuxedo C 言語リファレンス』の `tpinit(3c)` と `tpchkauth(3c)`
- 『BEA Tuxedo COBOL リファレンス』の `TPINITIALIZE(3cbl)` と `TPCHKAUTH(3cbl)`
- 第 1 章の 43 ページ「デフォルトの認証と認可」
- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」

ATMI アプリケーションへの参加

セキュリティが設定された ATMI アプリケーションでは、`TPINIT` バッファ (C の場合) または `TPINFDEF-REC` レコード (COBOL の場合) を使用して、BEA Tuxedo システムにセキュリティ情報を渡す必要があります。`TPINIT` バッファは、特殊な型付きバッファであり、クライアントが ATMI アプリケーションに参加しようとするときに、クライアントの ID および認証情報をシステムに渡すためにクライアント・プログラムで使用されます。COBOL の場合は、`TPINFDEF-REC` レコードが使用されます。

`TPINIT` は `atmi.h` ヘッダ・ファイルで定義され、`TPINFDEF-REC` は COBOL の `COPY` ファイルで定義されます。次の表は、それぞれの構造体を示しています。

TPINIT 構造体	TPINFDEF-REC 構造体	
char usrname[MAXTIDENT+2];	05 USRNAME	PIC X(30).
char cltname[MAXTIDENT+2];	05 CLTNAME	PIC X(30).
char passwd[MAXTIDENT+2];	05 PASSWD	PIC X(30).
char grpname[MAXTIDENT+2];	05 GRPNAME	PIC X(30).
long flags;	05 NOTIFICATION-FLAG	PIC S9(9) COMP-5.
long datalen;	88 TPU-SIG	VALUE 1.
long data;	88 TPU-DIP	VALUE 2.
	88 TPU-IGN	VALUE 3.
	05 ACCESS-FLAG	PIC S9(9) COMP-5.
注記 MAXTIDENT には最大30文字まで指定できます。	88 TPSA-FASTPATH	VALUE 1.
	88 TPSA-PROTECTED	VALUE 2.
	05 DATLEN	PIC S9(9) COMP-5.

次の表では、TPINIT バッファおよび TPINFDEF-REC レコードのフィールドを示します。

表 3-1 TPINIT バッファおよび TPINFDEF-REC レコードのフィールド

TPINIT のフィールド	TPINFDEF-REC のフィールド	説明
usrname	USRNAME	ユーザ名。* ヌルで終了する 30 文字までの文字列。 ユーザ名は呼び出し側を表します。クライアント・プログラムの作成者は、ホスト・オペレーティング・システムへのログイン時に使用したログイン名を使用できます。
cltname	CLTNAME	クライアント名。* ヌルで終了する 30 文字までの文字列。 クライアント名はクライアント・プログラムを表します。クライアント・プログラムの作成者は、このフィールドを使用して、クライアント・プログラム実行時のユーザの役割やジョブを指定することができます。

* このフィールドは、デフォルトの認証および認可で USER_AUTH、ACL または MANDATORY_ACL のセキュリティ・レベルが指定された場合に必要です。

** バイナリ形式の UBBCONFIG ファイル。tmloadcf(1) を使用して作成します。

*** 通常はユーザ・パスワードです。

表 3-1 TPINIT バッファおよび TPINFDEF-REC レコードのフィールド (続き)

TPINIT のフィールド	TPINFDEF-REC のフィールド	説明
passwd	PASSWD	アプリケーション・パスワード。* ヌルで終了する 8 文字までの文字列。 tpinit() または TPINITIALIZE() は、このパスワードを TUXCONFIG ファイル ** に格納されている設定済みのアプリケーション・パスワードと比較して、パスワードの検証を行います。
grpname	GRPNAME	グループ名。ヌルで終了する 30 文字までの文字列。このフィールドはセキュリティとは無関係です。 グループ名を使用すると、UBBCONFIG ファイルで定義されているリソース・マネージャのグループにクライアントを関連付けることができます。
flags	NOTIFICATION-FLAG TPU-SIG TPU-DIP TPU-IGN ACCESS-FLAG TPSA-FASTPATH TPSA-PROTECTED	通知フラグおよびアクセス・フラグ。このフィールドはセキュリティとは無関係です。 フラグを設定すると、クライアントに対して使用する通知メカニズムおよびシステムのアクセス・モードを指定できます。ここで指定した内容は、UBBCONFIG ファイルの RESOURCES セクションの値を上書きします (ただし、一部の例外を除く)。

* このフィールドは、デフォルトの認証および認可で USER_AUTH、ACL または MANDATORY_ACL のセキュリティ・レベルが指定された場合に必要です。

** バイナリ形式の UBBCONFIG ファイル。tmloadcf(1) を使用して作成します。

*** 通常はユーザ・パスワードです。

表 3-1 TPINIT バッファおよび TPINFDEF-REC レコードのフィールド (続き)

TPINIT のフィールド	TPINFDEF-REC のフィールド	説明
datalen	DATALEN	後に続くユーザ固有のデータ *** の長さ。 * C で記述したクライアント・プログラムの作成者は、送信予定のユーザ固有データのバイト数を指定して TPINITNEED を呼び出すと、このフィールドに指定するサイズを算出することができます。TPINITNEED は、atmi.h ヘッダ・ファイルに組み込まれているマクロです。
data	該当なし	ユーザ固有のデータ。*** 固定長を持ちません。 * tpinit() または TPINITIALIZE() は、ユーザ固有のデータを認証サーバに転送し、検証します。デフォルトの認証の場合、認証サーバは AUTHSVR です。

* このフィールドは、デフォルトの認証および認可で USER_AUTH、ACL または MANDATORY_ACL のセキュリティ・レベルが指定された場合に必要です。

** バイナリ形式の UBBCONFIG ファイル。tmloadcf(1) を使用して作成します。

*** 通常はユーザ・パスワードです。

クライアント・プログラムは、tpalloc(3c) を呼び出して TPINIT バッファを割り当てます。次は、8 バイトのアプリケーション固有のデータを tpinit() に渡す準備を行い、クライアントが ATMI アプリケーションに参加できるようにするサンプル・コードです。

リスト 3-2 TPINIT バッファを割り当てて ATMI アプリケーションに参加する

```
  .
  .
  .
TPINIT *tpinfo;
  .
  .
  .
if ((tpinfo = (TPINIT *)tpalloc("TPINIT",(char *)NULL,
     TPINITNEED(8))) == (TPINIT *)NULL){
     Error Routine
}
  .
  .
  .
tpinit(tpinfo)  /* ATMI アプリケーションへの参加 */
  .
  .
  .
```

ワークステーション・クライアントが tpinit() 関数または TPINITIALIZE() ルーチンを呼び出して ATMI アプリケーションに参加すると、次の主なイベントが発生します。

1. 「イニシエータ」側のワークステーション・クライアントおよび「ターゲット」側のワークステーション・リスナ (WSL) が、リンク・レベルの暗号化 (LLE: Link-Level Encryption) の *min-max* 値を交換します。これらの値は、イニシエータのワークステーション・クライアントとターゲットの WSH との間でリンクを確立するために使用されます。LLE については、第 1 章の 23 ページ「リンク・レベルの暗号化」を参照してください。
2. イニシエータのワークステーション・クライアントとターゲットの WSH は、セキュリティ・トークンの交換によってお互いを認証します。デフォルトの認証の場合、クライアントのセキュリティ・データが TPINIT バッファまたは TPINFDEF-REC レコードからターゲットの WSH に転送されると、認証は成功します。

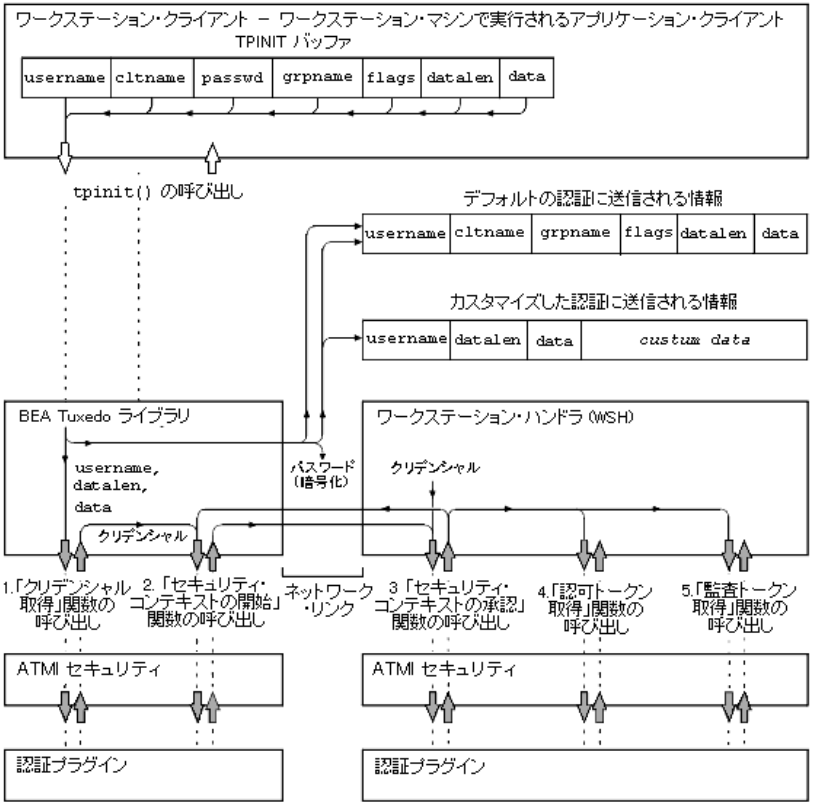
3. 認証に成功すると、イニシエータのワークステーション・クライアントは、usrname、cltname、および flags の 3 つのフィールド値を含む別のバッファをターゲットの WSH に送信し、認証済みのワークステーション・クライアントに関するこれらの情報が確実に伝わるようにします。

ネイティブ・クライアントが tpinit() 関数または TPINITIALIZE() ルーチンを呼び出して ATMI アプリケーションに参加すると、認証だけが行われます。基本的に、ネイティブ・クライアントは、自分自身を認証します。

クライアントのセキュリティ・データの転送

次の図は、ワークステーション・クライアントの TPINIT バッファからデータを転送する様子を示します。TPINFDEF-REC レコードからデータを転送する場合にもこの図が当てはまります。

図 3-2 ワークステーション・クライアントの TPINIT バッファからデータを転送する



注記 上の図に示す認可手順は、ネイティブ・クライアントが ATMI アプリケーションに参加しようとする場合も基本的に同じです。ただし、ネットワーク・リンクや WSH は無関係です。ネイティブ・クライアントは、自分自身を認証するためです。

上の図では、デフォルトの認証を使用するか、またはカスタマイズした認証を使用するかにより、BEA Tuxedo システムに送信される情報が異なる点に注意してください。デフォルトの認証の場合、cltname、grpname、および flags の各フィールドの値が、プラグイン・インターフェイス以外の方法で、ワークステーション・クライアントのデフォルトの認証プラグインに送信されます。一方、カスタマイズした認証の場合、クライアント・プログラムの作成者は、これらの値のほか、data フィールドで選択した可変長の別の値を組み込むこともできます。

デフォルトの認証を使用する場合、ワークステーション・クライアントの認証プラグインでは、passwd/PASSWD フィールドが使用され、ネットワーク経由で転送される情報が暗号化されます。このとき、暗号化アルゴリズムとして、56 ビットの DES (Data Encryption Standard) が使用されます。ターゲットの WSH にある認証プラグインでは、TUXCONFIG ファイルに格納されているアプリケーション・パスワードを使用して、この情報を復号化します。ネイティブ・クライアントの場合は、単に、passwd/ PASSWD フィールドの値と、TUXCONFIG ファイルに格納されているアプリケーション・パスワードが比較されます。

注記 ワークステーション・クライアントでは、passwd/PASSWD フィールドの値が、認証プラグイン・インターフェイス以外の方法で、認証プラグインに送信されます。WSH では、TUXCONFIG ファイルに格納されたアプリケーション・パスワードが、アプリケーションの起動時に、認証プラグイン・インターフェイスによって認証プラグインに送信されます。

ワークステーション・クライアントの認証に成功すると、tpinit() 関数は、最後の処理として、usrname、cltname、および flags の 3 つのフィールド値を含む別のバッファを WSH に送信し、認証済みのワークステーション・クライアントに関するこれらの情報が確実に伝わるようにします。TPINITIALIZE() ルーチンの場合も、最後の処理として、同様の情報を含む別のバッファを送信します。カスタマイズした認証プラグインでは、認証手順でこれらの情報が WSH に送信されない場合がありますが、WSH 側ではレポートを作成するとき、つまり tmadmin(1) printclient (pclt) コマンドを呼び出すときにこれらの情報を必要とします。

ワークステーションまたはネイティブ・クライアントは、セキュリティ・チェックにパスすると、サービス要求を発行したり、応答を受信することができます。

ATMI アプリケーションに参加する前のサービス要求の呼び出し

ターゲットの ATMI アプリケーションの SECURITY が NONE に設定されているか、または何も設定されていない場合に、クライアントが、tpinit() または TPINITIALIZE() を呼び出す前にサービス要求 (または ATMI 関数) を呼び出すと、BEA Tuxedo システムは、NULL パラメータで tpinit() または TPINITIALIZE() を自動的に呼び出します。この操作により、次のような結果になります。

- TPINIT または TPINFDEF-REC の機能は使用できません。
- クライアントの命名、任意通知型の通知、およびシステムのアクセス・モードには、デフォルト値が使用されます。
- クライアントをリソース・マネージャのグループに関連付けることはできません。
- アプリケーション・パスワードは指定できません。

ターゲットの ATMI アプリケーションの SECURITY が APP_PW、USER_AUTH、ACL、または MANDATORY_ACL に設定されている場合に、クライアントが、tpinit() または TPINITIALIZE() を呼び出す前にサービス要求 (または ATMI 関数) を呼び出すと、そのサービス要求は拒否されます。

関連項目

- 『C 言語を使用した BEA Tuxedo アプリケーションのプログラミング』および『COBOL を使用した BEA Tuxedo アプリケーションのプログラミング』の 第 4 章の 1 ページ「クライアントのコーディング」
- 『BEA Tuxedo C 言語リファレンス』の tpinit(3c) および tpalloc(3c)
- 『BEA Tuxedo COBOL リファレンス』の TPINITIALIZE(3cbl)
- 第 1 章の 43 ページ「デフォルトの認証と認可」
- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」

データの完全性と機密性を保護するためのセキュリティ・コードの記述方法

公開鍵によるセキュリティは、エンド・ツー・エンドのデジタル署名およびデータの暗号化で構成されています。これらの 2 つの機能は、BEA Tuxedo の ATMI 関数で実現できます。インターネット上でアプリケーションを使用する場合は、公開鍵でセキュリティ保護された ATMI アプリケーションの方が、保護されていないアプリケーションより安全です。

エンド・ツー・エンドのデジタル署名とデータの暗号化は、メッセージ・ベースのデジタル署名およびメッセージ・ベースの暗号化の機能によって実現できます。これらの機能は、*PKCS-7* 標準に基づいています。PKCS-7 は、RSA Laboratories が主要な通信会社の協力のもとに開発した、PKCS (Public-Key Cryptography Standards) という規格の 1 つです。

メッセージ・ベースのデジタル署名では、送信者の ID を特定のメッセージ・バッファと結び付けることにより、データの完全性を保ち、送信者がメッセージを送信した事実を否認できないようにします。メッセージ・ベースの暗号化では、指定した受信者だけがメッセージを復号化できるため、メッセージの機密性が保たれます。

デジタル署名および暗号化は、ATMI のメッセージ・バッファ単位で行われます。したがって、これらの機能は、既存の ATMI のプログラミング・インターフェイスおよび通信パラダイムと互換性があります。メッセージ・バッファは、署名することも暗号化することもできます。メッセージ・バッファに関連するデジタル署名の数と暗号化エンベロープの数の間に、関係を成立させる必要はありません。

注記 各暗号化エンベロープには、メッセージの受信者を識別し、受信者がメッセージを復号化するために必要な情報が含まれています。

公開鍵によるセキュリティの ATMI インターフェイス

公開鍵によるセキュリティの ATMI インターフェイスは、次の処理を行う関数群です。

- 主要なリソースをオープンおよびクローズします。
- 主要なオプション・パラメータを表示および変更します。
- メッセージ・バッファを署名および封印 (暗号化) します。
- メッセージ・バッファに関連付けられたデジタル署名および暗号化情報にアクセスします。
- 型付きメッセージ・バッファを、マシンに依存しないエクスポート可能な文字列表現に変換します (バッファに関連する暗号化エンベロープやデジタル署名の生成を含む)。

公開鍵によるセキュリティの ATMI インターフェイスには、C および COBOL の 2 つのインプリメンテーションがあります。ただし、ATMI の COBOL 言語バインディングでは、メッセージ・バッファがサポートされません。したがって、個別のバッファに対する明示的な署名、暗号化、およびクエリ操作は、COBOL のアプリケーションでは使用できません。一方、鍵管理のインターフェイスでは、COBOL 言語バインディングがサポートされているため、`AUTOSIGN` モードで署名を生成したり、`AUTOENCRYPT` モードで暗号化エンベロープを生成できます。署名の自動検証機能または自動暗号化機能に関するすべての操作は、COBOL のクライアント・プロセスおよびサーバ・プロセスに適用できます。

注記 COBOL の `TPKEYDEF` レコードは、メッセージ・ベースのデジタル署名と暗号化操作を実行するための公開鍵と秘密鍵の管理に使用されます。`TPKEYDEF` レコードの詳細については、『BEA Tuxedo COBOL リファレンス』の紹介部分にある「COBOL 言語 ATMI の戻り値とその他の定義」を参照してください。

次の表は、公開鍵によるセキュリティの ATMI インターフェイスをまとめたものです。各関数については、『BEA Tuxedo C 言語リファレンス』および『BEA Tuxedo COBOL リファレンス』も参照してください。

表 3-2 公開鍵によるセキュリティの ATMI インターフェイスの C 関数

使用する関数	目的
tpkey_open(3c)	デジタル署名の生成、メッセージの暗号化、またはメッセージの復号化のためのキー・ハンドルをオープンします。キーは、ハンドルを使用して表示および操作します。ハンドルには関連するデータがあり、ATMI アプリケーションはこのデータを使用して、ハンドルが指定する項目を検索したり、項目にアクセスします。 キーは、以下のいずれか、または複数の役割を果たします。 ■ 署名の生成 キーは、プリンシパル (ユーザまたはプロセス) の ID でデジタル署名を生成することを認可された状態で、呼び出しプロセスを識別します。プリンシパル名と、TPKEY_SIGNATURE フラグまたは TPKEY_AUTOSIGN フラグを指定して tpkey_open() を呼び出すと、プリンシパルの秘密鍵およびデジタル署名に対するハンドルが返されます。 ■ 署名の検証 キーは、デジタル署名に関連付けられたプリンシパルを表します。署名の検証では、tpkey_open() を呼び出す必要はありません。検証プロセスでは、デジタル署名付きメッセージに付属するデジタル証明書で指定された公開鍵を使用して、署名を検証します。 ■ 暗号化 キーは、暗号化されたメッセージの送信先のプリンシパルを表します。プリンシパル名と、TPKEY_ENCRYPT フラグまたは TPKEY_AUTOENCRYPT フラグを指定して tpkey_open() を呼び出すと、プリンシパルのデジタル証明書を使用して、プリンシパルの公開鍵に対するハンドルが返されます。
tpkey_open(3c)	■ 復号化 キーは、指定したプリンシパルに対して送信された秘密のメッセージの復号化を認可された状態で、呼び出しプロセスを識別します。プリンシパル名と、TPKEY_DECRYPT フラグを指定して tpkey_open() を呼び出すと、プリンシパルの秘密鍵およびデジタル証明書に対するハンドルが返されます。

表 3-2 公開鍵によるセキュリティの ATMI インターフェイスの C 関数 (続き)

使用する関数	目的
tpkey_getinfo(3c)	キー・ハンドルに関連付けられた情報を取得します。暗号サービス・プロバイダに固有の情報も含まれていますが、以下の属性に関しては、すべてのサービス・プロバイダでサポートされます。 ■ PRINCIPAL 指定されたキー (キー・ハンドル) に関連付けられたプリンシパルの名前。プリンシパルとは、ユーザまたはプロセスのことです。どちらになるかは、アプリケーション開発者が公開鍵によるセキュリティを設定する方法に応じて決まります。ATMI アプリケーションの UBBCONFIG ファイルの SEC_PRINCIPAL_NAME パラメータを使用して指定したプリンシパルは、1 つまたは複数のシステム・プロセスの ID になります。詳細については、第 2 章の 11 ページ「プリンシパル名の指定」および 第 2 章の 49 ページ「プラグインによる復号化キーの初期化」を参照してください。 ■ PKENCRYPT_ALG 公開鍵による暗号化のキーで使用される、公開鍵アルゴリズムの ASN.1 DER (Distinguished Encoding Rules) 形式のオブジェクト識別子。詳細については、tpkey_getinfo(3c) のリファレンス・ページを参照してください。 ■ PKENCRYPT_BITS 公開鍵アルゴリズムのキー長 (RSA の絶対値)。これは、512 ～ 2048 ビットの範囲内の値でなければなりません。

表 3-2 公開鍵によるセキュリティの ATMI インターフェイスの C 関数 (続き)

使用する関数	目的
tpkey_getinfo(3c)	■ SIGNATURE_ALG デジタル署名のキーで使用される、デジタル署名アルゴリズムの ASN.1 DER 形式のオブジェクト識別子。詳細については、tpkey_getinfo(3c) のリファレンス・ページを参照してください。 ■ SIGNATURE_BITS デジタル署名アルゴリズムのキー長 (RSA の絶対値)。これは、512 ～ 2048 ビットの範囲内の値でなければなりません。 ■ ENCRYPT_ALG バルク・データの暗号化のキーで使用される、対称鍵アルゴリズムの ASN.1 DER 形式のオブジェクト識別子。詳細については、tpkey_getinfo(3c) のリファレンス・ページを参照してください。 ■ ENCRYPT_BITS 対称鍵アルゴリズムのキー長。これは、40 ～ 128 ビットの範囲内の値でなければなりません。 ■ DIGEST_ALG デジタル署名のキーで使用される、メッセージ・ダイジェスト・アルゴリズムの ASN.1 DER 形式のオブジェクト識別子。詳細については、tpkey_getinfo(3c) のリファレンス・ページを参照してください。 ■ PROVIDER 暗号サービス・プロバイダの名前。 ■ VERSION 暗号サービス・プロバイダのソフトウェアのバージョン。
tpkey_setinfo(3c)	キー・ハンドルに関連付けられたオプションの属性パラメータを設定します。キー・ハンドルの属性のうち、主要な属性は、tpkey_getinfo() で既に説明しています。このほか、特定の暗号サービス・プロバイダに固有な属性を使用することもできます。
tpkey_close(3c)	既にオープンしたキー・ハンドルをクローズします。キー・ハンドルは、tpkey_open() を使用して明示的にオープンしたり、tpenvelope() を使用して暗黙的に (自動的に) オープンできます。

表 3-2 公開鍵によるセキュリティの ATMI インターフェイスの C 関数 (続き)

使用する関数	目的
tpsign(3c)	デジタル署名の型付きメッセージ・バッファをマークします。公開鍵ソフトウェアは、メッセージが送信される直前にデジタル署名を生成します。
tpseal(3c)	暗号化の型付きメッセージ・バッファをマークします。公開鍵ソフトウェアは、メッセージが送信される直前にメッセージを暗号化します。
tpenvelope(3c)	型付きメッセージ・バッファに関連付けられたデジタル署名および暗号化情報にアクセスします。tpenvelope() は、特定のメッセージ・バッファに添付されたデジタル署名および暗号化エンベロープに関するステータス情報を返します。また、各デジタル署名や暗号化エンベロープに関連付けられたキー・ハンドルも返します。デジタル署名のキー・ハンドルは署名者を識別し、暗号化エンベロープのキー・ハンドルはメッセージの受信者を識別します。
tpexport(3c)	型付きメッセージ・バッファを、マシンに依存しない (外部化された) エクスポート可能な文字列表現に変換します。 tpexport() は、このバッファを外部化された文字列表現に変換する直前に、型付きメッセージ・バッファに関連付けられたデジタル署名や暗号化エンベロープを生成します。 外部化された文字列表現は、任意の通信メカニズムを使用して、プロセス間、マシン間、またはドメイン間で送信できます。この文字列表現は恒久的な記憶域にアーカイブできます。
tpimport(3c)	外部化された文字列表現を型付きメッセージ・バッファに戻します。変換中、tpimport() は、必要に応じてメッセージを復号化し、関連するデジタル署名を検証します。

表 3-3 公開鍵によるセキュリティの ATMI の COBOL ルーチン

使用するルーチン	目的
TPKEYOPEN(3cbl)	デジタル署名の生成、メッセージの暗号化、またはメッセージの復号化のためのキー・ハンドルをオープンします。キーは、ハンドルを使用して表示および操作します。ハンドルには関連するデータがあり、ATMI アプリケーションはこのデータを使用して、ハンドルが指定する項目を検索したり、項目にアクセスします。 キーは、以下のいずれか、または複数の役割を果たします。 ■ 署名の生成 キーは、プリンシパル (ユーザまたはプロセス) の ID でデジタル署名を生成することを認可された状態で、呼び出しプロセスを識別します。プリンシパル名と、TPKEY-SIGNATURE および TPKEY-AUTOSIGN を設定して TPKEYOPEN( ) を呼び出すと、プリンシパルの公開鍵に対するハンドルが返され、AUTOSIGN モードで署名が生成できるようになります。公開鍵ソフトウェアは、メッセージが送信される直前にデジタル署名を生成し、メッセージに添付します。 ■ 署名の検証 キーは、デジタル署名に関連付けられたプリンシパルを表します。署名の検証では、TPKEYOPEN( ) を呼び出す必要はありません。検証プロセスでは、デジタル署名付きメッセージに付属するデジタル証明書で指定された公開鍵を使用して、署名を検証します。 ■ 暗号化 キーは、暗号化されたメッセージの送信先のプリンシパルを表します。プリンシパル名と、TPKEY-ENCRYPT および TPKEY-AUTOENCRYPT を設定して TPKEYOPEN( ) を呼び出すと、プリンシパルのデジタル証明書を使用して、プリンシパルの公開鍵に対するハンドルが返され、AUTOENCRYPT モードで暗号化が可能になります。公開鍵ソフトウェアは、メッセージを暗号化し、メッセージに暗号化エンベロープを添付します。暗号化エンベロープにより、受信側のプロセスはメッセージを復号化できます。
TPKEYOPEN(3cbl)	■ 復号化 キーは、指定したプリンシパルに対して送信された秘密のメッセージの復号化を認可された状態で、呼び出しプロセスを識別します。プリンシパル名と、TPKEY-DECRYPT を設定して TPKEYOPEN( ) を呼び出すと、プリンシパルの秘密鍵およびデジタル証明書に対するハンドルが返されます。

表 3-3 公開鍵によるセキュリティの ATMI の COBOL ルーチン (続き)

使用するルーチン	目的
TPKEYGETINFO(3cbl)	キー・ハンドルに関連付けられた情報を取得します。暗号化のサービス・プロバイダに固有の情報も含まれていますが、以下の属性に関しては、すべてのサービス・プロバイダでサポートされます。 ■ PRINCIPAL 指定されたキー (キー・ハンドル) に関連付けられたプリンシパルの名前。プリンシパルとは、ユーザまたはプロセスのことです。どちらになるかは、ATMI アプリケーション開発者が公開鍵によるセキュリティを設定する方法に応じて決まります。ATMI アプリケーションの UBBCONFIG ファイルの SEC_PRINCIPAL_NAME パラメータを使用して指定したプリンシパルは、1 つまたは複数のシステム・プロセスの ID になります。詳細については、第 2 章の 11 ページ「プリンシパル名の指定」および 第 2 章の 49 ページ「プラグインによる復号化キーの初期化」を参照してください。 ■ PKENCRYPT_ALG 公開鍵による暗号化のキーで使用される、公開鍵アルゴリズムの ASN.1 DER (Distinguished Encoding Rules) 形式のオブジェクト識別子。詳細については、TPKEYGETINFO(3cbl) のリファレンス・ページを参照してください。 ■ PKENCRYPT_BITS 公開鍵アルゴリズムのキー長 (RSA の絶対値)。これは、512 〜 2048 ビットの範囲内の値でなければなりません。

表 3-3 公開鍵によるセキュリティの ATMI の COBOL ルーチン (続き)

使用するルーチン	目的
TPKEYGETINFO(3cbl)	■ SIGNATURE_ALG デジタル署名のキーで使用される、デジタル署名アルゴリズムの ASN.1 DER 形式のオブジェクト識別子。詳細については、TPKEYGETINFO(3cbl) のリファレンス・ページを参照してください。 ■ SIGNATURE_BITS デジタル署名アルゴリズムのキー長 (RSA の絶対値)。これは、512 ～ 2048 ビットの範囲内の値でなければなりません。 ■ ENCRYPT_ALG バルク・データの暗号化のキーで使用される、対称鍵アルゴリズムの ASN.1 DER 形式のオブジェクト識別子。詳細については、TPKEYGETINFO(3cbl) のリファレンス・ページを参照してください。 ■ ENCRYPT_BITS 対称鍵アルゴリズムのキー長。これは、40 ～ 128 ビットの範囲内の値でなければなりません。 ■ DIGEST_ALG デジタル署名のキーで使用される、メッセージ・ダイジェスト・アルゴリズムの ASN.1 DER 形式のオブジェクト識別子。詳細については、TPKEYGETINFO(3cbl) のリファレンス・ページを参照してください。 ■ PROVIDER 暗号サービス・プロバイダの名前。 ■ VERSION 暗号サービス・プロバイダのソフトウェアのバージョン。
TPKEYSETINFO(3cbl)	キー・ハンドルに関連付けられたオプションの属性パラメータを設定します。キー・ハンドルの属性のうち、主要な属性は、TPKEYGETINFO( ) で既に説明しています。このほか、特定の暗号サービス・プロバイダに固有な属性を使用することもできます。
TPKEYCLOSE(3cbl)	TPKEYOPEN( ) を使用して既にオープンしたキー・ハンドルをクローズします。

公開鍵のセキュリティで推奨されている事項について

- デジタル署名の生成またはデータの復号化に使用したキー・ハンドルが不要になったら、`tpkey_close()` を使用して直ちに解放してください。
- リプレイ攻撃を防ぐため、デジタル署名は、特定の操作に関する詳しい情報を指定したメッセージ・バッファに対してのみ作成してください。たとえば、「預金残高を確認しました」というメッセージだけを含むバッファは、簡単に傍受され、再利用されてしまうため危険です。一方、操作に固有な情報を多く含むメッセージはより安全です。たとえば、「2001 年 7 月 31 日付けの John Smith 様の口座 987654321 の預金残高は、100 ドルです。確認コードは 123456789 です。」という詳しいメッセージであれば、傍受されても簡単に再利用することはできません。

関連項目

- 第 3 章の 26 ページ「署名付きメッセージの送信と受信」
- 第 3 章の 38 ページ「暗号化されたメッセージの送信と受信」
- 第 3 章の 56 ページ「デジタル署名および暗号化情報の調査」
- 第 3 章の 63 ページ「型付きメッセージ・バッファの外部化」
- 第 1 章の 29 ページ「公開鍵によるセキュリティ機能」
- 第 2 章の 40 ページ「公開鍵セキュリティの管理」
- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」

署名付きメッセージの送信と受信

メッセージ・ベースのデジタル署名では、エンド・ツー・エンドの認証が行われ、メッセージの完全性が保たれます。この機能のしくみについては、図 1-7「ATMI PKCS-7 のエンド・ツー・エンドのデジタル署名」を参照してください。

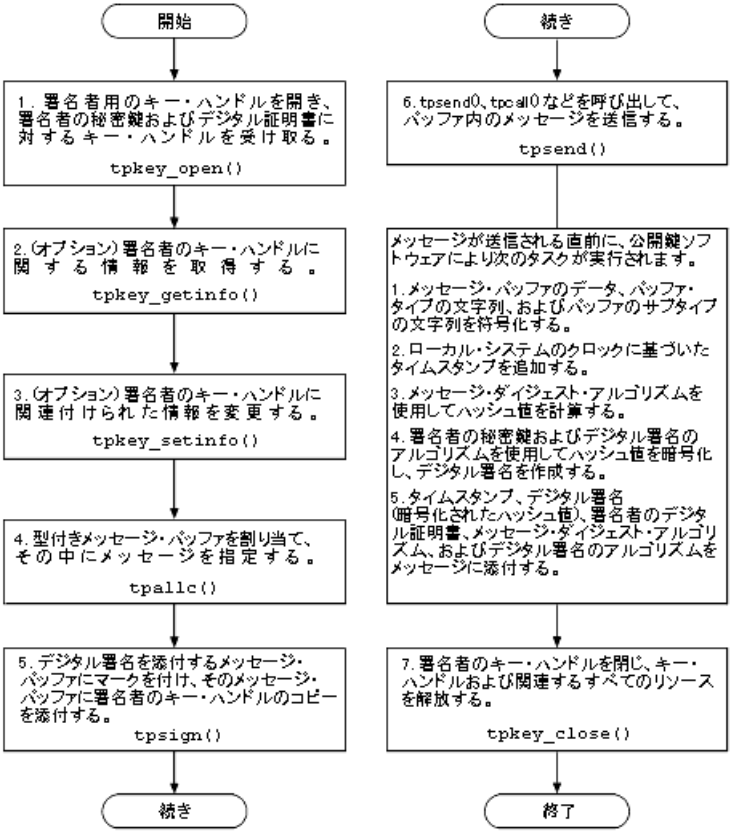
ATMI のメッセージ・バッファにデジタル署名を追加するには、送信側のプロセスまたはユーザがメッセージ・バッファに署名します。この署名には、メッセージ・バッファの内容から得た、暗号法的に安全なチェックサムと、署名者のローカル・クロックに基づいたタイムスタンプが含まれています。

このメッセージ・バッファに対するアクセス権がある場合は、署名者の署名が本物であるかどうか、メッセージ・バッファの内容が変更されていないかどうか、およびタイムスタンプが検証者のローカル・クロックで受け付ける時刻の範囲内かどうかを検証できます。さらに、時間に依存しない、サード・パーティによる検証を行うと、メッセージの非否認性を保証できます。つまり、送信側のプロセスまたはユーザは、メッセージを送信した事実を否認したり、メッセージが改ざんされたと主張することはできません。

署名付きメッセージを送信するためのコードの作成

次のフローチャートでは、署名付きメッセージを送信するコードを記述する手順を示します。

図 3-3 署名付きメッセージの送信手順



これらの手順およびメッセージ・バッファが署名されるしくみについては、以下の節を参照してください。

ステップ 1: デジタル署名用のキー・ハンドルをオープンする

まず、tpkey_open(3c) 関数または TPKEYOPEN(3cbl) ルーチンを呼び出して、送信側のプロセスが、署名者の秘密鍵および関連するデジタル証明書を使用できるようにします。秘密鍵は厳しくセキュリティ保護されているため、秘密鍵を所有することは、署名者の ID を所有することと同等の意味があります。

署名者の秘密鍵にアクセスするため、送信側のプロセスは、署名者として動作する権限があることを証明する必要があります。証明の条件は、公開鍵のプラグイン・インターフェイスのインプリメンテーションによって異なります。デフォルトの公開鍵のインプリメンテーションでは、呼び出しプロセス側が秘密のパスワードを入力する必要があります。

送信側のプロセスが tpkey_open() を呼び出してキー・ハンドルをオープンするとき、TPKEY_SIGNATURE フラグまたは TPKEY_AUTOSIGN フラグを指定して、そのキー・ハンドルがメッセージ・バッファのデジタル署名に使用されることを示します。通常、クライアントは tpinit() を呼び出した後でこの呼び出しを行い、サーバは tpsvrinit() を呼び出して初期化を行うときにこの呼び出しを行います。

TPKEY_AUTOSIGN フラグを使用してキー・ハンドルをオープンすると、署名の自動生成が可能になります。以降、送信側のプロセスは、メッセージ・バッファが送信されるたびに、メッセージ・バッファに自動的に署名します。TPKEY_AUTOSIGN フラグを使用すると、次の 3 つの利点があります。

- セキュリティ保護された ATMI アプリケーションでは ATMI 呼び出しが少なくなるため、アプリケーション・プログラマの作業が減ります。
- 最小限のコーディングを行うだけで、既存の ATMI アプリケーションにデジタル署名の技術を組み込むことができます。
- デジタル署名のないバッファがセキュリティ保護されていないネットワーク上で送信されてしまう、というプログラミング上のエラーの数を抑えることができます。

次のコード例では、署名者のキー・ハンドルをオープンする方法を示します。TPKEY は、atmi.h ヘッダ・ファイルで定義される特殊なデータ型です。

リスト 3-3 署名者のキー・ハンドルをオープンする例

```
main(argc, argv)
int argc;
char *argv[];
#endif
{
   TPKEY sdo_key;
   char *sdo_location;
```

```
  .
  .
  .
   if (tpkey_open(&sdo_key, "sdo", sdo_location,
      NULL, 0, TPKEY_SIGNATURE) == -1) {
      (void) fprintf(stderr, "tpkey_open sdo failed
         tperrno=%d(%s)\n", tperrno, tpstrerror(tperrno));
      exit(1);
   }
  .
  .
  .
}
```

ステップ 2: (オプション) キー・ハンドルの情報を取得する

署名者のキー・ハンドルの情報を取得して、キーの有効性を確認することができます。そのためには、tpkey_getinfo(3c) 関数または TPKEYGETINFO(3cbl) ルーチンを呼び出します。返される情報の中には、暗号サービス・プロバイダに固有の情報も含まれていますが、主要な属性は、すべてのプロバイダで共通です。

デフォルトの公開鍵のインプリメンテーションでは、メッセージ・バッファの署名を計算するための次の署名モードがサポートされています。

- RSA 公開鍵署名による MD5 メッセージ・ダイジェスト・アルゴリズム
- RSA 公開鍵署名による SHA-1 メッセージ・ダイジェスト・アルゴリズム

メッセージ・ダイジェスト・アルゴリズムは DIGEST_ALG のキー属性によって制御され、公開鍵署名は SIGNATURE_ALG のキー属性によって制御されます。サポートされている公開鍵のサイズは 512 ～ 2048 ビットであり、安全性とパフォーマンスを実現するためには十分な範囲です。公開鍵のサイズは、SIGNATURE_BITS キー属性によって制御されます。

デフォルトの公開鍵のインプリメンテーションでは、上記のアルゴリズムおよびキー・サイズの範囲で作成されたデジタル証明書の署名だけが認識されます。

次のコード例では、署名者のキー・ハンドルに関する情報を取得する方法を示します。

リスト 3-4 署名者のキー・ハンドルに関する情報を取得する例

```
main(argc, argv)
int argc;
char *argv[];
#endif
```

```
{
   TPKEY sdo_key;
   char principal_name[PNAME_LEN];
   long pname_len = PNAME_LEN;
  .
  .
  .
   if (tpkey_getinfo(sdo_key, "PRINCIPAL",
      principal_name, &pname_len, 0) == -1) {
      (void) fprintf(stdout, "Unable to get information
         about principal:%d(%s)\n",
            tperrno, tpstrerror(tperrno));
  .
  .
  .
      exit(1);
   }
  .
  .
  .
}
```

ステップ 3: (オプション) キー・ハンドルの情報を変更する

署名者のキー・ハンドルに関連付けられたオプション属性を設定するには、tpkey_setinfo(3c) 関数または TPKEYSETINFO(3cbl) ルーチンを呼び出します。キー・ハンドル属性は、暗号サービス・プロバイダによって異なります。

次のコード例では、署名者のキー・ハンドルに関連付けられた情報を変更する方法を示します。

リスト 3-5　署名者のキー・ハンドルに関連付けられた情報を変更する例

```
main(argc, argv)
int argc;
char *argv[];
#endif
{
   TPKEY sdo_key;
   static const unsigned char sha1_objid[] = {
      0x06, 0x05, 0x2b, 0x0e, 0x03, 0x02, 0x1a
   };
  .
  .
  .
```

```
   if (tpkey_setinfo(sdo_key, "DIGEST_ALG", (void *) sha1_objid,
      sizeof(sha1_objid), 0) == -1) {
      (void) fprintf(stderr, "tpkey_setinfo failed
         tperrno=%d(%s)\n",
            tperrno, tpstrerror(tperrno));
      return(1);
   }
  .
  .
  .
}
```

ステップ 4: バッファを割り当ててメッセージを指定する

型付きメッセージ・バッファを割り当てるには、tpalloc(3c) 関数を呼び出します。続いて、バッファにメッセージを指定します。

ステップ 5: デジタル署名を添付するバッファにマークを付ける

メッセージ・バッファにデジタル署名のマークを付ける (登録する) には、tpsign(3c) 関数を呼び出します。この関数を呼び出すと、署名者のキー・ハンドルのコピーがメッセージ・バッファに添付されます。TPKEY_AUTOSIGN フラグを指定してキーをオープンすると、tpsign() を明示的に呼び出さなくても、デジタル署名を添付するメッセージには自動的にマークが付きます。署名パラメータは保存され、後で使用するためにバッファに関連付けられます。

注記 COBOL のアプリケーションでは、AUTOSIGN 設定のメンバを使用してデジタル署名を作成します。TPKEYOPEN(3cbl) を参照してください。

次のコード例は、デジタル署名を添付するメッセージ・バッファにマークをつける方法を示しています。

リスト 3-6 デジタル署名を添付するメッセージ・バッファにマークを付ける例

```
main(argc, argv)
int argc;
char *argv[];
#endif
{
   TPKEY sdo_key;
   char *sendbuf, *rcvbuf;
  .
  .
```

```
   .
    if (tpsign(sendbuf, sdo_key, 0) == -1) {
       (void) fprintf(stderr, "tpsign failed tperrno=%d(%s)\n",
          tperrno, tpstrerror(tperrno));
       tpfree(rcvbuf);
       tpfree(sendbuf);
       tpterm();
       (void) tpkey_close(sdo_key, 0);
       exit(1);
    }
   .
   .
   .
}
```

ステップ 6: メッセージを送信する

メッセージ・バッファにデジタル署名のマークを付けた後、以下のいずれかの C 関数または COBOL ルーチンを使用して、メッセージ・バッファを送信します。

- tpcall() または TPCALL
- tpbroadcast() または TPBROADCAST
- tpconnect() または TPCONNECT
- tpenqueue() または TPENQUEUE
- tpforward()
- tpnotify() または TPNOTIFY
- tppost() または TPPOST
- tpreturn() または TPRETURN
- tpsend() または TPSEND

ステップ 7: 署名者のキー・ハンドルをクローズする

tpkey_close(3c) 関数または TPKEYCLOSE(3cbl) ルーチンを呼び出して、署名者のキー・ハンドルとそれに関連付けられたすべてのリソースを解放します。

デジタル署名の生成方法

デジタル署名の添付は、メッセージ・バッファが送信される直前に、公開鍵ソフトウェアによって行われます。デジタル署名されたバッファが複数回送信される場合は、送信のたびに新しい署名が生成されます。したがって、デジタル署名を行うメッセージ・バッファにマークを付けた後で、そのメッセージ・バッファを変更することができます。

公開鍵ソフトウェアは、次の 3 段階の手順でデジタル署名を生成します。

1. digest[message_buffer_data + buffer_type_string + buffer_subtype_string] = hash1
2. digest[hash1 + local_timestamp + PKCS-7_message_type] = hash2
3. {hash2}signer's_private_key = encrypted_hash2 = digital_signature

digest[ ] という表記は、メッセージ・ダイジェスト・アルゴリズム (この場合は MD5 または SHA-1) を使用して [ ] 内のハッシュ値が計算されることを示します。{ }*key* という表記は、*key* を使用して {} 内が暗号化または復号化されることを示します。この場合、計算されたハッシュ値は、署名者の秘密鍵を使用して暗号化されます。

署名のタイムスタンプ

デジタル署名には、ローカル・システムのクロックに基づいたタイムスタンプが組み込まれます。このようなタイムスタンプを組み込むことにより、受信者が署名を検証するときに、タイムスタンプ値の改ざんが検出されます。さらに、デジタル署名付きメッセージが宛先にルーティングされるとき、そのメッセージにはタイムスタンプのコピーが添付されます。

タイムスタンプは、秒単位まで表記されます。また、タイムスタンプは、PKCS-9 の `SigningTime` 形式で表記されます。

複数の署名について

1 つのメッセージ・バッファには、複数の署名を関連付けることができます。つまり、1 つのメッセージ・バッファに対して、任意の数の署名者が同時に署名できます。署名できるのは、ユーザまたはプロセスです。各署名者は、自分の秘密鍵を使用してメッセージ・バッファに署名します。

署名が異なる場合は、別のメッセージ・ダイジェスト・アルゴリズムまたはデジタル署名アルゴリズムを使用している可能性があります。同じメッセージ・ダイジェストおよびデジタル署名アルゴリズムを使用した署名が 2 つある場合、ハッシュ値はどちらか 1 つだけに対してだけ計算されます。

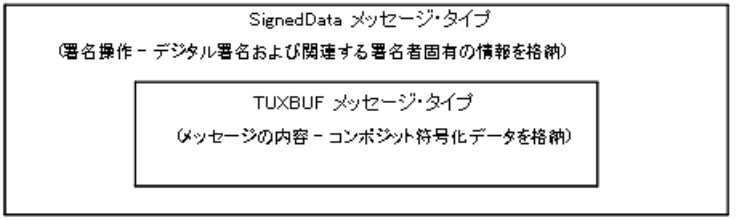
署名付きメッセージの内容

デジタル署名付きのメッセージ・バッファは、SignedData というメッセージ・タイプのバージョン 1 として、PKCS-7 形式で表現されます。BEA Tuxedo システムで使用されるメッセージ・タイプ SignedData は、次の項目で構成されます。

- 1 つ以上のデジタル署名。各デジタル署名は、署名者に固有な次の情報で構成されています。
  - 署名者の X.509v3 形式の証明書
  - メッセージ・ダイジェストおよびデジタル署名アルゴリズムの識別子
  - ローカル・クロックに基づくタイムスタンプ
- メッセージの内容。メッセージ・バッファのデータ、バッファ・タイプの文字列、およびバッファのサブタイプの文字列をまとめて BEA Tuxedo の符号化形式で表現したものです。この符号化形式により、メッセージ・バッファの署名は、どのマシンのアーキテクチャを使用しても検証できます。

次の図に示すように、メッセージの内容は、SignedData というメッセージ・タイプによって包含されています。

図 3-4 SignedData メッセージ・タイプ



署名付きメッセージの受信方法

署名付きメッセージ・バッファを受信するための ATMI アプリケーション・コードは必要ありません。公開鍵ソフトウェアは、添付されたデジタル署名を自動的に検証し、そのメッセージを受信側のプロセスに渡します。

受信側のプロセスの代わりに動作する公開鍵ソフトウェアは、署名付きメッセージ・バッファを受信すると、次のタスクを実行します。

1. 署名者のデジタル証明書、メッセージ・ダイジェスト・アルゴリズム、デジタル署名アルゴリズム、署名のタイムスタンプなど、受信されたメッセージに添付されているデジタル署名情報を読み取ります。

2. 署名者の公開鍵 (署名者のデジタル証明書に格納) とデジタル署名アルゴリズムを使用して、添付されたデジタル署名 (暗号化されたハッシュ値) を復号化します。
3. 次に示すように、受信したメッセージのハッシュ値を再計算します。
   a. digest[message_buffer_data + buffer_type_string + buffer_subtype_string] = hash1
   b. digest[hash1 + received_timestamp + PKCS-7_message_type] = hash2

   digest[ ] という表記は、メッセージ・ダイジェスト・アルゴリズム (この場合は MD5 または SHA-1) を使用して [ ] 内のハッシュ値が計算されることを示します。
4. 再計算されたハッシュ値と受信したハッシュ値を比較します。これらが同一でない場合、メッセージ・バッファが破棄されます。
5. 受信したタイムスタンプとローカル・システムのクロックを比較します。タイムスタンプが許容範囲内にない場合、メッセージ・バッファは破棄されます。
6. メッセージ・バッファがステップ 4 および 5 のチェックにパスすると、公開鍵ソフトウェアは、メッセージ・バッファのデータ、バッファ・タイプの文字列、およびバッファのサブタイプの文字列を復号化し、受信側のプロセスにメッセージを渡します。これは、送信側のプロセスで行われる符号化と逆の手順です。この BEA Tuxedo の符号化形式により、メッセージ・バッファの署名は、どのマシンのアーキテクチャを使用しても検証できます。

注記 添付されたデジタル署名をどれも検証できない場合、受信側のプロセスは、メッセージ・バッファを受け取りません。さらに、受信側のプロセスは、メッセージ・バッファをまったく認識しません。

デジタル署名を検証する

公開鍵ソフトウェアは、クライアント・プロセス、サーバ・プロセス、またはメッセージ・バッファの内容を読み取るシステム・プロセスに署名付きメッセージ・バッファがあると、メッセージに添付されたデジタル署名を自動的に検証します。ただし、パイプ役として機能するシステム・プロセス (メッセージの内容は読み取らない) の場合、メッセージに添付されたデジタル署名は検証されません。たとえば、ブリッジおよびワークステーション・ハンドラ (WSH) は、パイプ役として機能するシステム・プロセスの例です。

署名に記録されたタイムスタンプは、非同期のクロックに基づいているため、特に、PC またはパーソナル・コンピュータで署名が行われた場合は、完全に信頼できません。ただし、遠い過去または将来を示すタイムスタンプが記録された要求を、サーバ側で拒否することができます。タイムスタンプに基づいて要求を拒否する機能を使用すると、リプレイ攻撃から保護することができます。

入力バッファの署名を検証および送信する

メッセージ・バッファが入力パラメータとして ATMI 関数 (tpacall() など) に渡された場合、公開鍵ソフトウェアは、メッセージにあらかじめ添付された署名を検証してから、メッセージを転送します。この動作によって、複数のプロセスからの署名を付けた情報を、安全かつ確実に転送することができます。

サーバ側で、受信したメッセージ・バッファを変更して転送すると、元の署名は無効になります。この場合、公開鍵ソフトウェアは、無効な署名を検出して破棄します。このプロセスの例として、第 3 章の 53 ページ「入力バッファの暗号化エンベロープを破棄する」を参照してください。

出力バッファの署名を置換する

メッセージ・バッファが出力パラメータとして ATMI 関数 (tpgetrply() など) に渡された場合、公開鍵ソフトウェアは、このバッファに関連付けられた署名の情報を削除します。削除する情報には、保留中の署名 (メッセージ・バッファに登録された署名)、およびバッファを前回使用したユーザの署名が含まれます。

この操作が正常に終了すると、新しい署名の情報が新しいバッファの内容に関連付けられる場合があります。

関連項目

- 第 3 章の 38 ページ「暗号化されたメッセージの送信と受信」
- 第 3 章の 56 ページ「デジタル署名および暗号化情報の調査」
- 第 3 章の 63 ページ「型付きメッセージ・バッファの外部化」
- 第 1 章の 29 ページ「公開鍵によるセキュリティ機能」
- 第 2 章の 40 ページ「公開鍵セキュリティの管理」
- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」

暗号化されたメッセージの送信と受信

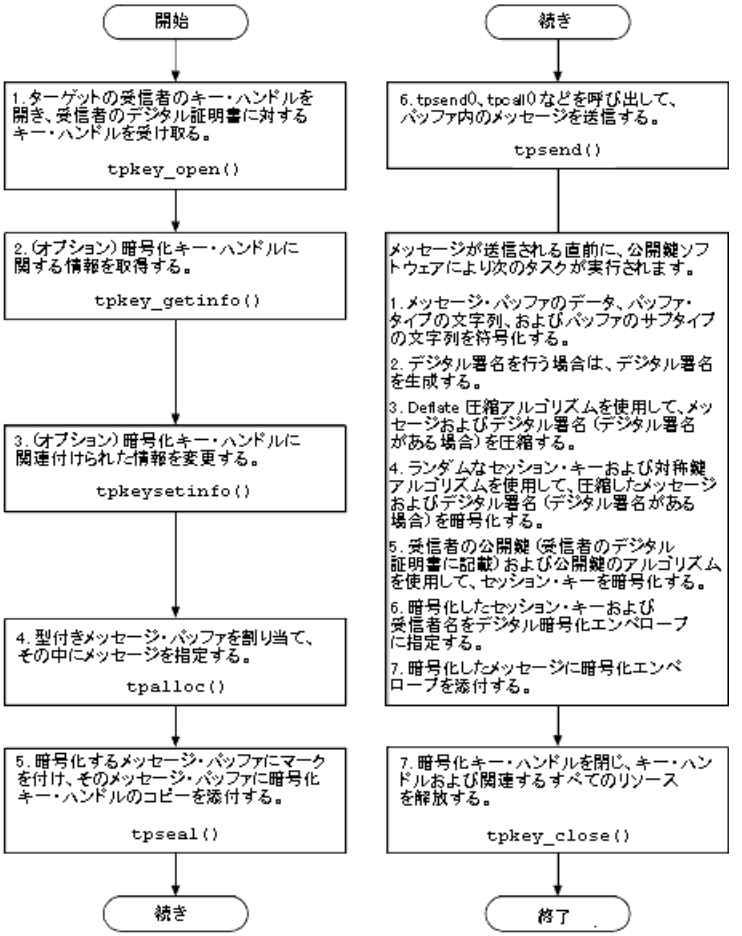
メッセージ・ベースの暗号化では、エンド・ツー・エンドでデータの機密性が保たれます。この機能のしくみについては、図 1-9「ATMI PKCS-7 のエンド・ツー・エンドの暗号化」を参照してください。

メッセージは、送信側のプロセスを離れる直前に暗号化され、その状態は受信側のプロセスで受信されるまで保持されます。メッセージは、オペレーティング・システムのメッセージ・キュー、システム・プロセス、ディスク・ベース・キューなどの中継ポイントのほか、サーバ間のネットワーク・リンクで転送される間もオペークです。

暗号化されたメッセージを送信するためのコードの作成

次のフローチャートでは、暗号化されたメッセージを送信するコードを記述する手順を示します。

図 3-5 暗号化されたメッセージの送信手順



これらの手順およびメッセージ・バッファが暗号化されるしくみについては、以下の節を参照してください。

ステップ 1: 暗号化用のキー・ハンドルをオープンする

まず、tpkey_open(3c) 関数または TPKEYOPEN(3cbl) ルーチンを呼び出して、送信側のプロセスが、ターゲット受信者のデジタル証明書を使用できるようにします。ターゲット受信者とは、クライアント、サービス、サーバ・グループ、ゲートウェイ・グループ、サーバ・マシン、または複数のサーバを含むドメイン全体のことです。

送信側のプロセスが tpkey_open() を呼び出してキー・ハンドルをオープンするとき、TPKEY_ENCRYPT フラグまたは TPKEY_AUTOENCRYPT フラグを指定して、そのキー・ハンドルがメッセージ・バッファの暗号化に使用されることを示します。通常、クライアントは tpinit() を呼び出した後でこの呼び出しを行い、サーバは tpsvrinit() を呼び出して初期化を行うときにこの呼び出しを行います。

TPKEY_AUTOENCRYPT フラグを使用してキー・ハンドルをオープンすると、暗号化の自動処理が可能になります。以降、送信側のプロセスは、メッセージ・バッファが送信されるたびに、メッセージ・バッファを自動的に暗号化します。
TPKEY_AUTOENCRYPT フラグを使用すると、次の 3 つの利点があります。

- セキュリティ保護された ATMI アプリケーションでは ATMI 呼び出しが少なくなるため、アプリケーション・プログラマの作業が減ります。
- 最小限のコーディングを行うだけで、既存の ATMI アプリケーションに暗号化の技術を組み込むことができます。
- 暗号化されていない (平文の) バッファがセキュリティ保護されていないネットワーク上で送信されてしまう、というプログラミング上のエラーの数を抑えることができます。

次のコード例では、暗号化キー・ハンドルをオープンする方法を示します。TPKEY は、atmi.h ヘッダ・ファイルで定義される特殊なデータ型です。

リスト 3-7 暗号化キー・ハンドルをオープンする例

```
main(argc, argv)
int argc;
char *argv[];
#endif
{
   TPKEY tu_key;
  .
  .
  .
   if (tpkey_open(&tu_key, "TOUPPER", NULL,
      NULL, 0, TPKEY_ENCRYPT) == -1) {
      (void) fprintf(stderr, "tpkey_open tu failed
```

```
            tperrno=%d(%s)\n", tperrno, tpstrerror(tperrno));
        exit(1);
    }
    .
    .
    .
}
```

ステップ 2: (オプション) キー・ハンドルの情報を取得する

暗号化キー・ハンドルの情報を取得して、キーの有効性を確認することができます。そのためには、tpkey_getinfo(3c) 関数または TPKEYGETINFO(3cbl) ルーチンを呼び出します。返される情報の中には、暗号サービス・プロバイダに固有の情報も含まれていますが、主要な属性は、すべてのプロバイダで共通です。

デフォルトの公開鍵のインプリメンテーションでは、バルク・データを暗号化するための 3 つのアルゴリズムがサポートされています。

- DES (DES-CBC)—CBC (Cipher Block Chaining) モードで実行する 64 ビット単位のブロック暗号です。56 ビット (64 ビットの暗号化キーから 8 パリティ・ビットを引いたもの) のキーを使用し、米国以外の国でも使用できます。DES は Data Encryption Standard の略です。
- 3DES (2 つの鍵による Triple DES)—EDE (Encrypt-Decrypt-Encrypt) モードで実行する 128 ビット単位のブロック暗号です。3DES では、56 ビット (実際には 112 ビット) のキーを使用します。米国から輸出することは禁止されています。
- RC2—40 ～ 128 ビットの範囲で、暗号鍵のサイズを変更できるブロック暗号です。DES より高速であり、40 ビットの暗号鍵は輸出できます。米国籍の企業の海外子会社および海外支店であれば、56 ビットの暗号鍵を使用することができます。公開鍵ソフトウェアでは、暗号鍵の長さを 128 ビットに制限していますが、米国では実質的に、RC2 にはどんな長さの鍵を使用することもできます。RC2 は Rivest's Cipher 2 の略です。

暗号化のレベルは、ENCRYPT_BITS のキー属性によって制御され、アルゴリズムは ENCRYPT_ALG のキー属性によって制御されます。ENCRYPT_ALG で固定キー長によるアルゴリズムが設定されると、ENCRYPT_BITS の値が自動的に調整されます。

次のコード例では、暗号化キー・ハンドルに関する情報を取得する方法を示します。

リスト 3-8 暗号化キー・ハンドルに関する情報を取得する例

```
main(argc, argv)
int argc;
```

```
char *argv[];
#endif
{
   TPKEY tu_key;
   char principal_name[PNAME_LEN];
   long pname_len = PNAME_LEN;
  .
  .
  .
   if (tpkey_getinfo(tu_key, "PRINCIPAL",
      principal_name, &pname_len, 0) == -1) {
      (void) fprintf(stdout, "Unable to get information
         about principal:%d(%s)\n",
            tperrno, tpstrerror(tperrno));
  .
  .
  .
      exit(1);
   }
  .
  .
  .
}
```

ステップ 3: (オプション) キー・ハンドルの情報を変更する

暗号化キー・ハンドルに関連付けられたオプション属性を設定するには、tpkey_setinfo(3c) 関数または TPKEYSETINFO(3cbl) ルーチンを呼び出します。キー・ハンドル属性は、暗号サービス・プロバイダによって異なります。

次のコード例では、暗号化キー・ハンドルに関連付けられた情報を変更する方法を示します。

リスト 3-9 暗号化キー・ハンドルに関連付けられた情報を変更する例

```
main(argc, argv)
int argc;
char *argv[];
#endif
{
   TPKEY tu_key;
   static const unsigned char rc2_objid[] = {
      0x06, 0x08, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x03, 0x02
   };
```

```
   .
   .
   .
    if (tpkey_setinfo(tu_key, "ENCRYPT_ALG", (void *) rc2_objid,
       sizeof(rc2_objid), 0) == -1) {
       (void) fprintf(stderr, "tpkey_setinfo failed
          tperrno=%d(%s)\n",
             tperrno, tpstrerror(tperrno));
       return(1);
    }
   .
   .
   .
}
```

ステップ 4: バッファを割り当ててメッセージを指定する

型付きメッセージ・バッファを割り当てるには、tpalloc(3c) 関数を呼び出します。続いて、バッファにメッセージを指定します。

ステップ 5: 暗号化するバッファにマークを付ける

メッセージ・バッファに暗号化のマークを付ける (登録する) には、tpseal(3c) 関数を呼び出します。この関数を呼び出すと、暗号化キー・ハンドルのコピーがメッセージ・バッファに添付されます。TPKEY_AUTOENCRYPT フラグを使用してキーをオープンすると、tpseal() を明示的に呼び出さなくても、暗号化するメッセージには自動的にマークが付きます。

注記 COBOL のアプリケーションでは、AUTOENCRYPT 設定のメンバを使用してメッセージ・バッファを暗号化します。TPKEYOPEN(3cbl) を参照してください。

次のコード例は、暗号化するメッセージ・バッファにマークを付ける方法を示しています。

リスト 3-10 暗号化するメッセージ・バッファにマークを付ける例

```
main(argc, argv)
int argc;
char *argv[];
#endif
{
   TPKEY tu_key;
   char *sendbuf, *rcvbuf;
  .
  .
  .
   if (tpseal(sendbuf, tu_key, 0) == -1) {
      (void) fprintf(stderr, "tpseal failed tperrno=%d(%s)\n",
         tperrno, tpstrerror(tperrno));
      tpfree(rcvbuf);
      tpfree(sendbuf);
      tpterm();
      (void) tpkey_close(tu_key, 0);
      exit(1);
   }
  .
  .
  .
}
```

ステップ 6: メッセージを送信する

メッセージ・バッファに暗号化のマークを付けた後、以下のいずれかの C 関数または COBOL ルーチンを使用して、メッセージ・バッファを送信します。

- `tpcall()` または `TPCALL`
- `tpbroadcast()` または `TPBROADCAST`
- `tpconnect()` または `TPCONNECT`
- `tpenqueue()` または `TPENQUEUE`
- `tpforward()`
- `tpnotify()` または `TPNOTIFY`
- `tppost()` または `TPPOST`
- `tpreturn()` または `TPRETURN`
- `tpsend()` または `TPSEND`

ステップ 7: 暗号化キー・ハンドルをクローズする

`tpkey_close(3c)` 関数または `TPKEYCLOSE(3cbl)` ルーチンを呼び出して、暗号化キー・ハンドルとそれに関連付けられたすべてのリソースを解放します。

メッセージ・バッファの暗号化方法

公開鍵ソフトウェアは、メッセージ・バッファが送信される直前に、メッセージを暗号化して暗号化エンベロープを添付します。暗号化エンベロープにより、ターゲット受信者はメッセージを復号化できます。封印されたバッファが複数回送信される場合は、送信のたびに暗号化が実行されます。したがって、暗号化するメッセージ・バッファにマークを付けた後で、そのメッセージ・バッファを変更することができます。

公開鍵ソフトウェアは、次の手順に従って、メッセージ・バッファの内容を暗号化し、暗号化メッセージの受信者用の暗号化エンベロープを生成します。

1. {message_buffer_data + buffer_type_string + buffer_subtype_string}session_key = encrypted_message
2. {session_key}recipient's_public_key = encrypted_session_key = encryption_envelope_for_recipient

{ }*key* という表記は、*key* を使用して { } 内 が暗号化または復号化されることを示します。ステップ 1 では、セッション・キーを使用してメッセージ・バッファが暗号化され、ステップ 2 では、受信者の公開鍵を使用してセッション・キーが暗号化されます。

複数のメッセージ受信者について

1 つのメッセージ・バッファには、複数の暗号化エンベロープを関連付けることができます。つまり、異なる秘密鍵を持つ複数の受信者が、暗号化されたメッセージを受信し、復号化することができます。受信者となるのは、ユーザまたはプロセスです。メッセージが複数の受信者に対して暗号化されると、メッセージは一度だけ暗号化されますが、セッション・キーは各受信者の公開鍵で暗号化されます。暗号化されたメッセージには、すべての暗号化エンベロープが添付されます。

1 つのメッセージ・バッファに複数の暗号化エンベロープが関連付けられた場合、すべての暗号化エンベロープは、そのアルゴリズムに対して同じ対称鍵アルゴリズムと同じキー・サイズを使用しなければなりません。

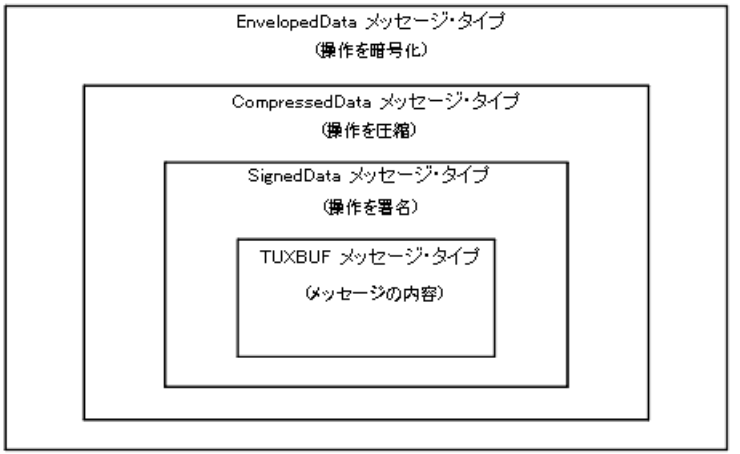
暗号化されたメッセージの内容

暗号化されたメッセージ・バッファは、`EnvelopedData` というメッセージ・タイプのバージョン 0 として、PKCS-7 形式で表現されます。BEA Tuxedo システムで使用されるメッセージ・タイプ `EnvelopedData` は、次の項目で構成されます。

- どの ATMI プロセスからも読み取ることができる、平文で表記された受信者のリスト
- 1 人または複数の受信者のための暗号化エンベロープ
- 公開鍵のアルゴリズムおよび関連するパラメータ (セッション・キーはこのアルゴリズムで暗号化されます)
- 対称鍵のアルゴリズムおよび関連するパラメータ (バルク・データはこのアルゴリズムで暗号化されます)
- 暗号化されたバルク・データ。メッセージ・バッファのデータ、バッファ・タイプの文字列、バッファのサブタイプの文字列、およびデジタル署名 (デジタル署名がある場合) をまとめたものであり、以下の変換処理が実行されます。
  - メッセージ・バッファ・データ、バッファ・タイプの文字列、およびバッファのサブタイプの文字列を BEA Tuxedo の符号化形式に変換し、コンポジット符号化データを生成します。この BEA Tuxedo の符号化形式により、メッセージ・バッファは、どのマシンのアーキテクチャを使用しても復号化できます。
  - Deflate 圧縮アルゴリズムを使用して、コンポジット符号化データおよびデジタル署名 (デジタル署名がある場合) を圧縮し、コンポジット圧縮データを生成します。
  - ランダムに生成されたセッション・キーおよび対称鍵アルゴリズム (このリストの前半で説明) を使用してコンポジット圧縮データを暗号化し、バルク・データを暗号化します。

次の図は、EnvelopedData メッセージ・タイプの場合のエンベロープの階層を示します。SignedData メッセージ・タイプは、メッセージに 1 つまたは複数のデジタル署名が関連付けられている場合にのみ、この階層に含まれます。

図 3-6 EnvelopedData メッセージ・タイプ



上の図に示すように、メッセージ・バッファは、署名することも暗号化することもできます。メッセージ・バッファに関連するデジタル署名の数と暗号化エンベロープの数の間に、関係を成立させる必要はありません。

メッセージ・バッファに対して署名と暗号化の両方が実行されると、まず、暗号化されていないデータに対する署名が生成されます。次に、添付される署名の数および署名者の ID が、バルク・データの暗号化機能により暗号化されます。

注記 署名を検証する前に、メッセージのデータを復号化するための適切な復号化キーを使用できる状態にしておく必要があります。

暗号化されたメッセージを受信するためのコードの記述

暗号化されたメッセージを受信するためのコードを記述するには、次の手順に従います。

1. tpkey_open() を呼び出して、ターゲット受信者のキー・ハンドルをオープンします。tpkey_open を呼び出すと、受信者の秘密鍵およびデジタル証明書に対するキー・ハンドルが返されます。
2. (オプション):tpkey_getinfo() を呼び出して、復号化キー・ハンドルに関する情報を取得します。
3. (オプション):tpkey_setinfo() を呼び出して、復号化キー・ハンドルに関連付けられた情報を変更します。
4. tpkey_close() を呼び出して、復号化キー・ハンドルをクローズします。tpkey_close() を呼び出すと、キー・ハンドルとそれに関連付けられたすべてのリソースが解放されます。

これらの手順およびメッセージ・バッファが復号化されるしくみについては、以下の節を参照してください。

ステップ 1: 復号化用のキー・ハンドルをオープンする

まず、tpkey_open(3c) 関数または TPKEYOPEN(3cbl) ルーチンを呼び出して、受信側のプロセスが、ターゲット受信者の秘密鍵および関連するデジタル証明書を使用できるようにします。受信側のプロセスとは、クライアント、サービス、サーバ・グループ、ゲートウェイ・グループ、サーバ・マシン、または複数のサーバを含むドメイン全体のことです。

アプリケーション管理者は、ATMI アプリケーションの UBBCONFIG ファイルを設定して、ATMI アプリケーションの起動時に復号化キー・ハンドルを自動的にオープンするように指定できます。この方法では、サーバごとに使用できる復号化キー・ハンドルは 1 つだけです。詳細については、第 2 章の 49 ページ「プラグインによる復号化キーの初期化」を参照してください。

起動時に受信側のプロセスの復号化キー・ハンドルをオープンするように ATMI アプリケーションが設定されていない場合、受信側のプロセスは自身で tpkey_open() を呼び出します。または、受信側のプロセスは、別の tpkey_open() を呼び出して、別の復号化キー・ハンドルをオープンすることもできます。

ターゲット受信者の秘密鍵にアクセスするため、受信側のプロセスは、ターゲット受信者として動作する権限があることを証明する必要があります。証明の条件は、公開鍵のプラグイン・インターフェイスのインプリメンテーションによって異なります。デフォルトの公開鍵のインプリメンテーションでは、呼び出しプロセス側が秘密のパスワードを入力する必要があります。

受信側のプロセスが tpkey_open() を呼び出してキー・ハンドルをオープンするとき、TPKEY_DECRYPT フラグを指定して、そのハンドルがメッセージ・バッファの復号化に使用されることを示します。通常、クライアントは tpinit() を呼び出した後でこの呼び出しを行い、サーバは tpsvrinit() を呼び出して初期化を行うときにこの呼び出しを行います。

次のコード例では、復号化キー・ハンドルをオープンする方法を示します。TPKEY は、atmi.h ヘッダ・ファイルで定義される特殊なデータ型です。

リスト 3-11 復号化キー・ハンドルをオープンする例

```
TPKEY tu_key;

tpsvrinit(argc, argv)
int argc;
char **argv;
#endif
{
   char  *tu_location;
  .
  .
  .
   if (tpkey_open(&tu_key, "TOUPPER", tu_location,
      NULL, 0, TPKEY_DECRYPT) == -1) {
      userlog("Unable to open private key:%d(%s)",
         tperrno, tpstrerror(tperrno));
   return(-1)
   }
  .
  .
  .
}
```

ステップ 2: (オプション) キー・ハンドルの情報を取得する

復号化キー・ハンドルの情報を取得して、キーの有効性を確認することができます。そのためには、tpkey_getinfo(3c) 関数または TPKEYGETINFO(3cbl) ルーチンを呼び出します。返される情報の中には、暗号サービス・プロバイダに固有の情報も含まれていますが、主要な属性は、すべてのプロバイダで共通です。

次のコード例では、復号化キー・ハンドルに関する情報を取得する方法を示します。

リスト 3-12 復号化キー・ハンドルに関する情報を取得する例

```
TPKEY tu_key;

tpsvrinit(argc, argv)
int argc;
char **argv;
#endif
{
   char principal_name[PNAME_LEN];
   long pname_len = PNAME_LEN;
  .
  .
  .
   if (tpkey_getinfo(tu_key, "PRINCIPAL",
      principal_name, &pname_len, 0) == -1) {
      (void) fprintf(stdout, "Unable to get information
         about principal:%d(%s)\n",
            tperrno, tpstrerror(tperrno));
  .
  .
  .
      exit(1);
   }
  .
  .
  .
}
```

ステップ 3: (オプション) キー・ハンドルの情報を変更する

復号化キー・ハンドルに関連付けられたオプション属性を設定するには、tpkey_setinfo(3c) 関数または TPKEYSETINFO(3cbl) ルーチンを呼び出します。キー・ハンドル属性は、暗号サービス・プロバイダによって異なります。

次のコード例では、復号化キー・ハンドルに関連付けられた情報を変更する方法を示します。

リスト 3-13 復号化キー・ハンドルに関連付けられた情報を変更する例

```
TPKEY tu_key;

tpsvrinit(argc, argv)
int argc;
char **argv;
#endif
{
   TM32U mybits = 128;
  .
  .
  .
   if (tpkey_setinfo(tu_key, "ENCRYPT_BITS", &mybits,
      sizeof(mybits), 0) == -1) {
      (void) fprintf(stderr, "tpkey_setinfo failed
         tperrno=%d(%s)\n",
            tperrno, tpstrerror(tperrno));
      return(1);
   }
  .
  .
  .
}
```

ステップ 4: 復号化キー・ハンドルをクローズする

tpkey_close(3c) 関数または TPKEYCLOSE(3cbl) ルーチンを呼び出して、復号化キー・ハンドルとそれに関連付けられたすべてのリソースを解放します。

メッセージ・バッファの復号化方法

公開鍵ソフトウェアは、BEA Tuxedo のクライアント・プロセス、サーバ・プロセス、またはメッセージ・バッファの内容を読み取るシステム・プロセスに暗号化されたメッセージ・バッファがあると、そのメッセージ・バッファを自動的に検証します。復号化の自動処理を成功させるには、受信側のプロセスで、添付された暗号化エンベロープのいずれかで指定されている復号化キー (TPKEY_DECRYPT) をオープンしておく必要があります。

受信側のプロセスの代わりに動作する公開鍵ソフトウェアは、暗号化されたメッセージ・バッファを受信すると、次のタスクを実行します。

1. 添付された暗号化エンベロープ内のターゲット受信者の名前を読み取ります。
2. セッション・キーを復元するには、受信者の秘密鍵と公開鍵アルゴリズムを使用して、受信者の暗号化エンベロープを復号化します。
3. 復元されたセッション・キーと対称鍵アルゴリズムを使用して、メッセージを復号化します。
4. メッセージを圧縮解除します。
5. デジタル署名があれば検証します。第 3 章の 35 ページ「署名付きメッセージの受信方法」を参照してください。
6. メッセージ・バッファがステップ 5 のチェックにパスすると、公開鍵ソフトウェアは、メッセージ・バッファのデータ、バッファ・タイプの文字列、およびバッファのサブタイプの文字列を復号化し、受信側のプロセスに平文化されたメッセージを渡します。これは、送信側のプロセスで行われる符号化と逆の手順です。この BEA Tuxedo の符号化形式により、メッセージ・バッファは、どのマシンのアーキテクチャを使用しても復号化できます。

注記 添付されたデジタル署名をどれも検証できない場合、または、メッセージ・バッファを復号化できない場合、受信側のプロセスはメッセージ・バッファを受け取りません。さらに、受信側のプロセスは、メッセージ・バッファをまったく認識しません。

ただし、パイプ役として機能するシステム・プロセス (メッセージの内容は読み取らない) の場合、メッセージは復号化されません。たとえば、ブリッジおよびワークステーション・ハンドラ (WSH) は、パイプ役として機能するシステム・プロセスの例です。

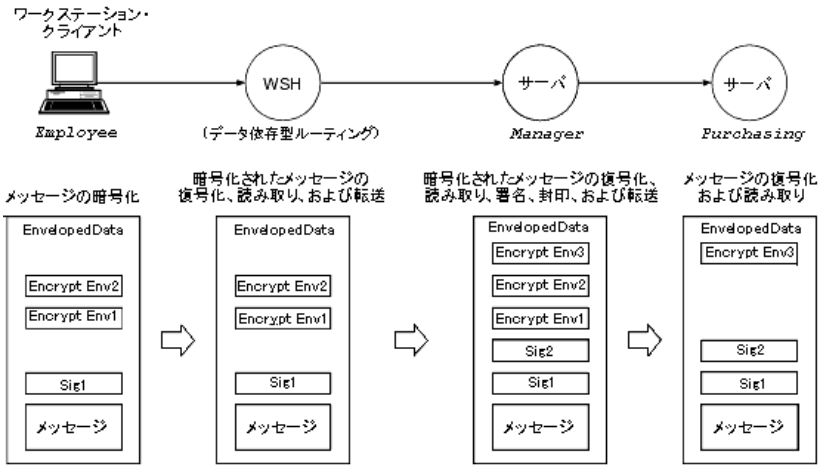
WSH は、パイプ役の特殊な例です。データ依存型ルーティング用に設定された WSH は、メッセージ・バッファを受信するとそれを読み取り、バッファのルーティング方法を決定します。つまり、まず、公開鍵ソフトウェアは、受信したメッセージ・バッファのコピーを作成し、そのコピーを復号化して WSH に渡します。WSH は、受け取ったコピーを解析し、そのメッセージ・バッファをルーティングする方法を決定すると、元のメッセージ・バッファをそのまま適切なサーバにルーティングします。データ依存型ルーティングと公開鍵セキュリティとの相互運用の詳細については、第 1 章の 60 ページ「データ依存型ルーティングとの互換性および相互運用性」を参照してください。

入力バッファの暗号化エンベロープを破棄する

メッセージ・バッファが入力パラメータとして ATMI 関数 (tpacall() など) に渡された場合、公開鍵ソフトウェアは、メッセージにあらかじめ添付された暗号化エンベロープを破棄します。この機能により、中継プロセスで変更された元のメッセージが、ターゲット受信者側で受け取られないようにすることができます。

このプロセスの例として、次の図に示すシナリオを考えてみます。

図 3-7　暗号化された署名付きメッセージを転送する例



この例は、Manager という名前のサーバ・プロセスが、Employee というクライアント・プロセスから、暗号化された署名付きメッセージ・バッファを受信する様子を示しています。サーバは、受信したメッセージ・バッファを復号化して読み取った後で、Purchasing サービス用に署名および封印し、Purchasing に送信します。

この操作を以下に詳しく説明します。

1. ワークステーション・ハンドラ (WSH) は、暗号化された署名付きメッセージ・バッファを Employee というクライアントから受け取ると、そのまま転送します。

   WSH プロセスには、データ依存型ルーティングが設定されています。これについては、第 3 章の 52 ページ「メッセージ・バッファの復号化方法」を参照してください。公開鍵ソフトウェアは、WSH プロセスで既にオープンした復号化キーを使用して、受信したメッセージ・バッファのコピーを復号化します。次に、復号化したメッセージのコピーを WSH に渡します。WSH は、復号化されたコピーを解析してから、受信したメッセージ・バッファをそのまま Manager プロセスに転送します。

   WSH プロセスにデータ依存型ルーティングが設定されていない場合、Employee プロセスは、WSH プロセスのメッセージ・バッファに対して tpseal() を呼び出す必要はありません。また、WSH プロセスも復号化キーをオープンする必要はありません。

   データ依存型ルーティングが設定されているかどうかにかかわらず、WSH はデジタル署名の検証を行いません。

2. メッセージ・バッファが Manager プロセスで受け取られると、公開鍵ソフトウェアは次の操作を行います。

   a. Manager プロセスで既にオープンした復号化キーを使用して、メッセージ・バッファを復号化します。

   b. Employee の署名を検証します。

   c. デジタル署名や暗号化情報のないメッセージを Manager に渡します。

   プロセスがメッセージ・バッファを受信するときは、メッセージの内容だけを受信します。そのメッセージ・バッファに関連付けられているデジタル署名や暗号化エンベロープは受信しません。

3. Manager は、tpenvelope() を繰り返し呼び出して、メッセージ・バッファに関連付けられたデジタル署名と暗号化の情報を調べます。tpenvelope() からは、次の情報が返されます。

   - デジタル署名の情報。署名者の公開鍵および TPSIGN_OK というデジタル署名のステータスが含まれます。
   - 暗号化の情報。WSH プロセスの公開鍵および Manager プロセス自身の公開鍵が含まれます。

4. Manager は、署名者の公開鍵を引数に指定して tpkey_getinfo() を呼び出し、署名者に関するさらに詳しい情報 (プリンシパル名など) を取得します。

5. Manager は、署名者を確認し、Employee からの要求 (メッセージで指定) が有効であると判断すると、次の操作を行います。
   a. tpsign() を呼び出し、Manager によるデジタル署名を添付するメッセージ・バッファにマークを付けます。
   b. tpseal() を呼び出して、メッセージ・バッファに Purchasing 用の暗号化のマークを付けます。
   c. tpforward() (またはデータの送信に使用する別の関数) を呼び出して、メッセージを Purchasing に送信します。

公開鍵ソフトウェアは、メッセージが送信される直前に、次のタスクを実行します。

1. Manager のデジタル署名を生成します。
2. Employee のデジタル署名を検証します。
3. メッセージの内容および関連するデジタル署名を暗号化します。
4. Purchasing 用の暗号化エンベロープを作成します。

出力バッファの暗号化エンベロープを置換する

メッセージ・バッファが出力パラメータとして ATMI 関数 (tpgetrply() など) に渡された場合、公開鍵ソフトウェアは、このバッファに関連付けられた暗号化の情報を削除します。削除する情報には、保留中の封印 (メッセージ・バッファに登録された受信者の封印)、および前回使用したバッファの封印が含まれます。

この操作が正常に終了すると、新しい暗号化の情報が、新しいバッファの内容に関連付けられる場合があります。

関連項目

- 第 3 章の 56 ページ「デジタル署名および暗号化情報の調査」
- 第 3 章の 63 ページ「型付きメッセージ・バッファの外部化」
- 第 1 章の 29 ページ「公開鍵によるセキュリティ機能」
- 第 2 章の 40 ページ「公開鍵セキュリティの管理」
- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」

デジタル署名および暗号化情報の調査

公開鍵ソフトウェアでは、次の順序指定が適用されます。

- デジタル署名の登録要求およびデジタル署名がメッセージ・バッファに添付される順序
- 暗号化の登録要求および暗号化エンベロープがメッセージ・バッファに添付される順序

プロセスは、ターゲットのメッセージ・バッファを引数に指定して tpenvelope() 関数を呼び出すことにより、この情報を取得します。tpenvelope() については、『BEA Tuxedo C 言語リファレンス』の tpenvelope(3c) リファレンス・ページを参照してください。

1 つのメッセージ・バッファに対して、デジタル署名の登録要求、デジタル署名、暗号化の登録要求、および暗号化エンベロープの複数のオカレンスが関連付けられる場合があります。これらのオカレンスは、最初の項目を 0 として、順番に連続する位置に格納されます。tpenvelope() の occurrence 入力パラメータは、要求された項目を示します。最後の項目より大きい値が occurrence に指定されている場合に tpenvelope() を呼び出すと、TPENOENT エラーが返されます。TPENOENT が返されるまで tpenvelope() を繰り返し呼び出すことにより、すべての項目を調べることができます。

送信側のプロセスでは通常、デジタル署名および暗号化の情報は、メッセージが送信されるまで保留状態になっています。受信側のプロセスでは、デジタル署名は既に検証され、暗号化と復号化は実行済みです。

送信側のプロセスが tpenvelope を呼び出したときの動作

送信側のプロセスが発信メッセージ・バッファを引数として tpenvelope() を呼び出すと、tpenvelope() は次のレポートを作成します。

- メッセージ・バッファに明示的に登録されたデジタル署名要求の状態は、TPSIGN_PENDING です。送信側のプロセスは、tpsign(3c) 関数を呼び出して、デジタル署名要求を明示的に登録します。
- メッセージ・バッファに暗黙的に登録されたデジタル署名要求の状態も、TPSIGN_PENDING です。送信側のプロセスは、TPKEY_AUTOSIGN フラグを指定して tpkey_open(3c) を呼び出し、デジタル署名要求を暗黙的に登録します。

- メッセージ・バッファに明示的に登録された暗号化 (封印) 要求の状態は、TPSEAL_PENDING です。送信側のプロセスは、tpseal(3c) 関数を呼び出して、暗号化要求を明示的に登録します。
- メッセージ・バッファに暗黙的に登録された暗号化 (封印) 要求のステータスも、TPSEAL_PENDING です。送信側のプロセスは、TPKEY_AUTOENCRYPT フラグを指定して tpkey_open() を呼び出し、暗号化要求を暗黙的に登録します。

tpenvelope() は、ステータスの情報のほか、デジタル署名または暗号化の登録要求に関連付けられたキー・ハンドルも返します。プロセスは、キー・ハンドルを引数として tpkey_getinfo(3c) 関数を呼び出し、キー・ハンドルに関する詳しい情報を取得できます。

受信側のプロセスが tpenvelope を呼び出したときの動作

プロセスがメッセージ・バッファを受信するときは、メッセージの内容だけを受信します。メッセージ・バッファに関連付けられているデジタル署名や暗号化エンベロープは受信しません。受信側のプロセスは、tpenvelope() を呼び出して、添付されたデジタル署名や暗号化エンベロープに関する情報を取得する必要があります。

受信側のプロセスが受信メッセージ・バッファを引数として tpenvelope() を呼び出すと、tpenvelope() は次のレポートを作成します。

- メッセージ・バッファに添付されたデジタル署名。デジタル署名は、以下のいずれかの状態になります。
  - TPSIGN_OK

    デジタル署名は検証されました。
  - TPSIGN_TAMPERED_MESSAGE

    メッセージ・バッファの内容が変更されたため、デジタル署名は無効です。
  - TPSIGN_TAMPERED_CERT

    署名者のデジタル証明書が変更されたため、デジタル署名は無効です。
  - TPSIGN_REVOKED_CERT

    署名者のデジタル証明書が取り消されたため、デジタル署名は無効です。
  - TPSIGN_POSTDATED

    タイムスタンプが遠い将来の時刻を示しているため、デジタル署名は無効です。

- TPSIGN_EXPIRED_CERT

  署名者のデジタル証明書の有効期限が切れたため、デジタル署名は無効です。

- TPSIGN_EXPIRED

  タイムスタンプが古すぎるため、デジタル署名は無効です。

- TPSIGN_UNKNOWN

  署名者のデジタル証明書が不明な認証局 (CA) によって発行されたため、デジタル署名は無効です。

■ メッセージ・バッファに添付された暗号化エンベロープ。暗号化エンベロープは、以下のいずれかの状態になります。

- TPSEAL_OK

  暗号化エンベロープは有効です。

- TPSEAL_TAMPERED_CERT

  ターゲット受信者のデジタル証明書が変更されたため、暗号化エンベロープは無効です。ターゲット受信者は、メッセージ・バッファを受信しません。

- TPSEAL_REVOKED_CERT

  ターゲット受信者のデジタル証明書が取り消されたため、暗号化エンベロープは無効です。ターゲット受信者は、メッセージ・バッファを受信しません。

- TPSEAL_EXPIRED_CERT

  ターゲット受信者のデジタル証明書の有効期限が切れたため、暗号化エンベロープは無効です。ターゲット受信者は、メッセージ・バッファを受信しません。

- TPSEAL_UNKNOWN

  ターゲット受信者のデジタル証明書が不明な認証局 (CA) から発行されたため、暗号化エンベロープは無効です。ターゲット受信者は、メッセージ・バッファを受信しません。

tpenvelope() は、ステータスの情報のほか、デジタル署名または暗号化エンベロープに関連付けられたキー・ハンドルも返します。プロセスは、キー・ハンドルを引数として tpkey_getinfo(3c) 関数を呼び出し、キー・ハンドルに関する詳しい情報を取得できます。

受信側のプロセスが、メッセージ・バッファを受信した後で tpsign() を呼び出してデジタル署名要求を登録した場合、tpenvelope() は登録のステータスを TPSIGN_PENDING としてレポートします。同様に、受信側のプロセスが、メッセージ・バッファを受信した後で tpseal() を呼び出して暗号化 (封印) 要求を登録した場合は、tpenvelope() は登録のステータスを TPSEAL_PENDING としてレポートします。

受信側のプロセスが署名付きメッセージ・バッファを受信した後でその内容を変更すると、添付された署名は無効になります。その結果、tpenvelope() は署名を検証できないため、TPSIGN_TAMPERED_MESSAGE という署名ステータスをレポートします。

コンポジット署名ステータスについて

メッセージ・バッファに複数のデジタル署名がある場合、公開鍵ソフトウェアは、tpenvelope() と同等の内部関数を呼び出して、各デジタル署名の状態を調べます。次に、特定の規則に従い、複数のデジタル署名の状態を合成した「コンポジット署名ステータス」を生成します。次の表は、コンポジット署名ステータスを生成する規則を示しています。

表 3-4 コンポジット署名ステータス

存在するステータス	存在しないステータス	結果のステータス
TPSIGN_TAMPERED_MESSAGE	...	TPSIGN_TAMPERED_MESSAGE
TPSIGN_TAMPERED_CERT	TPSIGN_TAMPERED_MESSAGE	TPSIGN_TAMPERED_CERT
TPSIGN_REVOKED_CERT	TPSIGN_TAMPERED_MESSAGE TPSIGN_TAMPERED_CERT	TPSIGN_REVOKED_CERT
TPSIGN_POSTDATED	TPSIGN_TAMPERED_MESSAGE TPSIGN_TAMPERED_CERT TPSIGN_REVOKED_CERT	TPSIGN_POSTDATED
TPSIGN_EXPIRED_CERT	TPSIGN_TAMPERED_MESSAGE TPSIGN_TAMPERED_CERT TPSIGN_REVOKED_CERT TPSIGN_POSTDATED	TPSIGN_EXPIRED_CERT

表 3-4 コンポジット署名ステータス (続き)

存在するステータス	存在しないステータス	結果のステータス
TPSIGN_OK	TPSIGN_TAMPERED_MESSAGE TPSIGN_TAMPERED_CERT TPSIGN_REVOKED_CERT TPSIGN_POSTDATED TPSIGN_EXPIRED_CERT	TPSIGN_OK
TPSIGN_EXPIRED	TPSIGN_TAMPERED_MESSAGE TPSIGN_TAMPERED_CERT TPSIGN_REVOKED_CERT TPSIGN_POSTDATED TPSIGN_EXPIRED_CERT TPSIGN_OK	TPSIGN_EXPIRED
TPSIGN_UNKNOWN	TPSIGN_TAMPERED_MESSAGE TPSIGN_TAMPERED_CERT TPSIGN_REVOKED_CERT TPSIGN_POSTDATED TPSIGN_EXPIRED_CERT TPSIGN_OK TPSIGN_EXPIRED	TPSIGN_UNKNOWN

TPSIGN_OK または TPSIGN_UNKNOWN のコンポジット署名ステータスが付いていないメッセージ・バッファは、受信されても破棄され、受信されなかったように扱われます。ATMI アプリケーションの UBBCONFIG ファイルの SIGNATURE_REQUIRED パラメータが Y (はい) に設定されている場合は、TPSIGN_OK のコンポジット署名ステータスが付いていないメッセージ・バッファを受信しても破棄され、受信されなかったように扱われます。詳細については、第 2 章の 43 ページ「受信メッセージに対する署名方針の適用」を参照してください。

ただし、前の段落で説明した署名付きメッセージ・バッファの処理の例外は、tpimport(3c) 関数です。tpimport(3c) 関数は、コンポジット署名ステータスとは関係なく、受信したメッセージ・バッファを送信します。

tpenvelope のコード例

次のコード例は、tpenvelope() を使用して、メッセージ・バッファに関連付けられているデジタル署名と暗号化の情報を調べる方法を示します。

リスト 3-14 tpenvelope を使用する例

```
main(argc, argv)
int argc;
char *argv[];
#endif
{
   TPKEY tu_key;
   TPKEY sdo_key;
   TPKEY output_key;
   char *sendbuf, *rcvbuf;
   int ret;
   int occurrence = 0;
   long status;
   char principal_name[PNAME_LEN];
   long pname_len = PNAME_LEN;
   int found = 0;
  .
  .
  .
   output_key = NULL;
   ret = tpenvelope(rcvbuf, 0, occurrence, &output_key,
      &status, NULL, 0);
   while (ret != -1) {
      if (status == TPSIGN_OK) {
         if (tpkey_getinfo(output_key, "PRINCIPAL",
            principal_name, &pname_len, 0) == -1) {
            (void) fprintf(stdout, "Unable to get information
               about principal:%d(%s)\n",
                  tperrno, tpstrerror(tperrno));
            tpfree(sendbuf);
            tpfree(rcvbuf);
            tpterm();
            (void) tpkey_close(tu_key, 0);
```

```
            (void) tpkey_close(sdo_key, 0);
            (void) tpkey_close(output_key, 0);
            exit(1);
         }
         /* リソースを必ず解放する */
         (void) tpkey_close(output_key, 0);
         output_key = NULL;
         found = 1;
         break;
      }
      /* リソースの解放を忘れないこと */
      (void) tpkey_close(output_key, 0);
      output_key = NULL;
      occurrence++;
      ret = tpenvelope(rcvbuf, 0, occurrence, &output_key,
         &status, NULL, 0);
   }
  .
  .
  .
}
```

関連項目

- 第 3 章の 63 ページ「型付きメッセージ・バッファの外部化」
- 第 1 章の 29 ページ「公開鍵によるセキュリティ機能」
- 第 2 章の 40 ページ「公開鍵セキュリティの管理」
- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」

型付きメッセージ・バッファの外部化

外部化された表現とは、通常はバッファが送信される直前にメッセージ・バッファに追加される ATMI のヘッダ情報が含まれないメッセージ・バッファのことです。署名付きメッセージ・バッファを外部化された表現に変換すると、署名付きデータを「パス・スルー (通過)」させたり、署名付きバッファを長期間保存しておき、バッファ送信した事実を否認できないようにすることができます。また、復号化キーを使用しないで、暗号化されたメッセージ・バッファを中継プロセス経由で伝送することもできます。

外部化された表現の作成方法

ATMI プロセスは、tpexport(3c) 関数を呼び出して、型付きメッセージ・バッファを外部化された表現に変換します。メッセージ・バッファに関連付けられた保留状態の署名は、そのメッセージ・バッファが ATMI 関数によって別のプロセスに送信された場合のように、tpexport() が呼び出されたときに生成されます。同様に、メッセージ・バッファに関連付けられた保留中の封印は、そのメッセージ・バッファが ATMI 通信関数によって別のプロセスに送信された場合のように、tpexport() が呼び出されたときに生成されます。

外部化された表現のメッセージ・バッファは、バイナリ形式の PKCS-7 形式で保存されます。文字列で指定する必要がある場合、呼び出しプロセスは、TPEX_STRING フラグを指定して tpexport() を呼び出す必要があります。

注記 外部化された表現の型付きメッセージ・バッファを作成する機能は、公開鍵セキュリティに固有のものではありません。プロセスは、tpexport() を呼び出して型付きメッセージ・バッファを外部化することができます。メッセージ・バッファに対してデジタル署名または暗号化のマークが付けられているかどうかは無関係です。

外部化された表現の変換方法

受信側のプロセスは、tpimport(3c) 関数を呼び出して、外部化された表現のメッセージ・バッファを型付きメッセージ・バッファに変換します。tpimport() 関数も、必要に応じて復号化を行い、関連するデジタル署名があれば検証します。

tpexport および tpimport のコード例

次のコード例は、tpexport() を使用して型付きメッセージ・バッファを外部化された表現に変換する方法、および tpimport() を使用して外部化された表現を型付きメッセージ・バッファに戻す方法を示します。

リスト 3-15 tpexport および tpimport を使用する例

```
static void hexdump _((unsigned char *, long));

#define MAX_BUFFER 80000

main(argc, argv)
int argc;
char *argv[];
#endif

{
   char  *databuf;
   char  exportbuf[MAX_BUFFER];
   long  exportbuf_size = 0;
   char  *importbuf = NULL;
   long  importbuf_size = 0;
   int go_on = 1;
  .
  .
  .
   exportbuf_size = 0;
   while (go_on == 1) {
      if (tpexport(databuf, 0, exportbuf, &exportbuf_size, 0)
         == -1) {
         if (tperrno == TPELIMIT) {
            printf("%d tperrno is TPELIMIT, exportbuf_size=%ld\n",
               __LINE__, exportbuf_size);
            if (exportbuf_size > MAX_BUFFER) {
               return(1);
            }
         }
         else {
            printf("tpexport(%d) failed:tperrno=%d(%s)\n",
               __LINE__, tperrno, tpstrerror(tperrno));
            return(1);
         }
      }
      else {
         go_on = 0;
      }
```

```
    }
   .
   .
   .
    hexdump((unsigned char *) exportbuf, (long) exportbuf_size);
    if (tpimport(exportbuf, exportbuf_size, &importbuf,
       &importbuf_size, 0) == -1) {
       printf("tpimport(%d) failed:tperrno=%d(%s)\n",
          __LINE__, tperrno, tpstrerror(tperrno));
       return(1);
    }
   .
   .
   .
}
```

関連項目

- 第 1 章の 29 ページ「公開鍵によるセキュリティ機能」
- 第 2 章の 40 ページ「公開鍵セキュリティの管理」
- 第 3 章の 3 ページ「セキュリティを備えた ATMI アプリケーションのプログラミング」