ネットワークセキュリティ ——分散オブジェクト通信環境へのSpiral Hellの適用

Network Security ——Applying the Spiral Hell to ORB（Object Request Broker）Environment

八津川 直伸

要 約 日本ユニシスが考案したエンティティ認証方式である「無限ワンタイム認証方式」（以降これをSpiral Hellと呼ぶ）を，分散オブジェクト通信機構（ORB：Object Request Broker）上のアプリケーション相互間（オブジェクト相互間）のエンティティ認証に適用し，分散環境下での安全性，パフォーマンスおよび実現性などの観点からその実用性を評価，検証した．毎回異なる一回限り有効な認証情報が無限に生成可能で，しかも外部の第三者のみならず認証サーバ側の内部管理者による「なりすまし」をも防御し，さらに処理シーケンスが極めて簡単なSpiral Hellは，真に重要機密通信時の相互エンティティ認証方式としてORB環境上においても有効であり，堅牢なエンティティ認証の要件を満たす．

Abstract This paper describes the evaluation and verification of the usability of the "Endless Onetime Authentication System"(called Spiral Hell in general) which is applied to the entity authentication among applications in the ORB environment from the viewpoint of the reliability, performance, and feasibility. Even though the Spiral Hell uses a very simple authentication sequence, it available to generate an infinite of different authentication tokens and prevent masquerade attacks by not only outside personnel but the internal personnel such as the sever-site administrator. Therefore, the Spiral Hell is still effective and satisfies the requirement for the robust entity authentication during the secret communications between applications on the ORB environment.

1. はじめに

本稿では，日本ユニシスが考案したエンティティ認証方式である「無限ワンタイム認証方式」[1]（以降これをSpiral Hellと呼ぶ）の実装について述べる．既存方式の問題点を解決し，仕組みが簡単で十分な認証強度を持つ本方式は，インターネット時代においてその適用領域が多数考えられるが，今回は，分散オブジェクト通信機構（ORB：Object Request Broker）上のアプリケーション相互間（オブジェクト相互間）のエンティティ認証に適用し，安全性，パフォーマンスおよび実現性などの観点からその実用性を評価，検証した．

2. 認証形態

まず，Spiral Hellの設計および実装にあたりエンティティ認証の形態にはどのようなものがあるか考えてみる．そのために，エンティティ認証の機能本質を整理してみると，以下のようになる．

1） エンティティ認証の目的

情報通信に関与した実体（エンティティ：人間，人間の代理として機能するプ

ロセス，ソフトウェア，ハードウェアあるいはメッセージなど）が正当なものであるか否かを確認することである．なお，確認後の処理はシステム（セキュリティポリシー）に依存する．

より厳密には，

・認証とは誰かが，お墨付き（信用）を何らかの方法で被認証実体に付与し，
　（注）必ずしも認証者がお墨付きを与えるとは限らない
・認証者がそのお墨付きを検証，およびそれによって保証された認証情報を検証すること．
　（例：免許証，クレジットカード）

と言える．

例えば免許証の場合は，免許証の顔写真に行政機関がお墨付き（信用）を付与し，認証情報は本人の顔そのものである．認証処理すなわち本人(エンティティ)の正当性確認は，認証者が免許証（お墨付き）をまず検証（本物か偽物かを確認）し，それによって保証された顔写真と本人の顔（認証情報）を比較照合することにより達成される．

2)　エンティティ認証の安全性

お墨付きを与えた機関の信用度と，第三者が認証情報を知ったり偽造したりすることの困難性に依存する．従って，お墨付きを与えた第三者が自社以外の民間会社である場合，その会社が不正行為を行ったり，破産することもあるので，後述の他律認証系の場合注意が必要である．

このような機能本質をもつエンティティ認証の実システムへの適用に際しては，どのような形式を採用するかが肝要であり，それによって実装方法が異なってくる．大きくは次の二つの形式を考慮する必要がある．

・認証システムの外形
・認証システムの信用系

2.1　認証システムの外形

本稿では，認証システムの外形として次の 3 種類を定義する．

1)　ローカル認証

認証者と被認証者が近接（対面）している．

（例）端末による利用者認証，入退室認証など

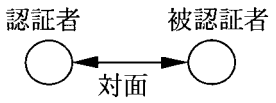


図 1　ローカル認証

2)　分散型リモート認証

各目的サーバ（認証者）が独自に遠隔のクライアント（被認証者）認証を行う．

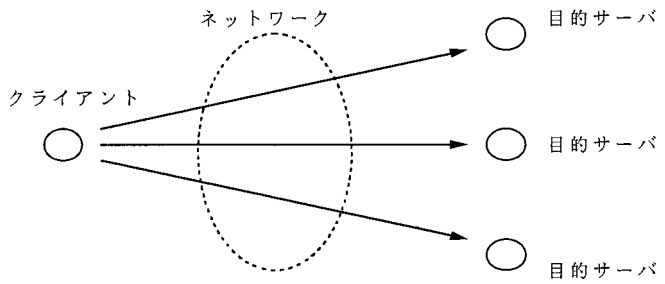


図 2　分散型リモート認証

3）　集中型リモート認証

各目的サーバは集中認証サーバ（認証者）と信頼関係にあり，クライアント（被認証者）の認証処理はこの集中認証サーバが一括して行う．

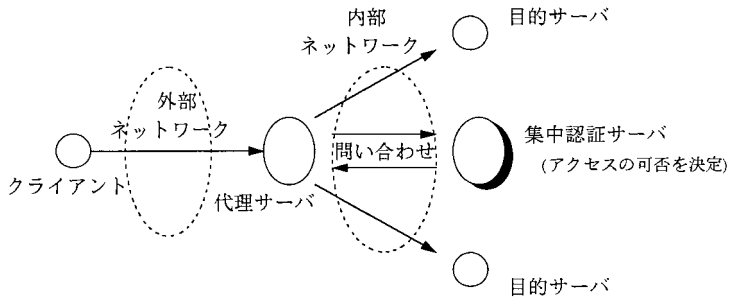


図 3　集中型リモート認証

各クライアントのアクセスプロトコルに対応した代理サーバが一旦コネクションのエンドポイントとなり，クライアントからの認証要求に応じて適当な集中型認証サーバにクライアントの正当性を問い合わせる．もし認証が成功すると，各クライアントはあたかも内部ネットワークに直接接続しているかの状態となり，それらは内部ネットワーク上の目的サーバにアクセス可能となる．トラフィックは全て代理サーバ経由となる．

この代理サーバと集中認証サーバの間のシーケンスは方式独自のプロトコルとなるが，デファクトとして RADIUS（Remote Authentication Dial In User Service）（RFC-2138）や TACACS（Terminal Access Controller Access Control System）（RFC-1492）などが挙げられる．この場合代理サーバと集中認証サーバは RADIUS または TACACS などに対応していなければならない．

2.2　認証システムの信用系

本稿では，認証システムの信用系として被認証実体に信用を付与する方法論により，次の二種類を定義する．

・自律認証系
・他律認証系

1)　自律認証系

認証者が自ら被認証者に信用を付与する．主に社内システムなどの特定者間の通信に適用される．具体的には認証サーバが自ら各クライアントの実在とその正当性を確認した上で利用者名（User-id など）と認証情報（Password など）を発行・登録する．従って，自律認証系の場合，第三者機関による信用付与（公開鍵証明書の発行など）は特に必要としない．

特定のグループ内に限り信用付与を一括して行う社内認証局（この場合も第三者が信用付与する訳ではないので自律認証系に含まれる）も考えられるが，CRL（Certificate Revocation List：廃止リスト）管理の手間やコストおよび期限切れの公開鍵証明書が流通することによるセキュリティ通信の信頼性低下などの問題を考えると，各認証サーバがユーザ登録，変更をこまめに行う方が安全である．

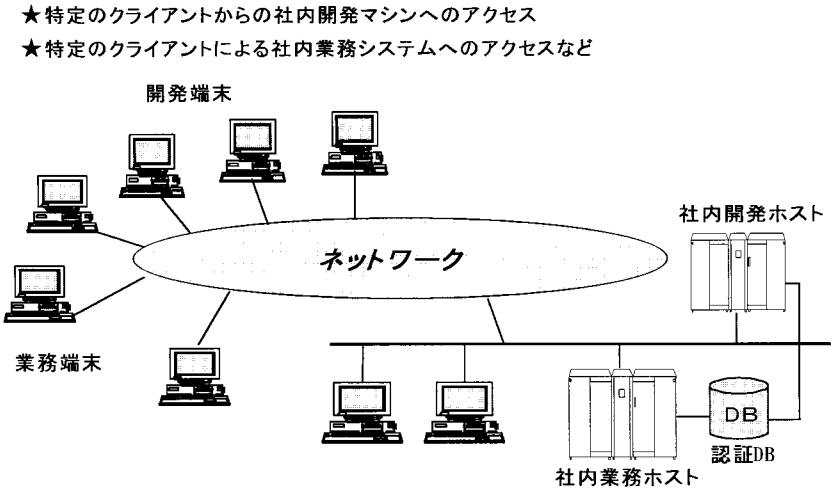


図 4　自律認証系のシステム例

2)　他律認証系

認証者が特定の第三者を信用し，その第三者が認証者に代わって被認証者に信用を付与する．通信当事者同士がお互いの素性を知らないという場合のような，不特定多数者間の通信に適用される．従って，相手認証は公開鍵証明書によって保証される認証情報のデジタル署名の検証によって成されることになるが，この場合，当事者以外の第三者が認証情報を盗聴しそれをそのまま再利用することによって正当な当事者になりすます（以降これを「リプレイ攻撃」と呼ぶ）などの不正アクセスが可能となる．なぜなら，デジタル署名の盗聴によるコピーはやはり本物であるからである．

他律認証系の典型的な例としては，インターネット環境を想定したセキュア E-mail（PEM : Privacy Enhanced Mail や S/MIME : Secure/Multipurpose Internet Mail Extensions）あるいは電子商取引プロトコル（SET : Secure Electronic Transaction）などが挙げられる．これらはディジタル署名と第三者認証局が発

行した公開鍵証明書によってメッセージ認証を行い，それによって通信相手が正当であることを推定しエンティティ認証を代替している．

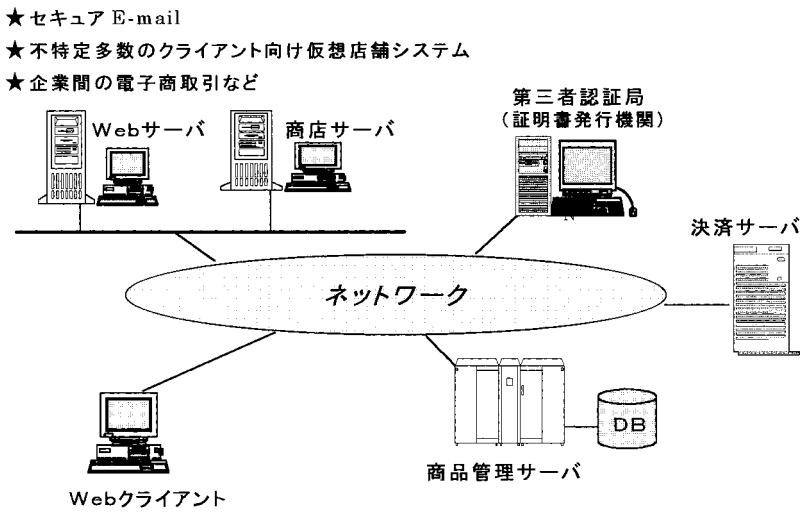


図 5　他律認証系のシステム例

3.　無限ワンタイム認証方式の概要

本章では，エンティティ認証に関する Common　Criteria（以降 C.C.と呼ぶ）[2]の要件および既存方式の問題点を鑑みて，理想的なエンティティ認証の要件を列挙し，その要件を満たすように設計された Spiral Hell の方式について述べる．

3.1　C.C.（Common Criteria）の要件

C.C. はセキュリティ評価基準の国際標準で，現在第 1 版が出ている．この C.C. がシステムに要求している利用者認証機能には次のような項目（抜粋）がある．なお，（ ）内は C.C. の項目識別番号である．詳細は文献[2]を参照いただきたい．

1）　システムは，利用者がシステムを利用するに先立って利用者の認証を実施しなければならない．

（FIA_UAU.1.1）

2）　システムは，認証データの不正な再利用を防止できなければならない．

（FIA_UAU.2.2）

3）　システムは，偽造された認証データを検知し，その利用を防止できなければならない．

（FIA_UAU.3.2）

4）　システムは，コピーされた認証データを検知し，その利用を防止できなければならない．

（FIA_UAU.3.3）

5）　システムは，必要に応じて利用者の再認証ができなければならない．

（FIA_UAU.7.1）

6) システムは，不正な参照，改ざんまたは破壊から，システムで保管している認証情報を保護できなければならない．
（FIA_ADP.1.1）

7) システムは，生の認証情報がシステムにある間はいかなる時も，認証情報の不正参照，改ざんまたは破壊から保護できなければならない．（FIA_ADP.2.1）

上記のなかで 4)，6)，7）は既存方式では実現が困難である．

3.2 既存システムの問題点

既存システムのほとんどは，エンティティを認証するために必要な情報を認証者側に予め登録しておき，認証されるべきエンティティがその情報を知っているか否かでそのエンティティの正当性を確認する方式である．しかしながら，以下のように種々問題点を抱えており，また C.C. の要件が満足できないものがほとんどである．詳細は文献[1]を参照いただきたい．

特に，システム管理者などのシステムに精通した人間の悪意の内部犯罪による脅威に対しては防御できないものがほとんどであり，たとえ防御できる方式であっても認証手続きが複雑になるなどの欠点がある．

以下に既存システムの問題点について整理を行った．

1) パスワード方式
- 覚え易い文字列を使用したり，人目に付き易いところにメモしがちで容易に盗まれる．
- 通信中に盗聴される．
- 辞書攻撃によってパスワードファイルが破られる可能性があり，内部犯罪も起き易い．
- C.C. の FIA_UAU.2.2，FIA_UAU.3.2，FIA_UAU.3.3，FIA_ADP.1.1，FIA_ADP.2.1 を満足できない．

2) ワンタイムパスワード方式，チャレンジレスポンス方式
- 認証者側が認証情報を生成するための秘密の情報を持っているので，悪意の管理者による不正行為（内部犯罪）が可能．
- C.C. の FIA_ADP.1.1，FIA_ADP.2.1 を満足できない．

3) 暗号利用方式（公開鍵暗号，共通鍵暗号）
- リプレイ攻撃により第三者による「なりすまし」が可能．
（注）デジタル署名は偽造できないが，だれでも複製が可能．
- C.C. の FIA_UAU.2.2，FIA_UAU.3.3，FIA_ADP.1.1，FIA_ADP.2.1 を満足できない．

4) ゼロ知識対話証明方式
- 対話シーケンスが冗長である．
- 認証プロセスが複雑であり，パフォーマンスと認証精度がトレードオフの関係となる．

5) 生体特徴利用方式（バイオメトリック：声紋，指紋，アイパターンなど）
- 認識確度が 100% ではなく，技術的な改善の余地がある．
- ネットワークを介した認証（リモート認証）ではリプレイ攻撃が可能．

・高価な生体特徴計測機器が必要.

・C.C. の FIA_UAU.2.2，FIA_UAU.3.3，FIA_ADP.1.1，FIA_ADP.2.1 を満足できない.

6) 所有物利用方式（キャッシュカード，PC カード，IC カードなど）

・ネットワークを介した認証（リモート認証）ではリプレイ攻撃が可能.

・安全維持の観点から強固な耐タンパー性（不正に情報を読み出したり，書き換えたりするのが困難な性質）を付与しているような IC カードなどでは，バッテリーの交換も不可能であり，そのためバッテリーが切れると新しい IC カードの調達と認証サーバへの再登録処理が必要といった運用が面倒なものもある.

・C.C. の FIA_UAU.2.2，FIA_UAU.3.3，FIA_ADP.1.1，FIA_ADP.2.1 を満足できない.

3.3 新認証方式の要件

前述した C.C. の要件を満足し，既存方式の問題点を払拭し，さらに仕組みが簡単で堅牢なインターネット時代に対応したエンティティ認証方式の要件として表 1 があげられる.

なお，Spiral Hell はこれらを満足するように設計されている.

表 1 新認証方式の要件

項番	要件	目的
1	盗聴などによって盗まれた認証情報が通信当事者以外の第三者によって再利用できないこと.	リプレイ攻撃を防ぐ
2	項番 1 を満たす認証情報がいつでも簡単に生成できること.	運用性の向上
3	認証サーバに認証情報が保存されていないこと.	内部犯罪を防ぐ
4	認証シーケンスは極力簡単であること.	相互接続性，パフォーマンス向上
5	認証情報は毎回異なり，しかもその情報は無限に存在すること.	運用性の向上およびリプレイ攻撃の防止
6	生体特徴利用のような特殊外部測定機器を必要としないこと.	低コスト化，インターオペラビリティの向上
7	自律認証系，他律認証系のどちらにも適用できる事.	可用性の向上
8	ローカル認証，リモート認証のどちらにも適用できること.	可用性の向上

3.4 Spiral Hell の方式

本節では，Spiral Hell の方式について述べるが，説明の簡単化のために，被認証者をクライアント，認証者をサーバと呼ぶことにする.

前提として，この方式は公開鍵暗号アルゴリズムを使用し，クライアントは自身の秘密鍵を，またサーバはそれに対応する公開鍵を保持しているものとする.

本方式では，ログインに先立って，認証処理に必要な初期情報 D_0 をサーバに登録

する必要がある．この登録は，クライアント毎に最初に一回だけ行えば良く，一旦登録すれば，その後この作業は不要である．この初期情報 D_0 は，乱数やクライアントの E-mail アドレスや利用者識別名などなんでもよいが，クライアントにおいては次回ログイン時の認証情報を生成するための元となるデータ（以降，認証情報生成種データと呼ぶ）であり，サーバにおいては次回ログイン時にクライアントから受信した認証情報を検査するためのデータ（以降，認証情報検査データと呼ぶ）となる．いずれにせよこのような利用者登録作業は一般にクライアントのアクセス権限の設定などを伴うので相応の権限を持ったシステム管理者が行う．初期情報登録後は，その旨クライアントに通知する．

以上の初期登録作業後，クライアントがサーバにログインする際のサーバによるクライアント認証処理は次のようになる．処理概要を図 6 に示す．

① クライアントがサーバにログインする際，クライアントは先ず利用者識別名（User-id など）をサーバに送信することにより，認証要求を行う．

② 認証要求メッセージを受け取ったサーバは，クライアントに対して認証情報要求メッセージを送る．

③ 認証情報要求メッセージを受け取ったクライアントは，サーバに返すべき認証情報として，自身が保存している認証情報生成種データを自身の秘密鍵で暗号化したものを送る．一回目のログイン時においては，認証情報生成種データは初期情報 D_0 であり，認証情報は D_1 である．

④ サーバは，クライアントから受信した認証情報をクライアントの公開鍵で復号する．

⑤ 次にサーバは，それを利用者毎に保存してあった認証情報検査データと比較する．一回目のログイン時においては，認証情報検査データは初期情報 D_0 である．

⑥ ⑤の比較結果が等しければログインを許可し，クライアントの認証情報検査データ格納場所に今受信した認証情報を次回ログイン時の検査用として上書き保存する．一方，不一致ならログインを拒否する．サーバは，許可あるいは拒否いずれの場合においてもその旨をクライアントに通知する．

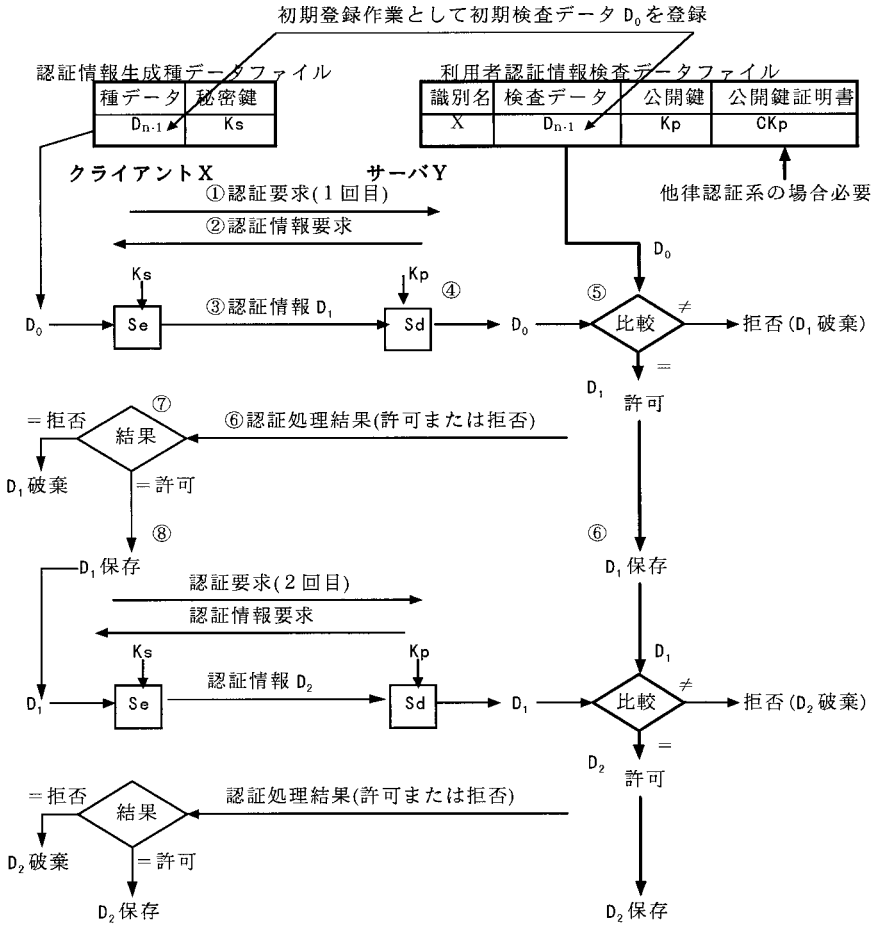
⑦ クライアントは，サーバから認証処理結果の通知を受けたなら，ログインが許可されたか否かを確かめる．

⑧ もし，認証結果が許可であれば，認証情報生成種データ格納場所に，③で送った認証情報を次回のログイン時の認証情報生成種データとして上書き保存する．

次回，クライアントが再ログインするときは，今保存した認証情報生成種データをクライアントの秘密鍵で再度暗号化し，それを新しい認証情報としてサーバに送る．二回目のログイン時においては，認証情報生成種データは D_1 であり，認証情報は D_2 となる．

⑨ 以降，①から⑧を繰り返す．

なお，利用者識別情報と認証情報を結合し一つのメッセージとして③をサーバに送ってもよい．その場合，①と②が省略できる．今回の実装ではこの①と②を省略して



各記号の意味は次のとおりである.

CKp：CAによって発行されたクライアントXの公開鍵証明書

Ks ：クライアントXの秘密鍵

Kp ：クライアントXの公開鍵

Se ：公開鍵暗号アルゴリズムの暗号化関数

Sd ：公開鍵暗号アルゴリズムの復号化関数

D_n ：n回目のログイン時においてクライアントXがサーバYに送信する認証情報

D_{n-1}：n回目ログイン時のクライアントXにおける認証情報生成種データ，サーバにおける認証情報検査データ

(注1)

D_0：初期登録値

D_n：D_{n-1} をKsで暗号化したもの（但し，$\infty > n > 1$）である.

(注2)

他律認証系で毎回，認証情報と共にクライアントXの公開鍵証明書をサーバに送るシステム仕様であれば，利用者認証情報検査データファイルにKpおよびCKpを保存する必要はない.

図 6　無限ワンタイム認証方式の処理概要

いる．また，CA（Certification Authority）の運用が行われている環境すなわち他律認証系であれば，③で認証情報と共にクライアントの公開鍵証明書をサーバに送ることにより，クライアントの公開鍵をサーバに配送してもよい．

以上述べたように，Spiral Hell の方式自体はシステムの環境や構成に応じてローカル，リモートおよび自律，他律認証系のいずれにも柔軟に適用可能である．

4. 分散オブジェクト通信環境への実装仕様

本章では，Spiral Hell（無限ワンタイム認証方式）の分散オブジェクト通信環境（以降 ORB（Object Request Broker）と略記する）上のアプリケーションへの実装仕様について述べる．

4.1 アプリケーションレベルにおけるセキュリティ機能の必要性

今後，電子商取引が拡大し，それらの当事者である利用者（エンティティ）間の信用維持が必須であることに鑑み，諸般の環境要素を考慮するとアプリケーションレベルにおけるセキュリティ機能の必要性として次のことが挙げられる．

1） ORB レベルのセキュリティサービスにおいても認証機能を提供するものがあるが，それはあくまでも ORB 機能層での認証である．システムの環境や構成によってはこれでも十分実用的であるが，機密性の高いアプリケーションがエンドツーエンドで認証を要求する場合は，やはりアプリケーションの責任において該セキュリティ機能を全うせざるを得ない．

2） 同じ CORBA であっても製品によってはセキュリティサービスを提供しないものもあり，たとえ提供しているものであっても CORBA のセキュリティ仕様[3]に定める準拠レベル“ Interoperability ”を満たさない製品はセキュリティ実装仕様がそれぞれ異なるため，それらを相互接続したシステム構成の場合，エンドツーエンドのセキュリティが維持できず，必然的にアプリケーションレベルのセキュリティ機能に依存せざるを得ない．

3） CORBA 環境とそれ以外の分散環境（DCOM や RPC 環境など）やメインフレームなどのレガシーシステム環境と相互接続する場合（図 7）は，ORB レベルでのセキュリティ維持が全く不可能であり，必然的にアプリケーション間でのセキュリティ機能が要求される．

4.2 セキュリティ機能コンポーネント概要

ORB 環境上のアプリケーションにセキュリティ機能を提供する方法として，次の二つが考えられる．本章では，以降アプリケーションを AP と略記する．

- ライブラリ形式（アプリケーションにセキュリティ機能モジュールがライブラリとして組み込まれ，各アプリケーションは必要に応じ該ライブラリを呼び出す）
- オブジェクト形式（セキュリティ機能モジュールは，ORB 上に一つのオブジェクトとして存在し，各アプリケーションは必要に応じ該オブジェクトを遠隔呼出しする）

各々の構成イメージを図 8 および図 9 に示す．

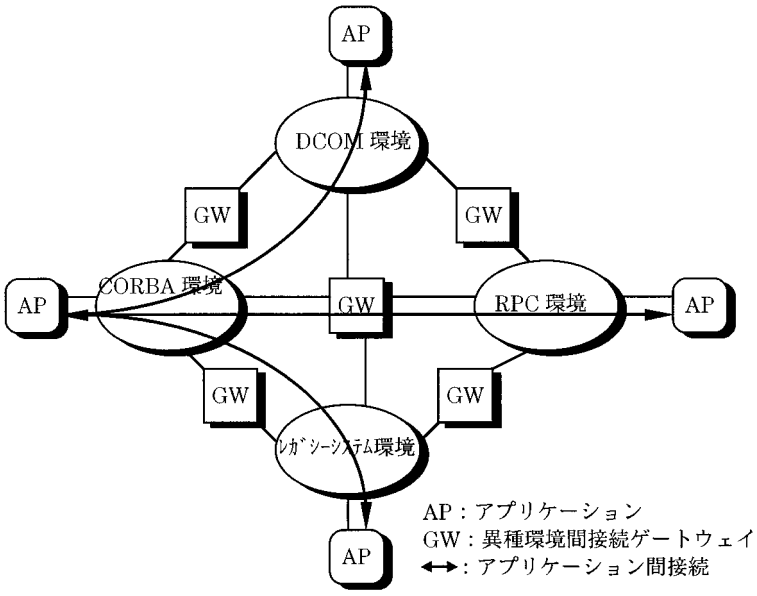


図 7　異機種環境を経由したアプリケーション間の接続

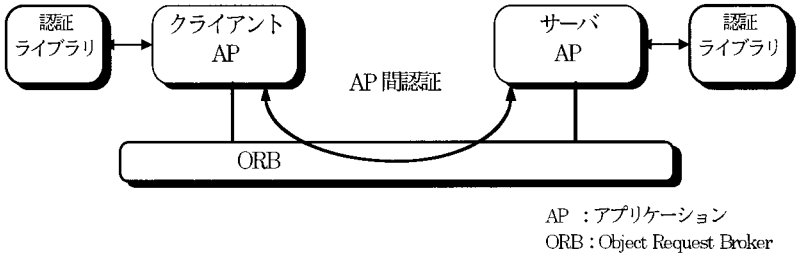


図 8　ライブラリ形式のセキュリティ機能

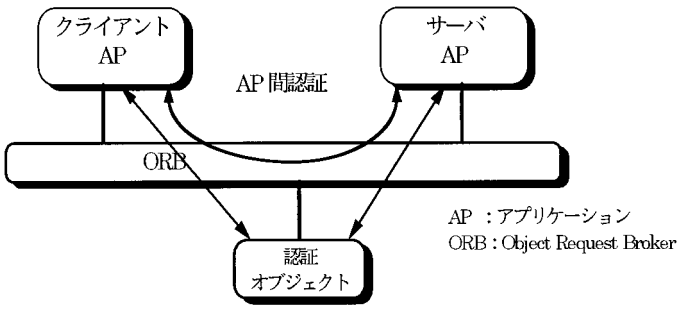


図 9　オブジェクト形式のセキュリティ機能

次に今回実装するセキュリティ機能コンポーネントの基本構成を図 10 に示す．この中で　セキュリティ管理モジュールは，利用者情報の登録・変更・削除や利用者の鍵生成・削除など，主に利用者との対話処理に用いられる．また，認証モジュールは ORB 上に存在する複数のアプリケーションオブジェクト相互間で Spiral　Hell による

認証が必要なときに API（Application Program Interface）を介して用いられる．

これらモジュールをライブラリ形式で使用するか，あるいはオブジェクト形式で使用するかはシステム全体の設計に依存する．

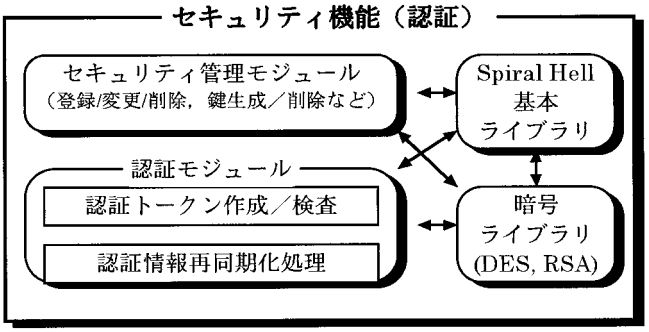


図 10　セキュリティ機能コンポーネントの基本構成

4.3　今回の実装における認証システムの形態

今回の実装では，認証システムの外形として「分散型リモート認証」を，また信用系としては「自律認証系」を前提として設計した．

4.4　実　装　概　要

4.4.1　Spiral Hell のエンティティ認証機能

図 11 に認証動作の概要を示す．自律認証系が前提であるので認証者側（サーバ）には既に利用者（クライアント）の情報が事前登録されている．

① ORB 上の各 AP（エンティティ）は何らかの手段により起動権限チェックを経て起動され，その際，やはり何らかの手段により起動した利用者の識別子（以降 User-id と記述する）が引き渡される．

② クライアント AP は，利用者識別子とアクセス先のサーバ識別子（以降 Server-id と記述する）をもって自ホストの認証ライブラリあるいは ORB 上の認証オブジェクト（以降，双方を合わせて認証モジュールと呼ぶ）に認証子作成要求を出す．

③ 認証モジュールはその User-id が登録済みか否かを確認し，もし登録済みであれば認証子（Token）を生成しそれをクライアント AP に返す．

④ クライアント AP は，認証子をアクセス先のサーバ AP に送信する．

⑤ 認証子の着信とともにサーバ AP は起動（Invoke）される．

⑥ サーバ AP は, 受信した認証子と共にその検査要求を認証モジュールに渡す．

⑦ 認証子検査要求を受けた認証モジュールは，User-id をキーとして保存してあった認証情報検査データをもとに今受けた認証子を検査し，検査結果をサーバ AP に返す．

⑧ サーバ AP は認証結果をクライアント AP に返すとともに，もし検査結果が NG であればサーバ AP を終了する（それ以降の処理を行わず起動待ちの状態に戻る）．

⑨　認証結果を受けたクライアント AP は，その結果に基づき予め定められた処理を行う．例えば，認証結果が OK であればそのままアプリケーションの実行を継続し，もし，NG であれば利用者にその旨を通知しその後の指示待ちの状態に移行するなどである．

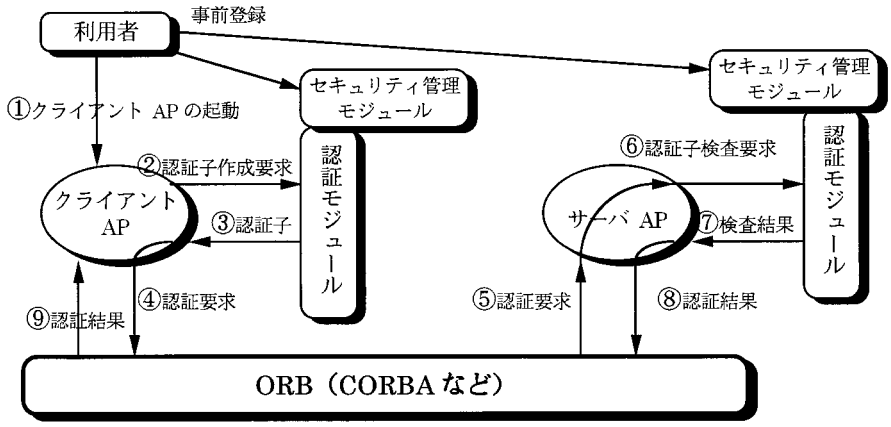


図 11　認証動作の概要

4.4.2　セキュリティ管理機能

ここでは，セキュリティ管理機能について述べる．

1）　利用者の鍵生成と基本登録

①　管理機能を使用するにあたっての管理機能利用者（管理者）/パスワード登録

②　利用者の秘密鍵，公開鍵の生成機能

RSA アルゴリズムによる秘密鍵と公開鍵を生成する．なお，同一 User-id につき複数の鍵を持つ事が可能で，その場合，鍵識別子により個々の鍵を識別する．4.5 節で述べる Spiral　Hell の各 API においても鍵識別子を引数として設定する．

③　認証機能を使用する利用者（User-id）の基本登録

全ての利用者は利用者自身について以下を登録する．

・利用者 User-id
・利用者 User-id が所有する秘密鍵（Ks-user）
・利用者 User-id の公開鍵（Kp-user）
・*利用者 User-id の公開鍵証明書（CKp-user）
・*CA の公開鍵（Kp-cert）
・*CA の公開鍵証明書（CKp-cert）

（注 1）＊印は他律認証系の場合に登録

（注 2）以降において前述の User-id は，認証者となる時 Server-id と呼び，被認証者となるとき Client-id と呼ぶ．

2）　認証者側としての登録

自分（User-id）が認証者（Server-id）の場合にアクセスしてくる被認証者（Client-

id）個々について以下を登録する．

・被認証者 Client-id（認証すべき利用者 id）

・認証情報検査データ（初期値：D_0）

・被認証者 Client-id の公開鍵（Kp-clnt）

・*被認証者 Client-id の公開鍵証明書（CKp-clnt）

（注）＊印は他律認証系の場合に登録

3) 被認証者としての登録

利用者（User-id）が被認証者（Client-id）の場合にアクセスするサーバ（Server-id）毎に以下を登録する．

・認証者 Server-id（リモートの認証者 id）

・認証情報生成種データ（初期値：D_0）

4) 登録状況表示機能

各種登録状況を検索，編集，表示する機能である．

5) セキュリティ監査機能

セキュリティ機能に関する各種イベントのログ採取ならびに保存・編集・表示機能である．

本実装ではイベントのログ表示機能のみ実装し，保存・編集機能は未実施である．

6) 認証情報再同期化機能

認証者と被認証者間で認証情報生成種データと認証情報検査データの同期が外れた場合に再同期をとるための機能である．例えば，サーバ AP が受信した認証子の検査結果が OK で検査データを更新し，そして検査結果をクライアント AP に返す直前でシステムダウンした場合などは，クライアントの種データとサーバの検査データが異なってしまうので，サーバがリカバリした後クライアントからの認証要求は全て NG となる．実システムにおいては，このような同期はずれに対処するための仕組みが必要となる．対処方法は種々考えられるが本実装ではその一つとして，種データと検査データの再同期化がオンラインで迅速に行えるような方法を考案した．図 12 にクライアント主動時の再同期化処理の概要を示す．

なお，再同期化要求子のデータフォーマットは図 13 のとおりである．

再同期化が必要な時，被認証者（クライアント）はこの要求子を作成し，さらにそれを自身の秘密鍵で暗号化したもの（以降これを再同期化要求子の電子署名と呼ぶ)を作成する．次にこの再同期化要求子の電子署名と Username および Hostname の平文を組み合わせたものを再同期化要求メッセージ（図 14）として認証者（サーバ）に送信する．

この再同期化要求メッセージを受信した認証者は，それに含まれる再同期化要求子の電子署名を被認証者の公開鍵で復号化することにより再同期化要求子を得，次の手順で再同期化要求子の正当性を検査する．

① 再同期化要求子のフィールド文字列が“ REINIT ”であるか？

② 再同期化要求子の Username，Hostname フィールドの内容が，再同期化要求メッセージ中の平文で送られた Username，Hostname フィールドの内容と

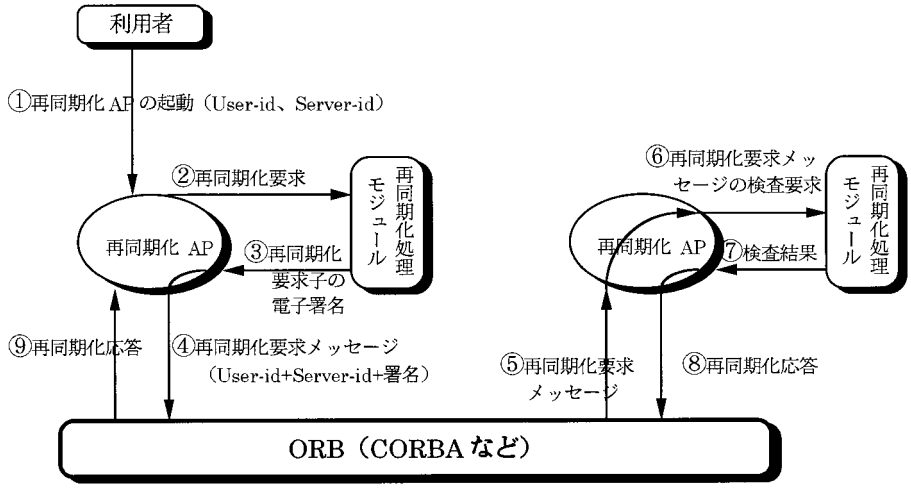


図 12　認証情報再同期化処理

10byte	8byte	8byte	12byte	26byte
Command	Username	Hostname	TimeStamp	Random

Command　：REINIT（固定文字列）　TimeStamp　：再同期化要求時刻
Username　：Client-id　Random　：多桁乱数
Hostname　：Server-id

図 13　再同期化要求子のデータフォーマット

Username(平文)	Hostname(平文)	再同期化要求子の電子署名(図 13を秘密鍵で暗号化したもの)

図 14　再同期化要求メッセージのデータフォーマット

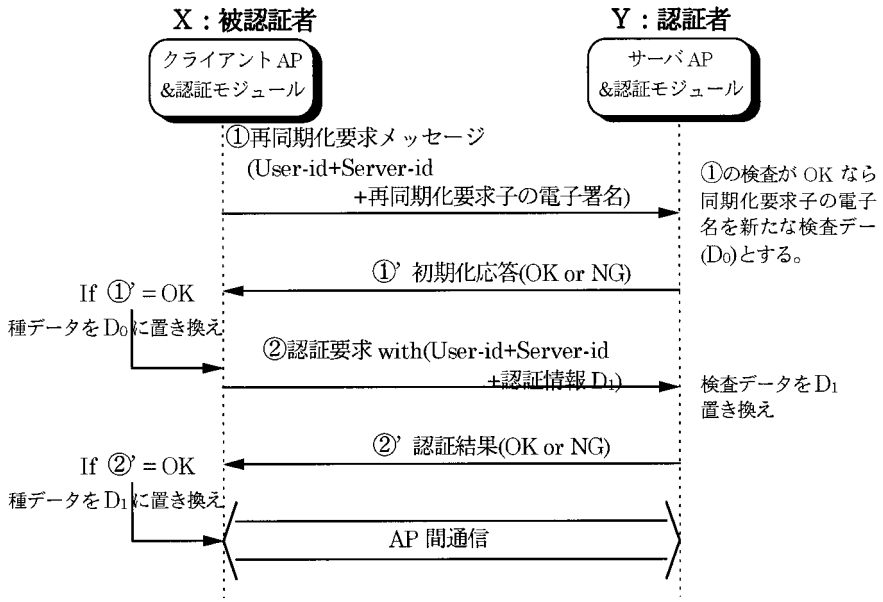
それぞれ同一であるか？

③　再同期化要求子の Timestamp フィールドの時刻と現在時刻との差が一定時間内であるか？（今回の実装では最大許容時間を 60 秒とした）

①から③の検査が全て YES であった場合，認証者はこの再同期化要求メッセージを真正なものと看做し，再同期化要求子の電子署名を被認証者に対する新たな検査データとして保存し，再同期化処理結果を再同期化応答メッセージとして被認証者に返信する．この応答を受けた被認証者は結果を確認し，もし OK であれば再同期化要求子の電子署名を新たな種データとして保存する．

以上の再同期化処理はオンラインで行うため，再同期化要求メッセージの偽造，改ざんなどの脅威が考えられるが，被認証者の秘密鍵が厳密に保管されている限り再同期化要求子の電子署名は偽造不可能であるし，Command，Username および Hostname の各フィールド検査により通信途上の改ざんも検知可能である．また

Timestamp フィールドの検査により，リプレイ攻撃を防ぐことも可能である．但し，許容時間（今回の実装では 60 秒）内にリプレイ攻撃を受けた場合は防ぐことができない．この点については，C.C の要件（FIA_UAU.3.3）を満足していないが，通常稼働時においては再同期化処理の機会は極めて少ないため攻撃する側にとっても再同期化要求メッセージを狙うのは至難の業と考えられる．たとえこの攻撃に成功したとしてもその攻撃者が真の利用者になりすましてサーバにログインできる訳ではない．なぜなら再同期化要求子の電子署名は，クライアントにおいては次回ログイン時の認証情報を生成するための種データ（サーバにおいては検査データ）に過ぎず，被認証者の秘密鍵を知らない限り攻撃者は依然として次回ログイン時の認証情報を生成し得ないからである．従って不正アクセスなどの実害は有り得ないが，この攻撃が成功した場合の唯一の実害は，次回以降正当なクライアントがサーバにログインできなくなる事である．その場合再度，再同期化処理を行えば済むが，再びこれが攻撃される可能性もある．非常に希であると想定されるが，このような事態に陥った際は確実にリカバリするために，実運用として 60 秒以上待ってからオフライン（電話など）で利用者がサーバ管理者に種（検査）データを直接通知すれ



(注 1)ここでは，アプリケーションＹがアプリケーションＸを認証する際の片方向認証を示す．
(注 2)①の再同期化要求は初期値の改ざん／偽造およびリプレイ攻撃防止のため再同期化要求子をＸの秘密鍵で暗号化（電子署名）している．なお，自律認証系であるので公開鍵証明書は不要である．
(注 3)①および①' の再同期化処理は，ＹがＸを認証する際に必要となる検査データの初期値を渡すために必要なシーケンスであり，一般的には一度行えば障害などで再同期化が必要となるまで不要である．
(注 4)この例では，認証シーケンス処理が全て OK の場合を記述している．

図 15　認証情報再同期化処理（その 2）

ばよい．種（検査）データはアルファベットや数字など一文字でも構わないのでサーバ管理者への通知は迅速かつ容易に行える．

なお，再同期化要求子の電子署名は，一般の電子署名のようにハッシュ関数を通さず平文を直接秘密鍵で暗号化している．これは，転送するデータ形式を Spiral Hell の認証トークンと共通にするためである．

図 15 に再同期化処理から AP 間認証および AP 間通信までの一連の流れを示す．

4.5 Spiral Hell 基本ライブラリ

本実装では，Spiral Hell の API（Application Program Interface）およびセキュリティ管理機能を実現するため C＋＋言語により Spiral Hell 基本ライブラリを作成した．アプリケーションは目的あるいは実装の都合に応じてこれら基本ライブラリを自由に呼び出し可能である．表 2 に Spiral Hell 機能実現のための基本ライブラリを示す．

- 管理機能用関数は，マンマシンインターフェース用で利用者とセキュリティ管理機能モジュールとの対話処理に用いられる．
- API 用関数は，ユーザアプリケーションが無限ワンタイム認証機能を実現するためのアプリケーションプログラムインターフェースである．

表 2 Spiral Hell 基本ライブラリ

	関数名	概要
管理機能用	1 void SPIRAL_HELL( )	Spiral Hell 管理機能メイン
	2 void GEN_SAVE_KEY( )	利用者の鍵の生成，変更，および保存
	3 void READ_KEY( )	利用者の鍵の読み出し
	4 void SAVE_SEED( )	種データまたは検査データの生成，変更，および保存
	5 void READ_SEED( )	種データまたは検査データの読み出し
	6 void DISPLAY( )	各種登録情報の表示
API用	1 int HELL_Make_Spiral_Token( )	スパイラルトークンの作成
	2 int HELL_Check_Spiral_Token( )	受信スパイラルトークンの検査
	3 int HELL_SeedWrite( )	種データまたは検査データの書き込み
	4 int HELL_Make_Reinit( )	再同期化要求子およびその電子署名の作成
	5 int HELL_Check_Reinit( )	再同期化要求子の電子署名の検査

なお，これらは Microsoft Visual C＋＋言語で開発し，次の環境で動作確認済みである．

- OS　：Windows 95 または NT
- 本体　：AT 互換機
- CPU　：Pentium 133 Mhz
- メモリ：32 M バイト

5. 評　　価

本章では，Spiral Hellの ORB 環境上のアプリケーション間エンティティ認証への試行実装における安全性や実現性およびパフォーマンスなどについての評価結果を述べる．

5.1 パフォーマンス

5.1.1 公開鍵長と認証処理時間

RSA 暗号アルゴリズムの鍵の長さに対する認証子（Spiral Token）の生成およびその検査時間を鍵のモジュロ長別に測定した．結果を表3に示すが，これから次のことが判る．

1) 同じモジュロ長では，認証子の生成，すなわち秘密鍵による種データの暗号化時間はどれもほとんど同一であるが，認証子の検査，すなわち公開鍵による認証子の復号化時間は，公開鍵長が短い程小さくなる．

2) 認証子生成時間はモジュロ長が長くなる程大きくなるが，認証子検査時間は公開鍵長が同一の場合モジュロ長が大きくなってもさほど変わらない．

（注）表3では秘密鍵長を記述していないが，秘密鍵長は公開鍵長に係わらずモジュロ長とほぼ同一である．

表 3　認証子の生成/検査時間

公開鍵長（バイト）	生成(秘密鍵による暗号化)時間（秒）			検査(公開鍵による復号化)時間（秒）		
	モジュロ長（バイト）			モジュロ長（バイト）		
	64	96	128	64	96	128
2	2.35	4.13	5.64	0.09	0.10	0.16
4	2.28	4.17	5.48	0.26	0.33	0.25
8	2.16	3.61	5.48	0.33	0.44	0.60
16	2.40	3.74	5.23	1.23	0.76	0.79
32	2.31	3.68	5.25	1.41	1.76	2.37
64	2.39	3.58	5.46	4.42	3.47	3.22
96	−	3.68	5.33	−	4.62	4.61
128	−	−	5.92	−	−	9.91

（注1）測定環境

（注1）測定環境

CPU　：Pentium 133 MHz

メモリ：32 Mbyte

OS　：WindowsNT Workstation 4.0

（注2）測定中の CPU 使用率：95%（タスクマネージャによる）

（注3）測定値は各々500回測定し1回あたりの平均値を示す．

上述の認証子生成または検査時間すなわち認証処理に要する時間は，使用する暗号アルゴリズムおよびその実装（コーディング）に大きく依存する．表3は，（財）情報処理相互運用技術協会が開発した RSA 暗号プログラムを使用した場合の測定

結果であり，同じ RSA 暗号でも他ベンダー製のものを使用したり，あるいは他の種類の公開鍵暗号（楕円暗号など）を使用した場合には当然結果が異なる．ORB 上のアプリケーションにとってこれら認証処理に要する時間はリモートのオペレーション呼び出し時に必要となるが，通常のシステムではさほど問題にならない．しかし，即時性が要求され大規模トランザクションが発生するような分散アプリケーションでは無視できなくなるかもしれない．その場合，多数のクライアントから特定のサーバにトラフィックが集中することを考えると認証子検査時間は極力小さいことが望ましく，従って公開鍵長は短いことが望ましい．また暗号強度面（攻撃者に対する耐力）を考慮すると鍵のモジュロ長は大きい方がよい．今回の実装においては，モジュロ長 64 バイトで公開鍵長 2 バイトがパフォーマンス上最も望ましい．

5.2 安　全　性

5.2.1 毎回異なる認証子

使用する公開鍵暗号アルゴリズムの入力と出力の間に相関が無く完全に乱雑化されるならば，Spiral Hell は正に無限ワンタイムの言葉どおり，生成される認証子は毎回異なり，しかもモジュロサイズ（64 バイト―128 バイト）長のサイクル（天文学的回数）間は同一の認証子が現れないはずである．RSA 暗号についてはこの関連の情報が無いため正確なことが述べられないが，実測では任意の種データを数万回繰り返し暗号化しても同一の認証子が現れなかった．仮にいつか同一の認証子が現れるとしても，「なりすまし」を狙っているセキュリティ攻撃者から見れば，膨大な回数の認証子を盗聴していなければ次の認証子が推測できず（なお，現段階では過去の認証子と同じものが現れる保証も無い），実用上問題無しと言える．

5.2.2 Spiral Hell の強度

3.4 節からも判るように Spiral Hell の認証強度，言い換えれば第三者によって認証子が偽造される可能性は使用する公開鍵暗号アルゴリズムの強度に完全に依存する．Spiral Hell の方式上，この認証強度と使用する公開鍵暗号アルゴリズムの強度は同値である．従って，使用する公開鍵暗号アルゴリズムの安全性が証明されていれば，それはすなわち Spiral Hell の安全性が証明されていることに他ならない．

今回の実装では，公開鍵暗号アルゴリズムに現在世界で最も広く利用されている RSA 暗号を採用した．この場合，暗号強度面においてはモジュロ長 128 バイトが最も望ましいが，暗号強度とパフォーマンスはトレードオフの関係となるので，システム全体のセキュリティーポリシーを勘案の上適正なモジュロ長を選択することになる．尚，今後強度面においてこの RSA 暗号の寿命（RSA 暗号の実用強度はあと数年と言われている）がきたら，その時は一定鍵長において RSA 暗号よりさらに強度な楕円暗号などに交換すればよい．原理的に Spiral　Hell のメカニズムそのものは未来永劫陳腐化することはない．

5.2.3 望ましいセキュリティモジュールの形式

厳密なセキュリティシステムを考えるなら，セキュリティ機能をオブジェクト化するのは根本的に間違いである．

例えば，既出の図 9 のようにクライアントまたはサーバのアプリケーションオブジェクトと認証子を生成する認証オブジェクトとが異なるホストにある場合，認証子生

成のために必要な秘密鍵をどのように安全に認証オブジェクトに渡すのか，あるいは生成された認証子をクライアントアプリケーションに返送する際にそれが第三者に盗聴されて，正規のクライアントより先にサーバへのログインに使用されたりする可能性，さらには各ホスト上のアプリケーションオブジェクトと認証オブジェクト間の認証はどうするのかなどの様々な問題が派生してくる．これらはクライアント側およびサーバ側の双方において同様のことが言える．このように処理を分散すればするほどセキュリティ上の脅威の機会（セキュリティホール）が増加するので，分散環境上にセキュリティシステムを実装する際には注意を要する．図 16 の例のように，オープンなネットワーク上に広域に亘って分散したクライアントとサーバ間を ORB で接続したシステム構成では，クライアントおよびサーバの双方とも認証モジュールはライブラリ形式が望ましい．しかし，内部ネットワークが完全にファイアウォールなどで外部からのアクセスが制御され，且つ内部ネットワーク上には悪意の攻撃者が居ないと言う前提であれば，認証モジュールのオブジェクト化も可能となろう．ただこのようなケースにおいても内部に悪意の攻撃者が皆無とは保証できない．実際セキュリティ犯罪の半数以上は内部犯罪である．また，セキュリティ機能を分散オブジェクト通信機構である ORB に委ねたり，あるいはもっと下位層の IPv 6 のようなネットワーク層に委ねることは可能であるが，アプリケーションからみればセキュリティ維持の保証が得られる訳ではない．特にエンドツーエンドの厳密なセキュリティ維持を要求するなら，アプリケーション相互がその責任においてセキュリティ機能を自らが全うするしかなく，その場合セキュリティホールを最小化するという意味でライブラリ形式が望ましい．

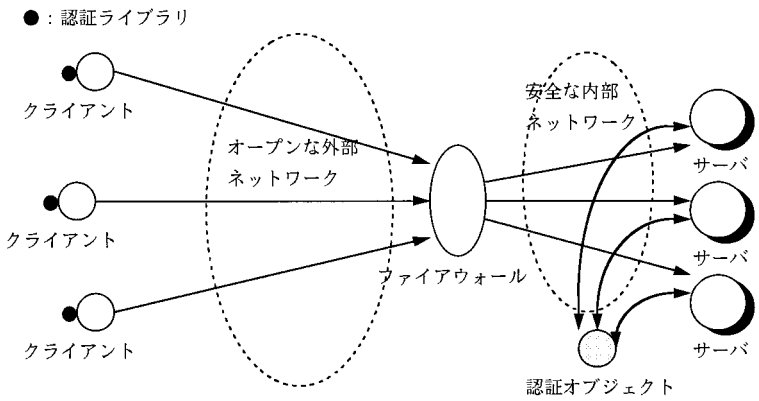


図 16　安全な内部ネットワークにおける認証オブジェクトの例

このように，分散環境本来のサーバ負荷分散化のためのセキュリティ機能のオブジェクト化やセキュリティ機能そのものを下位層に委ねることとセキュリティ強度はトレードオフの関係となり，システム構成時のセキュリティポリシーを如何に設定するかという問題に帰着する．今回の試行実装，すなわち ORB 上のアプリケーションに提供するセキュリティ機能についてはライブラリ形式が望ましい．

5.3 分散環境における Spiral Hell の実現性

5.3.1 API（Application Program Interface）の簡素化

アプリケーションが Spiral Hell を実装する際，極力アプリケーションが実装を意識しないで済むように， API は表 2 でも挙げたように以下の 5 種類に限定している．これにより容易に無限ワンタイム認証機能を実現することが可能である．

◆被認証者側 API

・int HELL_Make_Spiral_Token（）：認証子を生成.

・int HELL_Make_Reinit（）：再同期化要求子およびその電子署名の作成

◆認証者側 API

・int HELL_Check_Spiral_Token （）：認証子を検査.

・int HELL_Check_Reinit（）：再同期化要求子の電子署名の検査

◆被認証者側および認証者側共通 API

・int HELL_SeedWrite（）：認証結果が OK の時，認証子を次回の種データまたは検査データとして保存.

5.3.2 相互接続性

Spiral Hell の相互接続性に関し考慮すべきは，使用する暗号アルゴリズムの相互接続性である．本実装に使用した RSA のような暗号アルゴリズムは複数ベンダーから提供されているが，一般に各ベンダーは同じ RSA 暗号であっても自社以外の製品との相互接続性を保証していない．従って，認証者側と被認証者側で使用する暗号アルゴリズムのベンダーが異なっている場合，理論上は有り得ないが，認証子が正常に復号化されないケースがあるかもしれない．しかし，これは全てのセキュリティ製品についても言えることで，Spiral Hell に限ったことではない.

繰り返しになるが，エンドツーエンドの厳密なセキュリティを要求するなら，アプリケーション相互がその責任においてセキュリティ機能を全うするしかなく，その場合アプリケーション開発段階で暗号ライブラリのベンダーを統一することなどは，様々なベンダーの既存セキュリティ製品を組み合わせてシステム構築することに比較すれば容易である.

5.3.3 実装容易性

Spiral Hell の実装にあたって行う事は，鍵生成や利用者登録のためのセキュリティ管理機能のインストールと，5.3.1 項でも述べたように僅か 5 種類の API をアプリケーションに組み込むのみである．但し，現時点では Spiral Hell 機能を必要とするサーバ側のオペレーション単位に API の組み込みが必要である．しかし，実装のための工数やコストは，分散システムの規模にもよるがさほどでもないであろう.

5.4 総 合 評 価

既存の各種エンティティ認証方式および Spiral Hell（無限ワンタイム認証方式）を既に述べた「安全性」およびパフォーマンスをも含む「実現性」の観点から評価すると図 17 のようにプロットできる．Spiral Hell は，「安全性」および「実現性」ともに既存方式を上回っていると考えられる.

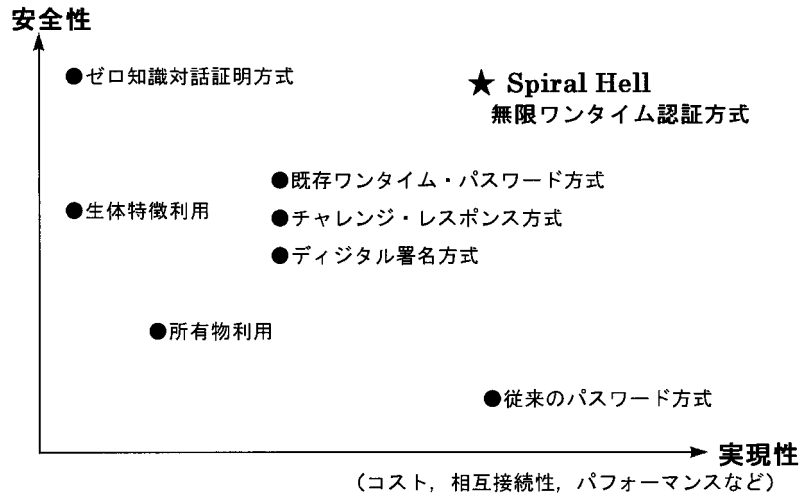


図 17　各種認証方式の安全性と実現性

6．課　　題

以下に今回の試行実装で実施できなかった課題を述べる．

6.1　再同期化処理の迅速化

今回の実装では，再同期化処理を行うため該処理専用のオブジェクトをクライアント側およびサーバ側の双方に作成し，利用者の要求に応じてセキュリティ管理モジュールがこれらオブジェクトを起動する形態を採った．しかし，迅速に再同期化処理を要求するシステム環境も想定されるため，同期はずれ発生時に利用者の手を煩わすことなくユーザアプリケーションの判断で自動的に再同期化処理を行い，一定回数リトライしてもNGの時にだけ利用者に通知するなどの方策が求められる．これについては，ユーザアプリケーションが若干複雑になるが容易に実現可能である．

6.2　複数のクライアントAPが同一User-idで稼動した時の考慮

同一のServer-idに対し，同一あるいは異なるクライアントアプリケーションが同一のClient-idで複数稼働した際には，認証処理結果待ちのクライアント側各スレッドが種データの更新をロック制御しなければならないのは致し方ないところである．

今回の実装では，ORB上のクライアントアプリケーションがサーバのオペレーションを呼び出すときに認証子を付加し，サーバ側ではオペレーションの実行に先立ち認証子を検査し，検査結果がOKならそのオペレーションを実行し，実効終了時に戻り値として認証結果をクライアントに返す仕組みとなっている．サーバ側は検査がOKの時点で検査データを更新するが，クライアント側は検査結果を受信してから種データを更新することになっている．従って，ロック制御する時間が長くなり，システム全体としてのパフォーマンスが劣化する．

これを回避するには，クライアント側のアプリケーション個々にUser-idを変えてやればよい．アプリケーション相互で認証を行うほど重要なアプリケーションが同一User-idで複数稼働させることはセキュリティ監査上も好ましくない．しかしながら，どうしても同一User-idでクライアント側のアプリケーションを複数稼働させざるを

得ない場合には，前述したロック時間短縮のために，例えば，サーバ側での認証子検査が OK のとき即時に（オペレーション実行前に）それをクライアントに通知してやるなど何らかの考慮が必要である．

6.3 双方向認証機能について

今回の実装では，サーバが認証者でクライアントが被認証者となる片方向認証について実装ならびに評価を行ったが，実際のシステムではクライアントもサーバの認証を要する双方向認証が必要なケースが想定される．Spiral Hell の方式そのものは原理的に双方向認証も可能である．すなわち図 6 の処理を逆方向にも適用すれば済む事である．しかし，ORB 上のアプリケーションに対して双方向認証機能を実装するためには，アプリケーション実行前にサーバ側から認証結果をクライアントに通知すると共に，クライアントに対する認証子を送信するなどの機能が必要となる．これに対処するためには ORB のコールバック機能などの利用が考えられるが，本実装では未実施である．

7. お わ り に

本稿では，エンティティ認証の本質を整理した上で Spiral Hell の自律認証系を前提とした ORB 環境への適用について，実装方法，安全性，パフォーマンスなどについて実測値を含め検討，考察ならびに評価した．ORB に依存するいくつかの課題は残るが，Spiral Hell の方式そのものに対する問題点は今回の試行実装では見当たらなかった．

既存方式の問題点を払拭し，しかも C.C. の要件をも満たし，さらに安全性が証明されている簡単で堅牢な方式 Spiral Hell は，真に重要機密通信時の相互エンティティ認証方式として ORB 環境においても有望であると考える．

参考文献 ［1］ 八津川直伸“ネットワークセキュリティ―無限ワンタイム認証方式の考案”，日本ユニシス技報 1997 年 8 月通巻 54 号, pp. 115–140
［2］ Common Criteria（CC）Project（http : //csrc.ncsl.nist.gov/nistpubs/cc/）
［3］ Object Management Group, CORBA Services : Common Object Services Specification V 1.0 November 1996.（Chapter 15. Security Service Specification）

執筆者紹介 八津川 直伸（Naonobu Yatsukawa）
1980 年福井大学工学部電子工学科卒業．同年日本電信電話公社入社，伝送システムの施設設計を担当．1985 年日本ユニシス(株)入社．以来，電気通信の自由化に伴い，コモンキャリアとしての基本電気通信事業および VAN 事業の企画推進，汎用機用通信制御装置のコミュニケーションソフトウェアの開発および保守，セキュリティ技術の研究を担当．
現在，プラットフォームビジネス部に所属．