第 10 部

認証機構

第 1 章

まえがき

近年、計算機の小型化、高性能なワークステーションの出現に伴い、計算機間での資源を共有することを目的として、LAN(Local Area Network)を用いて複数の計算機を互いに結合したネットワークシステムを構築することが、一般的に行われている。このような環境において、ネットワーク上の複数の計算機を協調して働かせるために、数々のネットワークアプリケーションが開発されてきた。これらのアプリケーションの多くは、処理を必要とするプロセス（クライアント）が処理を実行するプロセス（サーバ）にサービスを依頼し、その結果を受け取るような、いわゆるサーバ・クライアントモデルに基づいたものが多い。

これらのアプリケーションの中には、サービスを受けることのできる利用者を限定するものや、利用者によって提供するサービスの範囲を限定するものがある。前者の例として、有料のサービスが挙げられる。これは、料金を支払っている利用者のみがサービスを受けられる。後者の例は、パスワードの更新がある。普通、利用者は自分自身のパスワードしか変更できず、他者のパスワードは変更できないようになっている。このようなサービスにおいては、利用者によって正しい利用許可の発行を行う必要があり、利用者の認証が必要である。つまり、利用者を識別する機構を実装しなければならない。

このようなアプリケーションの運用を、ネットワーク環境で行うことを考える。対象となるネットワークシステムが非常に小規模の場合、単一の管理組織による利用者の登録の制限や、パスワードの管理などが可能である。このような環境では、利用者の認証は従来のパスワードを用いた方法を利用でき、これに基づいて利用許可を適切に発行することにより、サービスの運用を安全に行うことができる。また、盗聴などによる不正な情報の取り出しや、不正なユーザによる計算機の利用についても、対象となるネットワークの物理的あるいはネットワークプロトコル的な隔離・保護や、通信の積極的な制限が容易となる。

一方、最近では国内においても LAN 等の小規模のネットワークが相

互結合され、一つの大規模な広域ネットワークを形成しつつある。このようなネットワークでは、複数の管理組織が独立に管理を行っているネットワークシステム（サブネットと呼ぶ）を、相互に接続したものととらえることができる。このようなネットワークシステムでは、各管理組織が独自の方針にしたがって管理を行っている。このため、ネットワーク全体での統一的なセキュリティ管理を行うことは難しい。したがって、一部のサブネットワークにおいて、十分なセキュリティ管理が行われていないような場合、通信路での通信内容の傍受や他の利用者のアカウントの不正利用、パスワードの偽造などが発生しやすい。また、現在のネットワークアプリケーションにおいては、十分なセキュリティ対策を施しているとはいえない。例えば、UNIX システムにおける ftp や telnet 等のいくつかのネットワークアプリケーションでは、パスワードなどの保護すべき情報が、暗号化などの保護を行われない状態でやりとりされている。このような状況では、一般のユーザが、比較的簡単にこれらの情報を手に入れ、悪用することが可能である。

このような環境でのネットワークアプリケーションの実現およびその安全な運用を考えた場合、なんらかの通信路における通信内容の保護、利用者の識別方式 (identification)、利用者の正確な認証 (authentication)、利用者に対する適切な利用許可の発行 (authorization) などを行う機構が必要である。特に、これらの機能のなかで、利用者認証は最も基本的な機能であると考えられる。

1.1 WIDE プロジェクトと本研究の目的

1988 年に開始された **WIDE** (Widely Integrated Distributed Environment) プロジェクト [29] は、大規模な広域ネットワーク上での分散環境を提供することを目標にしている。**WIDE** プロジェクトでの研究は、二つのカテゴリに分けることができる。一つは、**WIDE** インターネットと呼ぶテストベッドネットワークを構築するためのネットワーク間接続技術の開発と、運用技術の確立が行われている。現在までに、ディジタル専用線、X.25 パケット網、ISDN などの通信サービスを利用して、プロジェクト参加組織のネットワークを相互接続するための技術開発が行われており、1990 年 2 月現在、国内の 6 大学 5 企業が **WIDE** インターネットに接続され、また、米国 ARPA インターネットとの接続も完了している。もう一つは、このような大規模な広域ネットワークを構築する際に起こる基本的な問題を同定することである。広域ネットワークを構築する際には、ネットワーク管理技術、セキュリティ管理技術、広域分散環境に適したプロセス間通信

技術などを研究する必要がある。**WIDE** プロジェクトでは、これらの研究を通じて、問題点を同定し、さらに **WIDE** インターネットに対して開発した技術を適用することで、有効性などを検証していく。

現在、**WIDE** インターネットには大学、研究所、企業などの様々な組織が接続されており、利用者にはネットワーク上に分散した多くの計算機によって各種のサービスが提供されている。サービスを提供する計算機にとって、利用者によって受けられるサービスを限定するためには、利用者を識別することが重要である。しかし、計算機や利用者が、すべて組織的な管理を受けているとはいえない。したがって、他人のアカウントを使用するなどの不正なサービスの利用が発生する可能性がある。このため、ネットワークプロトコルやネットワーク上のサービスに何らかのセキュリティ機構を提供しなければならない。また、組織の管理下にない計算機がネットワークサービスを受けるためには、その利用者とサービスを提供する主体が互いに認証を行うことが必要である。

本研究では、**WIDE** プロジェクトにおける、セキュリティに関する研究の一環として、SPLICE[1]/AS(Authentication System) を開発する。SPLICE/AS は、**WIDE** インターネットでのサービスに対して、サービスを受ける利用者を識別する手段を提供する。このシステムは、**WIDE** インターネットのセキュリティ管理の基本的な構成要素であり、この上位に、利用許可を与える (Authorization) 機構や、アカウント機構が構築される。今年度においては、鍵の管理を行うサーバを作成し、これを実際のネットワークアプリケーションに組み込むことを目標とする。

1.2　用語・表記

以後の議論を明確にするために、次のように用語を定義する。

分散環境: ここでの分散環境とは、独立した管理組織に管理された計算機ネットワーク（サブネットワークと呼ぶ）が相互に結合されているような広域分散環境を指す。

ユーザ・クライアント・サーバ: ユーザは分散環境における利用者である人間を表す。ユーザは分散環境において何らかのサービスを受ける場合、サーバと通信するためのプロセスを計算機上で起動してネットワーク上にあるサーバのプロセスに対して処理要求を送り、サーバは処理要求に対して処理を行って何らかの処理結果をクライアントに送り返

[1] Supercomputer,Personal workstation and Local area network InterConnected Environment

す。この一連の処理によって、ユーザは対象とする分散環境で提供されるサービスを利用することができる。

エンティティ(entity): 分散環境で通信を行う実体。ここでは、計算機上のプロセスを指す。

AS(Authentication Server): エンティティに対して利用者認証のサービスを提供するサーバ。利用者認証のサービスを要求するすべてのエンティティのサーバとして存在する。

ドメイン (domain): 一つの AS が管理する計算機やエンティティの論理的な集合。

公開鍵・秘密鍵・プライベート鍵: 公開鍵暗号方式において、所有者だけが知ることができる鍵を秘密鍵、公共に公開された鍵を公開鍵と呼ぶ。これと区別をつけるため、慣用暗号方式で用いる鍵をプライベート鍵と呼ぶ（プライベート鍵は利用者のパスワードに等しい)。各クライアントは、それぞれこれらの鍵を持つ。

また、プロトコルを記述するために以下のような表記方法を導入する。

A → B: $message$:A から B へのメッセージの送信を表す。$message$ はその通信内容

$message$ の内容は以下に示す表記を用いて記述する。

A	:A の名前
PK_A	:A の公開鍵
SK_A	:A の秘密鍵
CK_A	:A のプライベート鍵
$\{\text{messge}\}^{key}$	:公開鍵暗号方式で公開鍵 (あるいは秘密鍵) key を用いて message を暗号化したもの
$[\text{messge}]^{key}$	:慣用系暗号方式でプライベート鍵 key を用いて message を暗号化したもの
$(kind)$	:メッセージの種類・目的を示す識別子

以下に示す例では、X が自分の名前と Y の名前を、X のプライベート鍵で暗号化したものと、Y の名前をメッセージとして Y に送っていることを表す。

$\text{X} \longrightarrow \text{Y} : \text{Y}, \ [\text{X,Y}]^{CK_X}$

第 2 章

過去における研究

広域ネットワーク環境においては、第三者による通信内容の傍受を防ぐために、通信内容を暗号化することが一般的に考えられており、暗号化の技術を発展させて利用者認証に適用することが研究されてきた。これまで多くの研究者によって、暗号を利用した利用者認証プロトコルが提案されているが、代表的なものとして、Needham 等 [49] によって二つの方法が提案されている。

- 慣用系暗号方式
  慣用系暗号方式では、ユーザはそれぞれ一つの鍵を持っていて、自分と AS 以外に知られてはならない。あるユーザが他者と通信する場合、自分の鍵を使って暗号化しても他者が解読できないため、二人だけで共通の鍵を持つ必要がある。この共通の鍵の生成・配送のために、Authentication Server(AS) を配置し、AS が鍵の生成・配送に必要なサービスを提供する。ここで、ユーザ A の鍵を KA、ユーザ B の鍵を KB、共通の鍵を CK とする。A と B が CK を得るまでのプロセスは以下のようになる。

$$A \longrightarrow AS \quad : A, B, id \tag{2.1}$$

$$AS \longrightarrow A \quad : [id, B, CK, [CK, A]^{KB}]^{KA} \tag{2.2}$$

$$A \longrightarrow B \quad : [CK, A]^{KB} \tag{2.3}$$

  id は AS との通信に使われる一回限りの識別子で、返信のメッセージ中の id と比較して、通信が安全に行われたかどうかを確かめるものである。ただし、このプロセスでは認証はなされていない。つまり、通信相手が本物かどうかはこれだけでは分からない。これに認証機能を加えるための一つの方法として、平文にある操作を行い、得られた値を認証のための識別子とする方法が提案されている。まず、平文を入力として、その平文に対してある値を与える関数 f() を定義する。

$$\mathrm{CS} = \mathrm{f}(\mathrm{message})$$

ユーザ A は、ユーザ B に送るメッセージ $message$ に関数 f を適用し、CS という値が得られたとする。A はこれを AS に送り、AS はこれに自分の鍵で暗号化して A に送り返す。

$$A \longrightarrow AS \quad : A, [CS]^{KA} \tag{2.4}$$
$$AS \longrightarrow A \quad : [A, CS]^{KAS} \tag{2.5}$$

KAS は AS の鍵である。A はこれをメッセージの後につけて B に送る。

$$A \longrightarrow AS \quad : [message, [A, CS]^{KAS}]^{KB} \tag{2.6}$$

B はこれを復号すると、$message'$ と [A,CS]^{KAS} を得て、$message'$ に A と同じ操作を行い、CS′ という値を得たとする。B は [A,CS]^{KAS} を AS に送ると、AS は CS を送り返す。

$$B \longrightarrow AS \quad : B, [A, CS]^{KAS} \tag{2.7}$$
$$AS \longrightarrow B \quad : [A, CS]^{KB} \tag{2.8}$$

B はこれを復号化して、CS を得る。B は CS と CS′ を比較して、同じ値が得られれば、このメッセージが本当に A から送られたものだと判断できる。

- 公開鍵暗号方式
  公開鍵暗号方式では、ユーザはそれぞれ公開鍵・秘密鍵の二つの鍵を持つ。公開鍵で暗号化したものは秘密鍵で復号でき、その逆も可能である。したがって、秘密鍵は自分と AS 以外に知られてはならないが、公開鍵は誰に知られてもよい。よって、公開鍵の配送に関しては秘密保持のための暗号化は必要ない。また、公開鍵暗号では、秘密鍵はその所有者しか使えないので、認証を行うには秘密鍵で暗号化するだけでよい。例えば、A が B にメッセージ $message$ を送る場合、次のように通信を行う。

$$A \longrightarrow AS \quad : A, B \tag{2.9}$$
$$AS \longrightarrow A \quad : \{PK_B, B\}^{SK_{AS}} \tag{2.10}$$
$$A \longrightarrow B \quad : \{\{message\}^{SK_A}\}^{PK_B} \tag{2.11}$$

SK_{AS} は AS の秘密鍵、SK_A は A の秘密鍵、SK_B は B の公開鍵である。まず、A は B の公開鍵をもっていないので、AS にそれを要求する。AS は A に B の公開鍵を返すが、このとき、AS の秘密鍵で暗

号化することによって、A は確かに AS からのメッセージであることがわかる。次に、A は自分の秘密鍵で暗号化して、さらにそれを B の公開鍵で暗号化して B に送る。B はまず、自分の秘密鍵で復号化し、A の公開鍵で復号化し、意味のあるメッセージが得られればそれが本当に A からのものだということがわかる。この二つの鍵での暗号化を二重暗号化と呼び、通信内容の保護と認証が同時に行える。

また、Voydock 等 [82] は、ネットワークプロトコルにおける Security について体系的に説明し、DES(Data Encryption Standard) を用いた利用者認証方法を示している。しかし、Needham の研究も含めて、これらの研究においては、プロトコルを設計し、その安全性を議論しているが、実際のシステムでの実現や運用における問題についてはあまり考えられていない。

一方、実際にネットワークにおいて利用者認証の機構を構築した代表的なものとしては、MIT Athena Project の Kerberos [66] が挙げられる。Kerberos では、Voydock が示した方法を基礎として、UNIX システムを中心とした分散環境において利用者認証の機構を構築し、また、実際に Kerberos を適用したネットワークアプリケーションを多数開発している。Kerberos では、暗号化に必要な鍵のデータベースを管理するサーバはひとつのマスタ・サーバと複数のスレーブ・サーバから構成し、負荷が集中しないようにしている。しかし、鍵の変更が発生した時にマスタ・サーバからスレーブ・サーバに対して鍵の更新を行う必要があり、この更新を行うための有効なプロトコルは現在のところ実現されていない。また、ある Kerberos サーバ A が管理するネットワークシステムに属する利用者が、他の Kerberos サーバ B が管理するネットワークシステムで提供されているサービスを要求する時には、利用者はまず Kerberos サーバ B が管理するネットワークにアクセスするためのチケットを Kerberos サーバ A からもらい、その鍵をもって Kerberos サーバ B にアクセスする。すなわち、n 個の Kerberos サーバが存在するネットワークシステムでは n^2 個の鍵を Kerberos サーバにアクセスするために共有していなければならない。このため、Kerberos サーバの数が多くなると、これらの鍵の管理が難しくなるといった欠点がある。したがって、Kerberos をそのまま広域ネットワークに適用することは、多くの技術的困難を伴うと考えられる。

第 3 章

SPLICE/AS の概略

この章では、昨年度の基本設計を踏まえて SPLICE/AS の概略を述べる。

3.1 暗号を用いた利用者認証方式

実際に利用者認証システムの設計を行うにあたって、暗号方式の決定を行い、それに即した設計を行わなければならない。ここで問題となるのが、鍵の管理と鍵の配送である。我々は、効率良く鍵の管理・配送ができるような暗号方式を選択しなければならない。これに関して 2章で述べたように、Needham によって、二つの方法が提案されている。この二つの暗号方式を比較した場合、広域分散環境においては、鍵の配送に関して公開鍵暗号方式の方が有利であることは明らかである。以上のことから、我々は公開暗号方式がネットワークでの運用により適していると判断し、公開鍵方式を採用することにした。Needham が提案した方式に基づいて、SPLICE/AS では、鍵の管理・配送を行うための Authentication Server と、それに対するインターフェイスを開発する。

3.2 階層的ド メイン

一般に広域ネットワークでは、複数の管理組織が独立に管理しているサブネットワークが相互に接続された、一つの大規模なネットワークシステムととらえることができる。これを図 3.1のようにモデル化する。

AS は、ユーザとサーバの、公開鍵・秘密鍵およびプライベート鍵のデータベースを持っている。全ての利用者の公開鍵を単一の AS で管理した場合、ネットワークが小規模な場合であれば有効にサービスを提供できるが、ネットワークが広域化してくると、このデータベースが巨大になるばかりでなく、その AS に負荷が集中するという問題が発生すると考えられる。このトラヒックの集中に加えて、管理組織の問題もある。広域ネットワークでは、複数の管理組織がそれぞれ独自の方針でサブネットワークの管理

を行っている。このため、ネットワーク全体を一つの管理の単位とした場合、各管理組織の管理方針を反映させることは難しい。

そこで各管理組織ごとにローカルな AS を配置し、そこでのサービスを独立に提供してゆく方法が考えられる。こうすることにより負荷の分散や組織単位での管理が行え、実際のネットワークに即したサービスが行える。ここで、一つの AS によるサービスを受ける計算機やローカルなネットワークシステムの論理的な集合をドメインと呼ぶことにする。

ここで、ドメイン間での公開鍵の配送という問題が発生する。SPLICE/AS では、この解決策として、幾つかのドメインをサブドメインの集まりとみなして、新たな一つのドメインを構成することを考えた。そしてそのドメインの中の、サブドメイン上の AS の公開鍵を管理するため、Master Authentication Server(MAS) を置いた。MAS は、下位のドメインの AS での鍵の配送における安全性を暗号化の手法を用いて提供している。このようにドメインに階層的構造を持たせることにより、AS が上位の MAS の公開鍵を持っていれば、MAS を通じて他のエンティティの公開鍵を確実に手に入れることができる（具体的なプロトコルについては後述）。 また、鍵の更新も、更新を行う AS と MAS だけで行えばよい。さらに階層構造を増やすことにより、広域なネットワークに対応していくことができる。このとき AS および MAS は、図 3.2のように木構造の形でとらえることができる。このような木構造は、実際のネットワーク管理における組織間の関係に対しても、適合性がよい。SPLICE/AS では、この階層化したドメイン構造の考え方を導入し、Authentication Server を構築している。

3.3　名前構造

ネットワーク全体での利用者認証サービスを考えた場合に、各 AS はエンティティを階層型のドメイン構造の中で一意に識別する必要がある。SPLICE/AS が対象とするネットワークは、**WIDE** インターネットを想定している。**WIDE** インターネットにおいては、Internet Protocol が用いられている。Internet Protocol によれば、通信を行うエンティティは、インターネットアドレスとトランスポート層でのポート番号によって一意に識別することが可能である。そこで、以下の要素を組み合わせてエンティティを表すことにした。

```
struct entname {
        char *name;
        u_short id;
        char addr[4];
```

```
        char domain[256];
}
```

1. name
   対象となるエンティティがサーバであれば、それが提供するサービス名。クライアントであればそのクライアントを起動した利用者のアカウント名。

2. id
   対象となるエンティティがサーバであれば、そのポート番号（ポート番号は、4 オクテットの整数として扱う）。クライアントであれば、そのクライアントを起動した利用者のユーザ id。

3. addr
   対象となるエンティティが稼働している計算機のインターネットアドレス。インターネットアドレスは、4 オクテットから構成される。

4. domain
   エンティティの存在するドメインを、下位のものから順に”.”(ドット)で区切って並べたもの。例えば、大阪大学基礎工学部情報工学科のドメインは次のように表すことができる。

   ```
   ics.osaka-u.ac.jp
   ```

3.4 SPLICE/AS の実装環境

SPLICE/AS において、AS はこのシステムのもっとも重要な部分である。AS のおかれている環境下には、公開鍵・秘密鍵・プライベート鍵のデータベースなど、AS 以外にアクセスを禁止するようなファイルが集中している。これらのファイルでは、必要に応じて暗号化などの安全策をとっているが、データベースファイルからの情報の不正な取り出し、データベースファイルの破壊などの問題が発生する可能性がある。したがって、セキュリティの面から考えて、AS は、物理的にもネットワーク的にも安全な環境におかれる必要がある。

3.5 プロトコルの設計

SPLICE/AS で使用しているプロトコルは、Needham 等が提案した、公開鍵暗号を用いた利用者認証プロトコルを基礎としている。さらに SPLICE/AS

では、階層化された AS 間での鍵の配送、replay attack に対処するためのプロトコルの拡張を行っている（replay attack については後述）。

3.5.1　ログイン時の鍵入手プロトコル

ユーザは AS のサービスを受ける場合、通信するためのプロセスを起動するが、起動直後においては、ユーザは自分のドメインの AS の公開鍵および自分の公開鍵・秘密鍵を持っていない。そこで、利用者はまず、「AS にログイン」して、これらの鍵を取ってこなくてはならない。ここでは情報の保護のため、利用者のパスワードをプライベート鍵として、慣用系暗号を使用する。図 3.3に、このプロトコルを示す。

AS は利用者のパスワードをデータベースとして持っており、要求が来るとデータベースを検索してプライベート鍵を取り出し、メッセージの暗号化に使用する。id はセッション id と呼び、各鍵入手セッションごとに異なった値が設定される。これは送信者が返信を受け取った時、この値が同じかどうかを見て安全に通信が行われているかどうかを判断するのに使用される。

3.5.2 公開鍵入手のプロトコル

あるユーザが他者と安全な通信を行いたい場合、他者の公開鍵で暗号化をする必要がある。したがって、AS に他者の公開鍵の要求をしなければならない。ここでは同一ドメイン内での通信をを考える。図 3.4にこのプロトコルを示す。

ここで、返信のメッセージ中で、AS の秘密鍵で暗号化が行われている。これは認証のための暗号化である。基本的に AS の秘密鍵は AS しか使えないので、CL1 は、このメッセージが AS からのものである、ということがわかるわけである。また、メッセージとして送られるものは公開鍵であるので、情報の保護は行う必要はない。

3.5.3 複数ドメイン間での鍵入手プロトコル

ここでは、複数ドメインにまたがって公開鍵の要求を行う場合を考える。図 3.5にそのプロトコルを示す。認証を行なうため、4,5,6 ではすべて送信側の秘密鍵で暗号化されている。メッセージとして送られるのは公開鍵であるので、受信側の公開鍵で暗号化して情報を保護する必要はない。また、ドメインの数や管理組織の数が増加するのに応じて、階層の深さを大きくすることによって対応する。この場合、図 3.5の MAS が自分の上位の MAS にリクエストをかけることになる。

3.5.4 通信要求のプロトコル

ここでは、ある利用者が他者と通信を行う際、互いに認証を行うためのプロトコルを考える。図 3.6にこのプロトコルを示す（相手の公開鍵は既に入手済みとする）。

各メッセージには $timestamp$ と $life$ の二つが含まれている。$timestamp$ はメッセージが作られた時刻で、$life$ はメッセージの有効期間である。これらは replay attack の検出のために使われる。

replay attack とは、第三者がネットワーク上に流れているメッセージを傍受し、それを蓄積しておき、後になってそれを発信するものである。これに対する対策がなされていなければ、受信側では replay されたメッセージと正当なメッセージ区別をつけることができず、この不当なリクエストを受け入れてしまうことになる。

replay attack を検出するため、$life$ の値を、以下のように設定する。

life = (timestamp を発行からメッセージの送信までにかかる時間)
+ (メッセージのネットワーク内での最大存在時間)

受信側では、メッセージを受け取った時、$timestamp + life$ が現在の時間を過ぎていれば replay とみなせばよい。このような判断を正しく行なうためには、送信側のクロックと、受信側のクロックの時刻が同期している必要がある。しかしながら、実際にシステム上で正確にクロックの時刻を一致させるのは困難であり、この場合には送・受信者間でのクロックの誤差を考慮しなければならない。このため life の値としては

$$\text{life}' = \text{life} + (\text{クロックの最大誤差})$$

のように life$'$ を求めこれを用いる。

また、メッセージ 1 では、二重暗号化が用いられており、通信内容の保護・認証が同時に実現できる。メッセージ 2 では、id が正しければ、返信が正しい相手から返ってきたと判断できるので、二重暗号化は用いていない。

3.5.5　鍵の有効期間

我々が使用する暗号は時間をかけて計算を行えば破られる可能性がある。これは鍵の長さに依存しており、安全が保証できる程度に鍵の長さを設定しているが、あまりに長時間同じ鍵を使い続けるのは危険である。したがって、ある期間をおいて鍵の更新をする必要がある。

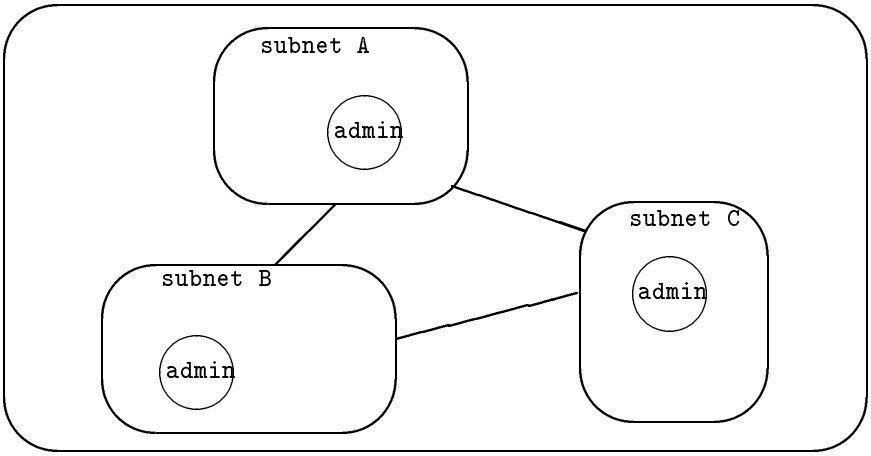


図 3.1: ネットワークモデル

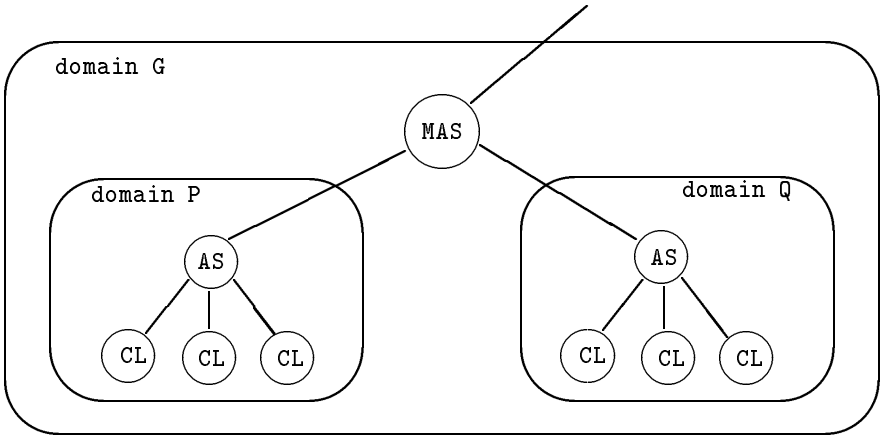
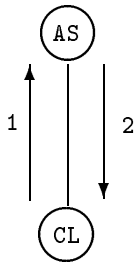
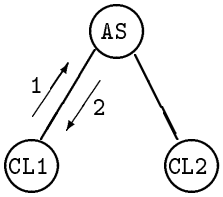


図 3.2: ドメインの階層構造



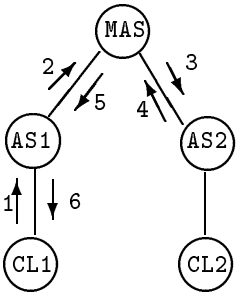
1．　CL → AS:CL,[CL,AS,id]^{PASSWD},($init_req$)
　　AS へログイン要求
2．　AS → CL:AS,[CL,AS,id,PK_{CL},SK_{CL},PK_{AS}]^{PASSWD},($init_rpl$)
　　クライアントへ PK_{CL},SK_{CL},PK_{AS} を返す

図 3.3: AS へのログイン



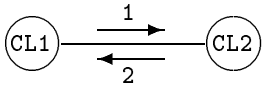
1．　CL1 → AS:CL1,CL2,id,($key\_request$)
　　AS に CL2 の公開鍵を要求
2．　AS → CL1:AS,$\{$AS,CL1,id,$PK_{CL2}\}^{SK_{AS}}$,(key_reply)
　　CL1 に CL2 の公開鍵を返す

図 3.4: 公開鍵の入手



1．　CL1 → AS1:CL1,CL2,id,($key\_request$)
　　CL1 から AS1 へ CL2 の鍵を要求
2．　AS1 → MAS:AS1,CL2,id,($key\_request$)
　　AS1 は CL2 の公開鍵を持っていないので、MAS に鍵を要求
3．　MAS → AS2:MAS,CL2,id,($key\_request$)
　　MAS は AS2 に鍵を要求
4．　AS2 → MAS:AS2,MAS,$\{$AS2,MAS,id,$PK_{CL2}\}^{SK_2}$,(key_reply)
　　AS2 は MAS に鍵を返す
5．　MAS → AS1:MAS,AS1,$\{$MAS,AS1,id,$PK_{CL2}\}^{SK_M}$,(key_reply)
　　MAS は AS1 に鍵を返す
6．　AS1 → CL1:AS1,CL1,$\{$AS1,CL1,id,$PK_{CL2}\}^{SK_1}$,(key_reply)
　　AS1 は CL1 に鍵を返す

図 3.5: 複数ドメインでの公開鍵の入手



1. CL1 → CL2 : CL1, CL2, $\{$CL1, $timestamp, life, \{$id, $(request)\}^{PK_{CL2}}\}^{SK_{CL1}}$
   CL2 へ通信要求を出す
2. CL2 → CL1 : CL2, CL1, $\{$CL2, id, $timestamp, life, (reply)\}^{PK_{CL1}}$
   CL1 へ reply を返す

図 3.6: 通信要求

第 4 章

SPLICE/AS の構成

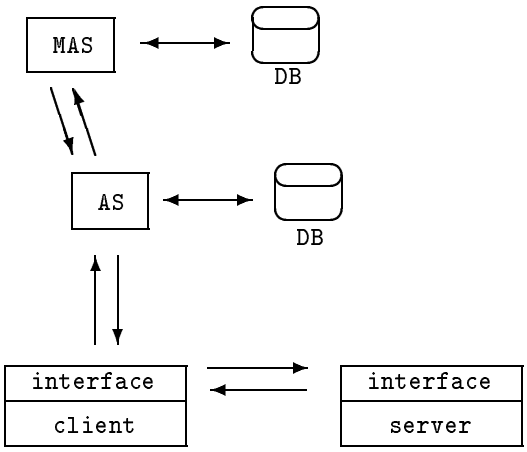


図 4.1: SPLICE/AS の構成

SPLICE/AS は以下のソフトウェアから構成される。

- プライベート鍵データベース
- 公開鍵データベース
- 秘密鍵データベース
- ユーザインターフェイス
- プログラムインターフェイス
- **Authentication Server(AS)**
- **Master Authentication Server(MAS)**

これらは図 4.1に示すような構成になっている。

4.1 AS の持つデータベース

AS は鍵の管理のために幾つかのデータベースを持っている。これらには二つの種類があり、

- ユーザの鍵の管理のためのデータベース
- AS の鍵の管理のためのデータベース

に分けられる。AS のサービス名は全て splice-as と設定されているために、サービス名だけで AS を区別することができない。AS どうしの識別には、その AS が存在するマシン名を用いる。以下、それぞれのデータベースについて説明する。

1. `ckdb`
   ユーザのプライベート鍵を登録するデータベース。データベースのエントリは以下の通り。ユーザのプライベート鍵は、DES のアルゴリズムを用いて暗号化を行う。まず、乱数などで暗号化のキーを生成し、それを一定長のブロックに分けた平文に順次適用し、暗号文の頭にキーをつける手法である。

   { ユーザの名前 : ユーザのプライベート鍵を暗号化したもの
   }

2. `pkdb`
   ユーザの公開鍵を登録するデータベース。データベースのエントリは以下の通り。

   { ユーザの名前 : ユーザの公開鍵 }

3. `skdb`
   ユーザの秘密鍵を登録するデータベース。秘密保持のため、秘密鍵はプライベート鍵で暗号化してある。データベースのエントリは以下の通り。

   { ユーザの名前 : { 利用者の秘密鍵 }^{CK} }

4. `asckdb`
   AS のプライベート鍵を登録するデータベース。データベースのエントリは以下の通り。AS のプライベート鍵は、ckdb と同様の手法でで暗号化されている。

{AS の存在するマシン名 : AS のプライベート鍵を暗号化したもの }

5. aspkdb
   AS の公開鍵を登録するデータベース。データベースのエントリは以下の通り。

   {AS の存在するマシン名 : AS の公開鍵 }

6. asskdb
   AS の秘密鍵を登録するデータベース。秘密保持のため、秘密鍵はプライベート鍵で暗号化してある。データベースのエントリは以下の通り。

   {AS の存在するマシン名 : {AS の秘密鍵 }^{CK}}

次に、データベースにアクセスするためのライブラリ群について説明する。これらは Berkeley UNIX で提供されている ndbm を利用して実現している。

1. プライベート鍵データベースにアクセスするためのライブラリ

   - ck_store(dbname,username,ck)
   - ck_delete(dbname,username)
   - ck_fetch(dbname,username)

   ck_store() はエンティティ名と公開鍵データを受け取り、それに対するエントリを新たにデータベースに追加する。もし、すでにそのエントリが存在すればそこにデータを上書きする。ck_delete() は指定されたエンティティのエントリを削除する。ck_fetch() は渡されたエンティティ名からそれに対応するデータを取りだしそのポインタを返す。もし、対応するエントリがなければ、null を返す。

2. 公開鍵データベースにアクセスするためのライブラリ

   - pk_store(dbname,username,pk)
   - pk_delete(dbname,username)
   - pk_fetch(dbname,username)

   機能的にはプライベート鍵データベースライブラリと全く同じルーチンが用いられている。

3. 秘密鍵データベースにアクセスするためのライブラリ

   - `sk_store(dbname,username,sk,ck)`
   - `sk_delete(dbname,username)`
   - `sk_fetch(dbname,username,ck)`

   秘密鍵のデータをプライベート鍵で暗号・復号化する所以外は、プライベート鍵データベースライブラリと機能的に全く同じである。これらのデータベース操作のためのライブラリは、データベースを保守するためのコマンド asdbm によって使用される。

4.2 ユーザインターフェイス

ここでは、ユーザインターフェイスとして提供されるユーティリティについて説明する。

4.2.1 asinit

```
%asinit okayama
Please answer your password.
passwd:(パスワードを入力)
Now,the SPLICE/AS service is available.
%
```

図 4.2: asinit の使用例

AS のサービスを受けるためには asinit コマンドを使用する。これを使用すると、まず利用者名を聞かれ、ついでパスワードを聞かれる。パスワードチェックが通れば、ユーザは各サービスが受けられる。図 4.2に、名前が okayama であるユーザの asinit の使用例を示す。

4.2.2 asdbm

公開鍵や秘密鍵・プライベート鍵の更新や、新規ユーザの登録・ユーザの登録削除などは asdbm コマンドを使用する。このコマンドは、AS(MAS) の存在する計算機上でしか行えない。一般ユーザが AS(MAS) の存在しない計算機（リモートマシンと呼ぶ）から鍵の更新を行う場合は、次に述べる aspasswd コマンドを使用する。また、新規登録・登録削除は管理者しか行えない。

4.2.3　aspasswd

リモートマシンにおいて、一般ユーザが、鍵の更新を行いたい場合にはaspasswdを使用する。これは、YPシステムで提供されているyppasswd[68]と概念的に同じである。

4.2.4　デーモン

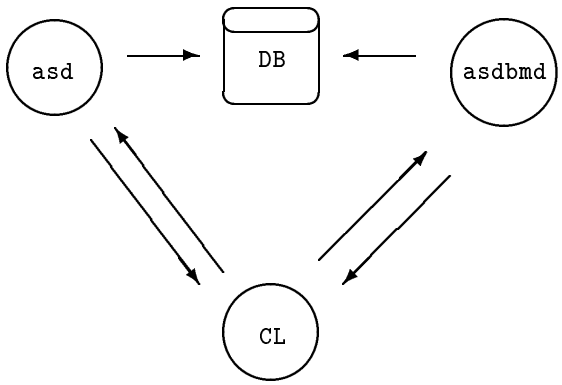


図 4.3: クライアント・asd・asdbmd の関係

AS 自身は、ユーザの鍵の更新や、新規ユーザの登録・削除などは行わない。これらを受け付けるサーバプロセスとして、asdbmd というデーモンがある。これは AS と同じ計算機上で走らせておき、asdbm コマンドや aspasswd コマンドを受け付けて処理を行う。また、AS は asd というデーモンを起動する。クライアント、asd、asdbmd の関係を図 4.3に示す。

4.3　プログラムインターフェイス

ここでは、SPLICE/AS のサービスをプログラムから直接利用するためのライブラリについて説明する。

- `as_initserv(socket,sevicename)`

  これは認証機構を必要とするサービスを提供するサーバの起動を行うためのものである。

- `get_pk(socket,sender,receiver)`

これは受信エンティティの公開鍵を得るためのものである。送信エンティティの名前および受信エンティティの名前を与えると、3.5.2および 3.5.3で述べたプロトコルにしたがって AS と通信し、公開鍵を返す。

- `authent(socket,sender,receiver,pk,sk)`

  送信エンティティの名前および受信エンティティの名前、相手の公開鍵、自分の秘密鍵を与えると、3.5.4で述べた通信要求のプロトコルにしたがって、通信相手の認証を行い、相手に返信する。

- `as_init(socket,asname)`

  これは 3.5.1で述べた AS へのログインのプロトコルにしたがってパスワードチェックが行われ、自分の公開鍵・秘密鍵および AS の公開鍵を得ることができる。前節の asinit コマンドもこれを利用して実現されている。

- `as_kchange(socket)`

- `as_pcahnge(socket)`

  as_kchange() は利用者の公開鍵・秘密鍵の更新を行い、as_pchange() は利用者のプライベート鍵（パスワード）の更新を行うものである。前節の asdbm コマンドや aspasswd コマンドは、これを利用している。

- `as_register(socket,username)`

- `as_expel(socket,username)`

  as_register() は新規のユーザ・サービスを AS に登録するためのもので、as_expel() はユーザ・サービスを AS から削除するためのものである。これらは特定のユーザ（管理者）だけが行える。前節の asdbm コマンドは、これらのライブラリを利用している。

4.4 Authentication Server(AS)

ここでは、SPLICE/AS の核をなす AS について述べる。AS は立ち上げ時に設定ファイル as.config を読み込み、それに応じて各種環境を設定する。設定ファイルでは、以下の項目について記述できる。

- MODE
  立ち上げ時のモードを選択する（AS モード・MAS モード、各モードの詳細については後述。）

- AS,ASDOMAIN
  自分のマシン名およびドメイン名

- MAS,MASDOMAIN
  （自分の上位の MAS が存在する場合）MAS のマシン名およびドメイン名

- ASLIST
  MAS モードの場合、自分の下位の AS(MAS) が存在しているマシンのリスト

- CKDB,PKDB,SKDB
  ユーザの公開鍵、秘密鍵、プライベート鍵データベースのファイル名

- ASCKDB,ASPKDB,ASSKDB
  AS の公開鍵、秘密鍵、プライベート鍵データベースのファイル名

また、as.config の記述例を図 4.4に示す。

```
MODE            as
PKDB            /usr/local/lib/as/pkdb
SKDB            /usr/local/lib/as/skdb
CKDB            /usr/local/lib/as/ckdb
ASPKDB          /usr/local/lib/as/aspkdb
ASSKDB          /usr/local/lib/as/asskdb
ASCKDB          /usr/local/lib/as/asckdb
AS              neptune
ASDOMAIN        ics.osaka-u.ac.jp
MAS             mars
MASDOMAIN       osaka-u.ac.jp
```

図 4.4: as.config の例

4.4.1 AS モード

サーバプロセスは設定ファイル as.config で AS モードを指定することによって、最も下位のドメインに位置する AS として起動される。AS は以下のような機能を持つ。

- 自分の管理しているドメイン内のエンティティに対し、必要としている公開鍵を提供する。
- 他のドメインの公開鍵を要求された時、その要求を上位の MAS に転送し、MAS から得た鍵を要求を行ったエンティティに与える。
- 利用者および管理者からの鍵の更新、新規登録、登録削除などの要求を受け付ける。

4.4.2 MAS モード

MAS は AS の上位のドメインに位置するものであるが、そのドメインにもクライアントが存在する場合もあり、また、下位の AS(MAS) もクライアントみなすことができる。したがって、MAS モードでは、AS モードの機能を含んでいなければならない。前節の AS モードの機能に加えて、MAS モードには次のような機能がある。

- 下位のサブドメインの管理を行っている AS(MAS) の鍵の情報をデータベースとして持ち、その登録・更新などの管理を行う。
- 3.5.3で述べたようなプロトコルにしたがって、下位の AS(MAS) からの公開鍵の要求を処理する。

4.5 暗号ライブラリ

SPLICE/AS で使用される暗号方式は公開鍵暗号方式と慣用系暗号方式がある。したがって、その二つについてそれぞれライブラリを用意した。

- `rsa_encrypt(key,plaintext,size)`
- `rsa_decrypt(key,ciphertext,size)`

  これらは公開鍵暗号方式のライブラリである。rsa_encrypt() は、公開鍵（秘密鍵）・平文・平文のサイズを与えると暗号文を返し、rsa_decrypt() は、秘密鍵（公開鍵）・暗号文・暗号文の長さを与えると平文を返す。

- `des_encrypt(ck,plaintext,size)`
- `des_decrypt(ck,ciphertext,size)`

  これらは慣用系暗号方式のライブラリである。des_encrypt() は、プライベート鍵・平文・平文のサイズを与えると暗号文を返し、des_decrypt() は、プライベート鍵・暗号文・暗号文のサイズを与えると平文を返す。

また、SPLICE/AS では公開鍵暗号方式に RSA、慣用系暗号方式に DES を使用しているが、この部分の変更によって他の暗号方式を使うことができる。

第 5 章

応用例

WIDE インターネットにおいて、もっとも利用者認証が必要とされるアプリケーションは、ファイル転送プログラム ftp である。そこで、SPLICE/AS を **WIDE** インターネットに実装するにあたり、Berkeley UNIX システムで提供されている ftp に対して、SPLICE/AS を利用するように改造を加える。ftp は、ネットワーク上のファイル転送プログラムで、サーバ・クライアント形式をとっている。ネットワーク上の各計算機では、サーバプロセスである ftpd（ftp デーモン）が動いており、ユーザは、ftp コマンドでクライアントプロセスを起動し、ターゲットの計算機上の ftpd と通信を行うことによってファイル転送を行う。

しかし、ftp コマンドでは、ネットワーク上の通信の際、メッセージに対して何の操作もなされていない。したがって、第三者がそのメッセージを傍受すれば、メッセージの内容がそのまま見えてしまう。これは、ftp のパスワードチェックに関しても同様であるので、そこからパスワードが他人に知られてしまう恐れがあり、ネットワークのセキュリティ上、非常に危険である。そこで、我々の開発した SPLICE/AS をこれに実装することによって、安全にファイル転送が行える環境を提供しようとするものである。

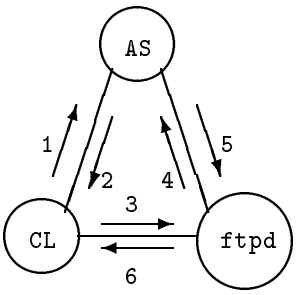
5.1 通信路確立のプロトコル

SPLICE/AS の実装にあたって、AS の果たす役割は、安全な通信路を確立することに尽きる。いったん通信路が確立してしまえば、クライアントとサーバの間で通信のための鍵を作るなどして、その通信路の安全を維持すればよい。図 5.1に、通信路確立のプロトコルを示す。

ここでは、すでにクライアントおよび ftpd は自分の公開鍵・秘密鍵および AS の公開鍵をもっているとする。また、クライアントと ftpd が同じドメイン上にない場合は、 3.5.3 で示されたようなプロトコルを使用する。

5.2　通信路確立後のクライアントと ftpd との通信

さて、一度安全な通信路が確立すれば、AS がこれに関与する必要はない。ここから後はクライアントと ftpd の間で通信を行えばよい。このとき、クライアントは二つの方法が選べる。一つはメッセージに対して暗号化を行わずに通信する方法で、もう一つは暗号化を行って通信する方法である。なぜならば、第三者に内容を知られたくないファイルのみに暗号化を行えばよいのであって、そうでなければ、暗号化をしない方が暗号化のオーバーヘッドがないため、高速にファイル転送が行えるからである。前者は従来通りのものなので、ここでは後者を考える。この暗号化には、慣用系暗号を使い、鍵の生成は、ftpd が行う。まず、クライアントは、通信要求の際に、暗号を使って通信したいことを ftpd に知らせる。すると、ftpd はそれを受けて、通信用の鍵 key を生成し、クライアントへ送り返す。これは図 5.1の後に図 5.2で示すような通信を行えばよい。



1. CL → AS:CL,ftpd,id_1
   AS に ftpd の公開鍵を要求
2. AS → CL:AS,$\{\mathrm{CL},\mathrm{AS},\mathrm{id}_1,\mathrm{PK}_{ftpd}\}^{SK_{AS}}$
   CL に ftpd の公開鍵を返す
3. CL → ftpd:CL,ftpd,$\{\mathrm{CL},timestamp,life,\{\mathrm{id}_2,(request)\}^{PK_{ftpd}}\}^{SK_{CL}}$
   ftpd に通信要求を出す
4. ftpd → AS:ftpd,CL,id_3
   AS に CL の公開鍵を要求
5. AS → ftpd:AS,$\{\mathrm{ftpd},\mathrm{AS},\mathrm{id}_3,\mathrm{PK}_{CL}\}^{SK_{AS}}$
   ftpd に CL の公開鍵を返す
6. ftpd → CL:ftpd,CL,$\{\mathrm{ftpd},\mathrm{id}_2,timestamp,life,(reply)\}^{PK_{CL}}$
   CL へ reply を返す

図 5.1: ftp における通信路確立のプロトコル

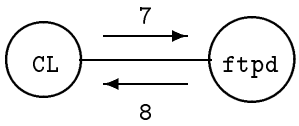
また、通信が長時間になると、途中で key の更新が必要となる。これは、ある一定の期間（key の長さに依存する）が経つと、ftpd が新たな key を生成して、クライアントにそれを知らせればよい。そのときのメッセージは次のようになる。

ftpd → CL : ftpd,CL,[ftpd,id,$timestamp,life,(change\_key)$,newkey]^{oldkey}

5.3 ftp への実装

これまでに述べたような機能を実現するために、以下の手順で実装を行う。

- クライアントプロセスを起動するためのコマンド ftp へ、4.3で示したユーザインターフェイスのライブラリを組み込む
- ftpd に 4.3で示したユーザインターフェイスのライブラリを組み込む
- ftpd に通信用の鍵の生成ライブラリを組み込む



7.　CL → ftpd:CL,ftpd,$\{$CL,$timestamp$,$life$,$\{$id,$(use_cryption)\}^{PK_{ftpd}}\}^{SK_{CL}}$
通信に暗号を使用することを要求

8.　ftpd → CL:ftpd,CL,$\{$ftpd,$timestamp$,$life$,$\{$id,$(reply)$,key$\}^{PK_{CL}}\}^{SK_{ftpd}}$
通信用の鍵 key を生成し、CL に送る

図 5.2: 通信用の鍵の要求

第 6 章

今後の課題

今後に残された課題として、以下の点が挙げられる。

- 鍵のキャッシュの問題
  あるユーザが通信を安全に行いたい場合、これまでに述べたように AS に相手の公開鍵を要求しなければならない。ここで、同じ相手としばしば通信を行う場合を考えると、その度に AS と通信を行う必要が生じる。このオーバーヘッドを軽減するためには、鍵のキャッシュが効果的であると考えられる。つまり、最初の通信で得た相手の公開鍵を持っておき、次回からはその鍵を使って直接相手と通信を行う。これによって、AS の負荷を軽減でき、ユーザも通信のオーバーヘッドが減少するというメリットが生じる。

  鍵のキャッシュを行う場合に考慮しなければならないのが、キャッシュする鍵の安全性と、鍵の更新である。鍵の安全性については、キャシュする鍵は公開鍵であるので、安全性の問題は発生しない。

  一方、鍵の更新では問題がある。例えば、ユーザ A は ftpd と頻繁に通信を行うため、ftpd の公開鍵をキャッシュしている場合を考える。ftpd が自分の公開鍵・秘密鍵の更新を行った場合、A がキャッシュ内の鍵を利用すると、ftpd の古い公開鍵で暗号化を行い、ftpd にメッセージを送ることになる。当然、ftpd はこのメッセージを正しく復号化することはできない。この時、ftpd では、このメッセージが古い鍵を使ったことに起因するものなのか、それともメッセージ自体が不正であったり、途中で第三者の妨害が入ったりしたのかを認識することはできない。また、A では ftpd からの反応がないので、コネクションを張るために何度も再送を行うことになる。

  以上のように、鍵のキャッシュに対しては何らかのチェック機構や、整合性を保つ機構を導入しなければ、鍵入手の効率化どころか、かえって通信の妨げとなってしまうと考えられる。したがって、このためのプロトコルの開発、システムの拡張が要求される。

- プロトコルの検討
  SPLICE/AS では、Needham 等によって提案された、公開鍵暗号を用いた利用者認証の手法に基礎とし、それを拡張することによって広域ネットワークに適応させている。これをネットワークアプリケーションに実装した場合、このプロトコルの安全性を検討し、実際のネットワークアプリケーションでの SPLICE/AS の利用のガイドラインを作ることが必要である。

- 性能評価
  SPLICE/AS の実装のガイドライン作りにあたって、AS の性能評価も必要になると考えられる。性能評価のポイントとしては、

  1. プロトコルのオーバーヘッド
  2. 階層レベル数とオーバーヘッドとの関係
  3. ユーザ数と鍵を取ってくるまでの時間の関係
  4. 暗号ルーチンの性能

  などが挙げられる。

- SPLICE/AS の運用
  SPLICE/AS は、1990 年 4 月より **WIDE** インターネットにおいて実際の運用を開始する予定である。現在の SPLICE/AS の管理・運用面から見たユーティリティは、不足していると考えられる。今後、実際の運用を通して、問題点を検討していく必要がある。

第 7 章

あとがき

本研究では昨年度の研究に基づいてネットワーク上のサービスに利用者認証のプロトコルの詳細設計およびソフトウェアの構成を行い、実際に利用者認証の機能を提供するサーバとして AS、さらに、それを分散環境に拡張するものとして MAS の実装を行った。

SPLICE/AS の作成に当たって公開鍵方式の暗号が必要であったため、その暗号化アルゴリズムが明らかにされている RSA 方式を用い、ソフトウエアでの実現を行った。

SPLICE/AS では、暗号化を行う部分は利用者認証システムの中で独立に考えることができ、同じ暗号系の暗号方式であれば容易に置き換えが行なえるような設計を行なうことが可能である。従って、将来ハードウエア等でさらに有効な公開鍵暗号が提供されれば、暗号ルーチンに対するインタフェースに変更を加える事により、それを用いる事ができる。

現時点では、各種ライブラリおよび AS、MAS の作成がほぼ完了しており、**WIDE** インターネットにおける実装を行うための細かい部分の変更・修正を行っている。

今後、この部分を完全なものとし、この応用として ftp などのアプリケーションプログラムに SPLICE/AS を組み込み、AS の性能向上に向けてより一層の改善を施していかなくてはならない。