プログラムを改変せず暗号通信を実現する機構の設計と実装

慶應義塾大学 環境情報学部 村井研究会

三谷 洋司

指導教員
徳田英幸
村井純
楠本博之
中村修

1999年2月22日

概 要

インターネットの普及により、コンピュータネットワークを利用した様々な通信が広く行われるようになり、それと共に不正アクセスなどが増加している。

ネットワークの盗聴やなりすましを防ぐためには、通信路の暗号化や認証が必要である。しかし、既存のプログラムに改変を行い、暗号化等に対応させるためには多大な労力を必要とする。

そこで、本研究では、ユーザーや管理者に出来るだけ負担をかけず、通信プログラムを暗号化に対応させ、安全な通信環境を作ることを目的とする。そのため、既存のプログラムに改変を行わずに暗号化通信に対応させる目的で、通信の暗号化を行う中継機構を設計し、実装を行なった。

また、通信の暗号化を行う中継機構の実装により、簡単な管理作業で暗号通信を実現するという目的を達成し、その結果、本研究の機構が他の機構による実装に劣らない通信性能を保ちながら管理作業の簡略化が行えることを確認した。

Abstract

This paper proposes a system to encrypt a communication and make the communication program secure without changing the original program itself and also without making user and administrator do more duty .

To avoid tapping and spoofing on the network , we must encrypt the communication route or use certification to ensure the communication . But , to change the existing software hardly . to do those encryptions , we need to work on the existing software .

So , to make it possible for existing software to encrypt communication without changing the original program itself , I propose a system to relay encrypted communication to the program with decrypted data, and vice versa .

目 次

図目次

表目次

1　序論

1.1　はじめに

インターネットの普及により、コンピュータネットワークを利用した様々な通信が広く行われるようになり、それと共に盗聴やなりすましによる不正アクセスが増加している。ネットワークの盗聴やなりすましを防ぐためには、通信路の暗号化や認証が必要である。しかし、既存のプログラムを改変し、暗号化等に対応させるためには多大な労力を必要とする。

本研究では、ユーザーや管理者に出来るだけ負担をかけず、通信を暗号化に対応した安全な物にすることを目的とする。そのためには、既存のプログラムに改変を行わず暗号化通信に対応させることが必要である。

これにより、プログラムの改変や再コンパイルをすることなく暗号化に対応出来るという利点と、プログラムのソースが手に入らなくてもそのプログラムを暗号化に対応させる事が出来るという利点や、管理のための作業を簡略化できるという利点が生じる。

本研究では、セッション層で暗号化を行う暗号化プロトコルSecure Sockets Layer(SSL)を利用して暗号化通信を実現するために、SSLとプログラムの中継を行う機構を設計し実装する。

1.2　本論文の構成

第2章で現在のネットワークにおけるセキュリティに関する問題点を指摘し、通信の暗号化の必要性と、それに関わる問題点を指摘する。

第3章では、問題点を解決するための新しい機構について述べ、第4章では、関連研究について述べる。
第5章と第6章では、本研究で実現する機構の設計と実装について述べる。

第7章で本機構の評価及び考察を行い、第8章で今後の課題と結論について述べる。

2 背景

2.1 現状

近年のパーソナルコンピュータやインターネットの普及により、企業や家庭でもコンピュータが普及し、インターネットという言葉も日常生活において使われるようになった。それに伴い、インターネットは学術ネットワークから、世界規模の情報基盤へと発展し、重要な社会のインフラストラクチャとして認識される様になった。

ネットワークの普及に伴い、インターネットに接続されたコンピュータシステムに不正に侵入し、情報の搾取・破壊や運用妨害を行なう、クラッカーと呼ばれる攻撃者による被害が増えつつある。また、その目的も単に興味本位の「悪戯」のレベルのものから、特定の企業情報を盗む産業スパイまで幅広いものになっている。

不正なシステム侵入に対する緊急対応を中心に、セキュリティ技術の教育・啓発活動を行なっている、JPCERT/CC(コンピュータ緊急対応センター)の統計によると、不正アクセスは増加傾向にあり、このまま放置すると、無防備なユーザーの被害が急激に増大すると予想されている(図 2.1)。

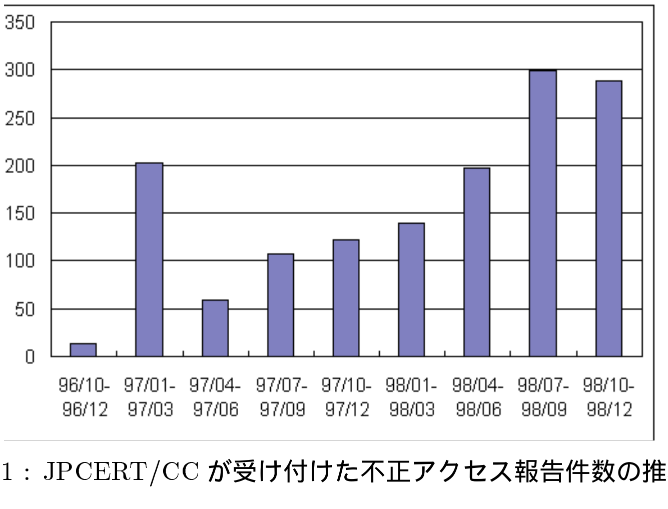

図 2.1：JPCERT/CC が受け付けた不正アクセス報告件数の推移[1]

---

[1] http://www.jpcert.or.jp/anm/stats.html

このような状況から、インターネットにおけるセキュリティ対策は重要である。しかし、インターネットの広域性と開放性から、問題の解決は難しい。
現在のインターネットでは、暗号化されていないデータがネットワーク上を通過するために、第三者が盗聴や改ざんを容易に行なう事が可能である。

例えば、TELNETやFTP、あるいは遠隔ログインプログラムなど、よく使われているプロトコルのほとんどが、パスワードを暗号化せずにネットワークに送信する。そのため、第三者が容易にパスワードの盗聴を行なうことができ、自分が損害を受けるばかりか、盗聴されたパスワードを不正使用される事で他人にも損害を与える可能性が有る。

このような危険性をなくし、安全な通信を実現するためには、インターネット上を通過するデータを暗号化し、データの正当性を保証する事が必要である。

しかし、暗号化による安全な通信を行なうためには、暗号化に対応した新しいプログラムを作成したり、既存のプログラムに暗号化のための改変を行う必要があり、多大な労力を必要とする。また、暗号化に対応したプログラムは、対応していないプログラムにくらべ、操作手順が増えるなどの操作上の負担をユーザーや管理者に課すことになる。そのため、暗号化による安全な通信プログラムがなかなか普及しないのが現在の状況である。

2.2 ネットワークにおけるセキュリティ

ネットワークやシステムへの侵入や攻撃の形態は以下のように分類することができ、それぞれに適切な対処を行えば、安全に通信を行うことが出来る。

2.2.1 盗聴、改竄

聴盗とは、ネットワーク上を流れる情報を盗み見ることである。

インターネットのような開放的な広域ネットワークは、多くの中間者による情報のリレーによって成り立っている。そのため、悪意のある第三者が居ればネットワークに流れる情報は簡単に見られてしまう。
そうなると、プライバシーが侵害されるばかりか、暗号化されていないパスワードを盗聴されることで、自分なりすまして犯罪を起こされる可能性もあり得る。また、中間者が通信の内容を傍受し、これを改竄して中継することも考えられる。

このような問題に対処するには、通信路の暗号化や通信内容の正当性のチェックなどを行う必要がある。暗号化することにより、通信内容を見られても安全であり、通信内容の正当性のチェックをすることで、通信内容を改竄されてもこれを発見することができるからである。

2.2.2　なりすまし

なりすましとは、盗聴や何らかの方法でパスワードを入手して、本人に成り代わってシステムに侵入したり、IPアドレスを偽って、本来通信すべきコンピュータに成り代わって通信を傍受・妨害することである。

これに対処するには、ユーザやコンピュータの認証が必要となる。

3　本研究の目的

本研究では、ユーザーや管理者に出来るだけ負担をかけずに通信プログラムを暗号化に対応させるため、既存のプログラムに改変を行わず暗号化通信に対応させる機構の開発を行なう。

本研究の目的を以下にあげる

- 通信路の暗号化
  安全な通信を実現するために、通信データを暗号化する。さらにまたデータの正当性を保証する。
- 既存のシステムやアプリケーションに変更を加えない
  管理作業の簡略化を行なう。
  従来行なわれてきた、ソースコードの改変や、暗号通信プログラムの繁雑な操作は、ユーザーや管理者に負担をかけるものである。管理作業の簡略化を行なうことのできる機構を実現する。
- 通信性能の保証
  本研究で提案した機構による実装が、他の機構による実装と比較して、速度などの点で大幅に劣るならば、暗号通信プログラムとして利用価値が無くなる。従って、通信性能も考慮したシステムを作成する必要がある。

  そのため、他の機構による実装と比較して、同等の性能を目指す。

4 関連研究

本章では、既存のプログラムに改変を行わずに通信路の暗号化やユーザの認証を実現することを目的としている関連研究について述べる。

4.1 SOCKS(SOCKSV5)

SOCKSとは、NEC Corporationによって開発されている、クライアント/サーバ環境のためのプロクシプロトコルである。認可されたアクセスは許可するが、望まれないアクセスを見つけて拒否するという、通信相手の認証を行うことができる。

さらに、SOCKSV5では、Username/Password及びGSS-API[2]による認証が可能になっており、この技術を元にインターネット上に暗号化された通路を確保し、インターネットのデータ通信による盗聴や改竄といった危険性に対してインターネットに接続するユーザー間を閉域化し、インターネットを仮想的なプライベート網として利用する技術が確立している。

また、SOCKS5はIETFににて標準化されており、現在、

- SOCKS Protocol Version 5 (RFC1928)
- Username/Password Authentication for SOCKSV5(RFC1929)
- GSS-API Authentication Method for SOCKS Version 5(RFC1961)

の3つがRFCになっている。

次に、SOCKSの機構について述べる。

SOCKSはSOCKSサーバーとSOCKSクライアントライブラリで構成されている。SOCKSを利用するためには、クライアントアプリケーションの代理をつとめるSocksProxyサーバを用意する必要がある。

また、クライアントがSOCKSクライアントライブラリによってSOCKSに対応させる必要があるが、SOCKSV5では、シェルスクリプトrunsocksなどを利用して既存のネットワークアプリケーションを動的にSOCKS化出来るようになっている。

---

[2] GSS-API(Generic Security Service API): The Open Groupが規定した、セキュリティメカニズムに依存しない汎用セキュリティインターフェイス

実際の動作の仕組みは、クライアントアプリケーションはSocksProxyサーバにコネクションリクエストを渡し、SocksProxyサーバはクライアントを認証してからコネクションを確立し、アプリケーションサーバへの接続を行うという機構になっている。（図 4.1）

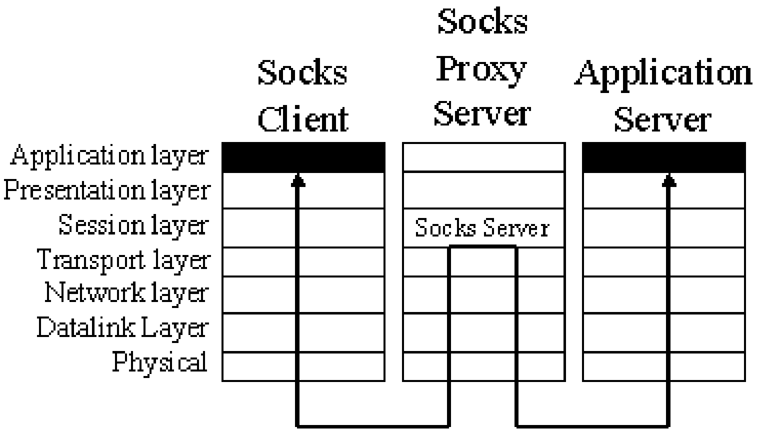

図 4.1：SOCKS によるプロクシの概要

この機構によって実現される主な利点と欠点は以下の様になる。

利点：

- クライアント/サーバ間の直接的なアドレスの授受を不要とし、内部アドレス構造を隠蔽する。
- セッション層におけるプロクシサーキットを確立する。
- アプリケーションゲートウエイのように、アプリケーション毎に中継デーモンを構築する必要がない。

欠点：

- クライアントがSOCKSプロトコルを利用するために、SOCKSライブラリを用いた再構築が必要となる。（SOCKSV5ではプログラムや再リンクは不要になっている。）
- UDPベースの通信をサポートできない。（SOCKSV5ではUDPベースの通信をサポートしている。）
- SOCKS自身では認証機能を持っていない。（SOCKSV5では認証機能をサポートしている。）

4.2 IPsec

IPsec は、IP パケットレベルでセキュリティを確保するプロトコルである。

IPsec は IPv4 にはオプションで導入でき、IPv6 では標準で備えられる。このセキュリティ機構は IP レイヤに実装されるステートレスな方式であり、パケットの認証と内容の暗号化をサポートする。この方式は IP レイヤ以上の TCP などのトランスポート層やアプリケーション層から独立しているので IP を使う任意のプロトコルやアプリケーションに透過的に適用できる。(図 4.2)

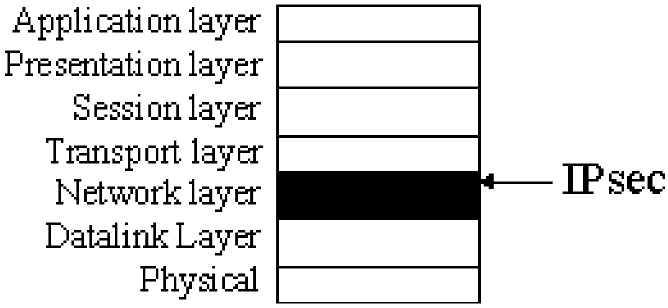

図 4.2:IPsec の OSI7 層モデルでの位置付け

また、IPsec は IETF において、IPSEC Protocol Working Group として、1992 年から議論されてきたものである。IPsec に関する RFC は、

- Security Architecture for the Intenet Protocol (RFC1825)
- IP Authentication Header(RFC1826)
- IP Encapsulating Security Payload(RFC1827)

を始めとして、多数の RFC になっている。

この機構によって実現される主な利点と欠点は以下の様になる。

利点 :

- IP を使う任意のプロトコルやアプリケーションに透過的に適用できる。

欠点 :

- ネットワーク層でのサービスのため、kernel の改造など、システムの改変が必要であり、管理の手間がかかる。

4.3 SSH(Secure Shell)

ssh(Secure Shell) はアプリケーション層でセキュリティを確保するプロトコルである。

ssh は IETF の Secure Shell Working Group で標準化されていおり、現在、drfat-ietf-secsh-architecture-02.txt と他の 3 つの Intenet Draft になっている。

ssh は、rsh/rlogin などの rcmds 系ツールに対してセキュリティ強化を行うパッケージとして開発され、ssh は rlogin（リモートログイン）rsh（リモートシェル）、そして rcp（リモートファイルコピー）に代わるものとして開発された。

ssh は強力な認証と安全ではない経路を通じての安全な通信を提供する。加えて、ssh は安全な任意 TCP 接続の安全な中継を提供している。

SSH の port forwading 機能とは、一種の proxy 機能である。
SSH client がある指定した SSH client の動いている host の port と、SSH daemon が動いている host を経由して任意の host の port を forward することができる。

図 4.3:SSH の port-forwarding の書式例

```
ssh -L 8023:remotehost:23 remotehost -l username
```

図 4.3 の場合、8023 は自分が localhost、つまり SSH client を動かしている host で開きたい port であり、23 は remort の host につないで欲しい port である。ここでは、23 つまり TELNET なので、localhost の 8023 と remote の 23 とを暗号化して forward するように指定している。-l username は username という user-name で login するという指定である。その後、localhost の 8023 番に繋ぎに行くことで、SSH によって暗号化された通信路を使用することができる (図 4.4)。

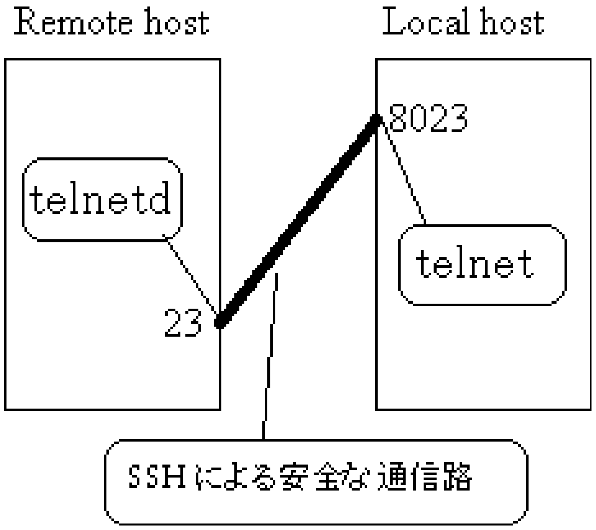

図 4.4: SSH の port-forwarding

つまり、port-forwarding の機構によって、既存のプログラムに改変を行わずに、通信路の暗号化を行うことができる。

この機構によって実現される主な利点と欠点は以下の様になる。

利点 :

- 既存のプログラムやシステムに全く改変を行わず、通信路を暗号化できる。

欠点 :

- port-forwarding の管理が手間がかかる。

4.4 stelsock

SSL Telnet Windows Sockets(STelSock) は、NEC Corporation によって開発された Windows Sockets DLL である。

これは、TELNET を利用するアプリケーションが、SSL telnetd と通信をするために、ライブラリを入れ替えることによって、Secure Sockets Layer プロトコルの暗号通信機能を利用できるようにしたものである。

この機構によって実現される主な利点と欠点は以下の様になる。

利点 :

- 既存のプログラムに改変を行わず、通信路を暗号化できる。

欠点 ：

- 使用できるプラットホームが少ない。
- システムに改変を加える必要がある。

4.5 bjorb

bjorb は、Hitachi Microsoftware Systems, Inc. によって開発された、TCP ベースの通信を Secure Sockets Layer で暗号化や復号化を行ないながら中継する secure relay ソフトウェアである。

SMTP, POP3, HTTP,NNTP など、固定ポートな TCP ベースの通信を Secure Sockets Layer で暗号化や復号化を行ないながら中継する。

Secure Sockets Layer 認証書によるクライアント認証を行う機能や、特定の IP アドレスやネットワークに対してアクセスを許可 / 拒否をする機能、など、豊富な機能が使用できる。

この機構によって実現される主な利点と欠点は以下の様になる。

利点 ：

- 既存のプログラムに改変を行わず、通信路を暗号化できる。

欠点 ：

- FTP プロトコルのような非 static port な通信には完全に対応していない。

4.6 Secure Sockets Layer

ここでは、トランスポート層とアプリケーション層の中間で暗号通信機能を提供するプロトコル、Secure Sockets Layer について説明する。

Secure Sockets Layer は、クライアント、サーバ間の通信において盗聴防止や改ざん防止、認証機能を提供するプロトコルである。

Netscape Communications 社の Alan O. Freier, Philip Karlton そして、Paul C. Kocher により Internet draft として提案されており、現在、The

SSL Protocol Version 3.0 が 1996 年 11 月に draft-freier-ssl-version3-02.txt として提案されている。そして、IETF の Trnasport Layer Security Working Group で標準化がされている。

DES や RC4 などの対称暗号アルゴリズムにより通信内容の暗号化を行い、盗聴防止を実現し、RSA などの非対称暗号アルゴリズムで通信相手を認証し、認証機能を実現し、MD5 などのハッシュ関数で改ざんが無いことを確認し、改ざんの防止を実現している。

Secure Sockets Layer は、トランスポート層とアプリケーション層の中間で暗号通信機能を提供するプロトコルである。(図 4.5) このため、Secure-HTTP のようにアプリケーション層に変更を加える必要がないので、プロトコルの汎用性が無くなることは無く、IPSec のようにネットワーク層に変更を加える必要がないので、トランスポート層以上に対する影響が生じることも無いという利点を生じる。

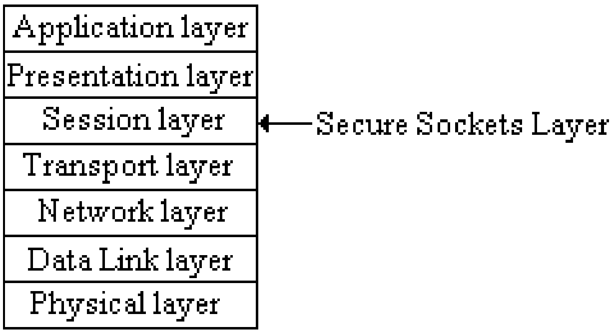

図 4.5:Secure Sockets Layer の構成

現在、Secure Sockets Layer プロトコルを元にいくつかの実装が公開されている。

- **SSLREF** Netscape 社が実装したパッケージ。アメリカ国外に持ち出すことはできない。
- **SSLeay** オーストラリアの Eric Young 氏が開発したパッケージ

また、Secure Sockets Layer プロトコルは、二つの層から構成されており、

- トランスポート層の直上に置かれる SSL Record Protocol
- その上でサービスを受ける SSL Handshake Protocol

SSL Record Layer Protocol は、上位のプロトコル群のカプセル化を担当しており、圧縮、配送などの役割を担っている。SSL Handshake Protocol（図4.6）は相手の認証や通信に使用する暗号のネゴシエーションを行っている。

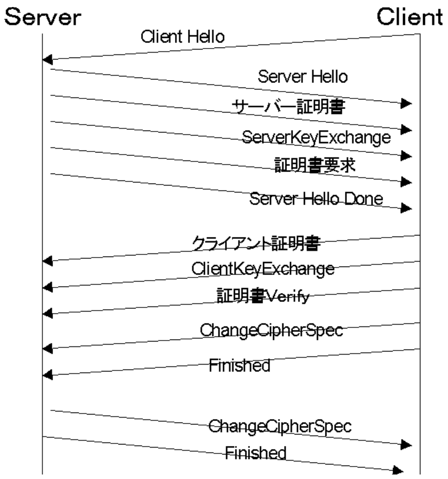

図 4.6: SSL Handsake protocol

このことにより、開発者はより簡単に認証や暗号通信を利用することが出来る。

5 設計

本章では、既存のプログラムに改変を行わず、暗号通信を可能とするための手法を説明する。

5.1 設計方針

暗号通信へのより容易な利用を目的とする場合、現状では、暗号化に対応したプログラムは、対応していないプログラムにくらべ、操作手順が増えるなどの操作上の負担をユーザーや管理者に課すことが多い。そのため、暗号化による安全な通信プログラムがなかなか普及しないのが現在の状況である。

従って、既存の暗号化プログラムでは、

- 管理者にとっては、システムの変更などや、アプリケーションの改変の負担が課される。
- ユーザーにとっては、操作手順が増えるなどの操作上の負担を課される。

という欠点が存在する。その欠点を解決するために、以下に設計方針を示す。

- 既存のシステムやアプリケーションに変更を加えない
  IPsec や IPv6 のセキュリティオプションなどの、ネットワーク層での暗号通信への対応を行う機構が提案されている。

  この方式では、kernel などのシステムの基本となるソフトウエアへの変更が必要となる。また、IPv6 の様なネットワーク層での変更は、トランスポート層以上への影響が大きくなることが考えられる。

  また、暗号通信に対応させるためにアプリケーション層でのプログラムの改変を行うためには、ソースの入手と改変、さらにコンパイルが必要となり、これらは非常に煩雑な作業となる。

  その他にも、ソースが入手できない場合が考えられ、このような場合には既存のプログラムを暗号通信に対応させることができない可能性がある。

  そのため、本研究では既存のシステムやアプリケーションに変更を加えず暗号通信に対応させる機構を実現する。

- 煩雑な管理作業を必要としない
  既存のシステムやアプリケーションに変更を加えずに暗号通信に対応させる機構には、SSH(Secure Shell) の port-forwarding 機構を用いる方法などがある。

この方式を用いて暗号通信を行うためには、第 4.2.3 節で示したように、暗号通信を行ないたいプログラムごとに煩雑な設定が必要となる。

そのため、本研究では、暗号通信に対応させるための一元的な管理機構を実現することにより、煩雑な管理作業を必要としない機構を実現する。

5.2　暗号化中継機構によるサーバプログラムの暗号通信対応

まず、Secure Sockets Layer がセッション層で暗号化サービスを提供していることと、ネットワーク上でのサービスを行う様々なサーバプログラムの多くが inetd 経由で起動されていることに注目した。

本研究では、Secure Sockets Layer がセッション層で暗号化サービスを提供していることを利用して、既存のシステムやアプリケーションに変更を加えず暗号通信に対応させる機構を実現する。また、inetd を利用することにより、数々のサーバプログラムの一元管理が可能とすることで、煩雑な管理作業を必要としない機構を実現する。

サーバプログラムが Secure Sockets Layer の提供する通信の暗号化機能を利用するには、通常はサーバプログラムのソースコードの改変とコンパイルが必要であり、手間がかかる。また、ソースコードを手に入れられない場合も考えられ、その場合は Secure Sockets Layer の提供する通信の暗号化機能を利用することができない。

そこで、暗号化を行う中継機構を実現することによりサーバプログラムに改変を加えることなく Secure Sockets Layer のセッション層での暗号化サービスを利用することができる。（図 5.1）

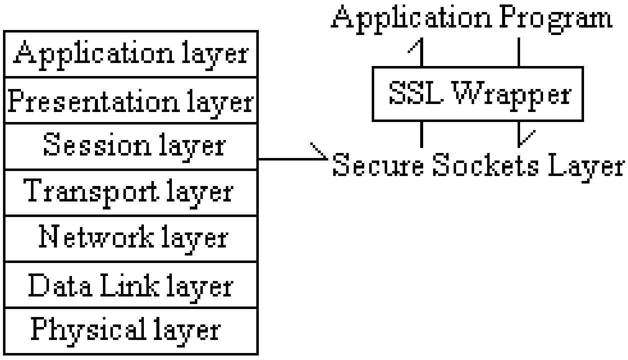

図 5.1: 暗号化中継機構の位置付け

5.3 暗号化中継機構の設計

本研究で実現する暗号・複合を行う中継機構は、暗号通信機能の無いサーバプログラムをSecure Sockets Layerに対応させることにより、Secure Sockets Layerで通信するクライアントに対しての暗号通信を可能にする。

そのために、暗号化中継機構は、暗号通信機能の無いサーバプログラムとは暗号化を行わず通信し、Secure Sockets Layerで通信するクライアントに対してはSecure Sockets Layerでの通信を行う。

図5.2で示すように、Rmote HostのSecure Sockets Layerで通信するクライアントが、Local Hostのinetdに起動される暗号化中継機構と暗号化された通信路で通信することで、Remote HostとLocal Hostの間での通信を盗聴されても安全にすることができる。さらに、暗号化中継機構の持つサーバ証明書により、Local Hostを認証することが可能になる。

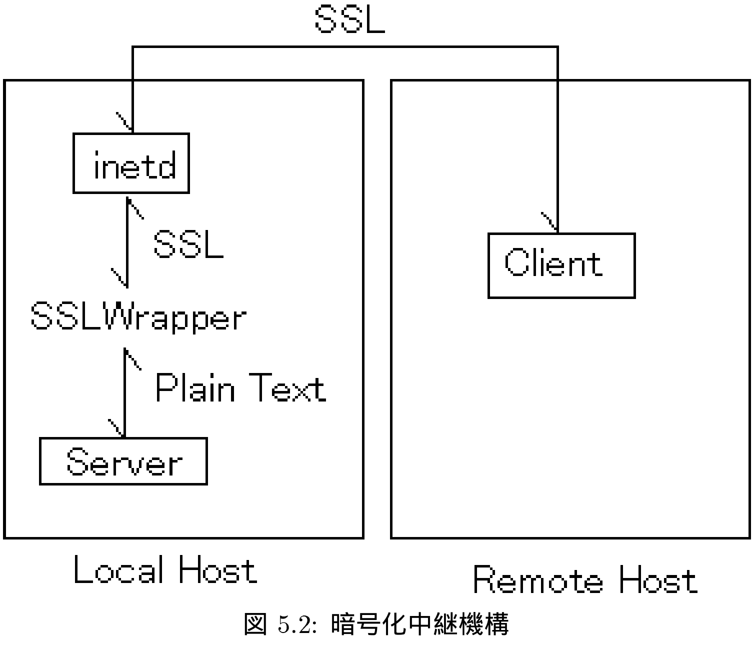

図5.2: 暗号化中継機構

Secure Sockets Layer の暗号化機能を利用するには、socket を使って相手との間の通信路を確保する必要がある。そこで、何らかの形でサーバプログラムとクライアントプログラムの間の通信路を中継機構が保持しなければならない。

そのため、暗号化中継プログラムは、inetd から起動する方法を取る。inetd.conf で暗号化中継プログラムの設定を行うため、数々のサーバプログラムの一元管理が可能となる。

inetd から起動された暗号化中継プログラムは fork した後、子プロセスはコマンドラインに指定されたサーバプログラムを exec し、親プロセスはサーバープログラムとクライアントプログラムの中間に立つ。そして、クライアントプログラムとは Secure Socket Layer での暗号化された通信を行い、サーバープログラムとは暗号化されていない平文での通信を行う (図 5.3)。

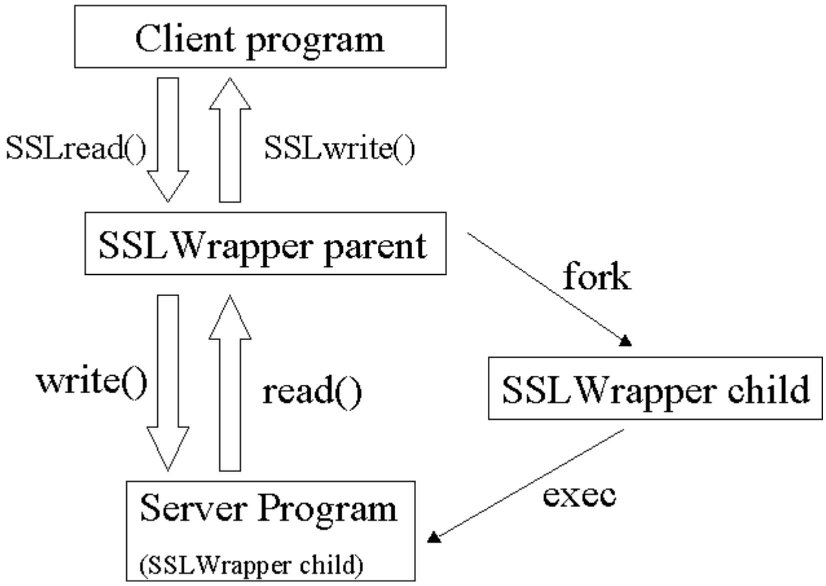

図 5.3:暗号化中継プログラムの機構

5.4 port が変わるサーバプログラムへの適用

暗号化中継プログラムはサーバプログラムの代わりに inetd から起動する方式を取るため、使用する port が変動するサーバプログラムに対して問題を生じる場合がある。

具体的な例として、ftpd は、データ転送時等に暗号化中継機構を通さない、ポート番号が不定な新しい通信路を作る。また、ftp などの新しい通信路を作るアプリケーションプログラムでは、クライアントが通信するポートを指定して通信を行う。

これに対処するため、暗号化中継プログラムがクライアントのポート番号指定を ftpd より先に取得し、暗号化中継プログラムがクライアントが指定したポートで待つ。そして ftpd には別のポートを指定し、そのポートと暗号化中継プログラムは平文で通信し、暗号化中継プログラムとクライアントは Secure Sockets Layer で通信を行う等の対処を行う必要がある。（図 5.4）

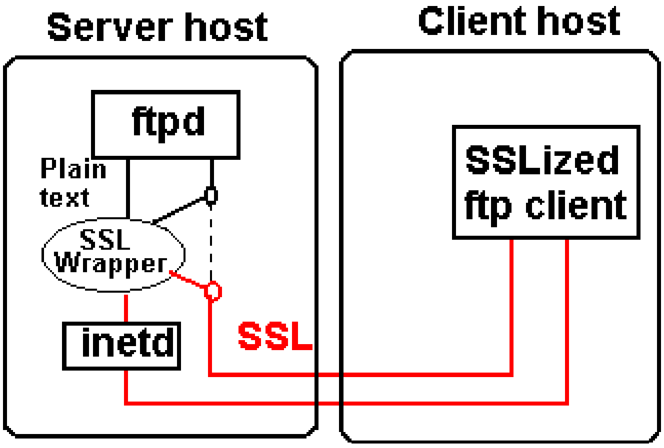

図 5.4:新しい通信路を作るプログラムへの対処

6　実装

前節では、暗号化中継機構についての設計をした。本節ではそれを踏まえ、本研究で行った実装方法について述べる。実装は、FreeBSD 2.2.7-RELEASE 上で行った。

今回は、使用する port が動的に変わらないサーバプログラムに対しての実装を行い、本方式により通信路の暗号化が実現できるか、管理作業の簡略化が有効であるか等を検証した。

まず、暗号化中継プログラムは、Secure Sockets Layer プロトコルでの、サーバとして動作するので、サーバー用証明書と鍵を必要とする。今回は、テスト用の自己署名を行った証明書を使用した。

クライアントから inetd への接続要求が来たら、inetd は本実装の暗号化中継プログラムを起動する。暗号化中継プログラムはクライアントに暗号化中継プログラムの証明書を送り、クライアントが証明書を持っている場合は、暗号化中継プログラムに証明書を送る。この時点で、通信路の暗号化が終了する。

以下は、暗号通信の初期化部分を抜き出したものである。

図 6.1: 暗号通信の初期化部分

```
#define SSL_KEY_FILE "/home/yoji/sslw.key"
#define SSL_CRT_FILE "/home/yoji/sslw.crt"



  SSLeay_add_ssl_algorithms();

  method = (ssl_ver == 3) ? SSLv3_method() : SSLv2_method();
  ctx = SSL_CTX_new(method);

  SSL_CTX_use_RSAPrivateKey_file
(ctx, SSL_KEY_FILE, SSL_FILETYPE_PEM);
  SSL_CTX_use_certificate_file
(ctx, SSL_CRT_FILE, SSL_FILETYPE_PEM);

  con = SSL_new(ctx);

  SSL_set_fd (con, fileno (stdin));
  if(SSL_accept (con) < 0){
      exit(1);
  }
```

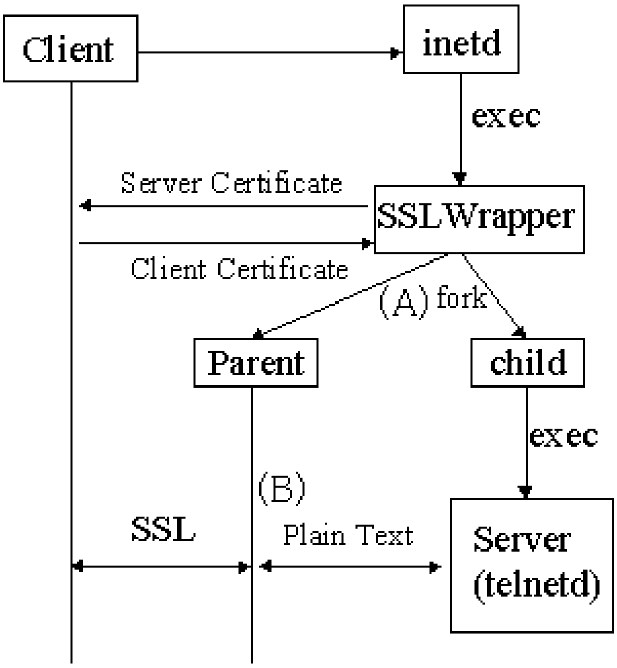

図 6.2:本実装の処理機構

(A)inetd から起動された暗号化中継プログラムは fork した後、子プロセスはコマンドラインに指定されたサーバプログラムを exec し、親プロセスはサーバープログラムとクライアントプログラムの中間に立つ。

次に、サーバプログラムと暗号化中継プログラムが通信するために socketpair() を実行することによって、暗号化中継プログラムがサーバープログラムとクライアントプログラム両方への通信路が確保される。

(B) クライアントプログラムとは Secure Socket Layer での暗号化された通信を inetd からの標準入出力に対して行い、サーバープログラムとは socketpair() によって得られたファイルディスクリプタに対して暗号化されていない平文での通信を行う。

以上のような方法によって、暗号化中継機構を実現している。

7　評価

本研究では、ユーザーや管理者に出来るだけ負担をかけず、通信プログラムを暗号化に対応した安全な物にするため、既存のプログラムに改変を行わず暗号化通信に対応させる機構の設計と実装を行った。

そこで本章では、行われた実装が本研究の目的を満たしているかどうか評価及び検討を行う。

7.1　評価項目

ここでは、本研究の目的に基づいた評価項目を設定した。評価を行う項目の詳細は以下の通りである。

- 通信路の暗号化機能の検証
  本研究の目標である安全な通信路の提供が確実に行われているかを検証する。
- 通信性能の比較
  本研究で提案した機構による実装を、他の機構による実装と比較し、検証する。ある大きさのファイルの転送に必要な処理時間を測定することにより、本研究の実装の性能を評価する。
- 管理作業の比較
  本研究の機構による実装の管理が、他の機構による実装の管理と比べてどの程度管理しやすいか評価する。

7.2 評価環境

本研究で用いた評価環境を図 7.1 に示す。
なお、二つのホストは 10Base-T で同一セグメント内の LAN に繋がっている。

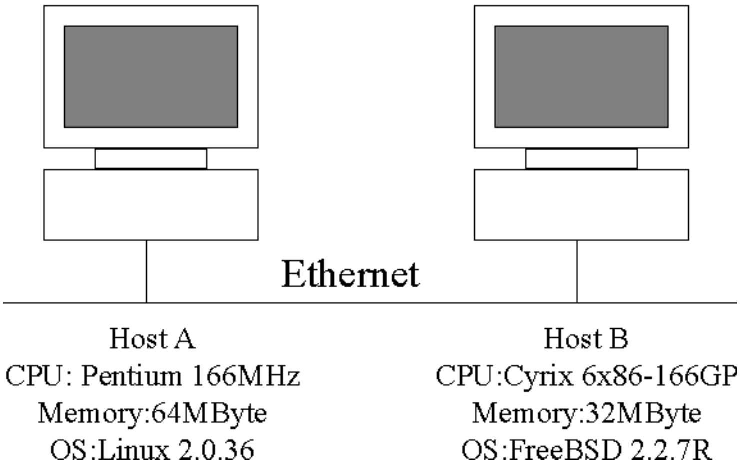

図 7.1:評価環境

ホスト B に本研究で開発した実装を inetd から起動させ、ホスト A からホスト B に telnet での通信と、Secure Sockets Layer による暗号化された通信を行った。

ホスト B での inetd の設定は以下の図の通りである。

図 7.2:inetd.conf の書式

```
telnet  stream tcp nowait root /usr/libexec/telnetd
telnets stream tcp nowait root /usr/local/bin/sslwrap \
   sslwrap /usr/libexec/telnetd
#telnets stream tcp nowait root
/usr/local/libexec/telnetsd
```

また、性能評価の比較対象として以下のものを利用した。

- **telnetd**
  カリフォルニア大学による telnetd(telnetd.c 8.2(Berkeley)) telnetd の

標準的な実装である。

- **SSLtelnet-0.13**
  Tim Hudson 氏が、カリフォルニア大学による telnetd(telnetd.c 5.48 (Berkeley)) に Secure Sockets Layer の暗号通信機能を付け加えた実装である。

- **SSL-MZtelnet-0.11.2**

  カリフォルニア大学による telnet クライアント (telnet.c 8.2 (Berkeley) 12/15/93) に Tim Hudson 氏と Christoph Martin 氏が Secure Sockets Layer の暗号通信機能を付け加えた実装である。

7.3 評価結果

7.3.1 通信路の暗号化の検証

まず初めに、安全な通信路の提供が確実に行われているかを検証するために、特定のportで行われている通信を傍受するパケットモニタプログラムを作成し、それを使って実際に傍受を行った。

まず、ホストBの暗号化中継プログラムを使わないtelnetdに対して、ホストAから行われている通信の傍受を行った。

図7.3:パケットモニタプログラムの出力1

```
12345678910abcdefghijklmnopqrstuvwxyz
```

結果は上の図の様になり、ホストAから入力した文字列を傍受することができた。

次に、ホストBの暗号化中継プログラムを使ったtelnetdに対して、ホストAから行われている通信の傍受を行った。これは、暗号化中継プログラムを使わないtelnetdに対してホストAから入力した文字列と同じ文字列を入力した。

図7.4:パケットモニタプログラムの出力2

```
(Jtf7s.3qC?iba~BU9?V;RJUfSgr:@b;vhM-*B>o?o7b0}b"[5gS3x=rP
!e\zI8xDj)u=Q!c7u]A+J
```

結果は上の図の様になり、ホストAから入力した文字は、暗号化され、内容を知ることが出来なかった。

このため、本研究で開発した機構が、盗聴による通信内容の漏洩に対して有効であることが確認された。

7.3.2 通信性能の比較

次に、本研究で提案した機構による実装の通信性能を、他の機構による実装と比較した。転送に必要な処理時間を測定することにより、本研究の実装の性能を評価する。

そのため、ホストBに1MByteと10MByteのファイルを用意し、暗号通信を行わない場合と、本研究の機構を用いた場合と、他の機構を用いた場合について、それぞれ1MByteと10MByteのファイルを10回転送して時間を計測した。

実際には、ホストAからホストBのtelnetdへの通信を行い、次に、ホストAからホストBの本研究での実装の暗号化機能を利用したtelnetdへの通信を行い、最後に、あらかじめ暗号化機能を実装したtelnetdへの通信を行った。それぞれ、1MByteと10MByteのファイルを10回転送し、それに要した処理時間を計測し、平均を計算した。

結果を、下の表7.1に示す。

表7.1: 転送に要した処理時間の平均

	1M Byte		10M Byte	
	平均　(単位 :秒)	標準偏差	平均　(単位 :秒)	標準偏差
telnetd	4.292	0.01020	43.650	0.06937
暗号中継プログラム + telnetd	5.255	0.06937	58.589	0.07758
SSLtelnetd	5.303	0.01405	56.536	0.08611

暗号通信を全く行っていないtelnetdは、1MByteの場合と10MByte場合両方で他の方式と比べて短い処理時間であることが確認出来る。

また、本研究での実装の暗号化機能を利用した場合と、あらかじめ暗号化機能を含んだ実装を使用した場合の1MByteの転送に要する手順が処理時間を比較すると、ほぼ同じと思われる。

さらに、本研究での実装の暗号化機能を利用した場合と、あらかじめ暗号化機能を含んだ実装を使用した場合の10MByteの転送に要する手順が処理時間を比較してみると、本研究での実装を使用した方が処理時間が長い。

これは、あらかじめ暗号化機能を含む場合に比べ、本研究での実装を使用すると、暗号化中継プログラムとtelnetdの間で通信をする必要が有るためと考えられる(図7.5)。

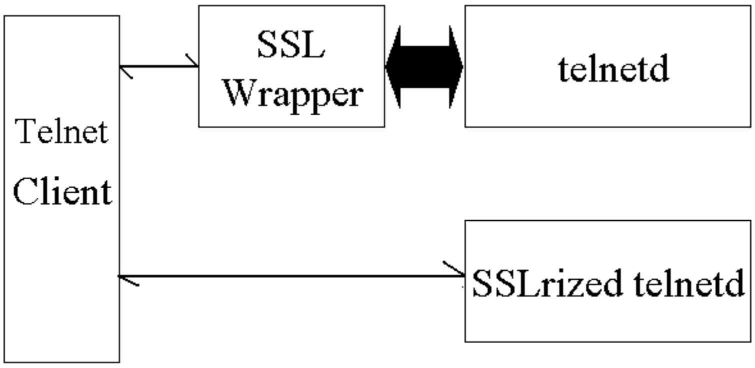

図 7.5: 暗号通信中継プログラムで生じる通信

このため、本研究で開発した機構は、暗号化機能を内蔵した機構に比べて、通信の性能が劣る部分が有ることが確認された。

7.3.3　管理作業の比較

次に、本研究で提案した機構による実装の管理方法を、他の機構による実装の管理方法と比較した。これは、ユーザーや管理者に出来るだけ負担をかけずに管理を行うという、本研究の目的が満たしているかどうか評価するためである。

比較結果は以下のようになった。

表 7.2: 管理作業の比較

	本研究の方式	IPsec	SOCKSV4	SOCKSV5	SSH (port-forwarding)	bjorb	stelsock
システムに改変を行わない	○	×	○	○	○	○	○
再コンパイルが不要	○	○	×	○	○	○	○
煩雑な設定が不要	○	○	○	○	×	○	○
一元管理が可能	○	○	○	○	×	○	×

- システムに改変を行わない

これは、kernel の再コンパイル等による、ネットワーク層への改変が必要かどうかを比較したものである。このような場合、カーネル等システムソフトウエアのソースコードが高価で手に入らない場合なども問題になる。比較した中では、IPsec がネットワーク層でのサービスを行っているため、この条件に該当する。システムを改変するためにはサービスを停止を必要とするので、管理作業が煩雑になると考えた。

- 再コンパイルが不要

これは、暗号化に対応していないプログラムを再コンパイルすることなしに暗号化に対応できるか考察したものである。SOCKSV4 では再コンパイルなどの作業が必要であった。

- 煩雑な設定が不要

基本的に、一度設定を行えば、次回からは複雑な操作を行わなくても、暗号化機能が利用できることを評価の基準とした。

その中でも、本実装の方式は、以下の様にinetd.confに登録を行うだけで、簡単に利用が可能である。

図 7.6:inetd.conf の書式例

```
#
# Internet server configuration database
#
# @(#)inetd.conf 5.4 (Berkeley) 6/30/90
#

telnet  stream tcp nowait root /usr/libexec/telnetd
telnets stream tcp nowait root /usr/local/bin/sslwrap \
   sslwrap /usr/libexec/telnetd
pop3s stream tcp nowait root /usr/local/bin/sslwrap \
   sslwrap /usr/libexec/popper
finger stream tcp nowait nobody  /usr/local/bin/sslwrap \
   sslwrap /usr/bin/finger
```

また、逆に、SSHのport-forwardingでは、通信路の暗号化を行うために以下のような設定と起動が必要で、煩雑である。

図 7.7:SSH の port-forwarding の書式例

```
ssh -L 8023:hostname:23 hostname -l username
```

- 一元管理が可能

本実装の方式は、inetd.confによって一元管理が可能である。
それに比べて、SSHのport-forwardingでは、暗号化したいプログラムごとにSSHを実行する必要があり、管理に手間が掛る。stelsockについては、暗号化するプログラムのディレクトリごとにプログラムを置く必要を考えたため、上の結果となった。

また、bjorbの場合、スタンドアローンで動くbjorb本体の管理が必要となるが、管理の手間は煩雑ではなく、本実装と同じように一元管理が可能である。そのため、本実装と同様に一元管理による管理作業の簡略化に成功している。

以上のように、本研究で提案した機構による実装の管理方法を、他の機構

による実装の管理方法と比較したが、本実装の管理方法は他の機構と比較しても、管理しにくいものではなかった。よって、容易な管理作業で、既存のアプリケーションに改変を加えない暗号化を達成することが出来た。

8　結論

8.1　本研究の成果

本研究では、ユーザーや管理者に出来るだけ負担をかけずに通信プログラムを暗号化に対応させるため、既存のプログラムに改変を行わず暗号化通信に対応させる機構の開発を行なった。

本研究で提案した設計 / 実装は以下の特徴を持つ。

- 通信路の暗号化機能
  本研究で開発した機構が、盗聴による通信内容の漏洩に対して有効で、サーバーの証明書による認証も可能である。
- 他の機構による実装に劣らない通信性能
  本研究で提案した機構による実装は、他の機構による実装とほぼ同等な通信性能を実現した。
- 管理作業の簡略化
  本研究の機構による実装の管理は、他の機構による実装の管理と比べてより簡易な管理機構を提供する。

8.2 課題

本章では、本研究の今後の課題について述べる。

- 動的に使用する **port** が変わるサーバプログラムへの対応

  第 5 章の設計の章で説明した通り、今回の実装では、使用する port が変動するサーバプログラムに対して問題を生じる。

  これに対処するため、暗号化中継プログラムがプロトコルごとの対応を行ない、クライアントのポート番号指定をより先に取得し、暗号化中継プログラムがクライアントが指定したポートでも通信を行なうような拡張を行ないたい。

- クライアントの **IP** アドレスを調べるサーバプログラムへの対応

  今回の実装では、暗号化中継プログラムがサーバプログラムと UNIX ドメインで通信する機構を取ったため、クライアントの IP アドレスをサーバへ渡すことが出来なかった。そのため、起動できないサーバプログラムなどが存在した。

  これに対処するため、暗号化中継プログラムが INET ドメインで通信を行なう機構を実装したい。

- 機能の強化

  今回の実装では、認証機能として、サーバの認証しかすることが出来なかったので、ユーザーの認証機能や、他の暗号アルゴリズムへの対応などを行ないたい。

  また、中継プログラムが他のホストへの接続を行なうなどのプロクシ機能などの、容易な管理作業を実現した上での機能面での強化を行ないたい。

9　謝辞

本論文を書くにあたり、慶應義塾大学環境情報学部の村井純博士と徳田英幸博士に指導頂きました。また、楠本博之博士と中村修博士、植原啓介氏にも多くのご助言を頂きました。諸氏に感謝致します。

慶應義塾大学徳田・村井・楠本・中村研究会の皆様には多くのお力添えを頂きました。特に、本論文の作成に当たって、慶應義塾大学政策・メディア研究科の関谷勇司氏、石橋啓一郎氏、土井裕介氏に多大なるご協力を賜りました。重ねてここにお礼を申し上げます。

最後に、学内・学外を問わず、励ましと協力を頂いた多くの友人諸氏に感謝と敬意を表し、私の謝辞とさせて頂きます。

10 Appendix

評価の結果

表 10.1: 評価の結果 (単位 分:秒)

	telnetd(1M)	telnetd(10M)
1	0:04.05	0:43.09
2	0:04.29	0:41.75
3	0:04.46	0:45.36
4	0:04.44	0:43.65
5	0:04.12	0:44.38
6	0:04.62	0:46.82
7	0:04.55	0:44.46
8	0:04.18	0:43.43
9	0:04.07	0:40.76
10	0:04.14	0:42.80

	SSLWrapper(1M)	SSLWrapper(10M)	SSLtelnetd(1M)	SSLtelnetd(10M)
1	0:05.46	0:59.01	0:05.00	0:54.95
2	0:05.30	0:57.13	0:05.24	0:55.93
3	0:05.41	0:57.55	0:05.25	0:56.58
4	0:04.94	0:59.10	0:05.40	0:58.12
5	0:05.22	0:59.73	0:05.29	0:55.53
6	0:04.93	0:58.29	0:05.27	0:56.22
7	0:05.04	0:58.61	0:05.40	0:56.24
8	0:04.94	0:58.31	0:05.21	0:56.43
9	0:05.86	0:58.91	0:05.21	0:58.79
10	0:05.45	0:59.25	0:05.76	0:56.57

参考文献

[1] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas and L. Jones. *SOCKS Protocol Version 5.* RFC 1928,*IETF*,March 1996

[2] M. Leech. *Username/Password Authentication for SOCKSV5* RFC 1929,*IETF*,March 1996

[3] M. Leech. *GSS-API Authentication Method for SOCKS Version 5* RFC 1961,*IETF*,March 1996

[4] R. Atkinson. *Security Architecture for the Internet Protocol* RFC 1825,*IETF*, August 1995

[5] R. Atkinson. *IP Authentication Header* RFC 1826,*IETF*, August 1995

[6] R. Atkinson. *IP Encapsulating Security Payload (ESP)* RFC 1827,*IETF*, August 1995

[7] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. *SSH Protocol Architecture¡draft-ietf-secsh-architecture-02.txt¿*. Internet Draft, *Network Working Group* , August 1998.

[8] Alan O. Freier, Philip Karlton. and Paul C. Kocher *The SSL Protocol Version 3.0¡draft-freier-ssl-version3-02.txt¿*. Internet Draft, March 1996.