第 11 部

セキュリティ技術

第 1 章

はじめに

最近のインターネット環境の急速な広がりと、さまざまな領域での利用によって、インターネットは学術研究面での効用だけではなく、商業利用におけるメリットも広く認識されつつある。最近では、新たな通信メディアとしてのインターネットの利用だけではなく、エレクトロニック・コマースのような、コンピュータネットワークの特性を最大限活かした商業活動も計画されるようになってきた。このような状況は、インターネットで取り扱われる通信が、金銭取引きといった直接的な「財」に関わる利用にも適用されようとしていることを意味する。そして、このようなインターネットの適用範囲の拡大によって、セキュリティ技術の積極的な開発と、開発された技術の広範囲な適用の必要性が、広く認識されるようになってきた。このようなことを背景として、WIDE Project Security Working Group では、インターネット環境で求められるセキュリティ機能に注目し、それらを実現することを具体的な方法として研究を進めている。

1994 年度における Security Working Group の活動では、以下の 3 点に主眼をおいて研究を行なってきた。

1. メッセージ交換システムにおけるセキュリティ
   昨年度より引続き、FJPEM に代表されるメッセージ交換システムにおけるセキュリティ機能の実現と、その機能の拡張を行なってきた。本年度の研究では、PEM の機能を包含した MIME と PGP を実際に対応する方式として取り上げている。第 2章では、この 2 方式の概要と統合の可能性について述べる。

2. One Time Password
   インターネット・サービスのセキュリティ上の大きな問題点として、telnet などを使用してリモートのホストにアクセスした場合に、パスワードが平文として交換されることがあった。この問題を解決する一つの方法として、使い捨てパスワード (OTP: One Time Password) 技術がある。WIDE Internet の主要なホストでは、OTP の一つの実装である S/KEY の運用を開始している。第 3章では、S/KEY の概要と運用技術、さらに本 WG で開発したツールについて述べる。

3. SecureTCP
   セキュリティ技術の一つの重要なものとして、通信路の暗号化技術がある。これま

で、通信路の暗号化は主にアプリケーション層で行なわれてきた。しかしながら、近年ではネットワーク層やトランスポート層での実装も進められている。本WGでは、インターネットプロトコル群のトランスポートプロトコルの一つである TCP の拡張を行ない、通信路暗号化機能を持った Secure TCP を開発した。第 4章では、Secure TCP について述べる。

第 2 章

PEM と MIME

我々のグループでは、1993 年から FJPEM の実装、配布を経て、PEM の運用実験を行ってきた。FJPEM は、RFC1421[80] で定めれた PEM(Privacy Enhanced Mail) に従って実装されている。PEM は、RFC822 メール [81] の本文への暗号化、本文への電子署名かつ暗号化の枠組を提供しているが、暗号化手続きと電子署名手続きの切り分け、MIME への対応などは考慮されていない。

Mew などの優れた MIME インターフェイスの出現により、現在では MIME はインターネットで広く利用されている。また、プライバシへの関心が高まるにつれ、PEM の普及も促進されるであろう。このような状況においては、当然 MIME と PEM を同時に利用したいという要求が現れる。たとえば、MIME の一部のパートに署名したいという要望などである。

MIME と PEM の統合は、IETF の PEM-DEV WG で議論されてきた。この節では、MIME と PEM の統合の動向について述べる。

2.1 2 つの候補

1993 年の終りから 1994 年の前半にかけて、PEM と MIME を統合する規格は 2 つ存在し、それぞれ ID として配布されていた。以下に 2 つの規格を示す。

Application 方式 RFC1421 PEM をそのまま MIME のパートに取り込む方式 [82]。また、PEM の Content-Domain: として MIME を定義しているので、PEM に MIME を取り込むことが可能である。

Multipart 方式 PEM の制御情報と対象オブジェクトを、パートに切り分け、両者をマルチパートに射影する方式 [83]。RFC1421 PEM とは互換性がない。

Application 方式は、MIME の Content-Type: として Application/pem-1421 を定義し、これを解釈するコードを書くだけで、既存 PEM を利用しながら容易に実装が可能である。しかしながら、PEM から MIME へ取り込まれた Base64 符号化方式を、依然として PEM の枠組で利用するため、この符号化方式の定義は冗長となっている。また、PEM の開始

と終了を表す境界は固定されているので、多重に PEM を施す場合は、境界の先頭を退避しなければならない。また、暗号化と電子署名を切り分けれない RFC1421 PEM の制約をそのまま受け継ぐことになる。Application 方式は、実装は容易であるが、PEM と MIME の完全な統合と言う意味では、両者の特徴をうまく生かしきれていない。

Multipart 方式は、Content-Type: Multipart のサブタイプとして、Signed と Encrypted を用意している。Multipart/Signed の第一パートは署名されたオブジェクト、また、第二パートとして制御情報をとる。RFC 1421 PEM では、必ず制御情報が前に来るため、PEM に対応していないメールインターフェイスでは、PEM で署名されたメールを読み辛かった。しかし、Multipart 方式では、署名されたオブジェクトを前にもってくることで、これを緩和している。Multipart/Encrypted の第一パートは制御情報、そして、第二パートとして暗号化されたオブジェクトをとる。

Signed、Encrypted ともに、対象としてテキスト以外のオブジェクトをとることができる。また、バイナリストリームとなったオブジェクトは、MIME の Content-Transfer-Encoding: を利用して符号化する。暗号化と署名が完全に切り離されているため、暗号化の後に署名をしてもよいし、また、逆の手順を踏んでもよい。Multipart 方式は、既存の PEM を利用することはできないが、PEM と MIME の統合の規格としては、美しくそして強力である。

1994 年の始めに Application 方式は取り下げられ、PEM と MIME の統合の規格は Multipart 方式に一本化された。我々は、一旦 Application 方式を実現したが、PEM-DEV の方針に従い、 Multipart 方式に追従することにした。

2.2 MOSS

Multipart 方式は、慣習的に PEM/MIME と呼ばれてきたし、文献 [83] のタイトルにも PEM という単語が含まれている。しかし、RFC 1421 PEM と互換性のないこの規格は、PEM という名称はふさわしくない。また、機能的に見ても、世界規模の証明書発行システムを仮定していないので、従来の PEM とは切り離して考える必要がある。

そこで、Multipart 方式は、PEM/MIME という通称を MOSS(MIME Object Security Services) に変更した [84]。現在このドラフトは、RFC となるための最終段階に達している。

我々のグループでは、MOSS を実現するコマンド郡を実装中である。また、MOSS のインターフェイスとして対応できるように、Mew の機能の強化を図っている。

2.3 PGP と MIME

PEM と同様に電子メールのプライバシを強化するツールとして PGP(Pretty Good Privary) がある。PGP においても、MIME と統合し、両者を共存させて行かなければならない。PGP と MIME の統合の企画として、節 2.1で説明した PEM の application 方式と全

く同様の方式が、文献 [85] において提案された。しかし、著者の多忙を理由に、この提案は取り下げられている。よって、現在は、PGP と MIME の統合の行方は予想がつかない。

第 3 章

使い捨てパスワードの運用

1994 年 2 月 3 日に発行された CERT Advisory CA-94:01 “Ongoing Network Monitoring Attacks” において、計画的なパスワードの盗聴事件が報告された。クラッカーは、SunOS 4.x に侵入後、ルートの権限を入手し、ネットワークインターフェイス NIT を通じて login や telnet セションを盗聴した。事件の 1ヵ月後に発行された CIAC Advisory Notice E-12 では、最終的に盗聴されたパスワードは、10 万を越えたと報告している。

この事件をきっかけに、リプレイ攻撃に弱い UNIX パスワードによる認証システムへの危機感が高まった。幸いなことに、現時点でも盗聴に強い認証システムである使い捨てパスワードが利用可能である。そこで、この節では、使い捨てパスワードシステムの中から特にフリーとして配布されている S/KEY[86] に焦点を当てる。ここでは、WIDE の S/KEY の運用実験を通じて得た経験や作成したアプリケーションについて述べる。また、S/KEY の動向に付いても簡単に触れる。

3.1 S/KEY

S/KEY は、一方向関数をうまく利用したチャレンジ＆リスポンスによる認証システムである。ユーザは、最初に一回だけ対象となる計算機のコンソールに足を運んで、コマンド keyinit を用いて S/KEY を初期化する。

初期化の際に必要な情報は、ユーザ名、シーケンス、種、秘密のパスワードである。通常、ユーザ名、シーケンス、および、種は自動的に選ばれるから、ユーザは秘密のパスワードを入力するだけでよい。ユーザ名、シーケンス、および、種は公開しても構わない情報である。秘密のパスワードだけは、公開してはならない。実際に秘密のパスワードを入力する際は、ネットワーク上を流れないように注意しなければならない。

秘密のパスワードと種から生成される文字列に、シーケンスの数だけ一方向関数が適応される。例えば、ユーザ名 kazu、シーケンスがデフォルトの 99、種が sh01234 で、ある秘密のパスワードを用いて初期化すると、次のようなエントリができる。

```
kazu 0099 sh01234           dc5da8b6714babbc  May 02,1995 21:25:23
```

これで初期化は完了である。つぎに、ユーザがこの計算機にログインしようとして、telnet したとする。ユーザ名を入力したときは、以下のようにチャレンジが発生する。

```
% telnet xxx
login: kazu
s/key 98 sh01234
(s/key required)
Password:
```

チャレンジとして、シーケンスが登録時より 1 減っていることに注意してほしい。ここでユーザは、(他のウインドウなどを利用して) 目の前の計算機で、key コマンドを用いて使い捨てパスワードを計算する。key コマンドは、引数としてシーケンスと種、入力として秘密のパスワードを要求する。ネットワーク経由で使い捨てパスワードを計算してはならないのは、このように秘密のパスワードが必要だからである。

```
% key 98 sh01234
Enter passphrase :
KANT SOD LOSE DOSE CUE JOIN
```

key は、入力された秘密のパスワードと種からできる文字列に、シーケンスで指定された回数だけ一方向関数を適応する。こうして計算された使い捨てを、リスポンスとして先程のウインドウにカット＆ペーストすれば、ログインが成功する (あるいは、失敗する)。

このとき、ログインしようとしている計算機では、以下のような計算が行われている。つまり、ユーザがチャレンジとして返した使い捨てパスワードに 1 回一方向関数を適応し、保存している (シーケンスが 1 大きい) 使い捨てパスワードと比較する。もし一致すれば、ユーザがチャレンジとして返した使い捨てパスワードと 1 小さいシーケンスを保存し、ユーザにログインを許可する。異なればログインを否定する。

サーバがチャレンジとして返すシーケンスは、保存しているシーケンスよりも 1 小さい。よって、ユーザは以前計算した使い捨てパスワードよりも、一回だけ少なく一方向関数を適応させた使い捨てパスワードを計算することになる。一方向関数を利用しているので、以前の使い捨てパスワードから現在の使い捨てパスワードを計算することはできない。言い替えれば、大きなシーケンスに対応する使い捨てパスワードから、小さいシーケンスに対するそれは計算できないのである。

これが、S/KEY がシーケンスを減らしていく本質的な理由であり、使い捨てパスワードを盗聴されても、次の使い捨てパスワードを推測される危険性はない。また、種により秘密のパスワードに対してアクセントを付けている。よって、シーケンスが 0 に近付いて再び初期化する場合や、別の計算機を利用する場合も、種を変更するだけで、秘密のパスワードを安全に共有できる。

3.2 Donkey

WIDE プロジェクトには、メンバーだけが利用できる計算機があり、この計算機においてS/KEY の運用を始めた。しかし、さまざまな組織に分散しているメンバーが、計算機のコンソールまで行って S/KEY を初期化するのは困難である。そこで、物理的にコンソールまで行かなくても、安全に S/KEY を初期化できる機構が必要があった。

S/KEY では、一旦コンソールで S/KEY を初期化すれば、後は、ネットワーク経由で安全に再び初期化することが可能である (keyinit -s)。これは、計算機上になんらネットワークを経由してはならない情報が保存されないからである。ではなぜ、最初に 1 回だけコンソールに行かなければならないのであろうか？それは、最初は S/KEY の登録をしてないので、計算機にログインできないからである。

そこで、keyinit で生成される情報を電子メールなどで管理者に送り、管理者に登録してもらえばそれで十分なことが分かる。この情報は、盗聴されても構わない。電子メールの改竄を防止するには、PEM や PGP などで署名しておけばよい。

しかしながら、keyinit は、/etc/skeykeys というルートしか書き込めない所定のファイルにエントリを登録するすることしかできない。また、ユーザは、目の前の計算機では S/KEY を利用しておらず、keyinit というコマンド自体が存在しないかも知れない。

そこで、S/KEY を運用している計算機にアカウントがあるユーザなら誰でも所有するコマンド key に、keyinit と同じ初期化の情報を標準出力に生成させる機能を持たせることがよいと結論を下した。この機能は、donkey というコマンドを作成することにより実現した。donkey の特徴を以下に示す。

- コマンド key と上位互換である
- コマンド keyinit と同じ情報を作成できる
- 一方向関数を、コマンドラインオプションで動的に指定できる。
- たくさんのプラットフォームをサポートしている。

donkey は、以下のように key の代替として利用できる。

```
% donkey 99 sh01234
Enter passphrase :
SILO TOLL NUT SORE MESS SLAT
```

従来の key では、コンパイル時に一方向関数として MD4 か MD5 を静的に決定しておく必要があった。しかし、donkey では、-f オプションで一方向関数を MD2、MD4、MD5 のなかから動的に選択できる。

```
% donkey -f md5 99 sh01234
Enter passphrase :
CHAD LAM LIAR LUCK QUOD GRIN
```

keyinit が生成する情報を作成するには、-i オプションを指定すればよい。

```
% donkey -i
Enter login name [default kazu]:
Enter sequence 1 to 999 [default 99]:
Enter new seed [default is17516]: sh01234
Enter passphrase :
Enter passphrase :
kazu 0099 sh01234          dc5da8b6714babbc  May 02,1995 21:25:23
SILO TOLL NUT SORE MESS SLAT
isindy03:~% key 98 sh01234
Enter passphrase :
KANT SOD LOSE DOSE CUE JOIN
```

donkey は以下に示す方法で入手できる。

```
ftp://ftp.aist-nara.ac.jp/pub/Security/tool/donkey/donkey-current.tar.gz
```

われわれは、Donkey をメンバーに配布し、S/KEY の運用を始めたところである。

3.3　OTP

S/KEY は、Bellcore 社の商標であるので、インターネットの標準として利用することができない。そこで、S/KEY は OTP(One Time Password) と名称を変更した。OTP は、インターネットの標準となることを目的としている。また、さまざまな一方向関数を統合する予定である。

OTP の最初の試みとして、S/KEY に関するセキュリティホールに関する対策が考えられた。S/KEY のセキュリティホールとは、以下の通りである。

正式なユーザを A、クラッカーを X とする。A がある計算機にログインしようとする。このとき、X はすかさずユーザ名 A で計算機にアクセスする。すると、X は、A が受け取ったチャレンジと全く同様のチャレンジを受け取れる。A は、使い捨てパスワードを返すが、X はこれを盗聴し、すかさず計算機に送りつける。この競争に勝てれば、X は計算機にログインすることができる。

このセキュリティホールの原因は、現在進行中のチャレンジと同じチャレンジを再び使ったことにある。実装に関していえば、全てのユーザ間で /etc/skeykeys というファイルを

共有していたので、フィールドロックをかけることができないのが問題であった。そこで、OTP では、ユーザごとにファイルを分け、ロックをかけれるように、同じチャレンジが同時に発生することを防いでいる。

第 4 章

Secure TCP

インターネット環境を考えた場合、通信路におけるデータの暗号化技術は、現在指摘されている、多くのセキュリティ問題の解決方法を与える。

通信路でのデータの暗号化は、具体的なアプリケーションの機能として、これまでアプリケーションプロトコルに実装されることが多かった。例えば、電子メールの暗号化を実現した PEM[80]、telnet に適用した PET (Privacy Enhanced Telnet) などが挙げられる。これは、各アプリケーションが取り扱う通信を、独自の方法で暗号化するものであり、アプリケーション毎に実装が必要となる。

一方、もう一つの動きとして、ネットワーク層プロトコルである IP のデータグラムを暗号化する手法も開発されている。例えば、 swIPe[41]、IP/Secure[87] 等は、その具体的な実装である。ネットワーク層のプロトコルで暗号化を実装した場合、ホスト／ホスト間の通信を簡単に暗号化できるため、パブリックなインターネット上に、安全な通信路を確保することができる。このため、アプリケーションの変更なく、安全な通信が保証することができる。このことから、次世代インターネットプロトコル (IPv6) では、その基本機能として、データグラムの暗号化機能が実装されることが決まっている。しかしながら、この方法にも問題点がある。セキュリティ機構を持ったホスト間で通信される全てのデータグラムに対して暗号化／復号化処理が適用されるため、通信性能の劣化が全ての通信に発生する。また全ての通信 (サービス) で、通信路の暗号化機能が必要になるわけではない。

このような状況を解決するには、TCP あるいは UDP といったトランスポート層プロトコルで通信路暗号化を実現することが考えられる。この方法であれば、

- アプリケーション層で、各アプリケーションが独自に通信路暗号化に対応する必要が無くなる。また、アプリケーションプロトコルを変更する必要はない。
- 暗号化が必要なサービスだけに適用することができる。

といった利点がある。このようなことから本研究では、インターネット上の多くのアプリケーションで使われている TCP プロトコルに通信路の暗号化の機能を付加した Secure TCP[88] を開発した。Secure TCP では、従来の TCP とのインターオペラビリティも確保していることから、あるアプリケーションが Secure TCP に移行したとしても、その移行は非常にスムースに行なうことができる。

本章では、Secure TCP の設計、実装、性能について述べる。

4.1 開発の目標

現在の TCP におけるセキュリティ上の問題点をまとめると次のようになる。

- メッセージ改ざん攻撃が可能
  TCP では、各セグメントのチェックサムがヘッダにあるが、これは保護されていないので、改ざんが可能である。また、メッセージを改ざんされた場合、発見する手段がない。

- 盗聴が可能
  現在の TCP では、セグメントの暗号化は行なっていないので、アプリケーション層で対応がない限り盗聴が問題となる。TCP では盗聴に対してなんの保護も講じていないことを利用した、telnet や rlogin のトラヒックをモニタし、パスワードを盗み出す、いわゆる sniffer attack は、深刻な問題である。

- 通信相手の認証がおこなわれない
  現在の TCP では、単にヘッダ中に書かれている IP アドレスとポート番号だけで通信相手を判別している。ヘッダにはなんの保護もしていないので、第三者が IP アドレスとポート番号を偽ってセグメントを生成し、通信することができてしまう。

Secure TCP では、上記の問題を解決するべく、以下のような機能を提供する。

- TCP セグメントの保全性 (integrity) の保証

- TCP セグメントの機密性 (confidentiality) の保証

- 通信開始時に、暗号化方式など を peer entity 間でネゴシエーションを行ない決定することによる拡張性の提供

- 従来の TCP との相互操作性 (interoperability) の確保

Secure TCP では、Kerberos などの認証機構を使用することで、peer entity 間の認証を実現している。

4.2 Secure TCP の設計

本章では Secure TCP の設計についてのべる。

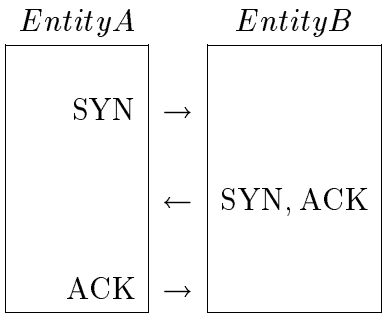

図 4.1: Three-way handshake of TCP

4.2.1 Three-Way Handshake の拡張

Secure TCP では、通信開始時に送信側と受信側のネゴシエーションによって、セグメントの暗号化方式などを決定する。さらに、このネゴシエーションを行ないつつも、従来の TCP との相互操作性を確保しなければならない。このため Secure TCP では、TCP がコネクション設定時に行なう three-way handshake を拡張して、このネゴシエーションを実現している。

現在の TCP では図 4.1に示すような手順で three-way handshake を行ない、コネクションを設定する。

Secure TCP では、図4.2に示すような three-way handshake を行なう。まず送信側 (entity A) は、通常の SYN セグメントに、送信側のネゴシエーション情報 $NEGO_A$ を付加したものを送る。

これを受信した側 (entity B) では、A から送られたネゴシエーション情報をチェックし、A との通信で使用する暗号化方式 (cipher) と、セグメントの保全性を検査するハッシュ関数 (MD5 など) を選択する。そして、A に対しては、$SYN + ACK + NEGO_B$ を送る。$NEGO_B$ は、受信側が選択した暗号化方式などを通告するためのデータである。同時に、B の公開鍵 KEY_p を送る。これは、セッション鍵を安全に交換するために使われる。

これを受信した A では、ACK を返すとともに、セッション鍵 KEY_S を送る。そして、B はセッション鍵を正しく受信したことを ACK により A に通告する。

通常の TCP との相互操作性は、以下のようにして確保されている。

- 送信側が通常の TCP の場合には、最初の SYN セグメントにネゴシエーション情報が含まれない。このため、受信側ではそれを発見した場合には、通常の TCP と同じ処理を行なう。
- 受信側が通常の TCP の場合には、SYN/ACK セグメントにネゴシエーション情報等が含まれない。これらを検知した場合には、通常の TCP の処理を行なう。

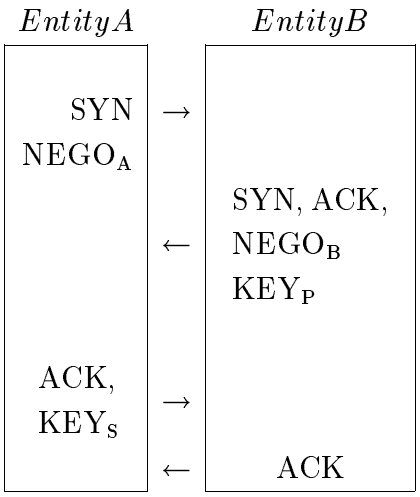

図 4.2: Extension of Three-way handshake

- ネゴシエーション情報の交換では、TCP option の拡張を使用しているので、これによる副作用は発生しない。通常の TCP では、拡張した TCP option を含むセグメントを受信しても、それを単に無視するだけである。

- 現在のコネクションが Secure TCP なのか通常の TCP によるものなのかをアプリケーションが調べるためのインタフェースを用意しており、これによりコネクションの種類によって、アプリケーションの動作を変えるといった処理を実装できる。

Secure TCP で行なった TCP Option の拡張の詳細については、[88] を参照してもらいたい。

4.2.2 暗号化方式の選択とセッション鍵の交換

暗号方式の選択では、表 4.1に示すように、強度にしたがった番号付けを行ない、送信側では、使用可能な方法のリストを送り、受信側ではそのリストの内、対応できるものの内の番号の最大のものを選択するように選択を行なう。また独自の暗号化方式を使用したい場合には、この表を拡張すればよい。

鍵交換では、公開鍵暗号を使って安全に鍵を交換するだけでなく、ピアの認証も同時に行なう。図 4.3には鍵交換プロトコルを示す。また、図中で使われている表記を表 4.2にまとめる。

	integrity		confidentiality
type	hash	cipher	cipher
0	—	—	—
1	MD5	DES	—
2	MD5	DES	DES(ECB)
3	MD5	DES	DES(CBC)
4	MD5	DES	DES(CFB 1byte)

表 4.1: Security Service Type

IP_X	Network Address of a host X
WK_c	Shared Session key (confidentiality)
WK_i	Shared Session key (integrity)
SK_X	Secret Key for a host X
PK_X	Public Key for a host X
VP	Validity Period of a Public Certificate
$(M)^K$	Public Enc. of Message M with Key K

表 4.2: Symbols used in key distribution

$HostJ$		CA
PK_J, IP_J	$\rightarrow$	$(PK_J, IP_J, VP)^{SK_{CA}}(\mathbf{Sign})$
	$\leftarrow$	$(PK_J, IP_J, VP)^{SK_{CA}}$

(a) Procedure for Public Key Certificate Issue

$HostA$		$HostB$
$(PK_B, IP_B, VP)^{SK_{CA}PK_{CA}}(\mathbf{dec})$ $\Rightarrow PK_B, IP_B, VP$	$\leftarrow$	$(PK_B, IP_B, VP)^{SK_{CA}}$
$\left\{ \begin{array}{l} (PK_A, IP_A, VP)^{SK_{CA}} \\ (WK_c, WK_i)^{SK_APK_B} \end{array} \right\}$	$\rightarrow$	$\left\{ \begin{array}{l} (PK_AIP_A, VP)^{SK_{CA}PK_{CA}} \\ (WK_c, WK_i)^{SK_APK_BSK_BPK_A} \end{array} \right\}(\mathbf{dec})$ $\Rightarrow \left\{ \begin{array}{l} PK_A, IP_A, VP \\ WK_c, WK_i \end{array} \right\}$

(b) Procedure for Key Exchange

図 4.3: Key Issue and Exchange Procedures

4.2.3　Secure TCP での状態遷移

以上のような拡張を行なったことで、従来の TCP の状態遷移を拡張した。図 4.4に、Secure TCP で使用している状態遷移図を示す。

4.2.4　暗号化処理

Secure TCP の暗号化処理では、実際のデータとヘッダの保護が目的となる。このため、Secure TCP では、 MD5 を使いハッシュを生成することでヘッダの検査を可能とし、さらにデータと計算したハッシュを暗号化することで改ざんを防止する。この処理は以下のように行なわれる。

1. 送信側のエンティティでは、まず送信する TCP セグメントについてのメッセージダイジェスト MD を計算する。

$$\begin{aligned}\mathrm{MD} = \ & \mathrm{hash}(\mathrm{TCP\ pseudo\ header} + \\ & \mathrm{TCP\ header} + \\ & \mathrm{TCP\ segment\ data})\end{aligned}$$

2. 次に MAC (Message Authentication Code) を計算する。これは、先に計算した MD をセッションキー K で暗号化したものである。

$$\mathrm{MAC} = (\mathrm{MD})^{\mathrm{K}}$$

3. 次に Secure TCP のセグメントを構成する。ここで Plaintext header では、暗号化したデータおよび MAC をつけた後のセグメントに対して、通常のチェックサムを計算する。

$$\begin{aligned}\mathrm{Segment} = \ & \mathrm{Plaintext\ header} + \\ & (\mathrm{Segment\ data})^{\mathrm{K}} + \\ & \mathrm{MAC(Encrypted\ segment)}\end{aligned}$$

4.3　Secure TCP の実装

Secure TCP は現在 BSDI BSD/386 1.1 上で稼働している。実装では、カーネル内の TCP 関連のモジュールに変更を加え、さらに Secure TCP のための情報保持のために STCPCB (Secure TCP Control Block) を新たに定義している。このため、`tcp_input.c`, `tcp_output.c`, `tcp_subr.c`, `tcp_usrreq.c`, `tcp.h`, `in_tcb.h` に変更を加えた。また現在実装されている暗号化方式は DES だけであり、DES(ECB) などのより強力な暗号化方式は実装が終っていない。

現在の実装で、従来の TCP との相互操作性を有しない部分は、以下の２点である。

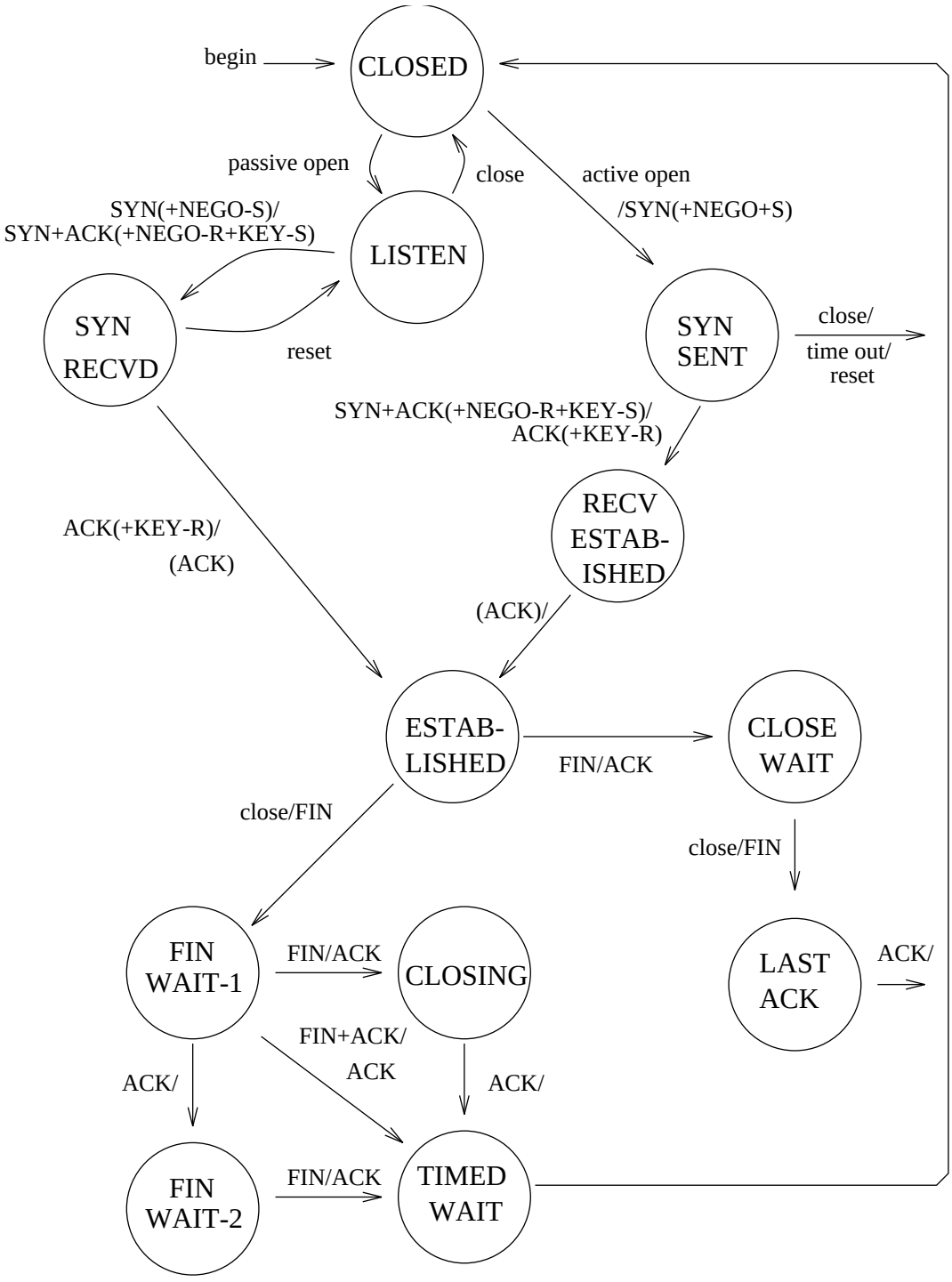

図 4.4: State Transition Machine

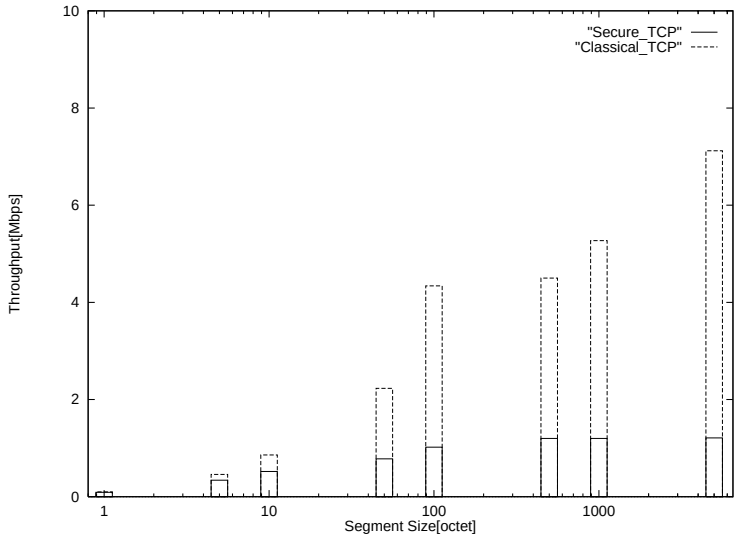

図 4.5: Throughput against various Segment Sizes

- RFC1644[89] で定義されている T/TCP (Transaction TCP) との相互操作性は保証されない。T/TCP では、コネクション設定時の SYN セグメントに、実際のデータを含ませて、トランザクション指向の効率の良い通信を行なおうとしている。しかしながら、Secure TCP では実際のデータを暗号化するためにネゴシエーションが必要であり、T/TCP のような通信方式には適用できない。

- 現在の TCP では、ピア・エンティティが同時に SYN パケットを送った場合には、両方のエンティティは単に ACK を返して、コネクションを設定する。これは稀にしかおきないが、このような状態遷移を Secure TCP では許していない。

4.3.1　性能

実装した Secure TCP の性能を、実際の環境で測定した。測定環境は、

- i486DX2 (66MHz), 16MB memory, ISA Ethernet adaptor (SMC Elite16C)

- BSD/386 1.1

である。この時の測定結果を図 4.5に示す。これから分かるように、従来の TCP ではセグメントサイズが大きくなればスループットが上昇する。一方、Secure TCP では、約 1Mbps で性能が頭うちになる。これは、現在の Secure TCP で使用している MD5, DES が共にソフトウェア実装されているためであった。したがって、ハードウェアで実装されている DES や、より性能のよいソフトウェア暗号器を利用すれば性能は改善されると予想される。

また、現在の実装で提供されている 1Mbps というパフォーマンスは、telnet などのインタラクティブな通信では、あまり暗号化処理の存在を気にさせない程度のオーバーヘッドしかない。また、現在の広域ネットワーク環境では、1Mbps のスループットは、十分な性能を提供しているといえる。しかしながら、今後、ネットワークの高速化が進むにつれ、このパフォーマンスの制限は大きな問題となるであろう。

4.4 まとめ

Secure TCP は、従来の TCP との相互接続性を保ちつつ、TCP ヘッダおよびデータの保護、エンティティ間の認証などのセキュリティ機能を提供している。

現在の実装では、1Mbps 程度の比較的良好な性能を提供している。しかしながら、より高速なネットワークでの性能を改善するために、ハードウェア実装による暗号化なども検討する必要がある。また、現在実装されている暗号化方式は DES だけだり、 DES-ECB などのより強力な方式、また DES 以外の暗号化方式も件とする必要があるだろう。

アプリケーションにとっては、Secure TCP を利用する場合には、Secure TCP によるコネクションの場合の処理と、通常の TCP によるコネクションの場合の処理の両方を用意する必要がある。Secure TCP を実際のアプリケーションに適用することを進めるとともに、Secure TCP を使用するためのライブラリとしての API を整備する必要があるだろう。

第 5 章

まとめ

ここでは、Security Working Group の 1994 年度における研究活動の概要について述べた。

第 2節で述べた RFC822 に基づいた電子メールの本文の保護は、プライバシーの保護や、メールという基本的なサービスの広がりを考えた場合、今後も重要なテーマと言える。このことから、今後も Working Group では、IETF における標準化動向を見据えながら活動を続けていく予定である。また、より多くの人に使ってもらうために、国内外の他の組織と共同した啓蒙活動を行なっていく予定である。

第 3節で述べた OTP の技術は、パスワード・クラッキングが多発している現在のインターネット環境を考えた場合重要な技術である。WIDE Project では OTP 技術を、WIDE Internet 内の主要なサーバで利用し始めている。

第 4節で述べた Secure TCP は、現在、第一段階の開発が終了したばかりであり、今後、実験と不足している機能の実装、認証システムとの連係などの課題が残されている。1995 年度では、これらの課題を解決すべく研究活動を進めていく予定である。