修士論文

題目

広域IPsecのための鍵交換デーモンに関する研究

指導教授

所 真理雄 客員教授　・天野 英晴 助教授

慶應義塾大学大学院

理工学研究科 計算機科学専攻

1997年度

89624457

祢宜 知孝

修士論文要旨

専攻	計算機科学	学籍 番号	89624457	フリガナ 氏名	ネギ トモノリ 祢宜 知孝

（内容の要旨は約25行程度で記入のこと）

(論文題目)

広域 IPsec のための鍵交換デーモンに関する研究

(内容の要旨)

近年，インターネットの商用利用が広まり，Electronic Commerce などが実用化されるようになった．しかし，一方でインターネットの普及に伴い，クラッカによるインターネット上でのアタックが数多く報告されている．そのため，盗聴や改竄に強い，より安全なネットワークの構築が急務となっている．

現在，安全なネットワークを提供すべく，IP 層におけるセキュリティ機構として IPsec(IP security protocol) が提案されているが，未だ広域での運用には至っていない．その理由には，IPsec や IPsec を広域で運用するために必要不可欠な鍵交換デーモンが広く世界に普及していないことが挙げられる．

そこで本論文では，IPsec の広域での運用に必要不可欠な鍵交換デーモン (Key Management Daemon) を ISAKMP/ Oakley(Internet Security Association and Key Management Protocol with Oakley) に基づいて実装し，評価を行う．ISAKMP/ Oakley は，鍵交換プロトコルとして世界標準規格の基となるものである．また ISAKMP/ Oakley には，通信遅延の影響が大きいという問題点が存在するため，本論文ではこの問題点に対する改善策を提案する．そして，提案する改善策を実装し，評価することによって改善策の有効性を示す．その他，ISAKMP/ Oakley には幾つかの問題点が存在するため，本論文ではそれぞれの問題点に対する解決策を提案する．

現在鍵交換デーモンの実装は幾つかの国で行われているが，これらは各国の輸出規制のために世界中で自由に利用できない．従って，世界中で自由に利用できる鍵交換デーモンは数少ない．本論文で実装する鍵交換デーモンが世界中で自由に利用できる鍵交換デーモンの一つになり，安全で快適なインターネット環境を提供する一翼を担う事が期待される．また，本論文で提案する改善策が ISAKMP/ Oakley の新たな仕様として議論される事が期待される．

慶應義塾大学大学院理工学研究科前期博士課程

Summary of M. S. Thesis

Major	Student No.	SURNAME, Firstname
Department of Computer Science	89624457	TOMONORI Negi

Title

Key Management Daemon for wide use IPsec

The Internet has become widely used as a medium for commercial activity, while electronic commerce and similar technologies are becoming an accomplished fact. However as the Internet have come into wide use, many attacks by crackers are reported. Therefore, it is crucial for the Internet community to develop and deploy solid security mechanisms such as integrity services or confidentiality services.

Currently, IPsec (IP security protocol), which provides the security mechanisms for IP layer, is proposed, but it isn't universally used. The reason is that Key Management Daemon, which automates the management and exchange of encryption key, is not yet available for wide area use.

The paper shows an implementation of Key Management Daemon which conforms to ISAKMP/ Oakley (Internet Security Association and Key Management Protocol with Oakley), and the evaluation of its performance. ISAKMP/ Oakley will be the standard protocol for Key Management Daemon on Internet. The paper shows the point at issue of ISAKMP/ Oakley as it will be seriously influenced by the traffic delay, and the solution to the issue. Then, the paper attempts to implement it and show the evaluation of the solution with the effectivity of it. The paper shows other issues of ISAKMP/ Oakley, too.

Most of the key management programs are implemented in the countries where export of cryptographic software is legally restricted. The implementation of Key Management Daemon in this paper will be one of the implementations which can be freely used in the world. Hence it will facilitate the comfortable and secure Internet. Furthermore, it is expected that proposed ideas will be discussed as the new specifications of ISAKMP/ Oakley.

Graduate School of Science and Technology, KEIO UNIVERSITY

(The summary of thesis should be written within 35 lines)

目 次

図目次

表目次

第 1 章

序文

1.1 はじめに

近年インターネットが世界中に急速に広まり，商用活動などにも利用されるようになってきた．しかし，インターネットの普及に伴い，クラッカによるアタックが数多く報告されるようになり，クラッカの話題がマスコミの見出しを賑わすようになった．つまりインターネットは，誰もが容易に利用できるまでに発展したと同時に，クラッカによるアタックの急増という問題に直面するに至った．クラッカが行うアタックには，盗聴，改竄，なりすまし，トラフィック解析など多くの種類がある[32]．また，クラッカによるアタックは年々進化してきており，現在規定されているプロトコルでは避け難いようなアタックも報告されている．現在規定されているプロトコルでは避け難いようなアタックの一つに，TCPにおけるTCP RST攻撃が知られている[1]．TCP RST攻撃は，すでに確立されているTCPのコネクションに対して，悪意を持った第三者が送信元アドレスを偽造したTCP RSTメッセージを送信し，確立されているコネクションを切断してしまうというものである．

そこで，これらの攻撃に対抗すべく，より安全なインターネットの構築が急務となり，**IPsec**[4]，[5]，[6]，[17]，[18]，[33]の研究が盛んに行われている．IPsecは，**IETF**(**Internet Engineering Task Force**) **ipsec working group**によって提案されているIPパケット毎の暗号化/認証を行うプロトコルである．そして，IPsecによりクラッカによる盗聴や改竄，偽造といった攻撃を防ぐ事が可能となる．

IPsecの規格化が行われているものの，現時点では未だ広域でのIPsecの運用に至っていない．IPsecが未だ広域での運用に至っていない理由の一つに，広域でのIPsecの運用に必要不可欠な鍵交換デーモン(**Key Management Daemon**)の実装が広く行き渡っていない事が挙げられる．鍵交換デーモンとは，通信先のホスト上で起動されている鍵交換デーモンと協調することによって，IPsecで使用される鍵の確立や鍵の確認，そして通信ホストの相互認証を行うプログラムである．

現在までに，多くの鍵交換のプロトコルが提案されている．具体的には，

- **NSA**(**National Security Agency**)が提案した**ISAKMP**(**Internet Security Association and Key Management Protocol**)[20]
- CISCO社が提案した**ISAKMP/ Oakley**[9]
- Qualcom社が提案した**Photuris**[15]
- SUN社が提案した**SKIP**(**Simple Key management for Internet Protocols**)[7]

が知られている．現在，ISAKMP/ Oakleyを基にした**IKMP**(**Internet Key Management Protocol**)を世界標準として規格化する事が宣言されている．そこで，本論文ではIKMPの基となるISAKMP/ Oakleyに基づいて，鍵交換デーモンの実装を行い，鍵交換デーモンの性能評価を行う．本論文で行う実装は，現在各国の輸出規制のために世界中で自由に利用できる鍵交換デーモンが数少ないなかで，世界中で自由に利用できる鍵交換デーモンとなる．

また本論文では，ISAKMP/ Oakley に存在する幾つかの問題点を指摘し，それぞれの問題点に対する改善策を提案する．そして，幾つか存在する問題点の内，ISAKMP/ Oakley は通信遅延の影響を大きく受けると言う点に着目し，この問題点に対する改善策として Moderate Mode を提案する．Moderate Mode では，プロトコル上の工夫により鍵交換デーモン間の通信回数を 1 往復少なくし，通信遅延の影響を削減する．提案した改善策を実装し，性能評価を行ない，その有効性を示す．

1.2 本論文の構成

本論文では，2章で IPsec の詳細について述べる．さらに 3章で本論文で実装する鍵交換デーモンで採用する ISAKMP/ Oakley の仕様について詳細に述べる．4章では，鍵交換デーモンの実装について述べ，実装した鍵交換デーモンの性能評価を行う．そして 5章では，ISAKMP/ Oakley における問題点と改善策を述べ，鍵交換デーモンの改良を行う．6章で，改良した鍵交換デーモンの評価・検討を行い，本論文で提案する改善策の有効性について述べる．最後に 7章にて今後の課題について述べ，8章で結論を述べる．

第 2 章

背景

クラッカによるアタックは年々進化しており，盗聴，改竄，なりすまし，トラフィック解析などは勿論のこと，近年はTCP におけるTCP RST アタックといった高度な技術を擁したアタックも報告されている．クラッカによるアタックの中には，現在規定されているプロトコルでは避け難いものもあり，これらのアタックに対抗すべく，IPsec が提案された．

勿論IPsec だけでクラッカによるアタックを完全に防ぐ事はできず，図2.1のようにIP 層の上位層における **TLS**(**Transport Layer Security**) [10] や **SSH**(**Secure Shell**)[34]，**PGP**(**Pretty Good Privacy**)[11] 等と組み合わせる事によってよりセキュリティをより強化する必要がある．しかし，IPsec はより安全なネットワークを構築する上で重要な役割を果たす．

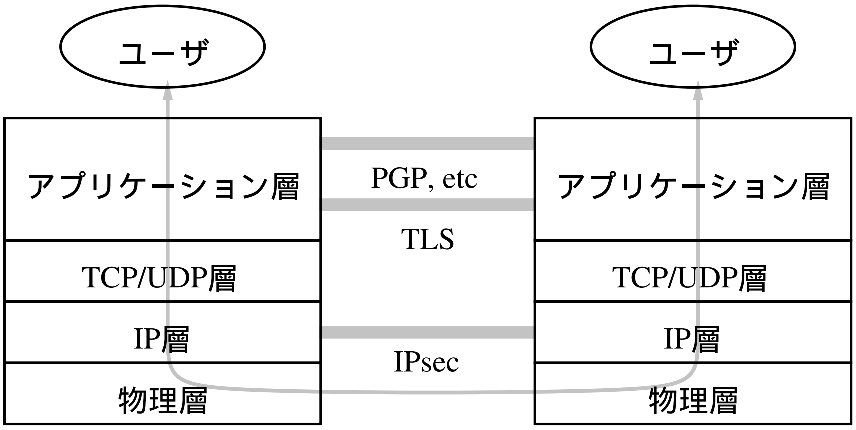


図2.1: 多段階セキュリティ機能

2.1 IPsec について

IPsec は，IETF(Internet Engineering Task Force) ipsec working group によって提案され，図2.2に示すようにIP 層におけるセキュリティ確保を目的とした新たなプロトコルである．TCP RST アタックといった高度な技術を擁したアタックの出現により，現在規定されているプロトコルでは防ぐ事が不可能となり，IPsec が誕生した．そして，IPsec では盗聴，改竄，偽造といったアタックを防ぐことが可能であり，IP パケット毎の暗号化/認証を行うことによってセキュリティを確保する．IPsec はIPv4 にもIPv6 にも対応している．そして，IPv6 をサポートしている全ての機器においてIPsec のサポートが義務付けられている．

現在IPsec の仕様には大きく分けて2 つある．1 つは，standard track rfc であるrfc1825 からrfc1829 で定義されている旧仕様のものである．そして，もう1 つはipsec working group によるinternet draft で定義され現在審議中の`draft-ietf-ipsec-*`で定義されている新仕様のものである．旧仕様と新仕様の違いは主としてリプレイアタックに対する強度の向上である．

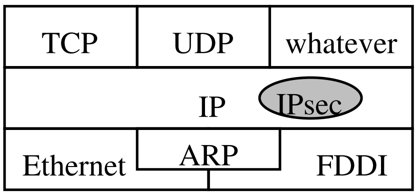


図 2.2: IPsec のレイヤリング

新仕様に関しては，現在も大きな基本ドキュメントの改定が行われている．現在の新仕様は`draft-ietf-ipsec-doc-roadmap-01.txt`[33] にはじまるドキュメント群 (`draft-ietf -ipsec-esp-v2-01.txt`[18]，`draft-ietf-ipsec-auth-header-01.txt`[17] 等) で定義されている．

2.2 IPsec の仕様

IPsec の新仕様が大きく改定されるなど未だ不確定なので，本論文では旧仕様に基づいて述べる．IPsec の仕様は大きく分けて **AH**(**Authentication Header**)[4] と **ESP**(**Encapsulated Security Payload**)[5] の 2 つに分けられる．

rfc1826[4] で規定される AH は，パケットが改竄・偽造されていない事を保証するための拡張ヘッダである．また rfc1827[5] で規定される ESP は，パケットの内容が途中経路上で盗聴される事を防ぐためのペイロードである．AH も ESP も **SA**(**Security Association**) という概念に基づいている．以降では，SA，AH，ESP について述べる．

2.2.1 SA

AH や ESP を利用するには，鍵，認証アルゴリズムや暗号化アルゴリズム，鍵の長さ，アルゴリズムの具体的な利用方法などのセキュリティ情報に関して送信者と受信者が合意する必要がある．これらのセキュリティ情報は，送信者と受信者間の SA の構成要素となる．

そして，SA を保持する単位として以下の 2 つがある．

- ホスト・ベース (SA をホスト毎に用意する)
- ユーザ・ベース (SA をユーザ毎に用意する)

図 2.3に示すように，ホスト・ベースでは，ホストを共有する全てのユーザが同じ鍵やアルゴリズムを用いることとなる．一方ユーザ・ベースでは，ホストが同じでもユーザやプロセスが異なれば，異なる鍵やアルゴリズムを用いることとなる．ホスト・ベースで SA を保持する場合，もし悪意のあるユーザが存在した時には，同一ホスト上に存在する他のユーザの通信が盗聴や改竄される危険性がある．rfc1825 では，

ホスト・ベースの SA 保持を必須とし，ユーザ・ベースの SA 保持は推奨とされている．以降では，簡単にするためにホスト・ベースを基に説明する．

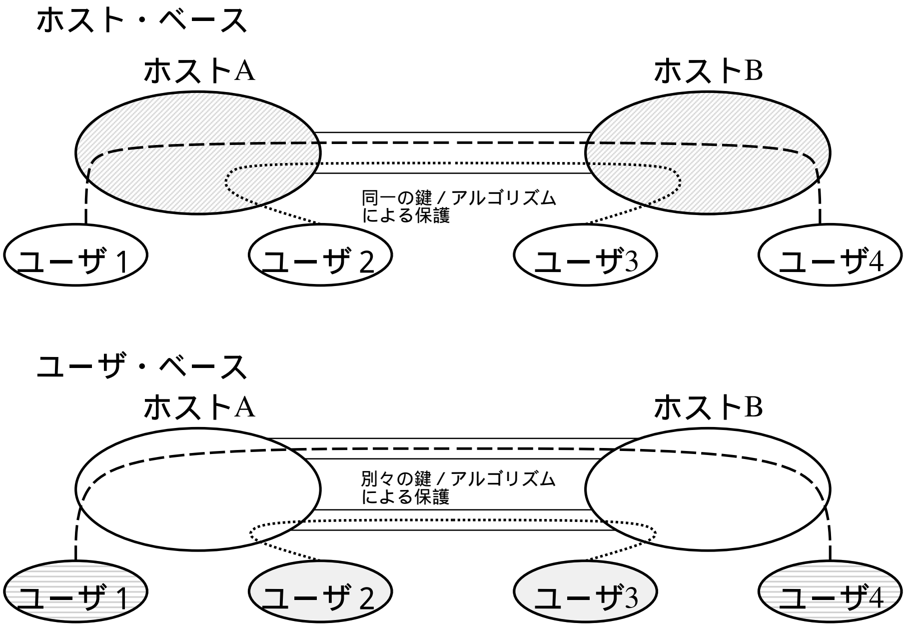


図 2.3: ホスト・ベースとユーザ・ベース

また，SA の構成要素として **SPI(Security Parameter Index)** と呼ばれるものがある．そして，SA は各ホストにおいて図 2.4のようなテーブルで保持される．SPI は，送信先ホストのアドレスと共にセキュリティ情報への指標として使用される．実際の SPI の利用方法については 2.2.2小節と 2.2.3小節で説明する．

SPI	仕様	送信元	送信先	アルゴリズム	鍵	· · · · ·
12345	AH	A	B	Keyed MD5	0123...	· · · · ·
23456	ESP	A	B	DES−CBC	9876...	· · · · ·
98765	AH	B	A	Keyed MD5	abcd...	· · · · ·
87654	ESP	B	A	DES−CBC	zyxw...	· · · · ·
⋮	⋮	⋮	⋮	⋮	⋮	⋮

図 2.4: SA テーブル

一般的な SA の特徴として，各 SA は一方向性であることが挙げられる．つまり，SA は一方向性であるので，1 つのセッションに SA は 2 つ必要となる．具体的に例を挙げて説明すると，図 2.5のようにホスト A とホスト B が双方向通信を行う際には，ホスト A からホスト B への通信に利用される SA とホスト

B からホスト A への通信に利用される SA の 2 つが必要となる．

図 2.5: SA の一方向性

また，SA は IPsec を利用した通信に先駆けて確立されていなければならない．しかし，IPsec は SA の確立方法に依存しない．そして，SA を確立する方法には主として 3 つの方法がある．

- 手動による方法··· マシン管理者等が手作業で鍵情報などを設定する方法である．接続されているマシンが少ない同一 LAN 内のような限られた範囲内では，もっとも効率的な方法であると考えられている．
- 鍵配付サーバによる方法··· **Kerberos**[16] の鍵配付センタ (**KDC**:**Key Distribution Center**) のような専門機関から鍵情報などを入手する方法である．KDC 等の専門機関がボトルネックとなるなどの弱点がある．
- 自動鍵配付による方法··· 自動的に鍵情報などを確立する方法である．具体的には鍵交換デーモンを用いる方法や SKIP という鍵交換プロトコルを用いる方法などがある．IPsec を広域利用するためには必要不可欠な方法である．鍵交換デーモンについては 3章にて詳述する．

2.2.2 AH

AH は，rfc1826 で定義されており，IP データグラムに改竄防止機能と認証機能を付与するための機構である．そして，この機構は AH と呼ばれる新しいヘッダ・フィールドを導入し，その中に IP データグラムの認証データを格納することによって実現されている．

図 2.6，図 2.7に AH を含む IP データグラムの例を示す．図 2.6が IPv4 での使用例であり，図 2.7が IPv6 での使用例である．

また，図 2.8に AH の書式を示す．図 2.8で示される各フィールドの意味は以下の通りである．

- 次ヘッダ: AH に続くデータの種類 (8 ビット)．
- AH ヘッダ長: Authentication Data フィールドの長さを 32 ビット単位で示した値 (8 ビット)．
- 予約: 将来の拡張のための予約フィールド (16 ビット)．

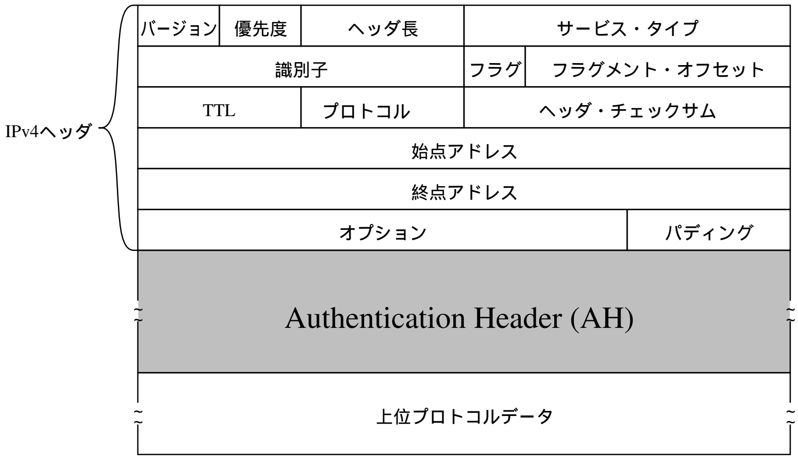


図 2.6: IPv4 での AH

- SPI: AH で使用した SA の SPI 値 (32 ビット)．
- Authentication Data: 認証データ (32 ビットの整数倍)．

AH の生成

Authentication Data は，IP データグラムに対して **MD5** などよりメッセージ・ダイジェストを算出し，さらに暗号化などの処理を施すことによって生成される．用いられるアルゴリズムは，送信者・受信者の好みによって変わるが，**Keyed MD5**[23] は必ずサポートされている必要がある．

Authentication Data は以下の手順により算出される．

1. 宛先ホストアドレスと送信者のホストアドレスより，使用する SA を決定する．これにより，認証データを計算するためのアルゴリズムや鍵が決定される．
2. IP データグラム中のペイロードと IP ヘッダ部分より，認証データが計算され，AH が生成される．ただし，受信者に届く通信過程で値が変更されるヘッダフィールドは，認証データの算出時には全て 0 であるとして扱われなければならない．認証データの算出時に 0 であるとして扱われるヘッダフィールドとして次のものが例として挙げられる．
   - IPv4
     - TTL(Time To Live)
     - ヘッダ・チェックサム
   - IPv6
     - ホップリミット

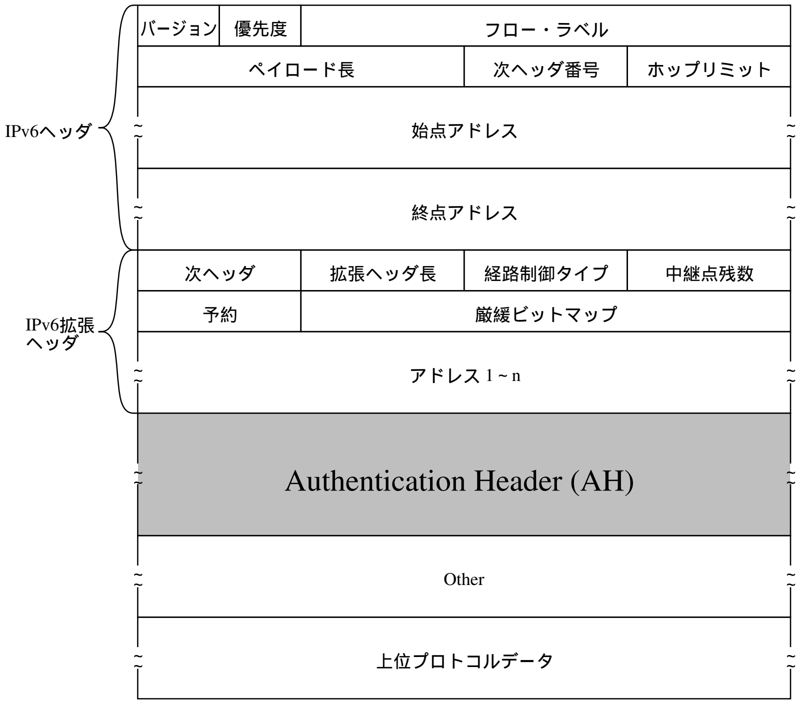


図 2.7: IPv6 での AH

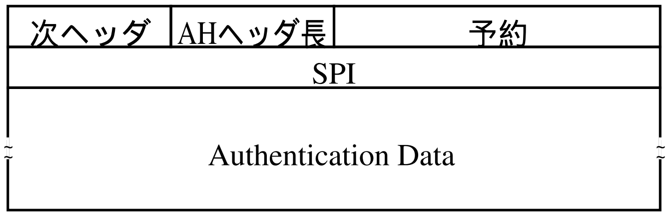


図 2.8: AH の書式

最後に，IP データグラムに AH が挿入されて，送信される．

一方，受信者は SPI と送信者のホストアドレスより，使用された SA を検索する．これにより，認証データを計算するためのアルゴリズムや鍵が決定され，送信者と同様の計算を行って，認証データを確かめることが可能となる．

2.2.3 ESP

ESP は，rfc1827 で定義されており，IP データグラム中のペイロードを暗号化することによって，盗聴や改竄や偽造を防ぐ機構である．図 2.9，図 2.10に ESP を含む IP データグラムの例を示す．図 2.9が IPv4 での使用例であり，図 2.10が IPv6 での使用例である．

また，図 2.11に ESP の書式を示す．図 2.11で示される各フィールドの意味は以下の通りである．

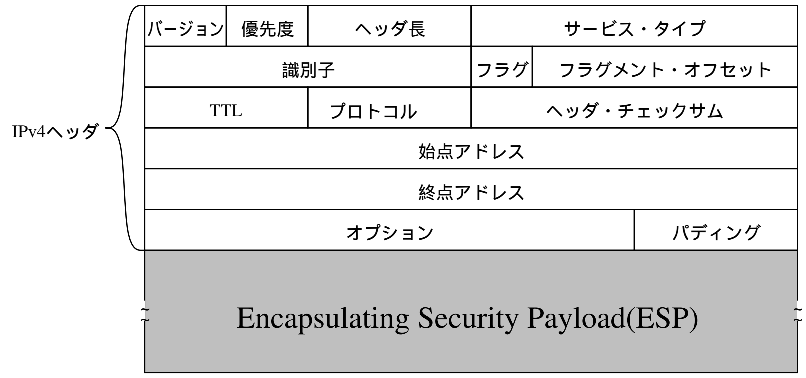


図 2.9: IPv4 での ESP

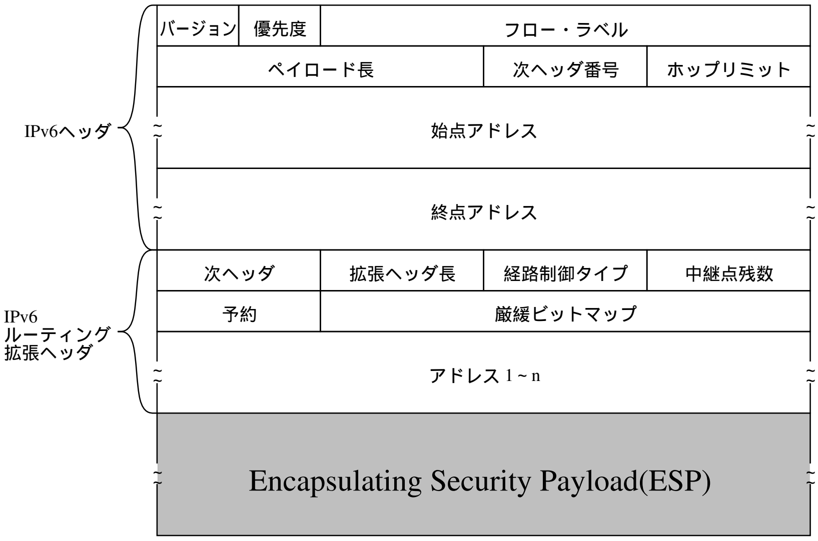


図 2.10: IPv6 での ESP

- SPI: ESP で使用した SA の SPI 値 (32 ビット)．
- Transform Data: 暗号化されたペイロード (可変長)．

ESP の生成

Transform Data は，IP データグラム中のペイロードに対して **DES**(**Data Encryption Standard**) などを用いて暗号化することによって生成される．用いられるアルゴリズムは，送信者・受信者の好みによって変わるが，**DES–CBC**[14] は必ずサポートされている必要がある．

Transform Data は以下の手順により算出される．

1. 宛先ホストアドレスと送信者のホストアドレスより，使用する SA を決定する．これにより，暗号化

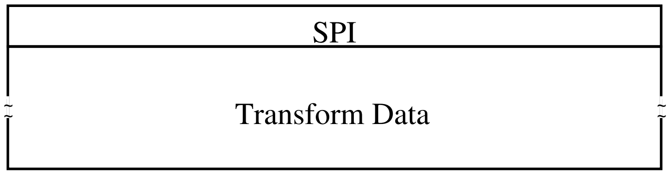


図 2.11: ESP の書式

に用いるアルゴリズムや鍵が決定される．

2. 決定したアルゴリズムや鍵を用いて IP データグラムを暗号化し，ESP に変換する．ESP に IP ヘッダを付加し送信する．

一方，受信者は ESP 中の SPI と送信者のホストアドレスより，使用された SA を検索する．これにより，暗号化に用いられたアルゴリズムと鍵が決定し，暗号化されたペイロードが復号される．

トンネルモードとトランスポートモード

ESP を用いる場合，暗号化処理の対象によって 2 種類のモードがある．一つはトンネルモードであり，もう一つはトランスポートモードである．

トンネルモード

1 つの IP データグラム全体が暗号化される．図 2.12のように，送信者は IP データグラムを完全に暗号化して ESP を生成し，その後あらためて IP ヘッダ等を付加して新しい IP データグラムを形成して送信する．受信者は，暗号化された IP データグラムを復号し，復号されたデータグラムをあらためて処理する．

トランスポートモード

TCP/UDP 等上位層のデータのみが暗号化される．図 2.13のように，上位層のデータは，暗号化されて ESP に変換された後に，IP ヘッダなどを付加されて送信される．受信者は，暗号化された上位層のデータを復号し，復号されたデータを上位層に渡す．

2.3 IPsec の広域運用

現在 IPsec は，必要性や重要性は充分認識されているにも関わらず，特に日本においてはほとんど広域運用されていない．これには以下のような理由が挙げられる．

1. IPsec について研究しているグループの多くは米国にあり，しかも米国の輸出規制により，日本など米国以外では IPsec を利用できない．

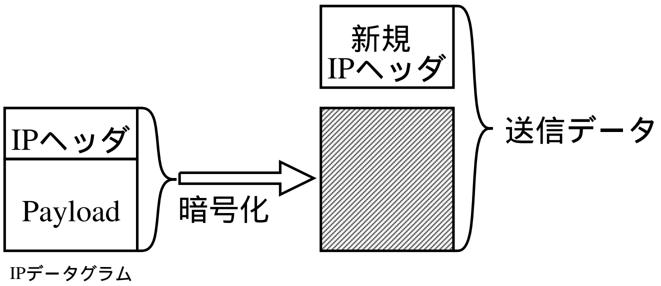


図 2.12: トンネルモード

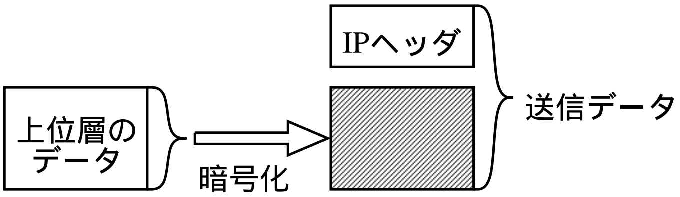


図 2.13: トランスポートモード

2. IPsec の広域運用に必要不可欠な鍵交換デーモンの実装が，IPsec の実装以上に少ない．しかも，鍵交換デーモンの実装は主に米国で行われており，米国の輸出規制によって日本など米国以外では鍵交換デーモンを利用できない．

現在日本では **WIDE** プロジェクト **IPv6/whips**[1] working group で IPsec をサポートした 4.4BSD カーネルの実装が幾つか行われている [1]．そして，鍵交換デーモンの実装も幾つか行われているが，IPsec 以上に数少なく，広く行き渡っていない．

---

[1] WIDE Happy IP Security

第 3 章

鍵交換デーモン (ISAKMP/ Oakley)

現在知られている鍵交換プロトコルとして，NSA(National Security Agency) が提案したISAKMP(Internet Security Association and Key Management Protocol)[20] やアリゾナ大学が提案した **Oakley** [26]，CISICO が提案した ISAKMP/ Oakley[9]，Qualcom が提案した Photuris[15]，SUN が提案した SKIP(Simple Key management for Internet Protocols)[8][7] 等が存在する．SKIP は，AH と同様拡張ヘッダとして定義されており，IPv6 の拡張ヘッダによって鍵が交換される．そして，提案されたいずれの鍵交換プロトコルも **Diffie–Hellman**[28] [30] 鍵交換アルゴリズムを利用している．

ここで，鍵交換プロトコルの歴史について述べる．長い間，Photuris と SKIP が IPv6 の鍵交換プロトコルの仕様として争われていた．しかし，1996 年 6 月のインターネットドラフトを最後に，IETF は Photuris の提案を取り下げ，Photuris の思想は ISAKMP と Oakley に引き継がれた．先に ISAKMP と Oakley は共に鍵交換プロトコルとして説明したが，実際には ISAKMP は鍵交換の枠組を提供し，Oakley は鍵交換機能を定義している．そして，ISAKMP の鍵交換の枠組と Oakley の鍵交換機能をうまく組み合わせたものが ISAKMP/ Oakley である．

鍵交換プロトコルが広く世界中で使用されるために，幾つか存在する鍵交換プロトコルの中からどれか 1 つを選択する必要があった．しかし，1996 年 6 月に Montreal で開催された IETF の ipsec working group の会議でも鍵交換プロトコルに関する合意は得られなかった．そこで会議後，妥協案を作成するためのチームが結成された．そして，12 月に San Jose で開催される IETF の会議までに妥協案が作成されないことを恐れたセキュリティ・エリア長の Jeffrey I. Schiller は，妥協案の作成期限を 9 月 1 日とした．しかし，残念ながら，作成期限までに妥協案が作成されなかったため，それ以上の混乱を避けるために，Schiller は自ら SKIP と ISAKMP/ Oakley の両プロトコルを評価し，決定を下した．そして 1996 年 9 月 19 日に IPv6 に必須の鍵交換プロトコルとして ISAKMP/ Oakley に決定した旨のメッセージが IPsec のメーリングリスト[1]に投稿された．以上の経過を 3.1にまとめた．

表 3.1: 鍵交換プロトコルが決定されるまで

1996. 6	IETF が Photuris の提案を取り下げる． Montreal にて IETF の会議が開催される． 鍵交換プロトコルの妥協案作成チームが結成される．
1996. 9. 1	鍵交換プロトコルの妥協案作成期限．
1996. 9.19	鍵交換プロトコルを ISAKMP/ Oakley に決定．
1996.12	San Jose にて IETF の会議が開催される．

ISAKMP/ Oakley が選ばれた理由として，SKIP が限られた範囲でしか利用できないことが挙げられている．ISAKMP/ Oakley も SKIP も共に Diffie–Hellman 鍵交換アルゴリズムを利用するが，ISAKMP/ Oakley では Diffie–Hellman 鍵交換アルゴリズムのパラメータを交換する機能があるのに対し，SKIP ではあらかあじめパラメータを決定しておく必要がある．つまり，SKIP では通信するコンピュータ群があ

---

[1] ipsec@tis.com

らかじめ共通のパラメータを共有する必要があり，結局管理できる範囲が限られてしまうのである [13]．

Schiller の決定により，ISAKMP/ Oakley を基に **IKMP(Internet Key Management Protocol)** を作成することが宣言されたが，未だ作成されていない．そこで本章では，IKMP の基となる ISAKMP/ Oakley について詳述する．

3.1 Diffie–Hellman 鍵交換アルゴリズム

ISAKMP/ Oakley 鍵交換プロトコルについて述べる前に，ISAKMP/ Oakley に限らず多くの鍵交換プロトコルで利用されている Diffie–Hellman 鍵交換アルゴリズムについて説明する．なお，以降 Diffie–Hellman 鍵交換アルゴリズムを Diffie–Hellman と記述する．

Diffie–Hellman は，整数領域における離散対数の性質を利用した鍵交換方式であり，離散対数方式とも呼ばれる．はじめに，Diffie–Hellman が利用している離散対数の性質について説明する．

離散対数問題とは，整数$\alpha/x/y$がある素数 qを法として

$$y = \alpha^x \pmod q \tag{3.1}$$

なる関係を満たすとき，$\alpha/q/y$の値から x の値を求めることを言う．この場合，“x は yのαを底とする離散対数である” と言う．なお，αが $\{1, 2, \cdots, q-1\}$ 上での原始元である時，すなわち $\{\alpha^0, \alpha^1, \cdots, \alpha^{q-2}\} \pmod q$ が全て異なる元である，という条件が満たされる時，全ての yについて式 (3.1) を満たす x が存在する事が保証されている．現在のところ離散対数問題の最適な解法は存在せず，離散対数問題を解く事は非常に困難である．

この離散対数問題を用いた実際の Diffie–Hellman の手順は次のようになる．例としてユーザ A とユーザ Bが，Diffie–Hellman を用いて鍵交換を行うとする．

1. ユーザ A とユーザ Bは使用する素数 qならびに原始元αに関して予め合意を取っておく．
2. ユーザ A，ユーザ Bはそれぞれ，$\{1, 2, \cdots, q-1\}$ の中から任意の整数 X_A，X_Bを選び，それを自分の秘密情報にする．さらに，X_A, X_Bから，

$$K_A = \alpha^{X_A} \bmod q \tag{3.2}$$

$$K_B = \alpha^{X_B} \bmod q \tag{3.3}$$

   を計算し，これらをそれぞれの公開情報とする．
3. ユーザ A はユーザ Bに公開情報 K_Aを，ユーザ Bはユーザ A に公開情報 K_Bを送信する．
4. ユーザ A は，ユーザ Bから受け取った公開情報 K_Bと自分の秘密情報 X_Aより，共有情報 K_{AB}を

$$K_{AB} = (K_B)^{X_A} \bmod q$$

$$\begin{aligned} &= (\alpha^{X_B})^{X_A} \bmod q \\ &= \alpha^{X_B \cdot X_A} \bmod q \\ &= \alpha^{X_A \cdot X_B} \bmod q \qquad (3.4) \end{aligned}$$

という計算より求める．

5. ユーザBも同様に，ユーザAから受け取った公開情報K_Aと自分の秘密情報K_Aより，共有情報K_{AB}を

$$\begin{aligned} K_{AB} &= (K_A)^{X_B} \bmod q \\ &= \alpha^{X_A \cdot X_B} \bmod q \qquad (3.5) \end{aligned}$$

という計算より求める．

離散対数問題を用いた鍵交換アルゴリズムでは，X_AもしくはX_BからK_AもしくはK_Bを求める計算

$$K = \alpha^X \bmod q \qquad (3.6)$$

が一方向関数となっている．従って，X_AもしくはX_Bが既知であれば，K_AもしくはK_Bを容易に求めることができる．しかし，逆にK_AもしくはK_BからX_A，X_Bを求めることが，非常に困難となる．この性質を利用して，両者が共有情報として鍵を安全に確立することが可能となる．

3.2 ISAKMP/ Oakley

先にも述べた通り，ISAKMP/ Oakley は，鍵交換の枠組を提供する ISAKMP と鍵交換の機能を定義する Oakley を組み合わせたものである．ISAKMP では，SA を確立，変更，削除をするための処理方法とパケットフォーマットを定義している．また，Oakley は ISAKMP の枠組の中で実際のパラメータの意味を定義している．

ISAKMP/ Oakley は，IPsec や TLS などあらゆるセキュリティプロトコルのための SA を確立する事が可能である事を目的として設計されている．また ISAKMP/ Oakley は，暗号アルゴリズム，鍵交換アルゴリズム，セキュリティ機構などを独立に扱うことが可能であるため，以下の点において有益である．

- 動的な環境に対応する事が可能である．
- より安全なセキュリティ機構やアルゴリズムの採用が容易である．

そして，ISAKMP/ Oakley は以下の攻撃を防ぐことが可能である．

- 過負荷によるサービス停止
- リプライ攻撃

- なりすまし
- コネクションハイジャック

ISAKMP/ Oakley は，**Phase 1**，**Phase 2** という 2 つのフェーズから構成され，Phase 1 と Phase 2 を経て SA が確立される．Phase 1 では，ISAKMP/ Oakley のための SA が確立される．ISAKMP/ Oakley のための SA を **ISAKMP SA** と呼ぶ．また Phase 2 では，ISAKMP SA による保護の下で SA を必要とするセキュリティプロトコル (IPsec や TLS など) のための SA が確立される (今後 SA と記述した時には，IPsec や TLS などのセキュリティプロトコルのための SA を意味する)．ISAKMP/ Oakley では，一度 Phase 1 を経て確立された ISAKMP SA の下で幾つもの SA を確立することが可能である．具体的には，一度ホスト A とホスト Bの間で ISAKMP SA が確立されると，ISAKMP SA の有効期限が切れるまで，ホスト A とホスト B間において IPsec 用の SA，TLS 用の SA と言った様に幾つもの SA を確立することが可能なのである．また，ISAKMP/ Oakley は，IPsec 用の SA にしてもホストベースの SA のみでなくユーザベースの SA を確立する事も可能であり，汎用性の高い鍵交換プロトコルである．

さらに，Phase 1 には **Main Mode** と **Aggressive Mode** の二つのモードがあり，そして Phase 2 には **Quick Mode** がある．また，例外として Phase 1，Phase 2 のどちらにも属さない **New Group Mode** がある．フェーズ毎にそれぞれのモードについて以下で詳しく述べる[2] が，その前に本論文で使用する表記法[3]について述べておく．また，表記の説明中で用いるヘッダやペイロードのフォーマットについては Appendix Aを参照されたい．

3.3 表記法

本節において本論文で使用する表記方法について述べる．

Initiator: SA 確立要求の送信者．

Responder: SA 確立要求の受信者．

HDR: ISAKMP のヘッダ (`ISAKMP Header:` A.1.1小節) を表す．

HDR*: ISAKMP のヘッダ (`ISAKMP Header:` A.1.1小節) を表すが，ヘッダ以後が暗号化されていることを示す．

SA: ISAKMP SA または SA を確立するために必要な情報を含むペイロード (`Security Association Payload:` A.1.4小節) を表す．

KE: 鍵を生成するために必要な情報を含んだペイロード (`Key Exchange Payload:` A.1.7小節) を表す．

---

[2] `draft-ietf-ipsec-isakmp-oakley-04.txt` に従う．

[3] ここで使用する表記方法は，Internet Draft で使用されている表記方法に基づく．

$\mathbf{N}_i$，$\mathbf{N}_r$: 乱数を含んだペイロード (`Nonce Payload:` A.1.13小節) を表す．

$\mathbf{ID}_{ii}$，$\mathbf{ID}_{ir}$: それぞれ Initiator，Responder の ID を含んだペイロード (`Identification Payload:` A.1.8小節) を表す．Phase 1 で使用する．

$\mathbf{ID}_{ui}$，$\mathbf{ID}_{ur}$: それぞれ Initiator，Responder の ID を含んだペイロード (`Identification Payload:` A.1.8小節) を表す．Phase 2 で使用する．

SIG: 電子署名を含んだペイロード (`Signature Payload:` A.1.12小節) を表す．

CERT: 認証書を含んだペイロード (`Certificate Payload:` A.1.9小節) を表す．

HASH: ハッシュ値を含んだペイロード (`Hash Payload:` A.1.11小節) を表す．

g^{x_i}，g^{x_r}: それぞれ Initiator，Responder の Diffie–Hellman の公開情報である．

$< x > y$: x が鍵 y で暗号化されていることを表す．

[x]: x ペイロードがオプションであることを表す．

$prf(key, msg)$: 鍵付き疑似乱数発生関数を表す．

$hash$: ハッシュ関数を表す．

SKEYID: Initiator と Responder が共有する秘密値を表す．

SKEYID_e: ISAKMP SA でメッセージを暗号化する際に鍵として使用される値を表す．

SKEYID_a: ISAKMP SA でメッセージの改竄を防ぎ，認証を行うために使用される値を表す．

SKEYID_d: SA の鍵を生成する際に種として使用される値を表す．

CKY: クッキーと呼ばれる疑似乱数を表す．HDR 中に含まれている．

M_ID: メッセージの ID を表す．HDR 中に含まれ，不正なメッセージの再送を防ぐために使用されるメッセージの通し番号である．

|: 連結を表す．

3.4 認証方式

各モードについて説明する前に，認証方法について述べる．不正な相手と SA を確立してしまっては無意味なので，ISAKMP/ Oakley は確実に通信相手を認証する必要がある．そこで，ISAKMP/ Oakley には，以下の 3 つの認証方式が提供されている．

- 電子署名を用いる方法
- 公開鍵暗号を用いる方法
- 模擬共有鍵を用いる方法

ISAKMP/ Oakley の仕様では，模擬共有鍵を用いた認証方式のサポートが義務づけられている．従って，模擬共有鍵を利用できる場合には，模擬共有鍵を用いた認証が行われる．しかし模擬共有鍵を利用できない場合に，電子署名と公開鍵暗号のいずれを優先して利用すべきであるかに関しては特に規定されていない．以下に各認証方式の概念について述べる．

3.4.1 電子署名を用いる方法

電子署名を用いた認証方法には，**RSA**(**Rivest, Shamir, Adleman**) のような公開鍵暗号アルゴリズムを用いる方法と **DSS**(**Digital Signature Standard**)[31] のような **DSA**(**Digital Signature Algorithm**)[29] を用いる 2 つの方法がある．

公開鍵暗号アルゴリズムを用いる方法では，メッセージのハッシュ値を自分の秘密鍵で暗号化した電子署名を元のメッセージと共に送信する．受信者は，送信されてきた電子署名を送信者の公開鍵で復号し，送信されてきたメッセージより求めたメッセージのハッシュ値と比較する．そして比較したものが一致することによって，相手を認証する．

DSS を用いる方法では，自分の秘密値を用いてメッセージ (普通はメッセージのハッシュ値) に電子署名し，メッセージと共に送信する．受信者は，相手の公開値を用いてメッセージと電子署名を照合する．正しく照合できることによって相手を認証する．

電子署名を用いる方法では，あらかじめ通信相手の公開鍵や公開値を知らなければならない．さもなくば，**S–DNS**(**Secure–Domain Name System**)[25] や **CA**(**Sertification Authority**) と言った信用できる機構より通信相手の公開鍵や公開値を確実に入手する必要がある．

3.4.2 公開鍵暗号を用いる方法

公開鍵暗号を用いる方法では，RSA のような公開鍵暗号アルゴリズムを用いて，相手の公開鍵である秘密情報を暗号化して送信する．正規の受信者のみが，暗号化されたメッセージを復号でき，秘密情報を知る事ができる．そして，入手した秘密情報を用いてその後のメッセージ交換を矛盾なく正しくできるかどうかで相手の認証を行なう．

公開鍵暗号を用いる方法では，電子署名を用いる方法同様に，あらかじめ通信相手の公開鍵を知らなければならない．さもなくば，S–DNS や CA と言った信用できる機構より通信相手の公開鍵を確実に入手する必要がある．

3.4.3 模擬共有鍵を用いる方法

模擬共有鍵を用いる方法では，通信者間である秘密情報を予め共有しておく．秘密情報は，特定の通信相手しか知られておらず，秘密情報を知っている者は確かな通信相手であるという考えに基づいて認証を行う．具体的には，秘密の共有値を共有鍵としてメッセージを暗号化し，相手が確実にメッセージを復号でき，矛盾なくメッセージの交換を行えるかどうかによって相手の認証を行う．

共有情報の交換方法に関しては，ホストの管理者どうしが電話で交換する方法や共有値を書き込んだフロッピーディスクを郵送するなど様々な方法が考えられる．しかし，ISAKMP/ Oakley では共有情報の交換方法に関しては配慮されておらず，定義されていない．

3.4.4 公開鍵・公開値の入手方法について

電子署名を用いる方法や公開鍵暗号を用いる方法では，互いに相手の公開鍵や公開値を必要とする．そして，通信相手の公開鍵や公開情報を知らない場合には，相手の公開鍵を入手し，且つ入手した公開鍵が正当なものである事を確かめる必要がある．そこで，S–DNS や CA，KDC 等の第 3 者認証機関に頼らなければならない．

今後，Electronic Commerce の発展に伴い CA の普及が予想される．また，CA より簡単な鍵配布方式として S–DNS の普及も予想される．現在，CA は実用実験として幾つか稼働しているが，未だ実用実験であるため，各 CA のプロトコルに互換性がないといった問題や，相互接続がおこなわれていないなどといった問題がある．従って，CA は未だ汎用的でなく一般に広く利用されるに至っていない．

一方 S–DNS は，1997 年 12 月の時点では RSA のライセンス問題や米国の輸出規制のため，アメリカ・カナダでしか利用できない．しかし，**ISC(Internet Software Consortium)** が RSA 社との間で，S–DNS への利用に対してのみ RSA のフリーライセンス契約に達した [3]．また，米国の輸出規制が緩くなってきており，近いうちに現在の DNS に代わって S–DNS が広まり，一般に S–DNS が使われることが予想される．

一方，KDC は S–DNS と同様鍵配布を行なうが，元は限られた範囲内での鍵配布のために考えられたものである．従って KDC は，広域運用には適しておらず一般的ではない．

3.5 ISAKMP/ Oakley Phase 1

Phase 1 は ISAKMP SA を確立するフェーズである．Phase 1 では，ISAKMP/ Oakley が使用する暗号アルゴリズムや認証アルゴリズム等を決め，ISAKMP/ Oakley がメッセージを暗号化や認証に用いるセッション鍵を算出し，通信相手を確認および認証する．

Phase 1 には，Main Mode と Aggressive Mode の 2 種類のモードがある．Main Mode では，通信相手を確認に利用される ID を暗号化して送信する事によって ID の保護が可能である．しかし，Aggressive Mode では ID を暗号化して送信する事ができないので，ID の保護が不可能である．ID を保護しない時には，なりすましや SA の確立妨害を被る可能性がある．また，ISAKMP/ Oakley には，3.4節で述べた様

に電子署名を用いる方法，公開鍵暗号を用いる方法，模擬共有鍵を用いる方法という3種類の認証方式が提供されており，各認証方式によって Phase 1 中で交換されるメッセージが異なる．以下では，3つの認証方式に応じて Main Mode と Aggressive Mode について説明する．

3.5.1 Main Mode

Main Mode は，ISAKMP で定義されている **Exchange Type** 中の **Identity Protection Exchange** (A.2.2章) に当たる．Main Mode での大まかな処理の流れは次の通りである．

1. ISAKMP SA で使用する暗号アルゴリズムや認証アルゴリズム等，ISAKMP SA を確立する環境を決定する．
2. 鍵生成に必要な Diffie–Hellman の公開情報や補助的なパラメータ等を交換する．
3. これまでにやり取りしたデータを元にセッション鍵を生成する．そして通信相手を認証すると同時に生成されたセッション鍵が正しいことを確認する．

以上の流れを図3.1に示す．

Main Mode では上記の処理を6回 (3往復) の通信を経て行なう．始めの2回 (1往復) の通信 (図3.1 ：1，2) では，ISAKMP SA で使用する暗号アルゴリズムや認証アルゴリズム等 ISAKMP SA を確立する環境を取り決める．1回目の通信 (図3.1 :1) で，Initiator は Responder に対して Initiator が利用可能な各種アルゴリズムを，使用する希望順に並べて送信する．Responder は列挙された各種アルゴリズムの中から，Responder が利用可能であり，かつ一番初めに該当したアルゴリズムを Initiator に送り返す (図3.1 ：2)．これで，利用する各種アルゴリズムが決定する．

次に，セッション鍵を生成するために必要な情報である，Diffie–Hellman の公開情報や補助的なパラメータを交換するためにさらに2回 (1往復) の通信 (図3.1 :3，4) を行なう．この3回目の通信 (図3.1 ：3) と4回目の通信 (図3.1 :4) によって，Initiator と Responder のみが知り得る Diffie–Hellman の共有情報を共有することができる．そして，Initiator と Responder は Diffie–Hellman の共有情報とこれまでの通信内容を基にセッション鍵を算出する．この時始めに全てのセッション鍵の元となる SKEYID が算出される．そしてこの SKEYID より SA で用いられる鍵の種となる SKEYID_d や ISAKMP SA で用いられる認証用の鍵 SKEYID_a，ISAKMP SA で用いられる暗号用の鍵 SKEYID_e が算出される．全てのセッション鍵の元となる SKEYID は認証方式によって算出方法が異なるが，SKEYID 以外の SKEYID_d，SKEYID_a ，SKEYID_e は認証方式に関係なく同じ算出方法で求められる．

そして，通信相手の確認・認証と算出したセッション鍵が正しい物であることを確かめるために，通算5回目の通信 (図3.1 :5) と6回目の通信 (図3.1 :6) を行なう．これまでのメッセージ交換によって両者が知り得る情報と SKEYID_a を元にハッシュ値を求める．そして，求めたハッシュ値を SKEYID_e を鍵として暗号化して送信する．送信されて来たメッセージを正しく復号でき，且つ送信されて来たハッシュ値

と自分が計算したハッシュ値が等しいか否かによってセッション鍵の確認を行う．さらに，セッション鍵は特定の通信相手のみが知り得る情報なども基にして生成されるために，セッション鍵を正しく生成出来なかった通信相手は，不正な相手であるということが判明する．これにより通信相手の確認・認証を行う．

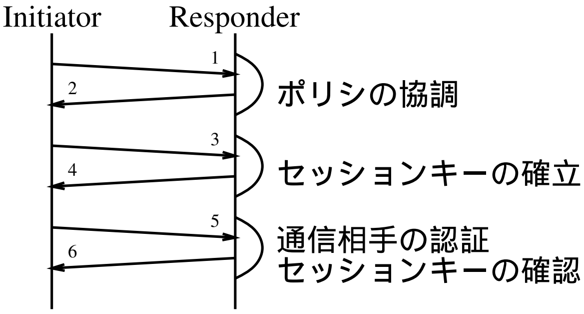


図 3.1: Main Mode の流れ

SKEYID_d，SKEYID_a，SKEYID_e の算出方法はそれぞれ式 (3.7)，式 (3.8)，式 (3.9) の様に定義される．尚，SKEYID の算出方法は，認証方式によって異なるため，後述する．

$$SKEYID\_d = prf(SKEYID, g^{x_i x_r}|CKY\_I|CKY\_R, 0) \tag{3.7}$$

$$SKEYID\_a = prf(SKEYID, SKEYID\_d|g^{x_i x_r}|CKY\_I|CKY\_R, 1) \tag{3.8}$$

$$SKEYID\_e = prf(SKEYID, SKEYID\_a|g^{x_i x_r}|CKY\_I|CKY\_R, 2) \tag{3.9}$$

以下では，各認証方式に対する Main Mode の処理の流れ及び特徴を述べる．

電子署名を用いた Main Mode

ここでは，Main Mode において認証方式として電子署名を用いる場合の特徴について述べる．電子署名を用いた認証方法については3.4.1小節で述べたので，ここでは説明しない．

処理の流れを図 3.2に示す．電子署名を用いる時の特徴は，SIG_I，SIG_R を送信する点である．SIG_I，SIG_R は，それぞれ HASH_I，HASH_R に対して電子署名を施した物であり，電子署名の方法には RSA を用いる方法や，DSS を用いる方法等があことは既に述べた．SKEYID や HASH_I，HASH_R はそれぞれ式 (3.10)，式 (3.11)，式 (3.12) より求められる．

$$SKEYID = prf(N_i|N_r, g^{x_i x_r}) \tag{3.10}$$

$$HASH\_I = prf(SKEYID, g^{x_i}|g^{x_r}|CKY\_I|CKY\_R|SA|ID_{ii}) \tag{3.11}$$

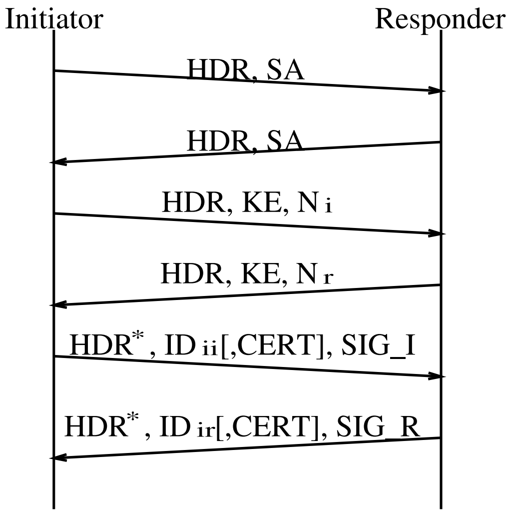


図 3.2: 電子署名を用いた Main Mode

$$HASH\_R = prf(SKEYID, g^{x_r}|g^{x_i}|CKY\_R|CKY\_I|SA|ID_{ir}) \tag{3.12}$$

公開鍵暗号を用いた **Main Mode**

ここでは，Main Mode において認証方式として公開鍵暗号を用いる場合の特徴について述べる．公開鍵暗号を用いた認証方法については 3.4.2小節で述べたので，ここでは説明しない．

処理の流れを図 3.3に示す．公開鍵暗号を用いる時の特徴は，3 回目，4 回目の通信で，セッション鍵の生成に必要な Diffie–Hellman の公開鍵と補助的なパラメータそれぞれを相手の公開鍵で暗号化することである．現在は公開鍵暗号アルゴリズムとして RSA のみが利用可能である．SKEYID は式 (3.13) より，また HASH_I，HASH_R は，それぞれ式 (3.11)，式 (3.12) より求められる．HASH(1) は CA や S–DNS によって発行された認証書に取り決めたハッシュ関数を作用させることにより求めたハッシュ値である．

$$SKEYID = prf(N_i|N_r, CKY\_I|CKY\_R) \tag{3.13}$$

模擬共有鍵を用いた **Main Mode**

ここでは，Main Mode において認証方式として模擬共有鍵を用いる場合の特徴について述べる．模擬共有鍵を用いた認証方法については 3.4.3小節で述べたので，ここでは説明しない．

処理の流れを図 3.4に示す．模擬共有鍵を用いる時の特徴は，模擬共有鍵は既に何らかの方法で共有している事が前提となっており，電子署名や公開鍵暗号を用いる方法と比べて処理が比較的簡単であることである．SKEYID は式 (3.14) より，また HASH_I，HASH_R は，それぞれ式 (3.11)，式 (3.12) より求められる．

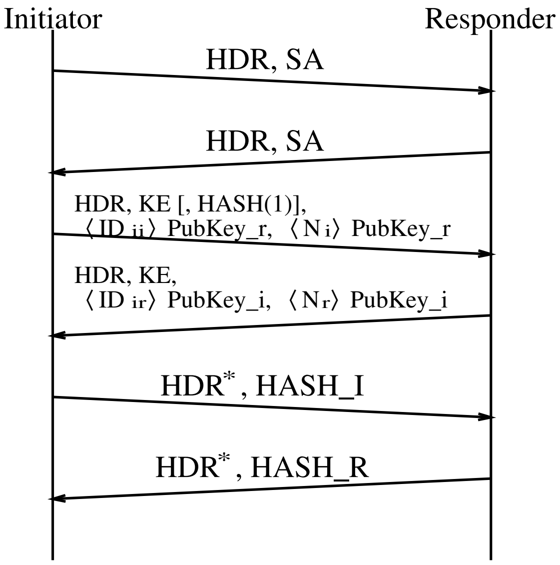


図 3.3: 公開鍵暗号を用いた Main Mode

$$SKEYID = prf(pre\_shared\_key, N_i|N_r,) \tag{3.14}$$

3.5.2 Aggressive Mode

Aggressive Mode は，ISAKMP で定義されている Exchange Type 中の **Aggressive Exchange**(A.2.4 章) に当たる．Aggressive Mode では，Main Mode より通信回数が少ない代わりに，セッション鍵の生成が完了する前に，ID を送信する．そのため，ID に対して暗号処理を施す事ができず，ID の保護を行うことができない．ただし，認証方法に公開鍵暗号を用いた時のみ，通信相手の公開鍵で暗号化するため，ID の保護を行うことが可能となる．Aggressive Mode での処理の大まかな流れは次の通りである．

1. ISAKMP SA で使用する暗号アルゴリズムや認証アルゴリズムを取り決め，同時にセッション鍵の生成に必要な Diffie–Hellman の公開情報や補助的なパラメータを交換する．そして，交換したデータを元にセッション鍵を生成する.
2. 通信相手を認証すると同時に生成したセッション鍵が正しいことを確認する．

以上の流れを図 3.5に示す．Aggressive Mode では上記の処理を 3 回 (1.5 往復) の通信を経て行う．1 回目の通信 (図 3.5 :1) で，Initiator は，ISAKMP SA を確立する環境を取り決めるために，Responder に対して Initiator が利用可能な各種アルゴリズムを，使用する希望順に並べて送信する．この時に同時に，セッション鍵の生成に必要な Diffie–Hellman の公開情報と補助的なパラメータを共に送信する．Responder は，列挙された各種アルゴリズムの中から，Responder が利用可能であり，かつ一番始めに該当したアル

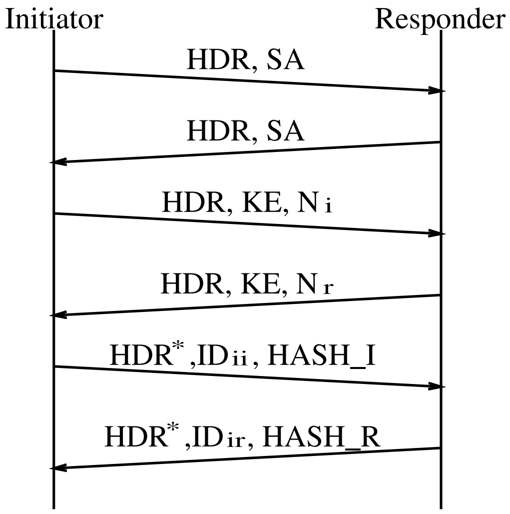


図3.4: 模擬共有鍵を用いたMain Mode

ゴリズムを使用するアルゴリズムに決定する．さらにResponderは，セッション鍵を計算した後，使用するアルゴリズムとDiffie–Hellmanの公開情報と補助的なパラメータ，そしてInitiatorから受け取った情報とセッション鍵を元に求めたハッシュ値をInitiatorに送信する(図3.5 :2)．これを受け取ったInitiatorは，セッション鍵を計算し，送信されてきたハッシュ値から，セッション鍵が正しいものであるかどうかを確かめる．さらに，受け取った情報とセッション鍵を元に求めたハッシュ値をResponderに送信する(図3.5 :3)．これにより，Responderも正しいセッション鍵が確立されたことを知ることができる．

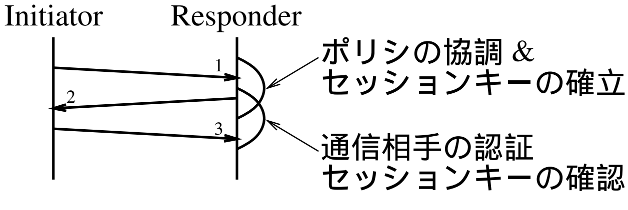


図3.5: Aggressive Modeの流れ

Aggressive ModeでもMain Modeと同様の方法でセッション鍵が算出される．始めに全てのセッション鍵の元になるSKEYIDが算出され，このSKEYIDを元にSAで用いられる鍵の種となるSKEYID_dや，ISAKMP SAで用いられる認証用の鍵SKEYID_a，ISAKMP SAで用いられる暗号用の鍵SKEYID_eが算出される．Main Modeと同様SKEYID_d，SKEYID_a，SKEYID_eはそれぞれ式(3.7)，式(3.8)，式(3.9)より算出される．そして，SKEYIDは認証方式によって算出方法が異なる．

以下で，各認証方式に対するAggressive Modeの処理の流れ及び特徴を述べる．

電子署名を用いた Aggressive Mode

ここでは，Aggressive Mode において認証方式として電子署名を用いる場合の特徴について述べる．電子署名を用いた認証方法については 3.4.1小節で述べたので，ここでは説明しない．

処理の流れを図 3.6に示す．電子署名を用いる時の特徴は，3.5.1小節で述べた電子署名を認証方式に用いた Main Mode の時と同様，HASH_I，HASH_R に対して電子署名を施した SIG_I，SIG_R を送信することである．電子署名の方法には RSA を用いる方法や，DSS を用いる方法等がある．SKYID や HASH_I，HASH_R はそれぞれ式 (3.7)，式 (3.11)，式 (3.12) より求められる．

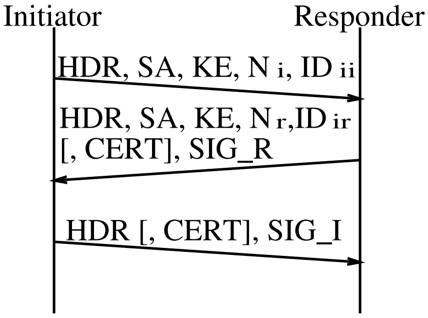


図 3.6: 電子署名を用いた Aggressive Mode

公開鍵暗号を用いた Aggressive Mode

ここでは，Aggressive Mode において認証方式として公開鍵暗号を用いる場合の特徴について述べる．公開鍵暗号を用いた認証方法については 3.4.2小節で述べたので，ここでは説明しない．

処理の流れを図 3.7に示す．公開鍵暗号を用いる時の特徴は，3.5.1小節で述べた公開鍵暗号を認証方式に用いた Main Mode の時同様，セッション鍵の生成に必要な Diffie–Hellman の公開情報や補助的なパラメータを相手の公開鍵で暗号化することである．しかもこの時，ID も相手の公開鍵で暗号化するので，ID の保護が Aggressive Mode にもかかわらず可能である．SKEYID や HASH_I，HASH_R はそれぞれ式 (3.13)，式 (3.11)，式 (3.12) より求められる．また，HASH(1) は CA や S–DNS によって発行された認証書に取り決めたハッシュ関数を作用させることにより求めたハッシュ値である．

模擬共有鍵を用いた Aggressive Mode

ここでは，Aggressive Mode において認証方式として模擬共有鍵を用いる時の処理の特徴について述べる．模擬共有鍵を用いた認証方法については 3.4.3小節で述べたので，ここでは説明しない．

処理の流れを図 3.8に示す．模擬共有鍵を用いる時の特徴は，3.5.1小節で述べた模擬共有鍵を認証方式に用いた Main Mode の時同様 Initiator と Responder が模擬共有鍵を既に何らかの方法で共有している事

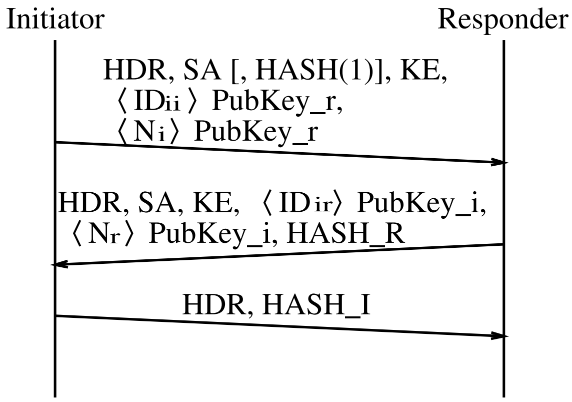


図 3.7: 公開鍵暗号を用いた Aggressive Mode

を前提としており，電子署名や公開鍵暗号を用いる方法と比べて比較的単純なことである．また，SKEYID や HASH_I，HASH_R はそれぞれ式 (3.14)，式 (3.11)，式 (3.12) より求められる．

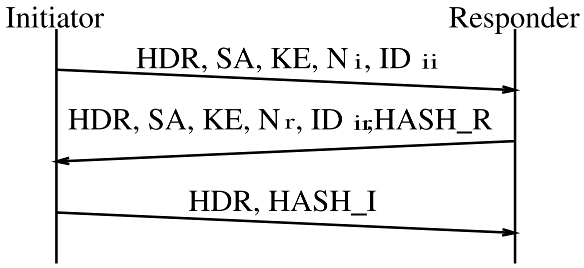


図 3.8: 模擬共有鍵を用いた Aggressive Mode

3.6 ISAKMP/ Oakley Phase 2

ISAKMP SA を利用して SA を確立するフェーズである．Phase 2 では，SA で使用する暗号アルゴリズムや認証アルゴリズム等を決め，SA で使用する鍵を算出し，最後に最終確認を行う．Phase 2 には Quick Mode というモード 1 種類だけがある．

3.6.1 Quick Mode

Quick Mode は，ISAKMP で定義されている Exchange Type 中の **Authentication Only Exchange**(A.2.3 章) に当たる．Quick Mode での大まかな処理の流れは次の通りである．

1. SA を確立する環境を決めるために，SA で使用する暗号アルゴリズムや認証アルゴリズム等を取り決める．

2. 取り決めた環境に従って，SA で使用する鍵を生成する．
3. 最後に最終確認を行って SA の確立を終了する．

以上の流れを図 3.9に示す．

実際の処理の流れを図 3.10に示す．Quick Mode では上記の処理を 3 回 (1.5 往復) の通信を経て行う．始めに Initiator は，SA を確立する環境を取り決めるために，Initiator が利用可能な各種アルゴリズムを，使用する希望順に並べて送信する．同時に，SA で使用する鍵を生成するための補助的なパラメータと，メッセージの正当性を保証するための HASH(1) を共に Responder に送信する．

Responder は列挙された各種アルゴリズムの中から，Responder が利用可能であり，かつ一番始初めに該当したアルゴリズムを選択する．そして，選択したアルゴリズムと SA で使用する鍵を生成するための補助的なパラメータとメッセージの正当性を保証するための HASH(2) を共に Initiator に送信する．そして，この時点で，Initiator と Responder は SA で使用する鍵を算出する．最後に Initiator は，最終確認のために HASH(3) を Responder に送信する．

HASH(1)，HASH(2)，HASH(3) は，それぞれ式 (3.15)，式 (3.16)，式 (3.17) より求められる．これらは，ISAKMP SA が有効期間内であることと，メッセージが改竄されることなく正確に送受信されていることを互いに確認するために送信される．また，SA で使用する鍵は式 (3.18) より求められる．

式 (3.18) に示したように，SA で使用する鍵の算出方法には 2 種類ある．SA で使用する鍵が **PFS(Perfect Forward Secrecy)** を要求するか否かによって生成方法が異なる．さらに，PFS を要求する場合には，Initiator と Responder は，Main Mode 中に交換した Diffie–Hellman の公開情報とは異なった Diffie–Hellman の公開情報を Quick Mode において Key Exchange Payload を用いて交換する必要がある．$g(qm)^{x_i x_r}$ は，Quick Mode 中で交換された Diffie–Hellman の公開情報より求められる Initiator と Responder の共有値である．

ここで，PFS は以下の性質を満たす必要がある．

- あるデータの送信保護に用いられた鍵から，いかなる付加的な鍵も生成してはならない．
- 一度鍵生成に利用されたデータは，それ以上鍵を生成する際に利用されてはならない．

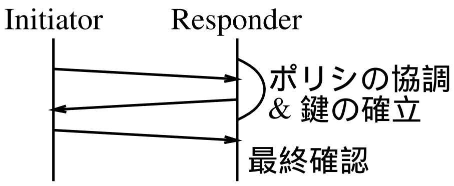


図 3.9: Quick Mode の流れ

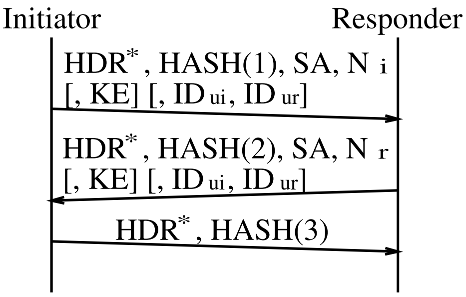


図 3.10: Quick Mode

$$HASH(1) = prf(SKEYID\_a, M\_ID|SA|N_i[|KE][|ID_{ui}|ID_{ur}]) \tag{3.15}$$

$$HASH(2) = prf(SKEYID\_a, M\_ID|N_i|SA|N_r[|KE][|ID_{ui}|ID_{ur}]) \tag{3.16}$$

$$HASH(3) = prf(SKEYID\_a, 0|M\_ID|N_i|N_r) \tag{3.17}$$

$$KEY = \begin{cases} prf(SKEYID\_d, SPI|N_i|N_r) \\ prf(SKEYID\_d, g(qm)^{x_i x_r}|SPI|N_i|Nr) \end{cases} \tag{3.18}$$

3.7 New Group Mode

New Group Mode は，ISAKMP SA を利用して，Diffie–Hellman の素数 q と原始元 α の組の変更を行う．New Group Mode は ISAKMP/ Oakley では定義されているが，ISAKMP で定義されている Exchange Type のいかなる種類にも当てはまらない．

New Group Mode の処理の流れを図 3.11に示す．New Group Mode では，始めに Initiator は，サポートしている Diffie–Hellman の素数 q と原始元 α の組を示す識別子を候補として列挙する．この候補は，Initiator の望む順に並べられて，SA ペイロードとして Responder に送信される．この時に，メッセージの正当性を保証するための HASH(1) が一緒に送信される．これを受け取った Responder は列挙された Diffie–Hellman の素数 q と原始元 α の組の中から，Responder がサポートしており，かつ一番初めに該当した物を選択する．そして，Responder は選択した Diffie–Hellman の素数 q と原始元 α の組の識別子を Initiator に送信する．この時にも，メッセージの正当性を保証するための HASH(2) が一緒に送信される．これで，Diffie–Hellman の素数 q と原始元 α の組が変更される．HASH(1)，HASH(2) は，それぞれ式 (3.19)，式 (3.20) より算出される．

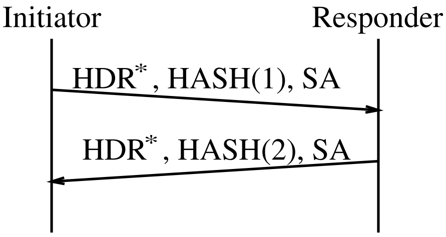


図 3.11: New Group Mode

$$HASH(1) = prf(SKEYID\_a, M\_ID|SA) \tag{3.19}$$

$$HASH(2) = prf(SKEYID\_a, M\_ID|SA) \tag{3.20}$$

3.8 ISAKMP/ Oakley のまとめ

以上 ISAKMP/ Oakley の仕様について説明したが，本小節では各認証方式について比較をする．ここでは，以下の点に着目をして比較を行う．

- 暗号処理の回数
- スケーラビリティ
- 管理コスト

3.8.1 暗号処理の回数

各認証方式における処理の回数を比較する．なお，比較する処理対象は以下の 3 点であり，Main Mode，Quick Mode を経て 1 つの SA を確立するために必要な処理全てを考える．ただし，認証アルゴリズムには鍵付きハッシュ関数を用い，さらに電子署名には RSA を用いるとする．

- 公開鍵暗号
- 慣用暗号
- 鍵付きハッシュ

比較の結果を表 3.2に示す．表 3.2では，それぞれ
合計処理回数 (Main Mode 中での処理回数 + Quick Mode 中での処理回数)
という書式で処理回数を示した．

表 3.2: 比較 1: 暗号処理回数

認証方式	公開鍵暗号		慣用暗号		鍵付き ハッシュ
	暗号化	復号	暗号化	復号	
RSA 電子署名	2 (2 + 0)	2 (2 + 0)	5(2 + 3)	5(2 + 3)	20(12 + 8)
公開鍵暗号	4 (4 + 0)	4 (4 + 0)	5(2 + 3)	5(2 + 3)	20(12 + 8)
模擬共有鍵	0 (0 + 0)	0 (0 + 0)	5(2 + 3)	5(2 + 3)	20(12 + 8)

表 3.2では，認証方法に電子署名や公開鍵暗号を用いる時の CA や S–DNS へアクセスする際に発生する処理を考慮していない．従って実際には，認証方法に電子署名や公開鍵暗号を用いる時にはより多くの処理を必要とする．

3.8.2 スケーラビリティ

各認証方式におけるスケーラビリティを比較する．なお，ここでいうスケーラビリティとは広域分散環境下において利用可能であるか否かである．認証方法に電子署名や公開鍵暗号を用いる際には，CA や S–DNS を利用できるものとする．比較の結果を表 3.3に示す．

表 3.3: 比較 2: スケーラビリティ

認証方式	スケーラビリティの有無
電子署名	有
公開鍵暗号	有
模擬共有鍵	無

模擬共有鍵を用いた認証方式では，予め通信相手と模擬共有鍵を共有している必要があり，実際には限られた範囲内での ISAKMP/ Oakley の運用に制限されてしまい，スケーラビリティはない．一方，電子署名や公開鍵暗号を用いた認証方式では，CA サーバや S–DNS サーバへのアクセス集中が発生する可能性などの問題があるものの，CA や S–DNS から通信相手の公開鍵を入手することが可能であり，CA や S–DNS と同等のスケーラビリティを有することになる．

3.8.3 管理コスト

各認証方式のおける管理コストを比較する．認証方式に電子署名や公開鍵暗号を用いる際には，CA や S–DNS を利用できるものとする．比較の結果を表 3.4に示す．

模擬共有鍵を用いた認証方式では，予め模擬共有鍵を登録しておく必要があり，通信先のホストが増える度に管理者は，ホスト毎に模擬共有鍵の登録を行わなければならない．従って，模擬共有鍵の管理が大変になる．一方，電子署名や公開鍵暗号を用いた認証方式では，CA や S–DNS に通信相手が登録されて

表 3.4: 比較 3: 管理コスト

認証方式	管理の難易
電子署名	易
公開鍵暗号	易
模擬共有鍵	難

いれば SA の確立が可能である．従って，CA や S–DNS を利用する場合において，ホストを 1 台増す時には，管理者は CA や S–DNS へホストの登録さえ行えば良い．

第 4 章

鍵交換デーモンの実装と評価

本章では，3章で説明したISAKMP/ Oakley に従って鍵交換デーモンを実装し，その性能評価を行う．

4.1 実装概要

本論文では，3章で説明した鍵交換デーモンを実装した．はじめに鍵交換デーモンを実装した環境について述べる．

1. 2で述べるOS を AT 互換機上にインストール．
2. Free BSD 2.2.2–RELEASE に伊藤 純一郎氏によって作成されたIPsec カーネルへの変換パッチ[1] [2]を当てたOS 上で実装．
3. ISAKMP/ Oakley の基本的な骨組みが実装された**Pluto**[3] をベースに実装．

本論文で使用した，伊藤 純一郎氏によって作成されたIPsec カーネルは，以下に挙げる特徴を持っている．

- FreeBSD 用の実装．
- IPv4/v6 対応．
- **PF_KEY**(**PF_KEY key management application programming interface**) v1[4]の部分的サポート．
- Simple API[5]をサポート
- ESP で使用できる暗号アルゴリズムはDES CBC モード [14]．
- AH で使用できる認証アルゴリズムはKeyed–MD5[6] とKeyed–SHA1[24]，HMAC–MD5，HMAC–SHA1[19]．
- ESP のトランスポートモードのみをサポート．

また，本論文でベースとしたPluto は，米国在住の**Angelos D. Keromytis** 氏が米国の輸出規制を逃れるために，ギリシャのアテネで実装したものである．つまりPluto は，自由に使用できる数少ない鍵交換デーモンの実装の内の一つである．そして，Pluto は以下に挙げる特徴を持った実装である．

- `draft-ietf-ipsec-isakmp-07.txt`
  `draft-ietf-ipsec-isakmp-oakley-03.txt` に従った実装．
  `draft-ietf-ipsec-ipsec-doi-02.txt`

---

[1] `ftp://ftp.itojun.org/pub/ipsec/FreeBSD221-ipsec.970714.tar.gz.pgp`

[2] 最新バージョンは`ftp://ftp.itojun.org/pub/ipsec/FreeBSD222-ipsec.970914.tar.gz.pgp` (98/ 2 現在)

[3] `ftp://ftp.funet.fi/pub/unix/security/net/ip/pluto-6.alpha.tar.gz`

[4] Appendix Bを参照．

[5] Appendix Cを参照.

- Linux 用の実装．
- IPv4 対応．
- IPsec 用の SA のみを確立．
- Main Mode と Quick Mode のサポート．
- ISAKMP SA で使用する暗号アルゴリズムを DES CBC モードに固定．
- ISAKMP SA で使用する認証アルゴリズムを Keyed–SHA1 に固定．
- SA で使用する暗号アルゴリズムを DES CBC モードに固定．
- SA で使用する認証アルゴリズムを HMAC–MD5 に固定．
- 認証方式として模擬共有鍵方式のみの提供．
- 模擬共有鍵を予め`/etc/isakmp-secrets`ファイルに登録しておく必要性有り．`/etc/isakmp-secrets`ファイルの書式は 4.2.3小節に示す．
- 使用する Diffie–Hellman の素数 q，原始元αの組を **First Oakley Default Group** に固定．First Oakley Default Group については，4.2.5で説明する．
- PF_KEYv1/v2 の未サポート．
- 脆弱鍵および準脆弱鍵のチェックの未サポート．脆弱鍵，準脆弱鍵とは暗号文から容易に推測できてしまう鍵の事である．
- エラー発生時の不適切な対応．

Pluto は，実用を考慮して実装されておらず，IPsec で利用することが不可能である．そこで本論文では，IPsec で利用することが可能であり，かつ様々な認証方式を利用できるように実装を行った．本論文で行った鍵交換デーモンの実装は，上記で述べた Pluto がベースになっており，以下に挙げる特徴を持つ．

- `draft-ietf-ipsec-isakmp-08.txt`
  `draft-ietf-ipsec-isakmp-oakley-04.txt` に従った実装．
  `draft-ietf-ipsec-ipsec-doi-02.txt`
- Free BSD 用の実装．
- IPv4 対応．
- IPsec 用の SA のみを確立．
- Main Mode と Quick Mode のサポート．

- ISAKMP SA で使用する暗号アルゴリズムを DES CBC モードに固定．
- ISAKMP SA で使用する認証アルゴリズムを Keyed–SHA1 に固定．
- SA で使用する暗号アルゴリズムを DES CBC モードに固定．
- SA で使用する認証アルゴリズムを Keyed–MD5 に固定．
- 認証方式として模擬共有鍵方式，電子署名方式，公開鍵暗号方式を提供．ただし，SSH[6]の秘密鍵，公開鍵を利用．4.2.4小節で詳しく説明する．
- 模擬共有鍵を予め`/etc/isakmp-secrets`ファイルに登録しておく必要性有り．`/etc/isakmp-secrets`ファイルの書式は 4.2.3小節に示す．
- 使用する Diffie–Hellman の素数 q，原始元αの組を First Oakley Default Group に固定．
- PF_KEY v1 の部分的サポート．4.2.2小節で詳しく説明する．
- Diffie–Hellman の公開値計算を ISAKMP/ Oakley の処理と並行化．4.2.5小節で詳しく説明する．
- 脆弱鍵および準脆弱鍵のチェックの未サポート．
- エラー発生時の不適切な対応．

4.2 実装の詳細

本論文で実装した鍵交換デーモンは，メッセージ到着によって処理が開始し，受け取ったメッセージに対応する処理の状態によって振舞いが異なる．その様子を図 4.1に示す．使用する認証方式によっても，振舞いに多少の違いが現われるので，振舞いが異なる部分に関しては，図 4.1において

1. 電子署名を用いる方法
2. 公開鍵暗号を用いる方法
3. 模擬共有鍵を用いる方法

の順に表してある．

図 4.1に示した様に，カーネルから SA 確立要求が届いた時には，ISAKMP SA が既に確立されているか否かで振舞いが異なる．また，通信ポートへ他の鍵交換デーモンからのメッセージが到着した時には，SA の確立処理の状態によって振舞いが異なる．図 4.1で説明すると，SA の確立処理の状態が，“OAKLEY_MAIN_I_1” から “OAKLEY_QUICK_R_1” のいずれであるかによって振舞いが異なる．“MAIN” は Main Mode を示しており，“QUICK” は Quick Mode を示している．また，“I” は Initiator を示しており，“R” は Responder を示している．

---

[6] `ftp://ftp.cs.hut.fi/pub/ssh/ssh-1.2.21.tar.gz`

4.2.1 通信方法

ISAKMP の仕様書において，鍵交換デーモン間の通信には，通信プロトコルとして UDP を用い，通信ポートに 500 番ポートを使用することが定義されている．また，PF_KEY の仕様書において，鍵交換デーモンとカーネル間の通信には，RAW ソケットを用いることが定義されている．さらに，ISAKMP の仕様書において，鍵交換デーモンとカーネル間の通信では，通信ポートに 501 番ポートを使用することが定義されている．本論文での実装は，以上の仕様に従っている．

4.2.2 PF_KEY v1 の部分的サポート

本論文で利用した IPsec カーネルは，PF_KEY v1 の部分的サポートをしているため，鍵交換デーモンも PF_KEY v1 の部分的サポートを行った．主に以下の点において PF_KEY v1 をサポートしている．PF_KEY v1 の仕様については，Appendix Bを参照されたい．尚，Appendex Bで述べている PF_KEY の仕様は version 2 の物だが，基本的には version 1 と変わらない．

- カーネルと鍵交換デーモン間の通信には PF_KEY プロトコルファミリを使用．
- 鍵交換デーモン間の通信には PF_INET プロトコルファミリを使用．
- カーネルへの SADB_REGISTER メッセージの送信．
- カーネルからの SADB_ACQUIRE メッセージの受信．
- カーネルへ SADB_ADD メッセージを用いた SA の登録．

そして，本論文の実装では，以下の点はサポートしていない．

- カーネルからの SADB_REGISTER メッセージの受信．
- カーネルからの SADB_ADD メッセージの受信．
- SADB_GETSPI メッセージの送受信．
- カーネルへの SADB_UPDATE メッセージを用いた SA の登録．
- カーネルからの SADB_EXPIRE メッセージの受信．

本来ならば，自ホストから通信先ホストへの SA は，SADB_UPDATE メッセージによって登録されなければならない．しかし，本論文での実装では SADB_GETSPI メッセージの送受信をサポートしていないため，全ての SA は SADB_ADD メッセージによって登録される．

4.2.3 認証方式の選択

本論文で行った実装では，認証に模擬共有鍵を用いるためには，模擬共有鍵を予め`/etc/isakmp-secrets`に登録しておかなければならない．そして，`/etc/isakmp-secrets`に模擬共有鍵が登録されていないホストとSAを確立する際には，認証に電子署名や公開鍵暗号が用いられる．模擬共有鍵が登録されていないホストとSAを確立する時に，認証に電子署名を用いるか公開鍵暗号を用いるかは，鍵交換デーモンを“make”する時に決まる．“make”の際に“`-DRSAENC`”を引数とすると，認証に公開鍵暗号が用いられる．

また，`/etc/isakmp-secrets`ファイルは

IPアドレス
模擬共有鍵

という書式となっている．以下に`/etc/isakmp-secrets`ファイルの例を示す．

```
207.74.188.194
thisisasecret
199.249.254.9
password
131.113.83.33
neginegi
```

4.2.4 SSHの利用

認証方式として電子署名や公開鍵暗号を用いる時には，送信元ホストの秘密鍵や公開鍵，そして送信先ホストの公開鍵が必要である．本来ならば，公開鍵は，CAやS–DNSなどに登録されており，CAやS–DNSによって管理されていなければならない．しかし，現在CAやS–DNSは未だ運用実験段階であり，利用することが困難であるので，本実装ではSSHで使用される秘密鍵や公開鍵を利用することにした．

SSHでは自ホストの秘密鍵と公開鍵を`ssh_host_key`ファイルに保持している．また，SSHでは`make-ssh-known-hosts`と言うコマンドにより，他ホストの公開鍵を収集し，`ssh_known_hots`ファイルに保存する事が可能である．そこで，本論文では，`ssh_host_key`ファイルから送信元ホストの秘密鍵と公開鍵を入手し，`ssh_known_host`ファイルから送信先ホストの公開鍵を入手するように実装した[7]．ただし，SSHの秘密鍵や公開鍵の利用は，ISAKMP/ Oakleyの仕様ではないため，送信先ホストの公開鍵は必ず`ssh_known_host`に存在する事を仮定し，エラー処理を行うよう実装していない．

4.2.5 Diffie–Hellmanの秘密値計算の並行化

Diffie–Hellmanの公開値を計算するには，最低256 bitの乱数の発生と多倍長整数計算を必要とする．ISAKMP/ Oakleyでは，Diffie–Hellmanの計算を行う上で予め合意を取っておく必要がある素数qと原始

---

[7] 実装では`ssh_host_key`ファイルも`ssh_known_host`ファイルも`/usr/local/etc`に存在するものとした．

元αの組が 2 つ定義されている．始めの組は，

$$
\left\{
\begin{array}{lll}
q & = \mathrm{0x} & \texttt{FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1} \\
 & & \texttt{29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD} \\
 & & \texttt{EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245} \\
 & & \texttt{E485B576 625E7EC6 F44C42E9 A63A3620 FFFFFFFF FFFFFFFF} \\
\alpha & = & 2
\end{array}
\right.
\tag{4.1}
$$

であり，デフォルトとして定義されている．そして，この組は First Oakley Default Group と呼ばれる．そして，2 つめの組は，

$$
\left\{
\begin{array}{lll}
q & = \mathrm{0x} & \texttt{FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1} \\
 & & \texttt{29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD} \\
 & & \texttt{EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245} \\
 & & \texttt{E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED} \\
 & & \texttt{EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE65381} \\
 & & \texttt{FFFFFFFF FFFFFFFF} \\
\alpha & = & 2
\end{array}
\right.
\tag{4.2}
$$

であり，**Second Oakley Group** と呼ばれる．

本論文で行った実装では，式 (4.1) や式 (4.2) で示した値を扱うために **GMP Library(GNU Multiple Precision arithmetic Library)**[8]を用いている．GMP Library は，多倍長整数計算を可能にするライブラリであり，Intel Pentium や x86 系の CPU を搭載したマシンを始めとして，多くのマシンで利用することができる [12]．しかし，GMP Library を利用することによって，処理に時間がかかってしまう．

そこで，Diffie–Hellman の秘密情報と公開情報を計算する処理を ISAKMP/ Oakley 本来の処理と並行に行うようにした．また，必要に応じて Diffie–Hellman の秘密情報や公開情報を計算するのではなく，CPU が行わなければならない処理が無く，CPU に余裕がある間に幾つかの Diffie–Hellman の秘密情報と公開情報を計算しておき，貯蔵しておくようにした．実際の実装では，メモリ中に 5 つまで貯蔵可能である．この様に処理に時間がかかり，且つ並行に処理が可能な部分を並行化することにより，処理時間の短縮を行った．

4.2.6 Pluto との比較

本小節では，本論文で実装した鍵交換デーモンと Pluto の実装を比較する．比較結果を表 4.1に示す．なお以下の点に着目し，比較を行った．

- 対応 OS $\cdots$ どの OS 上で利用可能かを比較する．
- ネットワーク $\cdots$ 対応している IP のバージョンを比較する．
- 対応 SA$\cdots$ 鍵交換デーモンが確立可能である SA の種類を比較する．

---

[8] ftp://prep.ai.mit.edu/pub/gnu/gmp-2.0.2.tar.gz

- 利用可能モード ··· Phase 1 と Phase 2 において利用可能なモードを比較する．
- 使用暗号アルゴリズム ··· 鍵交換デーモンが ISAKMP SA と SA の確立時に，使用する暗号アルゴリズムとして決定するものを比較する．
- 使用認証アルゴリズム ··· 鍵交換デーモンが ISAKMP SA と SA の確立時に，使用する認証アルゴリズムとして決定するものを比較する．
- 使用 Oakley Groupαの組 (Oakley Group) ··· 使用する Diffie–Hellman の素数 qと原始元αの組を比較する．
- 認証方式 ··· 利用可能な認証方式を比較する．
- PF_KEY のサポート ··· PF_KEY のサポートを比較する．
- 脆弱鍵のチェック ··· 脆弱鍵のチェックを行うか否かを比較する．
- エラーの対応 ··· エラーが発生した時の対応処理が仕様に則しているか否かを比較する．
- 処理の高速化 ··· Diffie–Hellman の処理を並行化することによる，処理の高速化などを行っているか否かを比較する．
- SA の登録 ··· kernel へ SA を登録できるか否かを比較する．

4.3 鍵交換デーモンの評価

本論文で実装した鍵交換デーモンが SA を確立するために必要とした時間を測定した．測定環境は次の通りである．

- AT 互換機 × 2
    - Intel Pentium 200MHz
    - 256KByte キャッシュ
    - 64MByte メモリ
- Free BSD 2.2.2–RELEASE with IPsec
- IPsec 実験専用 LAN(10Mbps Ethernet)
- SSH に関係したファイルは全てローカルディスクに存在する

表 4.1: Pluto との比較

	本論文での実装	Pluto
対応 OS	Free BSD	Linux
ネットワーク	IPv4	IPv4
対応 SA	IPsec	IPsec
利用可能モード	Main Mode & Quick Mode	Main Mode & Quick Mode
使用暗号アルゴリズム	DES–CBC(for ISAKMP SA)	DES–CBC(for ISAKMP SA)
	DES–CBC(for SA)	DES–CBC(for SA)
使用認証アルゴリズム	Keyed–SHA1(for ISAKMP SA)	Keyed–SHA1(for ISAKMP SA)
	Keyed–MD5(for SA)	HMAC–MD5(for SA)
使用 Oakley Group	First Oakley Default Group	First Oakley Default Group
認証方式	電子署名 公開鍵暗号 模擬共有鍵	模擬共有鍵
PF_KEY のサポート	部分的 (v1)	無
脆弱鍵のチェック	無	無
エラーの対応	不適切	不適切
処理の高速化	○	×
SA の登録	可	不可

測定環境を図 4.2に示す．

システムに最低限必要なプロセスのみが起動しており，無関係なプロセスが起動しておらず，また不必要な通信が起こらず，かつ通信遅延が 0 であるという理想環境に近い状態で測定した．このような環境で，“`ping`” コマンドを実行し，一方のホストからもう一方のホストへ “ECHO Request” を送信することによって，カーネルから鍵交換デーモンへ SA の確立要求が発生するようにした．

4.3.1 模擬共有鍵を認証方式に用いた時の測定結果

認証に模擬共有鍵を用いた時の測定結果を表 4.2，表 4.3，表 4.4に示す．いずれの測定結果も 10 回測定を行った平均である．

表 4.2: 模擬共有鍵を用いた時の全処理時間

送信者側	処理時間 [sec]	受信者側	処理時間 [sec]
Main Mode	0.150	Main Mode	0.145
Quick Mode	0.009	Quick Mode	0.006

表 4.2は，送信者側・受信者側でそれぞれ Main Mode，Quick Mode を処理するのにかかった時間を測定した結果を示している．この測定結果には，通信遅延はもちろんのこと，鍵交換デーモン以外のプロセ

表 4.3: 模擬共有鍵を用いた時の Main Mode の各処理時間

送信者側	処理時間 [sec]	受信者側	処理時間 [sec]
I–M–1	0.003	R–M–1	0.001
I–M–2	0.005	R–M–2	0.052
I–M–3	0.049	R–M–3	0.003
I–M–4	0.001		

表 4.4: 模擬共有鍵を用いた時の Quick Mode の各処理時間

送信者側	処理時間 [sec]	受信者側	処理時間 [sec]
I–Q–1	0.004	R–Q–1	0.004
I–Q–2	0.001	R–Q–2	0.001

スに対する処理にかかった時間も含まれている．しかし，理想環境に近い状態で測定したので，本論文で実装した鍵交換デーモンにおいて認証に模擬共有鍵を用いる場合には，最低表 4.2で示された処理時間がかかると言える．

また，表 4.3は Main Mode において送信者側・受信者側で各処理にかかった時間を測定した結果を示している．同様に表 4.4は，Quick Mode において送信者側・受信者側で各処理にかかった時間を測定した結果を示している．これらの測定結果は，純粋に各処理にかかった時間であり，通信遅延や鍵交換デーモン以外のプロセスに対する処理時間を含んでいない．表 4.2，表 4.3，表 4.4に示した結果をまとめたものを図 4.3に示す．

4.3.2 電子署名を認証方式に用いた時の測定結果

認証に RSA 電子署名を用いた時の測定結果を表 4.5，表 4.6，表 4.7に示す．いずれの測定結果も 10 回測定を行った平均である．

表 4.5: 電子署名を用いた時の全処理時間

送信者側	処理時間 [sec]	受信者側	処理時間 [sec]
Main Mode	0.304	Main Mode	0.293
Quick Mode	0.009	Quick Mode	0.006

表 4.5は，送信者側・受信者側でそれぞれ Main Mode，Quick Mode を処理するのにかかった時間を示している．この測定結果には，通信遅延はもちろんのこと，鍵交換デーモン以外のプロセスに対する処理にかかった時間も含まれている．しかし，理想環境に近い状態で測定したので，本論文で実装した鍵交換デーモンにおいて認証に電子署名を用いる場合には，最低表 4.5で示された処理時間がかかると言える．

表 4.6: 電子署名を用いた時の Main Mode の各処理時間

送信者側	処理時間 [sec]	受信者側	処理時間 [sec]
I–M–1	0.003	R–M–1	0.001
I–M–2	0.005	R–M–2	0.052
I–M–3	0.121	R–M–3	0.079
I–M–4	0.008		

表 4.7: 電子署名を用いた時の Quick Mode の各処理時間

送信者側	処理時間 [sec]	受信者側	処理時間 [sec]
I–Q–1	0.004	R–Q–1	0.004
I–Q–2	0.001	R–Q–2	0.001

また，表 4.6は Main Mode において送信者側・受信者側で各処理にかかった時間を示している．同様に表 4.7は，Quick Mode において送信者側・受信者側で各処理にかかった時間を示している．これらの測定結果は，純粋に各処理にかかった時間であり，通信遅延や鍵交換デーモン以外のプロセスに対する処理時間を含んでいない．表 4.5，表 4.6，表 4.7に示した結果をまとめたものを図 4.4に示す．

さらに，電子署名を用いて認証を行うために必要な処理にかかる時間を測定した．本論文で行った実装では，電子署名を行うために以下の処理が行われる．

署名

1. `/usr/local/etc/ssh_host_key` ファイルから，秘密鍵を取り出す．ただし秘密鍵を保護するために，`/usr/local/etc/ssh_host_key` ファイルは暗号化されているので，復号しなければばならない．また，`/usr/local/etc/ssh_host_key` ファイルには秘密鍵自体が保存されているのではなく，秘密鍵の基となる情報が保存されている．従って，基となる情報から秘密鍵を算出する必要がある．
2. 秘密鍵を用いて HASH_I (HASH_R) に対して署名を行う．

検証

1. `/usr/local/etc/ssh_known_host` ファイルから，通信相手の公開鍵を取り出す．
2. 公開鍵を用いて検証を行う．

以上の処理を行うために必要とした処理時間を表 4.8に示す．

署名や検証の処理時間に関してさらに詳しく測定を行った．始めに，`/usr/local/etc/ssh_host_key` ファイルから秘密鍵の基となる情報を取り出す時間と`/usr/local/etc/ssh_known_host` ファイルから公開鍵を取り出す時間を測定した．測定結果を表 4.9 に示す．

表 4.8: 電子署名の処理時間

署名		検証	
データサイズ	処理時間	データサイズ	処理時間
20 → 128 [Byte]	0.071 [sec]	128 → 20 [Byte]	0.006 [sec]

表 4.9: 電子署名処理中の鍵取り出し時間

処理	処理時間
秘密鍵の取り出し	0.003 [sec]
公開鍵の取り出し	0.004 [sec]

次に，秘密鍵を用いた署名のみの処理時間と公開鍵を用いた検証のみの処理時間を測定した．ただし，署名を行う際には，秘密鍵の基となる情報から秘密鍵を求める処理も行う．測定結果を表 4.10に示す．

表 4.10: 署名・検証の実処理時間

署名のみ		検証のみ	
データサイズ	処理時間	データサイズ	処理時間
20 → 128 [Byte]	0.0667 [sec]	128 → 20 [Byte]	0.0015 [sec]

RSA では，秘密鍵を用いた暗号処理と公開鍵を用いた暗号処理は，全く同じである．従って，表 4.8，表 4.9，表 4.10より，署名に多くの処理時間を要するのは，秘密鍵の算出に時間を要しているためであることがわかる．

4.3.3 公開鍵暗号を認証方式に用いた時の測定結果

認証に RSA 公開鍵暗号を用いた時の測定結果を表 4.11，表 4.12，表 4.13に示す．いずれの測定結果も 10 回測定を行った平均である．

表 4.11は，送信者側・受信者側でそれぞれ Main Mode，Quick Mode を処理するのにかかった時間を示している．この測定結果には，通信遅延はもちろんのこと，その他のプロセスに対する処理にかかった時間も含まれている．しかし，理想環境に近い状態で測定したので，本論文で実装した鍵交換デーモンにおいて認証に公開鍵暗号を用いる場合には，最低表 4.11で示された処理時間がかかると言える．

また，表 4.12は Main Mode において送信者側・受信者側で各処理にかかった時間を示している．同様に表 4.13は，Quick Mode において送信者側・受信者側で各処理にかかった時間を示している．表 4.11，表 4.12，表 4.13に示した結果をまとめたものを図 4.5に示す．

さらに，公開鍵暗号を用いて認証を行うために必要な処理にかかる時間を測定した．本論文で行った実

表 4.11: 公開鍵暗号を用いた時の全処理時間

送信者側	処理時間 [sec]	受信者側	処理時間 [sec]
Main Mode	0.626	Main Mode	0.614
Quick Mode	0.010	Quick Mode	0.006

表 4.12: 公開鍵暗号を用いた時の Main Mode の各処理時間

送信者側	処理時間 [sec]	受信者側	処理時間 [sec]
I–M–1	0.003	R–M–1	0.001
I–M–2	0.017	R–M–2	0.241
I–M–3	0.325	R–M–3	0.002
I–M–4	0.001		

装では，公開鍵暗号を行うために以下の処理が行われる．

暗号化

1. `/usr/local/etc/ssh_known_host` ファイルから，通信相手の公開鍵を取り出す．
2. 公開鍵を用いて ID_{ii} (ID_{ir})，N_i (N_r) に対してにそれぞれ暗号化を行う．

復号

1. `/usr/local/etc/ssh_host_key` ファイルから，秘密鍵を取り出す．ただし秘密鍵を保護するために，`/usr/local/etc/ssh_host_key` ファイルは暗号化されているので，復号しなければならない．また，`/usr/local/etc/ssh_host_key` ファイルには秘密鍵自体が保存されているのではなく，秘密鍵の基となる情報が保存されている．従って，基となる情報から秘密鍵を算出する必要がある．
2. 秘密鍵を用いて復号を行う．

以上の処理を行うために必要とした処理時間を表 4.14に示す．

暗号化や復号の処理時間に関してさらに詳しく測定を行った．始めに，`/usr/local/etc/ssh_host_key` ファイルから秘密鍵の基となる情報を取り出す時間と`/usr/local/etc/ssh_known_host` ファイルから公開鍵を取り出す時間を測定した．測定結果を表 4.15 に示す．

表 4.13: 公開鍵暗号を用いた時の Quick Mode の各処理時間

送信者側	処理時間 [sec]	受信者側	処理時間 [sec]
I–Q–1	0.004	R–Q–1	0.003
I–Q–2	0.001	R–Q–2	0.001

表 4.14: 公開鍵暗号の処理時間

暗号化		復号	
データサイズ	処理時間	データサイズ	処理時間
8 → 128 [Byte]	0.005 [sec]	128 → 8 [Byte]	0.088 [sec]
16 → 128 [Byte]	0.006 [sec]	128 → 16 [Byte]	0.088

表 4.15: 公開鍵暗号処理中の鍵取り出し時間

処理	処理時間
秘密鍵の取り出し	0.003 [sec]
公開鍵の取り出し	0.004 [sec]

次に，暗号化のみの処理時間と復号のみの処理時間を測定した．ただし，復号を行う際には，秘密鍵の基となる情報から秘密鍵を求める処理も行う．測定結果を表 4.16に示す．

表 4.16: 暗号化 ·復号の実処理時間

暗号化		復号	
データサイズ	処理時間	データサイズ	処理時間
8 → 128 [Byte]	0.0003 [sec]	128 → 8 [Byte]	0.0846 [sec]
16 → 128 [Byte]	0.0005 [sec]	128 → 16 [Byte]	0.0850 [sec]

4.3.2小節で述べたように，RSA では秘密鍵を用いた暗号処理と公開鍵を用いた暗号処理は，全く同じである．従って，表 4.14，表 4.15，表 4.16より，復号に多くの処理時間を要するのは，秘密鍵の算出に時間を要しているためであることがわかる．

4.3.4 評価のまとめ

以上，4.3.1小節，4.3.2小節，4.3.3小節において，各認証方式を用いた時の Main Mode および Quick Mode の処理時間を測定した．そこで，4.3.1小節，4.3.2小節，4.3.3小節で示した測定結果より，ISAKMP/ Oakley の全処理にかかる時間と通信遅延の関係を考察する．始めに，模擬共有鍵を認証に用いる時を考える．表 4.3および表 4.4より，純粋に ISAKMP/ Oakley の処理にかかっている時間は 0.124 sec であることがわかる．そして 9 回の通信が発生するので，1 回の通信による遅延を t_dとすると，全処理時間 t_{all}は

$$t_{all} = 9t_d + 0.124 \tag{4.3}$$

と表される．

次に，電子署名を認証に用いる時を考える．表 4.6および表 4.7より，純粋に ISAKMP/ Oakley の処理にかかっている時間は 0.279 sec であることがわかる．そして 9 回の通信が発生するので，1 回の通信による遅延を t_dとすると，全処理時間 t_{all}は

$$t_{all} = 9t_d + 0.279 \tag{4.4}$$

と表される．

最後に，公開鍵暗号を認証に用いる時を考える．表 4.12および表 4.13より，純粋に ISAKMP/ Oakley の処理にかかっている時間は 0.599 sec であることがわかる．そして 9 回の通信が発生するので，1 回の通信による遅延を t_dとすると，全処理時間 t_{all}は

$$t_{all} = 9t_d + 0.599 \tag{4.5}$$

と表される．

式 (4.3)，式 (4.4)，式 (4.5) をグラフに表すと図 4.6になる．図 4.6より，通信遅延を軽視できないことがわかる．

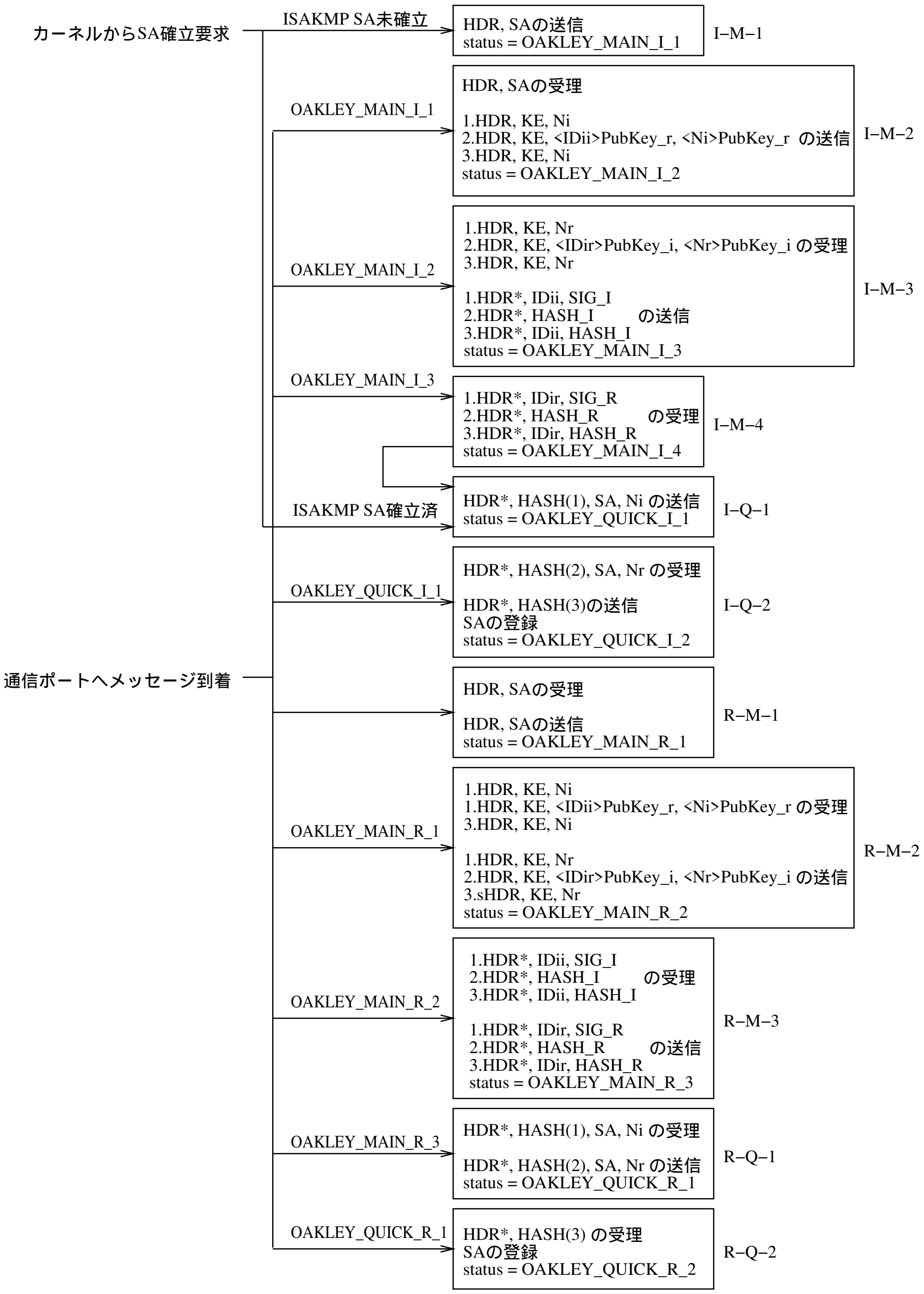


図 4.1: 鍵交換デーモンの実装

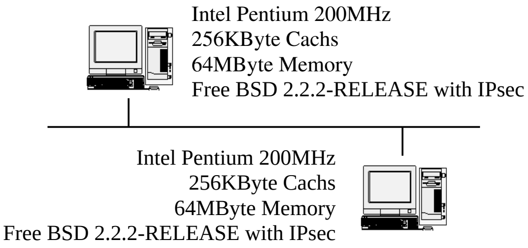


図 4.2: 測定環境

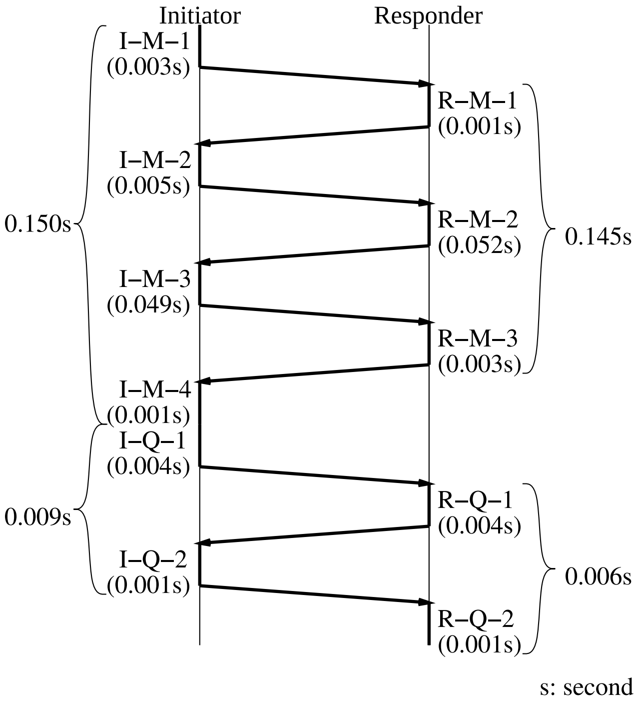


図 4.3: 模擬共有鍵の時

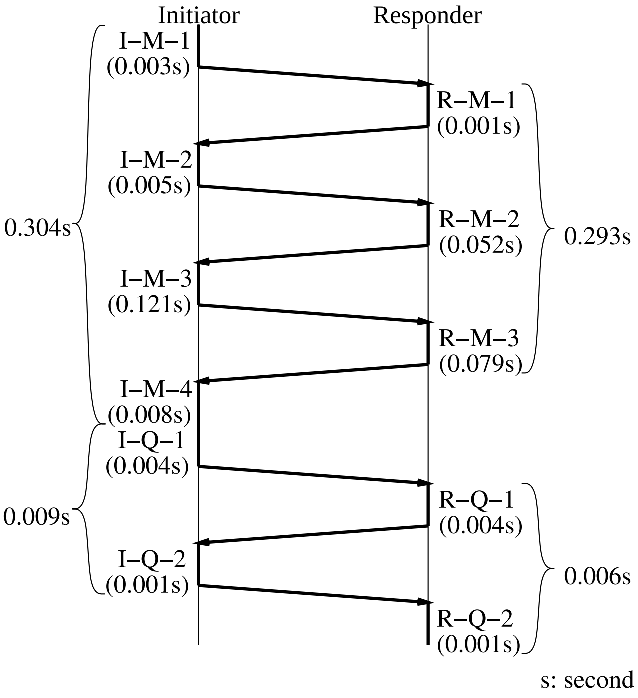


図 4.4: 電子署名の時

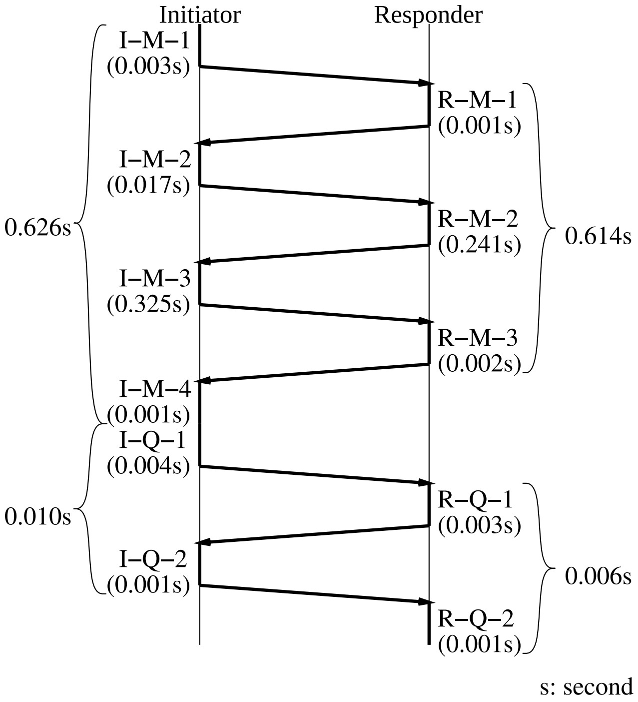


図 4.5: 公開鍵暗号の時

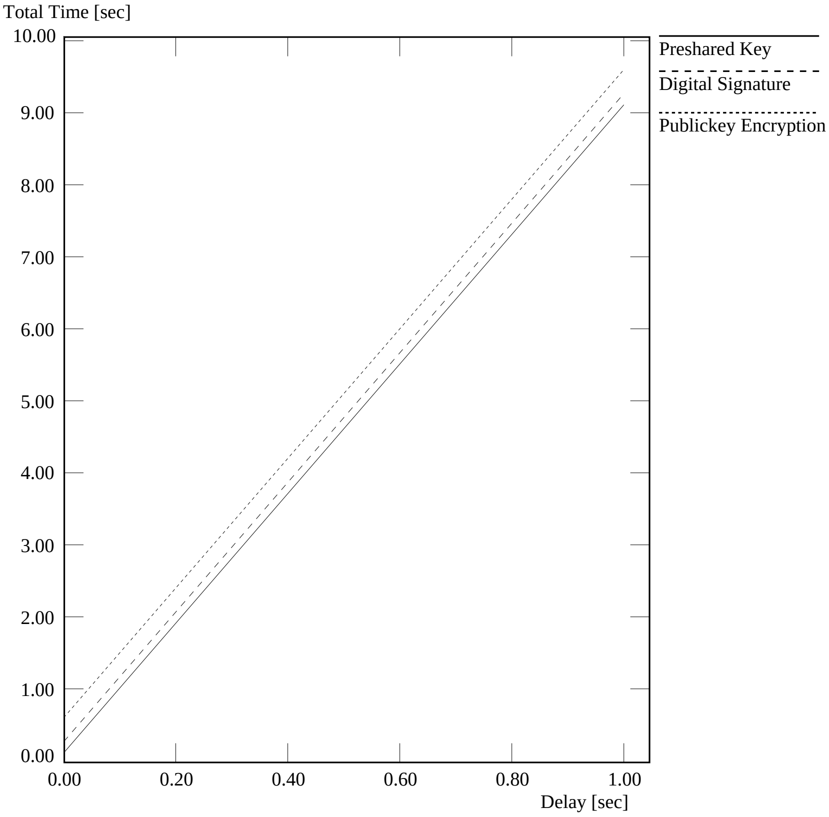


図 4.6: 通信遅延と全処理時間 (Main Mode + Quick Mode)

第 5 章

鍵交換デーモンの問題点と改善策

4章ではISAKMP/ Oakley の仕様に基づいて鍵交換デーモンの実装を行い，さらに実装した鍵交換デーモンの性能評価を行った．しかし，現在定義されているISAKMP/ Oakley は，仕様が未だ完全に固まっておらず，さらに幾つかの問題点を含んでいる．本章では，ISAKMP/ Oakley の問題点を指摘し，問題点を解決すべく改善策を提案する．

5.1 問題点

ISAKMP/ Oakley の問題として以下の点が挙げられる．

1. 大きな通信遅延の影響
2. CA やS–DNS との接続の不明確
3. SA の同時確立
4. RSA を用いることによる処理負荷の増加

以下では，それぞれの問題点について詳しく議論する．

5.1.1 大きな通信遅延の影響

ISAKMP/ Oakley では，Main Mode において3 往復の通信を，Quick Mode において1.5 往復の通信を行う．従って，ISAKMP SA の確立されていないホスト間でIPsec を用いた通信を開始すると，鍵交換デーモンによってSA の確立のために4.5 往復 (計9 回) の通信が生じる．つまり，ISAKMP/ Oakley は9 回分の通信遅延の影響を被ることになる．

ISAKMP/ Oakley では一度ISAKMP SA が確立されると，ある一定期間 (ISAKMP SA の有効期限が切れるまで) の間は，複数のSA を確立することが可能であり，SA の確立全てに4.5 往復の通信を必要するわけではない．既にISAKMP SA が確立されているホストとの通信では，1.5 往復の通信で済む．

しかし，実際のネットワーク利用には，次のような特長があると考えられる．

- LAN 内の通信は，NFS などによって特定のホストとの通信が定常的に発生する．
- LAN 外との通信は，HTTP やFTP などによって不特定多数のホストとの通信が不定期に起こる．

2 つのうち，より強度のネットワークセキュリティを要求されるのは，LAN 外との通信である．また，当然LAN 外との通信では，LAN 内での通信に比べて通信遅延が大きい．

以上の点を考慮すると，IPsec が不可欠であるLAN 外との通信において，且つ不特定多数のホストと不定期に通信が起こり，しかも通信遅延の大きいLAN 外との通信において，少しでも早いSA の確立が要求される．しかし，不特定多数のホストと不定期に通信が起こるので，ISAKMP SA の確立を行ってからSA を確立する可能性が高い．つまり，通信遅延の大きなLAN 外との通信は，ISAKMP SA の確立から始まる可能性が高いので，通信遅延の影響を敏感に被るのである．

5.1.2 CA や S–DNS との接続の不明確

ISAKMP/ Oakley では，認証方式に電子署名を用いる方法や公開鍵暗号を用いる方法を利用する時には，通信相手の正当な公開鍵を入手するためには CA や S–DNS にアクセスしなければならない．だが ISAKMP/ Oakley は，鍵交換プロトコルを定義したものであり，CA や S–DNS へのアクセスについては考えられていない．ISAKMP/ Oakley では，“CA や S–DNS へのアクセスに関しては，CA や S–DNS の範疇であるので議論しない” とされている．

しかし，CA や S–DNS に効率良くアクセスをしなければ ISAKMP/ Oakley の処理性能に影響が出てしまう．従って，確かに CA や S–DNS へのアクセスプロトコルは，CA や S–DNS に対する仕様書の範疇であるが，CA や S–DNS へアクセスするタイミングなどは ISAKMP/ Oakley の仕様書において考慮されるべきである．CA や S–DNS へのアクセスに関する問題は，多分に実装に依存する面が大きいが，実装に際して CA や S–DNS へのアクセスに対する指針を与える必要があるであろう．

5.1.3 SA の同時確立

IPsec ではセキュリティレベルを変更することが可能である．セキュリティレベルに関しては C.2節を参照してもらいたい．IPsec のセキュリティレベルを “**If available**(C.2節参照)” に設定すると，SA が不必要に重複して確立されてしまう．ホスト A とホスト Bにおいて IPsec のセキュリティレベルを “If available” に設定して，ホスト A からホスト Bへ “ECHO Request” を送信することを例にその理由を以下に述べる．

はじめに，ホスト A とホスト Bの間に ISAKMP SA すら確立されていないとする．ホスト A がホスト Bへ “ECHO Request” を送信しようとすると，ホスト A のカーネルはホスト A 上で起動している鍵交換デーモンに SA の確立を要求する．そして，ホスト A のカーネルは鍵交換デーモンへ SA の確立を要求すると同時に，ホスト Bへ “ECHO Request” を送信する．この “ECHO Request” を受け取ったホスト B は，“ECHO Reply” を送信しようとする．そして，ホスト B上で起動している鍵交換デーモンに SA の確立を要求すると同時に，ホスト Bへ “ECHO Reply” を送信する．よって，ホスト A 上で起動している鍵交換デーモンによって，ホスト A からホスト Bへの SA とホスト Bからホスト A への SA が一組確立される．さらに，ホスト B上で起動している鍵交換デーモンによって，ホスト Bからホスト A への SA とホスト A からホスト Bへの SA が一組確立される．

これは，IPsec のセキュリティレベルが “If available” の時にはホスト A とホスト Bが同時に Initiator となってしまうために起こる．IPsec のセキュリティレベルが “**Use**(C.2節参照)” 以上の時には，ホスト A とホスト B間の SA が確立されるまで，“ECHO Request” が送信されないので，このような事は起こらない．

5.1.4 RSAを用いることによる処理負荷の増加

ISAKMP/ Oakleyでは認証方式にRSA電子署名やRSA公開鍵暗号を用いる場合，RSAの処理に多くの時間を要する事は問題である．

一度に多数の鍵交換デーモンからISAKMP SAを確立するためのメッセージが届く可能性があるホストでは，RSAの処理は大きな負荷となる．一度に多数の鍵交換デーモンからISAKMP SAを確立するためのメッセージが届く可能性のあるホストとして，WWWサーバやanonymous FTPサーバなど各種サーバが考えられる．

勿論各種サーバには優れた処理能力を有したマシンが当てられているであろう．しかし，優れた処理能力を有したマシンであっても，大量のRSAの処理は無視できない負荷に成り得る．

5.2 改善策

5.1節では，ISAKMP/ Oakleyの問題点を指摘した．そこで，本節では5.1節で指摘した問題点の改善策について提案・議論する．

5.2.1 通信遅延に対する改善策

通信遅延の影響を少なくする方法として，Main Modeの代わりにAggressive Modeを利用する方法が考えられる．3章で述べたように，ISAKMP SAが確立されていないホストとの通信を開始する時にAggressive Modeを利用すれば，Aggressive Modeに1.5往復，Quick Modeに1.5往復の計3往復の通信でSAを確立することが可能である．しかし，3.5.2小節で述べたようにAggressive Modeには認証に模擬共有鍵や電子署名を用いる場合は，IDの保護が行えないという弱点がある．

そこで，Main Modeより少しでも通信回数が少なく，且つIDの保護も行えMain Modeと同等の安全性を持った方法が望まれる．このような要求を満たす方法として，Phase 1のモードの一つとして新たに**Moderate Mode**を提案する．以下でModerate Modeについて述べる．

Moderate Mode

Moderate Modeでの大まかな処理の流れは次の通りである．

1. ISAKMP SAで使用する暗号アルゴリズムや認証アルゴリズム等，ISAKMP SAを確立する環境を決定する．この時，同時にDiffie–Hellmanの公開情報や補助的なパラメータを交換する．そして，やり取りしたデータを元にセッション鍵を生成する．

2. 通信相手を認証すると同時に生成したセッション鍵が正しいことを確認する．

以上の流れを図5.1に示す．

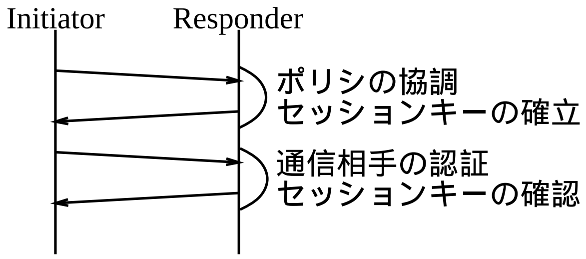


図 5.1: Moderate Mode の流れ

Moderate Mode では上記の処理を 4 回 (2 往復) の通信を経て行う．ISAKMP SA を確立する環境を取り決めるために，1 回目の通信で Initiator は，Responder に対して Initiator が利用可能な各種アルゴリズムを，使用する希望順に並べて送信する．この時にセッション鍵の生成に必要な Diffie–Hellman の公開情報と補助的なパラメータを共に送信する．Responder は，列挙された各種アルゴリズムの中から, Responder が利用可能であり，かつ一番始めに該当したアルゴリズムを使用するアルゴリズムに決定する．さらに Responder は，セッション鍵を計算した後，使用するアルゴリズムと Diffie–Hellman の公開情報と補助的なパラメータを Initiator に送信する．

次に，Initiator は，Responder から受け取ったメッセージを元にセッション鍵を計算する．さらに，Initiator は通信相手の確認・認証と算出したセッション鍵が正しい物である事を確かめるために，これまでの通信によって両者が知り得る情報とセッション鍵を元にハッシュ値を計算し，Responder に送信する．Responder は受け取ったハッシュ値の確認を行う．正しいハッシュ値が送信されて来ている場合には，Responder はこれまでの通信によって両者が知り得る情報とセッション鍵を元にハッシュ値を計算し Initiator に送信する．最後に Initiator は送信されて来たハッシュ値の確認をする．

Moderate Mode でも Main Mode や Quick Mode と同様の方法でセッション鍵を求める事が可能である．始めに全てのセッション鍵の元になる SKEYID を算出する．この SKEYID は，Main Mode や Quick Mode の時同様 Moderate Mode 中で使用する認証方式によって求め方が異なる．そして，SKEYID を元に SA で用いられる鍵の種となる SKEYID_d，ISAKMP SA で用いられる認証用の鍵 SKEYID_a，ISAKMP SA で用いられる暗号用の鍵 SKEYID_e を算出する．Main Mode・Quick Mode と同様 SKEYID_d，SKEYID_a，SKEYID_e はそれぞれ式 (3.7)，式 (3.8)，式 (3.9) より算出される．以下で，各認証方式に対する Moderate Mode の処理の流れについて議論する．

模擬共有鍵を用いた Moderate Mode

認証に模擬共有鍵を用いた時の処理の流れを図 5.2に示す．Moderate Mode では，Main Mode で 2 往復の通信を経て交換する情報を 1 往復の通信で交換する．SKEYID は式 (3.14) より，そして HASH_I，HASH_R は，それぞれ式 (3.11)，式 (3.12) より求める．

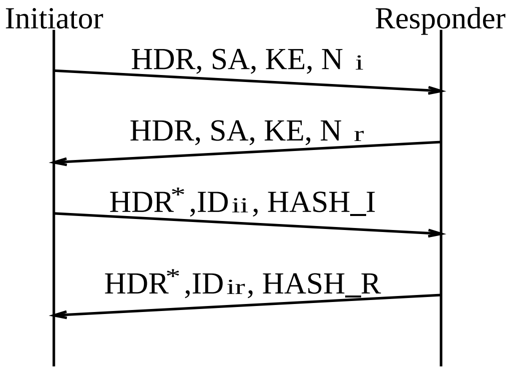


図 5.2: 模擬共有鍵を用いた Moderate Mode

電子署名を用いた Moderate Mode

認証に電子署名を用いた時の処理の流れを図 5.3に示す．Moderate Mode では，認証に模擬共有鍵を用いる時同様，Main Mode で 2 往復の通信を経て交換した情報を 1 往復の通信で交換する．SIG_I，SIG_R は，それぞれ HASH_I，HASH_R に対して電子署名を施したものであり，電子署名の方法には RSA を用いる方法や，DSS を用いる方法などがある．また，SKEYID や HASH_I，HASH_R はそれぞれ式 (3.7)，式 (3.11)，式 (3.12) より求める．

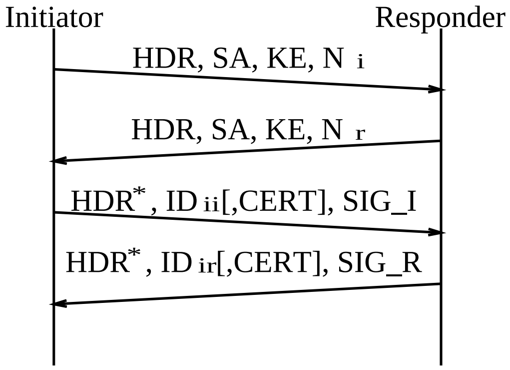


図 5.3: 電子署名を用いた Moderate Mode

公開鍵暗号を用いた Moderate Mode

認証に公開鍵暗号を用いた時の処理の流れを図 5.4に示す．SKEYID は式 3.13より，そして HASH_I，HASH_R は，それぞれ式 (3.11)，式 (3.12) より求める．そして，HASH(1) は認証書に対してハッシュ関数を作用させることにより求めたハッシュ値である．

しかし，認証に公開鍵暗号を用いる時には Moderate Mode は必要ない．Moderate Mode は，Main

Mode より通信回数が少なく，且つ ID の保護が行えるモードとして提案したものである．だが，認証に公開鍵暗号を用いる時には，Aggressive Mode でも ID の保護が行える．しかも Aggressive Mode は Main Mode と Moderate Mode より少ない通信回数で ISAKMP SA を確立する事が可能である．従って，認証に公開鍵暗号を用いる時には Moderate Mode より Aggressive Mode を用いる方が好ましい．

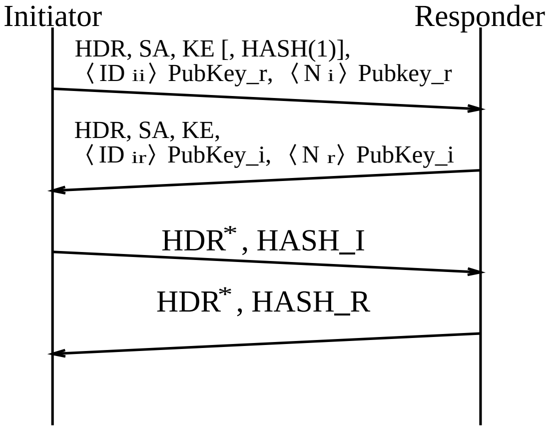


図 5.4: 公開鍵暗号を用いた Moderate Mode

Moderate Mode の問題点

以上で Moderate Mode について説明したが，Moderate Mode にも問題がある．問題の 1 つは，Main Mode の時のように最初の ISAKMP SA を確立する環境を取り決める際に，Diffie–Hellman の素数 q と原始元 α の組を合意の下に変更できない点である．Main Mode では，ISAKMP SA を確立する環境を取り決めた後に，Diffie–Hellman の公開情報を交換するのだが，Moderate Mode では，ISAKMP SA を確立する環境と取り決めると同時に Diffie–Hellman の公開情報を交換してしまう．従って，Main Mode では Initiator と Responder は，Initiator と Responder が合意した Diffie–Hellman の素数 q と原始元 α の組より公開情報を算出し，その後公開情報を交換する事が可能である．しかし，Moderate Mode では Initiator が始めに公開値を算出しなければならないので，Initiator が一方的に Diffie–Hellman の素数 q と原始元 α を決めることになる．全く未知の通信先と Moderate Mode をを利用して確実に ISAKMP SA を確立するためには，全ての鍵交換デーモンが知っている First Oakley Default Group もしくは Second Oakley Group を用いらざるを得なくなる．従って，Moderate Mode は Main Mode と比較して柔軟性に劣る．ただし，このことは Aggressive Mode でも同様である．

2 つ目の問題は，Moderate Mode は ISAKMP で定義されている Exchange Type のいずれにも属さない点である．ISAKMP/ Oakley は，ISAKMP と Oakley を組み合わせたものであるので，ISAKMP/ Oakley の各モードは ISAKMP で定義されている Exchange Type に属する事を基本方針として定義され

ている．Main Mode や Aggressive Mode は，3.5.1小節や 3.6.1小節で述べたように，それぞれ Identity Protection Exchange や Aggressive Exchange という Exchange Type に属している．従って，Moderate Mode も ISAKMP で定義されている Exchange Type のいずれかに属していることが仕様的に好ましいが，Moderate Mode はいずれの Exchange Type にも該当しない．

Main Mode および Aggressive Mode との比較

Moderate Mode を Main Mode や Aggressive Mode と以下の点において比較する．

- 通信回数: 各モードが必要とする通信回数を比較する．
- ID 保護の可否: 暗号化によって ID の保護が可能か否かを比較する．
- Diffie–Hellman の素数・原始元変更の可否: Diffie–Hellman の素数 q と原始元 α の組を変更できるか否かを比較する．
- ISAKMP Exchange Type との対応: ISAKMP で定義されている Exchange Type のいずれに属するかを比較する．

比較結果を表 5.1に示す．

表 5.1: 各モードの比較

	Moderate Mode	Main Mode	Aggressive Mode
通信回数	2 往復	3 往復	1.5 往復
ID 保護	可	可	不可
素数・原始元の変更	不可	可	不可
Exchange Type	不対応	Identity Protection Exchange	Aggressive Exchange

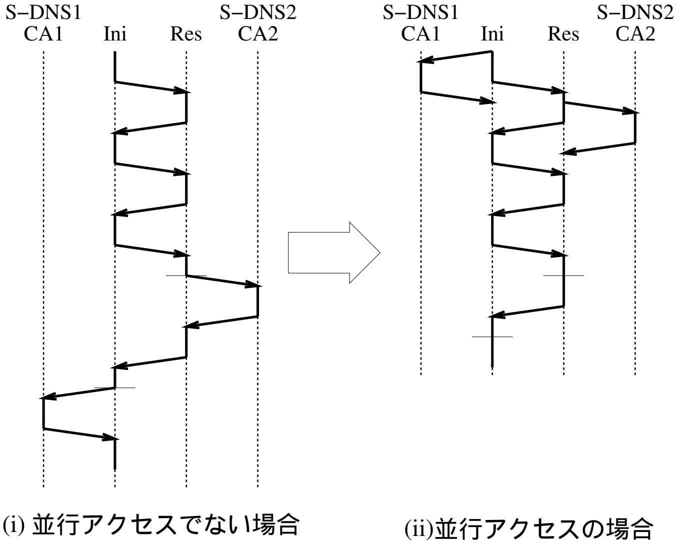
5.2.2 CA や S–DNS との接続が不明確に対する改善策

5.1.2小節で述べたように，実装に際して CA や S–DNS へのアクセス方法に対する指針が必要である．そこで，ISAKMP/ Oakley において CA や S–DNS との接続が不明確であることにたいする改善策として，本論文では CA や S–DNS へのアクセスに関する 1 つの指針を提案する．

ISAKMP/ Oakley の処理において，公開鍵や認証書などが必要になった後に CA や S–DNS へアクセスすることは，時間の浪費につながり処理効率が劣化することは自明である．そこで，CA や S–DNS へのアクセスを，ISAKMP/ Oakley の処理と並行に行うようにすべきである．鍵交換デーモンは，カーネルから SA の確立要求が届いた時点で通信相手を知ることができる．そこで，通信相手を知った時点で，CA や S–DNS へ認証書や公開鍵の要求を送信しておく．そして，送信されてきた認証書や公開鍵を保持して

おき，必要になった時点で使用する．ただし，認証書や公開鍵を必要とする時点で，CA や S–DNS から返答が返って来ていない時には，返答を待たざるを得ない．

以降では Main Mode を例に説明する．電子署名を認証に用いる Main Mode では，Initiator は通算 6 回目のメッセージを受信した際に，Responder は通算 5 回目のメッセージを受信した際に通信相手の公開鍵を必要とし，CA や S–DNS から公開鍵を入手することになる．必要に応じて CA や S–DNS へアクセスすると図 5.5の左のような処理の流れになる．ところが，カーネルから SA の確立要求が届いた時点で CA や S–DNS にアクセスをするようにすると，図 5.5の右のような処理の流れになる．図 5.5より，大幅な処理時間の短縮を実現できる可能性がある事がわかる．



(i) 並行アクセスでない場合 (ii)並行アクセスの場合

図 5.5: 並行アクセス (電子署名)

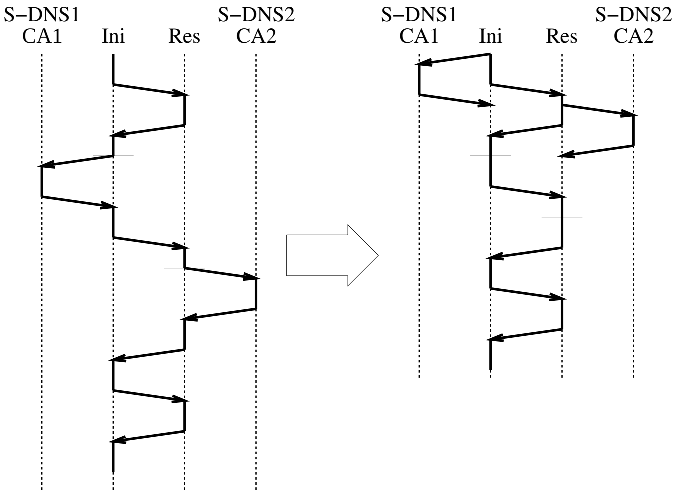
次に，公開鍵暗号を認証に用いる Main Mode では，Initiator は通算 3 回目のメッセージを送信する際に，Responder は通算 4 回目のメッセージを送信する際に通信相手の公開鍵を必要とし，CA や S–DNS から公開鍵を入手することになる．必要に応じて CA や S–DNS へアクセスすると図 5.6の左のような処理の流れになる．ところが，カーネルから SA の確立要求が届いた時点で CA や S–DNS にアクセスをするようにすると，図 5.6の右のような処理の流れになる．図 5.6より，処理時間の短縮を実現できる可能性がある事がわかる．

5.2.3 SA の同時確立に対する改善策

この問題に対する改善策は現在のところ存在しないように思われる．IPsec メイリングリスト[1]でもこの問題に対して議論されたが，同時に 2 組の SA が確立されるのは ISAKMP/ Oakley のプロトコル上避ける事が不可能であるという結論に達した．

---

[1] ipsec@tis.com



(i) 並行アクセスでない場合 (ii)並行アクセスの場合

図 5.6: 並行アクセス (公開鍵暗号)

実現の可能性がある改善策としては，セキュリティレベルが “If available” に設定されている場合にはカーネルに鍵交換デーモンに SA の確立を要求するか否かを判断させる事が考えられる．つまり，セキュリティレベルが “If availabel” に設定されている場合には，送信者としてメッセージの送信を開始したのか，それともメッセージを受信して返答としてメッセージを送信するのかをカーネルが判断する．そして，メッセージに対する返答としてメッセージを送信する時には，鍵交換デーモンに SA の確立要求を送信しないようにする．

しかしこの改善策では，メッセージの送信を開始したホストのセキュリティレベルが “**None**(C.2節参照)” に設定されており，メッセージに対する返答を行うホストのセキュリティレベルが “If availabe” の場合には，いつまでも SA が確立されない．つまり返答を送るホストは，セキュリティレベルが “If available” に設定されているにもかかわらず，いつまでも IPsec を用いた安全な通信を行うことができない．

5.2.4 RSA による処理負荷の増加に対する改善策

RSA を用いることによる処理負荷の増加は，RSA の処理を専門に行う RSA チップを利用することによって解決できる．一度に多数の鍵交換デーモンから ISAKMP SA の確立開始メッセージが届くようなサーバでは，この RSA チップを利用するようにすれば良い．鍵交換デーモンが起動している全てのホストにおいて RSA チップを利用できる事が好ましいが，今のところ RSA チップは高価な物である．従って，一度に大量の RSA の処理を行う可能性のあるサーバなどを中心に RSA チップの利用を行えば良い．

近年の Electronic Commerce の発展と共に，RSA チップの需要が高まって来ており，多くの企業が RSA チップの開発・生産を開始している．従って，今後 RSA チップの大量生産によって，RSA チップが簡単

に安価で入手できるようになるであろう．

5.3 改善策の実装

本論文では，これまで述べた 4 つの問題のうち，通信遅延に対する改善策に着目し，実装を行った．本節では通信遅延に対する改善策の実装について述べる．

なお改良は，本論文で実装を行った鍵交換デーモンに対して行った．また，5.2.1小節で述べたように，認証に公開鍵暗号を用いる Moderate Mode は Aggressive Mode と比較して利点が無いため，認証に公開鍵暗号を用いる Moderate Mode の実装は行わなかった．つまり，模擬共有鍵を認証方式に用いた Moderate Mode と電子署名を認証方式に用いた Moderate Mode の 2 つを実装した．

そして，鍵交換デーモンの “make” 時に，引数に “`-DMODER`” を指定すると，Main Mode の代わりに Moderate Mode が選択されるよう実装した．また，Main Mode の時同様認証に模擬共有鍵を用いるか電子署名を用いるかは，`/etc/isakmp-secrets` ファイルに模擬共有鍵が登録されているか否かによって決定される．

4.2節で述べたように，本論文で行った鍵交換デーモン実装は，メッセージ到着によって処理が開始し，受け取ったメッセージに対応する処理の状態によって振舞いが異なる．そしてその様子を図 4.1に示した．図 4.1のおける Main Mode の処理の流れを図 5.7のように変更した．図 4.1には，Quick Mode の流れも示してあり，Quick Mode の流れは同じである．また，Moderate Mode でも Main Mode 同様使用する認証方式によって，振舞いが多少異なる．振舞いが異なる部分に関しては図 5.7において

1. 公開鍵暗号を用いる方法
2. 電子署名を用いる方法

の順に表してある．

図 5.7に示した様に，カーネルから SA 確立要求が届いた時には，ISAKMP SA が既に確立されているか否かで振舞いが異なる．また，通信ポートへ他の鍵交換デーモンからのメッセージが到着した時には，SA の確立処理の状態によって振舞いが異なる．図 5.7で説明すると，SA の確立処理状態が，“OAKLEY_MODELETE_I_1” から “OAKLEY_QUICK_R_1” のいずれであるかによって振舞いが異なる．“MODELETE” は Moderate Mode を示しており，“QUICK” は Quick Mode を示している．また，“I” は Initiator を示しており，“R” は Responder を示している．

Moderate Mode 中で決定する ISAKMP SA で使用する暗号アルゴリズムや認証アルゴリズムは Main Mode 中で決定するものと同じである．つまり，以下の暗号アルゴリズムや認証アルゴリズムに決定される．

- ISAKMP SA で使用する暗号アルゴリズムは DES CBC モードである．
- ISAKMP SA で使用する認証アルゴリズムは Keyed–SHA1 である．

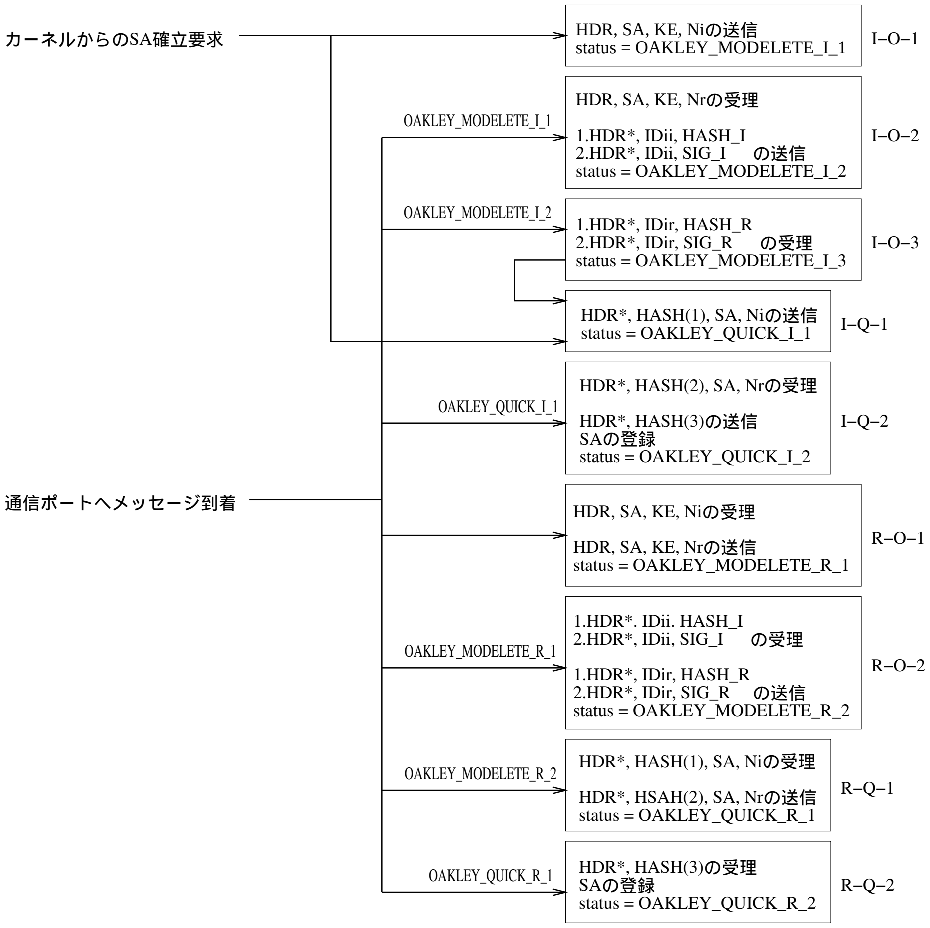


図 5.7: Moderate Mode の実装

その他の特徴は，4章で述べたものと同じである．

第 6 章

改善策に対する評価・検討

5章では，問題点を挙げ，その改善策を述べた．そして，ISAKMP/ Oakleyは通信遅延の影響を大きく受けるという問題点を改善するためにModerate Modeを実装した．本章では，Moderate Modeを用いた時にSAを確立するために要する時間を測定し，評価する．さらに評価結果からModerate Modeの有効性を検討する．

6.1 評価

5章で提案した改善策の実装に対して測定および評価を行った．測定環境は4.3節で述べた環境と同じであるが，再度述べておく．

- AT互換機 × 2
    - Intel Pentium 200MHz
    - 256KByteキャッシュ
    - 64MByteメモリ
- Free BSD 2.2.2–RELEASE with IPsec
- IPsec実験専用LAN(10Mbps Ethernet)
- SSHに関係したファイルは全てローカルディスクに存在する

測定環境を図4.2に示す．

起動しているプロセスはシステムに最低限必要なもののみであり，また不必要な通信が起こらず，かつ通信遅延が0であるという理想環境に近い状態で測定した．このような環境で，“`ping`”コマンドを実行し，一方のホストからもう一方のホストへ“ECHO Request”を送信することによって，カーネルから鍵交換デーモンへSAの確立要求が発生するようにした．

6.1.1 模擬共有鍵を用いたModerate Mode

認証に模擬共有鍵を用い，Moderate Modeを利用して通信回数を減らした時の測定結果を表6.1，表6.2，表6.3に示す．

表6.1: 模擬共有鍵を用いた時の全処理時間 (Moderate Mode)

送信者側	処理時間 [sec]	受信者側	処理時間 [sec]
Moderate Mode	0.149	Moderate Mode	0.140
Quick Mode	0.009	Quick Mode	0.006

表 6.2: 模擬共有鍵を用いた Moderate Mode の各処理時間

送信者側	処理時間 [sec]	受信者側	処理時間 [sec]
I–O–1	0.008	R–O–1	0.053
I–O–2	0.049	R–O–2	0.002
I–O–3	0.001		

表 6.3: 模擬共有鍵を用いた Quick Mode の各処理時間 (Moderate Mode)

送信者側	処理時間 [sec]	受信者側	処理時間 [sec]
I–Q–1	0.004	R–Q–1	0.004
I–Q–2	0.001	R–Q–2	0.001

表 6.1は，送信者側・受信者側でそれぞれ Moderate Mode，Quick Mode の処理に要した時間を示している．この測定結果には，通信遅延や鍵交換デーモン以外のプロセスに対する処理にかかった時間も含まれている．しかし，理想環境に近い環境で測定したので，模擬共有鍵を認証に用いて，Moderate Mode と Quick Mode を経て SA を確立する時には，最低表 6.1で示された処理時間がかかると言える．

また，表 6.2は，Moderate Mode において送信者側・受信者側で各処理にかかった時間を測定した結果を示している．同様に表 6.3は，Quick Mode において送信者側・受信者側で各処理にかかった時間を測定した結果である．これらの測定結果は，純粋に各処理にかかった時間であり，通信遅延や鍵交換デーモン以外のプロセスに対する処理時間を含んでいない．表 6.1，表 6.2，表 6.3に示した結果をまとめると図 6.1として表される．

6.1.2 電子署名を用いた Moderate Mode

認証に電子署名を用い，Moderate Mode を利用して通信回数を減らした時の測定結果を表 6.4，表 6.5，表 6.6に示す．

表 6.4: 電子署名を用いた時の全処理時間 (Moderate Mode)

送信者側	処理時間 [sec]	受信者側	処理時間 [sec]
Moderate Mode	0.309	Moderate Mode	0.287
Quick Mode	0.009	Quick Mode	0.006

表 6.4は，送信者側・受信者側でそれぞれ Moderate Mode，Quick Mode の処理に要した時間を示している．この測定結果には，通信遅延や鍵交換デーモン以外のプロセスに対する処理にかかった時間も含まれている．しかし，理想環境に近い環境で測定したので，電子署名を認証に用いて，Moderate Mode と

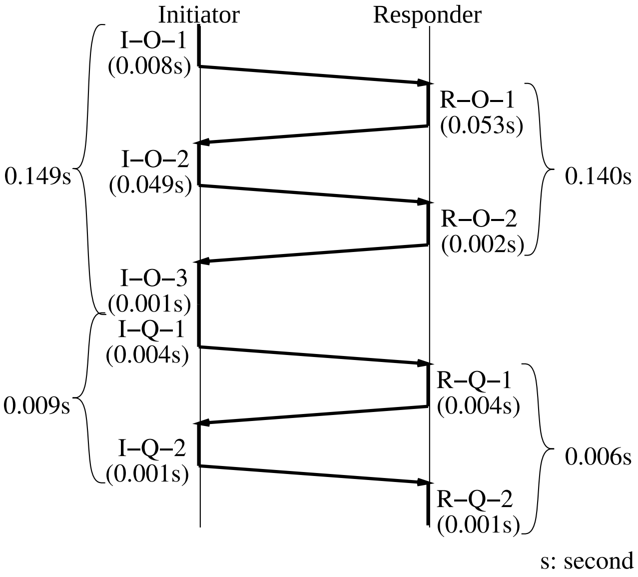


図 6.1: 模擬共有鍵を用いた Moderate Mode の測定結果

表 6.5: 電子署名を用いた Moderate Mode の各処理時間

送信者側	処理時間 [sec]	受信者側	処理時間 [sec]
I–O–1	0.008	R–O–1	0.052
I–O–2	0.120	R–O–2	0.080
I–O–3	0.007		

Quick Mode を経て SA を確立する時には，最低表 6.4で示された処理時間がかかると言える．

また，表 6.5は，Moderate Mode において送信者側・受信者側で各処理にかかった時間を測定した結果を示している．同様に表 6.6は，Quick Mode において送信者側・受信者側で各処理にかかった時間を測定した結果を示している．これらの測定結果は，純粋に各処理にかかった時間であり，通信遅延や鍵交換デーモン以外のプロセスに対する処理時間を含んでいない．表 6.4，表 6.5，表 6.6に示した結果をまとめると図 6.1として表される．

表 6.6: 電子署名を用いた Quick Mode の各処理時間 (Moderate Mode)

送信者側	処理時間 [sec]	受信者側	処理時間 [sec]
I–Q–1	0.004	R–Q–1	0.004
I–Q–2	0.001	R–Q–2	0.001

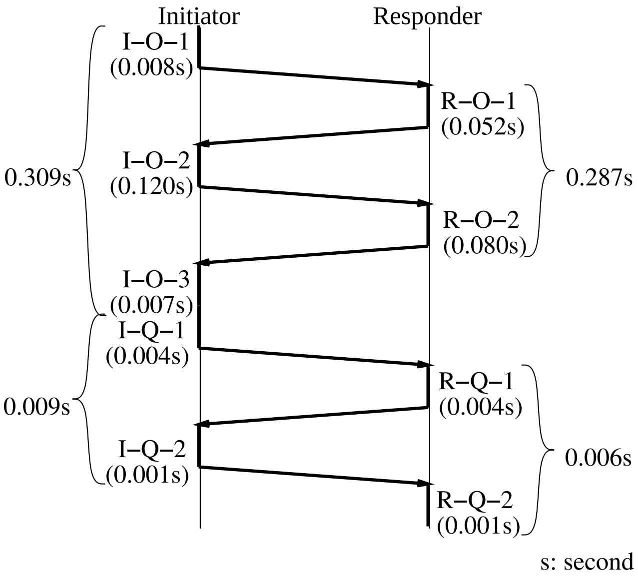


図 6.2: 電子署名を用いた Moderate Mode の測定結果

6.2 検討

本節では，6.1節で測定した Moderate Mode の処理時間に対する検討および Moderate Mode の安全性に対する検討を行う．

6.2.1 処理時間に対する検討

本小節では，Moderate Mode の処理時間に対する検討を行う．表 6.2と表 6.3より，認証に模擬共有鍵を利用し，Moderate Mode を経た時の ISAKMP/ Oakley の処理時間は 0.123 sec であることがわかる．そして，7 回の通信が発生するので，1 回の通信による遅延を t_dとすると，全処理時間 t_{all}は

$$t_{all} = 7t_d + 0.123 \tag{6.1}$$

と表される．

また，表 6.5と表 6.6より，認証に電子署名を利用し，Moderate Mode を経た時の ISAKMP/ Oakley の処理時間は 0.277 sec であることがわかる．そして，7 回の通信が起こるので，1 回の通信による遅延を t_dとすると，全処理時間 t_{all}は

$$t_{all} = 7t_d + 0.277 \tag{6.2}$$

と表される．

式 6.1と式 6.2をグラフに表すと図 6.3となる．Moderate Mode を経る場合には，全処理時間は平均通信遅延の 7 倍に比例して増える．

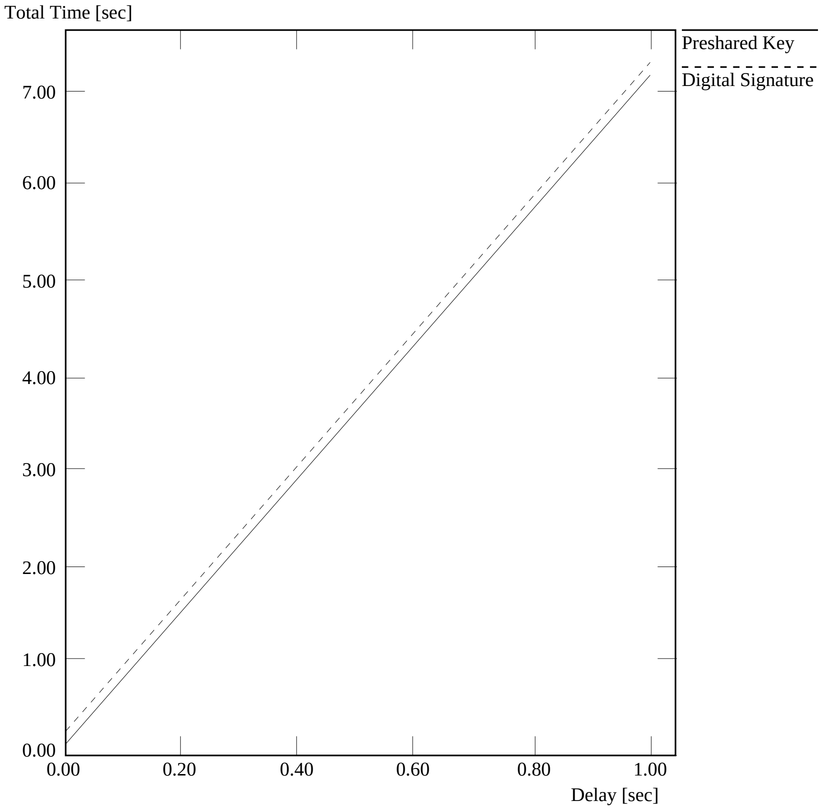


図 6.3: 通信遅延と全処理時間 (Moderate Mode + Quick Mode)

Main Mode を経て SA を確立する時と Moderate Mode を経て SA を確立する時を比較するために，図 4.6と図 6.3を同一グラフ上に表すと図 6.4になる．自明ではあるが，Main Mode を経て SA を確立する場合にかかる全処理時間と Moderate Mode を経て SA を確立する場合にかかる全処理時間の差は，通信遅延が大きくなるに従って大きくなる．

しかし，表 4.2と表 6.1を，そして表 4.5と表 6.4を比較する限り，Main Mode を経て SA を確立する場合にかかる処理時間と Moderate Mode を経て SA を確立する場合にかかる処理時間の差はわずかであり，差はほとんど無い．これは通信遅延が 0 に近い環境で測定したためである．正確な通信遅延を求めることはできないが，参考までに簡単な方法で通信遅延を求めた．本論文では通信遅延を，“`ping`” を実行することによって測定できる round–trip time を 2 で割るという方法より求めた．この方法で通信遅延を求めると，round–trip time が 5.61×10^{-4} sec であり，通信遅延は 2.86×10^{-4} sec となる．つまり，Main Mode を経て SA を確立する場合にかかる処理時間と Moderate Mode を経て SA を確立する場合にかかる処理時間の差はわずか 5.62×10^{-4} sec となり，処理時間の差は明確に現われない．

ところが，実際に広域で鍵交換デーモンによる SA の確立を行うと，通信遅延は大きい．処理時間の差が明確に現われる．例えば，慶應義塾大学大学院理工学研究科所研究室から米国ホワイトハウスへ通信

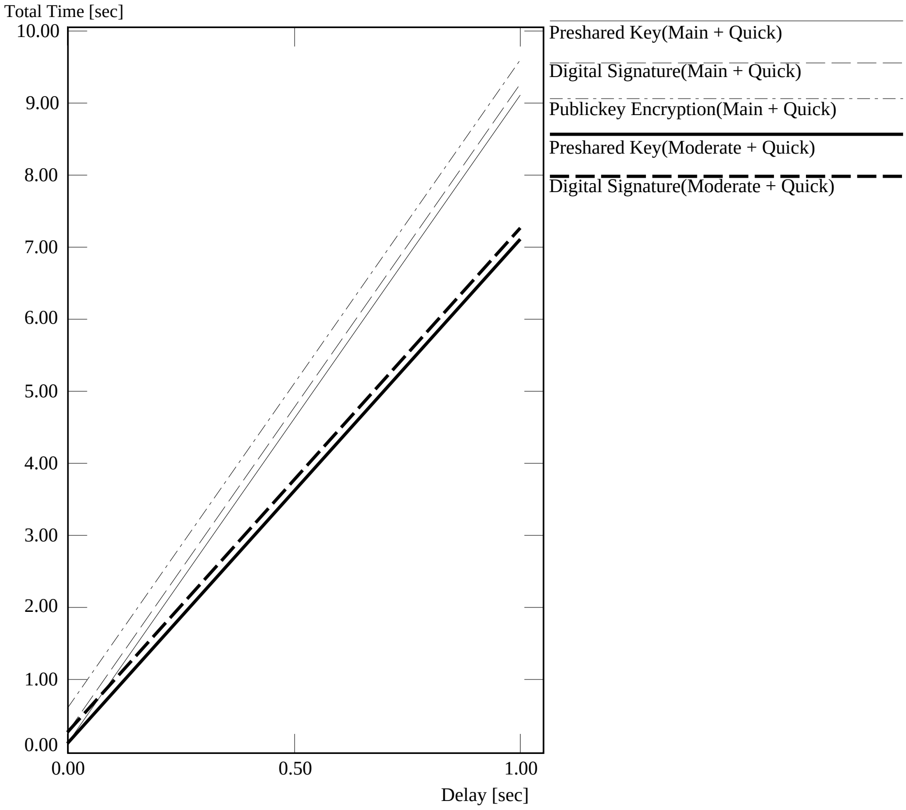


図 6.4: 通信遅延と全処理時間

を行う際の通信遅延を先に述べた方法より求めると，round–trip time が 0.289 sec であり，通信遅延は 0.145 sec である．従って，Main Mode を経て SA を確立する場合にかかる処理時間と Moderate Mode を経て SA を確立する場合にかかる処理時間の差は 0.290 sec となり，処理時間の差は明確である．また，慶應義塾大学大学院理工学研究科所研究室からフィンランドにある SSH 社へ通信を行う際の通信遅延は，round–trip time が 0.466 sec であり，通信遅延は 0.233 sec である．従って，処理時間の差は 0.466 sec となる．

6.2.2 安全性に対する検討

本小節では，Moderate Mode の安全性に対する検討を行う．本研究で提案した Moderate Mode では，各通信で送信される情報の内容および状態 (施されている処理など) は Main Mode の時と変わらない．唯一異なるのは，合意の下に Diffie–Hellman の素数 q と原始元 α の組を変更するための情報が，送信されないという点のみである．従って，Moderate Mode において送信される情報量および情報の信頼性，正当性，機密性は Main Mode の時と同等である．

さらに，交換された情報から全てのセッション鍵の種となる SKEYID を求める方法は，Main Mode や

Aggressive Mode の時と全く同じである．つまり Moderate Mode は，Main Mode において定義されている SKEYID 算出方法を用いて SKEYID を算出する．当然，Main Mode において SKEYID の算出方法は利用する認証方法によって異なるので，Moderate Mode においても認証方法によって SKEYID の算出方法が異なる．そして Moderate Mode において，SA で用いられる鍵の種となる SKEYID_d や ISAKMP SA で用いられる認証用の鍵 SKEYID_a，ISAKMP SA で用いられる暗号用の鍵 SKEYID_e は Main Mode や Aggressive Mode の時と同様の方法で求められる．従って Moderate Mode によって求められる SKEYID，SKEYID_d，SKEYID_a，SKEYID_e は，Main Mode によって求められるものと同等である．

Moderate Mode では，Diffie–Hellman の素数 qと原始元αの組を合意の下に変更する事が不可能であるため，必然的に利用される Diffie–Hellman の素数 qと原始元αの組は ISAKMP/ Oakley によって定義されている幾つかの組に決定される．これにより，Diffie–Hellman によって求められる共有情報の集合が小さくなってしまい，安全性の劣化がみられる．しかし，以下の 2 点よりこの安全性の劣化は無視できると考えられる．

1. 離散対数問題では，素数 qの値が大きければ大きい程離散対数問題を解くことが困難である．そして，ISAKMP/ Oakley で定義されている Diffie–Hellman の素数の値は，現在の計算機の処理速度では容易に離散対数問題を解くことができないくらい大きな値である．しかも，4.2.5小節で述べたように ISAKMP/ Oakley には Diffie–Hellman の素数と原始元の組が 2 つ定義されており，Diffie–Hellman によって求められる共有情報の集合が大きく，離散対数問題を解くことがより困難になっている．よって，ISAKMP/ Oakley で定義されている Diffie–Hellman の素数と原始元の組だけを利用しても十分安全であると言える．

2. Moderate Mode では，Initiator が始めに送信するメッセージ中に Diffie–Hellman の公開情報が含まれている必要があり，Initiator は最初に使用する Diffie–Hellman の素数と原始元の組を決定する必要がある．よって，Moderate Mode において合意の下で Diffie–Hellman の素数と原始元の組を変更する事は不可能であるが，Initiator が一方的に決定する事は可能である．Moderate Mode でも，ISAKMP/ Oakley によって定義されていない Diffie–Hellman の素数と原始元の組を利用する事が可能である．ただし，ISAKMP/ Oakley で定義されていない Diffie–Hellman の素数と原始元の組を利用する際に，Responder が Initiator によって指定された Diffie–Hellman の素数と原始元の組をサポートしていない場合には，エラーが起こる．

以上より，Main Mode における 6 回の通信を 4 回に削減した結果，Diffie–Hellman の素数と原始元の組を合意の下に変更することが不可能となったのみで，ISAKMP/ Oakley の安全性には影響が無い事がわかる．従って，本研究で提案した Moderate Mode は，Main Mode とほぼ同等の安全性を保持しており，Aggressive Mode 以上の安全性を保持している．

6.2.3 まとめ

Moderate Mode は，Main Mode と同等の安全性があり，かつ少ない通信回数で SA を確立することが可能である．従って，通信遅延が大きい通信先と SA を確立する場合には Moderate Mode は有効である．しかも，Moderate Mode は通信遅延が大きければ大きい程有効である．しかし Moderate Mode は Diffie–Hellman の素数 q と原始元 α の組を送信元と送信先との合意の下で変更できないため，確実性の面で Main Mode に劣る．

第 7 章

今後の課題

本章では，今後の課題として検討していかなければならない事項について述べる．今後の課題として以下の点が挙げられる．

1. マルチキャスト用の SA の確立
2. ISAKMP/ Oakley や PF_KEY への完全準拠
3. Aggressive Mode のサポート
4. IPv6 のサポート
5. IKMP への対応
6. CA や S–DNS との接続
7. 相互接続実験
8. 実運用実験
9. Moderate Mode の標準化
10. Main Mode/Quick Mode/Moderate Mode の選択方法
11. 認証方式の選択方法
12. 鍵交換アルゴリズムの選択方法

以降，各事項に関して詳しく論ずる．

マルチキャスト用の SA 確立

現在 ISAKMP/ Oakley の仕様は，ユニキャスト用の SA を確立する方法のみを定義しており，マルチキャスト用の SA を確立する方法を定義していない．しかし近い将来，マルチメディア社会においてテレビ会議システム等が普及する事が予想される．テレビ会議システム等のアプリケーションでは，マルチキャストの利用が効率面から考えても実用的である．さらに，極秘会議などをテレビ会議システムを用いて行えるように，セキュリティを考慮したマルチキャストの技術が必要となる事が予想される．このセキュリティを考慮したマルチキャストを実現する方法として，マルチキャスト技術としてに IP マルチキャストを利用し，IPsec と組み合わせる事が最も実用的である．

そこで鍵交換デーモンにマルチキャスト用の SA を確立する事が求められる．従って，今後鍵交換デーモンがマルチキャスト用の SA を確立する方法を検討していく必要がある．例えば，IP マルチキャスト用の SA を確立する鍵交換デーモンに対する提案の 1 つに [2] がある．

[2] では，マルチキャストグループを管理するサーバを用意し，このサーバ上で起動している鍵交換デーモンにマルチキャストグループ用の鍵を任意に生成させ，管理させる．マルチキャストグループに参加したいホストは，サーバ上で起動している鍵交換デーモンと協調することにより，ユニキャスト用の SA を確立する時と同様の方法で ISAKMP SA を確立し，その後 ISAKMP SA の保護の下にサーバからマルチキャストグループ用の鍵を受け取る．

ISAKMP/ Oakley や PF_KEY への完全準拠

本論文で行った鍵交換デーモンの実装は，現時点では不十分であり，ISAKMP/ Oakley の仕様や PF_KEY の仕様を完全に満たしていない．例えば現在の鍵交換デーモンは，エラーが発生しても通信相手にエラーの発生を通知するなどといった適切な対処を行っていない．しかし，ISAKMP/ Oakley ではエラーの発生へのに対する処理が仕様として定義されている．

現在 ISAKMP/ Oakley の仕様は internet draft で定義されており，現在審議中のため未だ不確定である．今後 ISAKMP/ Oakley の動向を注意深く観察しながら，ISAKMP/ Oakley の仕様を完全に満たすよう実装して行く予定である．

また，PF_KEY への準拠も不完全である．かつ，実装した鍵交換デーモンは PF_KEY version 1 に基づき実装してあるので，PF_KEY version 2 に基づいた実装に変更しなければならない．PF_KEY の仕様も ISAKMP/ Oakley 同様，現在 internet draft で審議中であり今だ不確定である．しかも，PF_KEY の仕様は IETF ipsec working group から提案されているものではない．従って，今後 PF_KEY の動向を注意深く観察しながら，PF_KEY の仕様を完全に満たすよう実装して行く予定である．

Aggressive Mode のサポート

本論文で行った鍵交換デーモンの実装は，Main Mode，Quick Mode，Moderate Mode のみをサポートしており，Aggressive Mode をサポートしていない．ISAKMP/ Oakley の仕様では，Aggressive Mode はサポートする事を推奨しているのみで、Main Mode や Quick Mode のようにサポートする事を義務付けられていない．

3.5.2小節で述べたように Aggressive Mode は ID の保護が行えず，Main Mode と較べてアタックに弱い半面，少ない通信回数で SA を確立できる．従って，利用方法によっては Aggressive Mode は非常に有効であり，サポートされている方が好ましい．従って，今後 Aggressive Mode をサポートし Aggressive Mode を利用できるように実装を行う．

IPv6 のサポート

IPsec は，IPv6 の仕様を定義する中で IP 層におけるセキュリティ確保を目的として提案されたものである．つまり IPv6 では必ず IPsec を利用できなければならない．さらに，鍵交換デーモンは IPsec の SA

を確立する事を目的として提案されたものである．つまり，IPsec と鍵交換デーモンは表裏一体と言える．従って，鍵交換デーモンは IPv6 を必ずサポートする必要がある．

本論文で行った鍵交換デーモンの実装では IPv4 のサポートのみにとどまっている．しかし，今後 IPv4 から IPv6 への急速な移行が予想される．従って，早急に IPv6 をサポートする必要があり，今後 IPv6 に対応するように実装を行うべきである．

IKMP への対応

3章で述べたように，Schiller の決定によって ISAKMP/ Oakley を基に IKMP を作成する事が宣言された．現時点では IKMP の完成に至っていないが，近い内に IKMP が作成され，世界標準規格となると予想される．本論文で行った鍵交換デーモンの実装は ISAKMP/ Oakley に従っており，いずれ IKMP の仕様に従う必要がある．

今後，IKMP が完成した際には，ISAKMP/ Oakley の仕様と IKMP の仕様の動向を探りながら，徐々に IKMP の仕様に対応していく予定である．一日でも早く IKMP の仕様が決まる事を期待する．

CA や S–DNS との接続

本論文で実装を行った鍵交換デーモンでは，電子署名や公開鍵暗号を認証方式とする時に必要とされる秘密鍵と公開鍵は，SSH で使用されているものを利用した．しかし，本来は通信相手の公開鍵を CA や S–DNS 等信頼できる第 3 者認証機関から入手する必要がある．

現在日本国内では，CA や S–DNS が普及しておらず，容易に利用できない．しかし，セキュリティ技術の発展や Electronic Commerce の普及に伴い，CA や S–DNS が普及する事が予想される．そこで，今後正しく CA や S–DNS 等の信頼できる第 3 者認証期間から公開鍵を入手するように改良していく必要がある．

相互接続実験

本論文で行った実装は，ISAKMP/ Oakley の仕様に基づいており，他の ISAKMP/ Oakley の仕様に基づいて実装された鍵交換デーモンと協調して SA を確立することが可能なはずである．

しかし，本論文では他の鍵交換デーモンとの相互接続実験を行うに至らなかった．よって，本論文で実装した鍵交換デーモンが，他の鍵交換デーモンと協調して SA を確立することが可能であるか否かが定かではない．

今後，他の ISAKMP/ Oakley の仕様に基づいて実装された鍵交換デーモンと相互接続実験を行い，本論文で実装した鍵交換デーモンが，正しく ISAKMP/ Oakley の仕様に基づいている事を確かめなければならない．そして，もし相互接続実験に失敗した時には，正しく ISAKMP/ Oakley の仕様に基づくよう改良しなければならない．現在，**SSH Communications Security** 社による，ISAKMP/ Oakley 鍵交

換デーモンの相互接続実験サイト[1]がある．今後このようなサイト等を利用して相互接続実験を行う予定である．

実運用実験

本論文では，実装した鍵交換デーモンの実運用実験を行わなかった．従って，実験専用の LAN 内で，かつ性能測定を行う程度の利用では現われなかった実装上の問題が存在する可能性がある．今後，本論文で実装した鍵交換デーモンの耐久テスト等を兼ねて，実運用を行い，問題点の洗い出しや性能評価を行う必要がある．

本論文では実装した鍵交換デーモンを無料公開した．そこで，本論文で実装した鍵交換デーモンの利用希望者に協力を仰ぎ，実運用実験を行いたいと思う．

Moderate Mode の標準化

本論文では，Moderate Mode の提案および実装を行った．しかし，Moderate Mode が多くの鍵交換デーモンに実装され，利用されるためには Moderate Mode が世界標準になる必要がある．つまり，今後 IPsec メイリングリストや IETF ipsec working group へ Moderate Mode を提案し，Moderate Mode が世界標準として認められなければならない．従って，Moderate Mode を再検討した後，IPsec メイリングリストへ提案する予定である．

Main Mode/Moderate Mode/Aggressive Mode の選択方法

5.3節で述べた様に，本論文で行った鍵交換デーモンの実装では Main Mode と Moderate Mode の切り替えは，鍵交換デーモンの “make” 時に “`-DMODER`” を引数に指定するか否かで行われる．しかし，使用するモードを切り替えるために鍵交換デーモンを作り直すことは好ましくない．Main Mode と Moderate Mode と Aggressive Mode の切り替えを動的に行える事が好ましい．

Main Mode や Moderate Mode，Aggressive Mode のいずれを利用するかは，管理者が通信先ホスト毎に設定できる事が望ましい．そこで，Moderate Mode を用いて SA を確立するホストを列挙した設定ファイルと Aggressive Mode を用いて SA を確立するホストを列挙した設定ファイルを作成し，鍵交換デーモンがこれらのファイルを検索するようにする．そして，Moderate Mode や Aggressive Mode を用いて SA を確立するホストとして登録されている場合には，Moderate Mode や Aggressive Mode を用いて SA を確立し，Moderate Mode や Aggressive Mode を用いて SA を確立するホストとして登録されていない時には Main Mode を用いて SA を確立するようにする．今後，以上で述べた事を満たすように改良を行う．

---

[1] http://isakmp-test.ssh.fi

認証の選択方法

4.2.3小節で述べた様に，本論文で行った鍵交換デーモンの実装では，“`/etc/isakmp-secrets`” ファイルに模擬共有鍵が登録されていれば認証に模擬共有鍵が使用され，模擬共有鍵に登録されていない時には，認証に電子署名もしくは公開鍵暗号が使用される．そして，認証に電子署名と公開鍵暗号のいずれが使用されるかは，鍵交換デーモンの “make” 時に “`-DRSAENC`” を引数に指定するか否かによって決まる．“make” 時に “`-DRSAENC`” を指定すると公開鍵暗号が認証に用いられる．

しかし，この様な方式は好ましくない．本来 ISAKMP/ Oakley は，鍵交換のプロトコルのみを定義しており，ISAKMP/ Oakley 中で使用するアルゴリズムに関して制限していない．従って，様々なアルゴリズムを認証に用いる事ができる．実際に，今でも電子署名のアルゴリズムに RSA もしくは DSS を用いることが可能である．今後，様々な電子署名のアルゴリズムや公開鍵暗号のアルゴリズムが考案され，鍵交換デーモンで使用される事が予想される．様々なアルゴリズムを使用できるようになった時に，いずれのアルゴリズムを用いて認証を行うのかが問題となってくる．そこで，今後様々な認証方式をどのように選択し，使用するかを検討していかなければならない．

鍵交換アルゴリズムの選択方法

現在 ISAKMP/ Oakley で使用できる鍵交換アルゴリズムは Diffie–Hellman 鍵交換アルゴリズムのみである．しかし，今後様々な鍵交換アルゴリズムが考案され，鍵交換デーモンで使用されるようになる事が予想される．そこで，今後どのように使用する鍵交換アルゴリズムを選択するかを検討していかなければならない．

第 8 章

結論

近年，インターネットの商用利用が広まり，Electronic Commerce などが実現するようになった．これに伴い，クラッカによるアタックに強い安全なネットワークが必要とされるようになった．そこで，安全なネットワークを実現するために多くのセキュリティプロトコルやセキュリティシステムが提案され実装されてきた．これらによって，比較的安全なネットワーク環境が提供されるようになったが，暗号プロトコルの標準化のために IPsec が提案された．

本論文では，IPsec の広域での運用に必要不可欠な鍵交換デーモンを ISAKMP/ Oakley に基づいて実装し，実装した鍵交換デーモンの測定および評価を行った．また ISAKMP/ Oakley の問題点を 4 点指摘し，それぞれの問題点に対する改善策を述べた．本論文で指摘した問題点は以下の 4 点である．

- ISAKMP/ Oakley は通信遅延の影響を被りやすい．
- CA や S–DNS へのアクセス方法が明確に定義されていない．
- SA が不必要に重複して確立されてしまう場合がある．
- RSA を用いる場合には，RSA の処理により負荷が増加する．

本論文では “ISAKMP/ Oakley は通信遅延の影響を被りやすい” と言う問題に対して，Moderate Mode という新たなモードを提案し，実装および評価を行った．そして，Moderate Mode の安全性と有効性を示した．

本論文では，実験専用の LAN を用いて評価を行った．そのため，本論文では実運用下における評価を行うに至らず，実用的な結果を得ることができたとは言いがたい．そこで，今後実運用下において評価を行い，実用的な結果を得る．また本論文で行った実装は現時点では，エラー発生時における処理などの点において ISAKMP/ Oakley や PF_KEY などで定義されている仕様に完全には準拠していない．今後，ISAKMP/ Oakley や PF_KEY などの標準化の動向を見ながらこれらに完全に準拠した鍵交換デーモンに改良していく予定である．

現在多くの国では，暗号を用いたセキュリティ技術の輸出が規制されている．従って鍵交換デーモンの実装は，輸出規制のために広く行き渡っておらず，自由に利用できる鍵交換デーモンは数少ない．そのため，IPsec が広く一般に利用されていない．日本には輸出規制がないので，未だ不完全なものではあるが，IPsec の普及を促進する一翼を担う事を希望し，本論文で行った実装を世界に向けて無料公開した[1]．今後 IPsec と鍵交換デーモンの普及により，より安全で快適なインターネット環境が提供されることを期待する．

---

[1] `http://www.mt.cs.keio.ac.jp/person/negi/implement.html`

謝辞

まず始めに，この研究を行う機会を与えて下さり，親切な御指導と御助言を頂きました慶應義塾大学理工学部客員教授 所 真理雄博士に深く感謝致します．

また，本研究に関し貴重な時間を割き多大なる御助言と御助力を頂きました，慶應義塾大学大学院所研究室後期博士課程3年の伊藤 純一郎氏，後期博士課程2年の福田 健介氏，佐々木 貴宏氏に感謝致します．

そして，今日まで，互いに励まし合い，助け合い，それぞれの研究を成し遂げて来た慶應義塾大学大学院所研究室 修士2年の池谷 直紀氏，勝野 恭治氏，北脇 晴之氏，久曽神 宏氏，桑折 隆之氏，三輪 泰孝氏，矢崎 啓之氏，Pornchai LAOHAKULVATHIT 氏に感謝致します．

さらに，貴重な時間を割き本論文の査読をしてくれた慶應義塾大学大学院所研究室修士1年の舌間 一宏氏に感謝いたします．

また，学科・研究室の違いはあれ，共に遊び，学び，研究をしてきた，慶應義塾大学大学院笹瀬研究室修士2年の篠井 健一郎氏，遠藤 俊樹氏，その他多くの友人に感謝致します．

最後となりましたが，本研究を影から支えて下さった私の両親や兄弟に深く深く感謝致します．

1998年2月

Appendix A

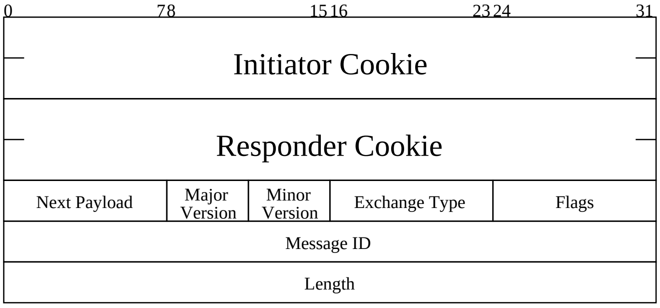
ISAKMPの仕様

A.1 Payload Format

本節では，ISAKMPで定義されているPayloadのフォーマットについて記述する．

A.1.1 ISAKMP Header Format

ISAKMPでは図A.1に示すような決まったHeader Formatが定義されている．



図A.1: ISAKMP Header Format

Initiator Cookie (8 Bytes) SAの確立や削除などの処理を開始した者のクッキー．

Responder Cookie (8 Bytes) SAの確立や削除などの処理に応じる者のクッキー．

Next Payload (1 Byte) ISAKMP Headerの次に繋がっているPayloadの識別子．識別子は表A.1で定義される．

Major Version (4 bits) ISAKMPのバージョン識別子．

Minor Version (4 bits) ISAKMPのマイナーバージョン識別子．

Exchange Type (1 Byte) ISAKMP Exchange Typeの識別子．Exchange Typeによってメッセージや Payloadの順が決定する．識別子は表A.2で定義される．また，Exchange TypeはA.2で示す．

Flags (1 Byte) 特定のオプション．最下位(0 bit目)がEncryption Bitであり，1 bit目がCommit Bitである．

- **Encryption Bit (1 bit)** 1がセットされている時には，ISAKMP Header以降のメッセージが暗号化されている事を示す．
- **Commit Bit (1 bit)** 鍵生成の同期を取るために使用される．1をセットされたメッセージの受け手側は，送り手側からの鍵の生成を終了した旨を伝えるメッセージを待たなければならない．

Message ID (4 Bytes) メッセージの識別子．Phase 2で使用され，Phase 1では0でなければならない．

Length (4 bytes) メッセージの全長．

表 A.1: Payload Type

Next Payload Type	Value
NONE	0
Security Association (SA)	1
Proposal (P)	2
Transform (T)	3
Key Exchange (KE)	4
Identification (ID)	5
Certificate (CERT)	6
Certificate Request (CR)	7
Hash (HASH)	8
Signature (SIG)	9
Nonce (NONCE)	10
Notification (N)	11
Delete (D)	12
RESERVED	13 – 127
Private USE	128 – 255

表 A.2: Exchange Type

Exchange Type	Value
NONE	0
Base	1
Identity Protection	2
Authentication Only	3
Aggressive	4
Informational	5
ISAKMP Future Use	6 – 31
DOI Specific Use	32 – 255

A.1.2 generic Payload Header

generic Payload Header は A.1.4小節から A.1.15小節で定義される ISAKMP Payload Header の共通部分である．generic Payload Header のフォーマットを図 A.2に示す．

Next Payload (1 Byte) 次に繋がっている Payload の識別子．識別子は表 A.1で定義される．

Reserved (1 Byte) 未使用．0 でなければならない．

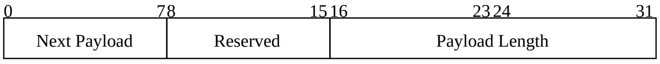


図 A.2: generic Payload Header

Payload Length (2 Bytes) generic Payload Hader を含めた現 Payload 長．

A.1.3 Data Attributes

Data Attributes は ISAKMP の Payload ではないが，ISAKMP の Payload の中に含まれて様々な情報を示すために使用される．Data Attributes のフォーマットを図 A.3に示す．

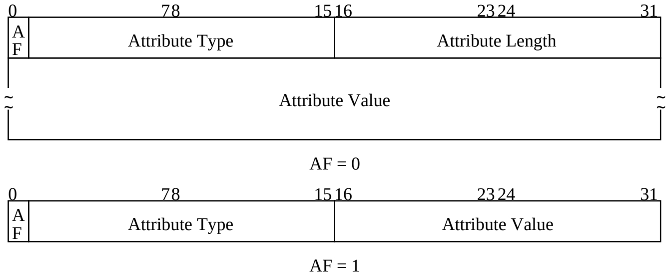


図 A.3: Data Attributes

Attribute Type (2 Bytes) 情報の属性値．属性値は **DOI(Domain of Interpretation)** として [27] などで定義される．ただし，最上位ビット (16 ビット目) は Data Attributes フォーマット (AF: Attribute Format) の識別に用いられる．AF が 0 の時には Type/Length/Value (TLV) フォーマットを示し，AF が 1 の時には Type/Value (TV) フォーマットを示す．

Attribute Length (2 Bytes) Attribute Value の長さ．AF が 1 の時には Attribute Value の長さは 2 Bytes と決まっており，Attribute Length は示されない．

Attribute Value (任意長) 情報の値．値の意味は，[27] などで定義される．AF が 0 の時には，Attribute Value は任意長であるが，4 Bytes の整数倍になっていなければならない．実際に値が 4 Bytes に満たない時には，0 を埋めて 4 Bytes の整数倍に整える．また，AF が 1 の時には 2 Bytes と決まっている．

A.1.4 Security Association Payload Format

Security Association Payload は，SA を確立する環境に関する情報を交換するために使用される．Security Association Payload のフォーマットを図 A.4に示す．

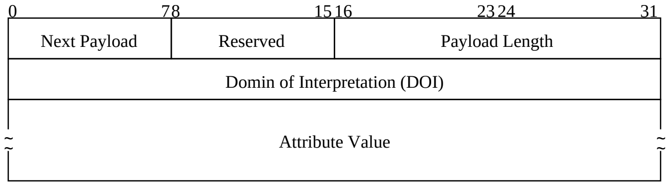


図 A.4: Security Association Payload Format

Next Payload (1 Byte) 次に繋がっている Payload の識別子．識別子は表 A.1で定義される．ただし，次に Proposal Payload や Transform Payload が続いていたとしても，これらの識別子は示されない．

Reserved (1 Byte) 未使用．0 でなければならない．

Payload Length (2 Bytes) Security Association Payload と Proposal Payload と Transform Payload を含めた長さ．

Domain of Interpretation (4 Bytes) SA の確立が行われる環境を示すために使用される DOI の識別子．Internet のための DOI を示す識別子として 1 が割り当てられている．

Situation (任意長) SA が確立される環境を示す DOI 値．DOI 値は [27] などで定義されている．

A.1.5 Proposal Payload Format

Proposal Payload は，SA を確立するために必要な情報を含んだ Payload である．Proposal Payload のフォーマットを図 A.5に示す．

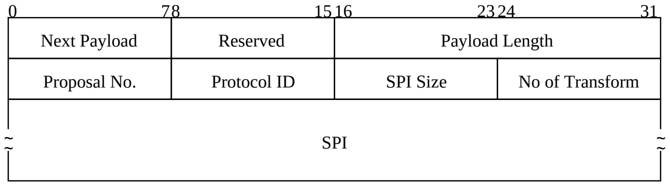


図 A.5: Proposal Payload Format

Next Payload (1 Byte) 次に繋がっている Payload の識別子．識別子は表 A.1で定義される．ここには，識別子として “2” もしくは “0” しか示されない．

Reserved (1 Byte) 未使用．0 でなければならない．

Payload Length (2 Bytes) Proposal Payload と全ての関係する Transform Payload を含めた長さ．

Proposal No. (1 Byte) 現 Proposal Payload の番号．

Protocol ID (1 Byte) どのセキュリティシステムのための SA を確立するかを示す識別子．識別子は [27] などで定義される．

SPI Size (1 Byte) SPI 長．

Number of Transfroms (1 Byte) 現 Proposal Payload に関係する Transform Payload の数．

SPI (任意長) SPI 値．

A.1.6 Transform Payload Format

Transform Payload は，SA で使用される，暗号アルゴリズムや認証アルゴリズム，ハッシュ関数などの情報を交換するために使用される．Transform Payload のフォーマットを図 A.6に示す．

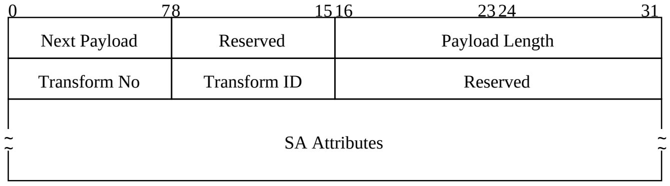


図 A.6: Transform Payload Format

Next Payload (1 Byte) 次に繋がっている Payload の識別子．識別子は表 A.1で定義される．ここでは，識別子として “3” もしくは “0” しか示されない．

Reserved (1 Byte) 未使用．0 でなければならない．

Payload Length (2 Bytes) generic Payload Hader を含めた現 Payload 長．

Transform No. (1 Byte) 現 Transform Payload の番号．

Transform ID (1 Byte) SA で使用使用される暗号アルゴリズムやハッシュ関数を示す識別子．

Reserved (2 Bytes) 未使用．0 でなければならない．

SA Attributes (任意長) 確立する SA に関する付加情報．付加情報として，Diffie–Hellman の素数 qと原始元αの組や，有効期限などがある．

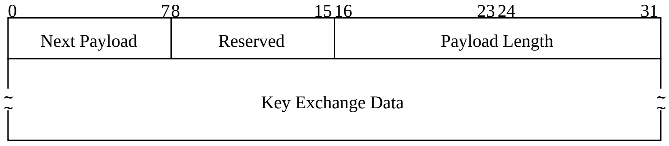


図 A.7: Key Exchange Payload Format

A.1.7 Key Exchange Payload Format

Key Exchange Payload は，鍵交換アルゴリズムが必要とする情報を交換するために使用される．Key Exchange Payload のフォーマットを図 A.7に示す．

Next Payload (1 Byte) 次に繋がっている Payload の識別子．識別子は表 A.1で定義される．

Reserved (1 Byte) 未使用．0 でなければならない．

Payload Length (2 Bytes) generic Payload Hader を含めた現 Payload 長．

Key Exchange Data (任意長) セッションキーを生成するために必要なデータ.

A.1.8 Identification Payload Format

Identification Payload は，身元を示す情報を送信するために使用される．Identification Payload のフォーマットを図 A.8に示す．

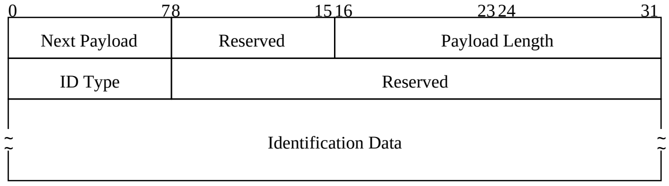


図 A.8: Identification Payload Format

Next Payload (1 Byte) 次に繋がっている Payload の識別子．識別子は表 A.1で定義される．

Reserved (1 Byte) 未使用．0 でなければならない．

Payload Length (2 Bytes) generic Payload Hader を含めた現 Payload 長．

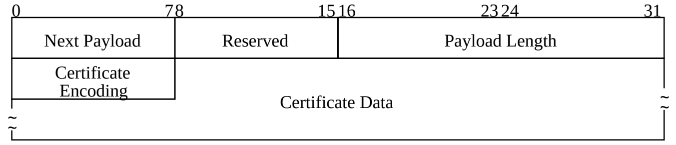
ID Type (1 Byte) 身元を示す情報の種類を示す．

Reserved (3 Bytes) 未使用．0 でなければならない．

Identification data (任意長) 身元を示す情報．

A.1.9 Certificate Payload Format

Certificate Payloadは，認証書や認証情報を送信するために使用される．Certificate PayloadはISAKMPのいかなるメッセージにも含まれ得，いかなる時にも受理されなければならない．Certificate Payloadのフォーマットを図A.9に示す．



図A.9: Certificate Payload Format

Next Payload (1 Byte) 次に繋がっているPayloadの識別子．識別子は表A.1で定義される．

Reserved (1 Byte) 未使用．0でなければならない．

Payload Length (2 Bytes) generic Payload Haderを含めた現Payload長．

Certificate Encoding (1 Byte) 認証書や認証情報の種類．認証書や認証情報の種類は表A.3で示すものがある．

Certificate Data (任意長) 認証書や認証情報．

表A.3: Certificate Type

Certificate Type	Value
NONE	0
PKCS #7 wrapped X.509 certificate	1
PGP Certificate	2
DNS Signed Key	3
X.509 Certificate – Signature	4
X.509 Certificate – Key Exchange	5
Kerberos Tokens	6
Certificate Revocation List (CRL)	7
Authority Revocation List (SRT)	8
SPKI Certificate	9
RESERVED	10 – 255

A.1.10 Certificate Request Payload Format

Certificate Request Payload は，認証書や認証情報を要求するために使用される．Certificate Request Payload は ISAKMP のいかなるメッセージにも含まれ得，いかなる時にも受理されなければならない．Certificate Request を受け取った者は，ただちに認証書もしくは認証情報を送信しなければならない．Certificate Request Payload のフォーマットを図 A.10に示す．

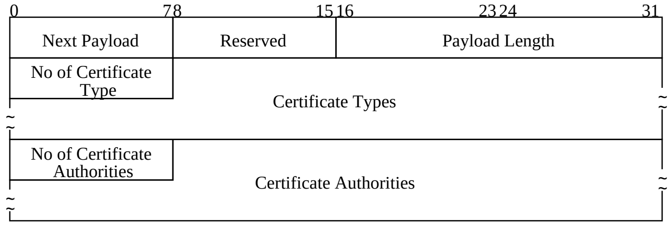


図 A.10: Certificate Request Payload Format

Next Payload (1 Byte) 次に繋がっている Payload の識別子．識別子は表 A.1で定義される．

Reserved (1 Byte) 未使用．0 でなければならない．

Payload Length (2 Bytes) generic Payload Hader を含めた現 Payload 長．

Number of Certificate Types (1 Byte) 要求する認証書・認証情報の種類数．

Certificate Types (任意長) 要求する認証書・認証情報の種類のリスト．リストは，認証書・認証情報の要求者が希望する種類順に並べられる．認証書・認証情報の種類は表 A.3で示すものがある．

Number of Certificate Authorities (1 Byte) 要求する認証書・認証情報の数．

Certificate Authorities (任意長) 認証書・認証情報で認証したい相手のリスト．

A.1.11 Hash Payload Format

Hash Payload は，ハッシュ関数によって算出されるメッセージのハッシュ値を含む．Hash Payload のフォーマットを図 A.11に示す．

Next Payload (1 Byte) 次に繋がっている Payload の識別子．識別子は表 A.1で定義される．

Reserved (1 Byte) 未使用．0 でなければならない．

Payload Length (2 Bytes) generic Payload Hader を含めた現 Payload 長．

Hash Data (任意長) メッセージなどに対してハッシュ関数を施したハッシュ値．

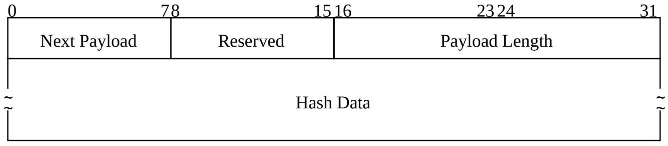


図 A.11: Hash Payload Format

A.1.12 Signature Payload Format

Signature Payload は，メッセージなどの電子署名を含む．Signature Payload のフォーマットを図 A.12 に示す．

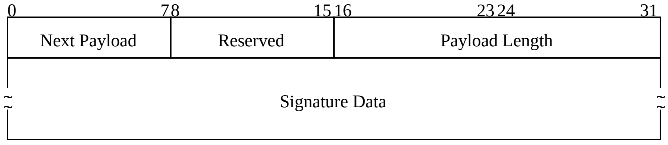


図 A.12: Signature Payload Format

Next Payload (1 Byte) 次に繋がっている Payload の識別子．識別子は表 A.1で定義される．

Reserved (1 Byte) 未使用．0 でなければならない．

Payload Length (2 Bytes) generic Payload Hader を含めた現 Payload 長．

Signature Data (任意長) メッセージなどに対する電子署名．

A.1.13 Nonce Payload Format

Nonce Payload は乱数情報を含む．Nonce Payload のフォーマットを図 A.13に示す．

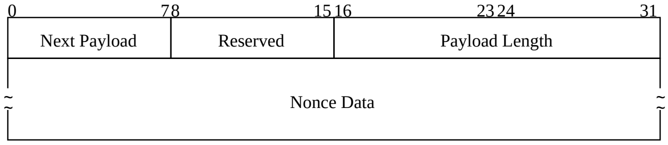


図 A.13: Nonce Payload Format

Next Payload (1 Byte) 次に繋がっている Payload の識別子．識別子は表 A.1で定義される．

Reserved (1 Byte) 未使用．0 でなければならない．

Payload Length (2 Bytes) generic Payload Hader を含めた現 Payload 長．

Nonce Data (任意長) 乱数情報．

A.1.14 Notification Payload Format

Notification Payload は，エラーの通知等さまざまな情報を通知するために使用される．Notification Payload のフォーマットを図 A.14に示す．

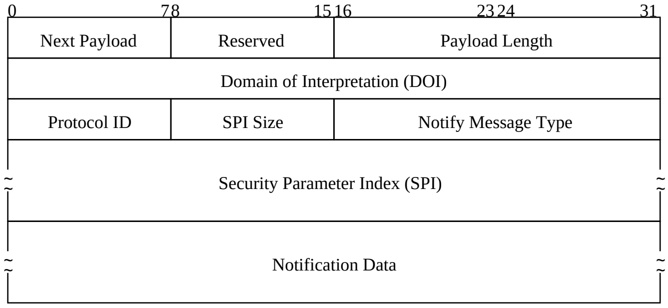


図 A.14: Notification Payload Format

Next Payload (1 Byte) 次に繋がっている Payload の識別子．識別子は表 A.1で定義される．

Reserved (1 Byte) 未使用．0 でなければならない．

Payload Length (2 Bytes) generic Payload Hader を含めた現 Payload 長．

Domain of Interpretation (4 Bytes) 通知を行う環境を示すために使用される DOI の識別子．Internet のための DOI 識別子を示す識別子として 1 が割り当てられている．

Protocol ID (1 Byte) プロトコルの識別子．

SPI Size (1 Byte) SPI 長．

Notify Message Type (2 Bytes) 通知の種類．種類を表 A.4と表 A.5に示す．

SPI (任意長) SPI 値．Phase 1 では SPI 値は，Initiator Cookie と Responder Cookie の組となる．

Notification Data (任意長) エラーなどに対する詳細情報．

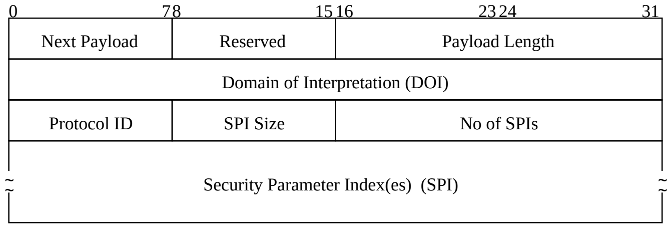


図 A.15: Delete Payload Format

A.1.15 Delete Payload Format

Delete Payload は，SA を削除するために使用される．1 つの Delete Payload で複数の SA を削除できるが，ただし同じセキュリティシステムに対する SA でなければばならない．Delete Payload のフォーマットを図 A.15に示す．

Next Payload (1 Byte) 次に繋がっている Payload の識別子．識別子は表 A.1で定義される．

Reserved (1 Byte) 未使用．0 でなければならない．

Payload Length (2 Bytes) generic Payload Hader を含めた現 Payload 長．

Domain of Interpretation (4 Bytes) SA の削除を行う環境を示すために使用される DOI の識別子．Internet のための DOI を示す識別子として 1 が割り当てられている．

Protocol ID (1 Byte) いずれのセキュリティシステムに対する SA を削除するのかについての情報．

SPI Size (1 Byte) SPI 長．

Number of SPIs (2 Bytes) 削除する SPI(SA) の数．

SPI(es) (任意長) 削除する SPI(SA) 群．

A.2 ISAKMP Exchange Type

ISAKMP では，次の 5 つの Exchange Type が定義されている．

A.2.1 Base Exchange

身元を示す情報の保護を行わない代わりに，鍵生成に関する情報と認証を行うために必要な情報を同時に送信することによって通信回数の削減を行う．Base Exchange を図 A.16に示す．

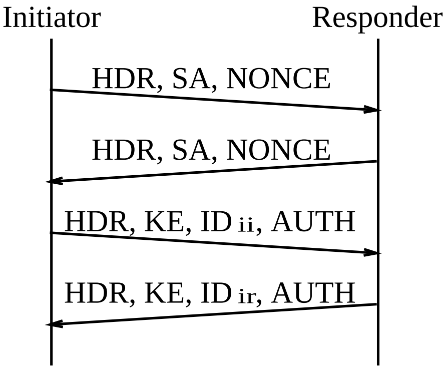


図 A.16: Base Exchange

A.2.2 Identity Protection Exchange

通信回数が多い代わりに，鍵生成に関する情報を身元を示す情報や認証情報から分離させることにより，通信者の身元に関する情報の保護を行う．Identity Protection Exchange を図 A.17に示す．

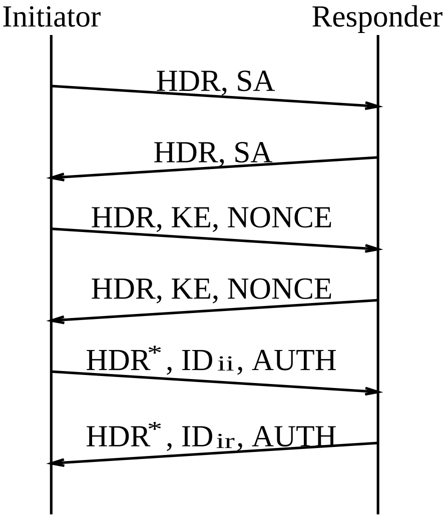


図 A.17: Identity Protection Exchange

A.2.3 Authentication Only Exchange

鍵生成に関する情報は送らず，認証を行うために必要な情報のみを送信する．この方法は Phase 2 で用いられ，ISAKMP SA によって送信されるメッセージは保護される．Authentication Only Exchange を図 A.18に示す．

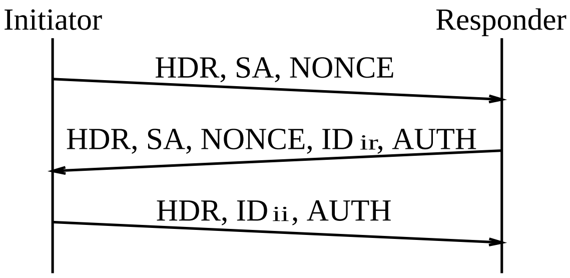


図 A.18: Authentication Only Exchange

A.2.4 Aggressive Exchange

身元を示す情報の保護を行わない代わりに，SA を確立する環境に関する情報と鍵生成に関する情報と認証を行うために必要な情報を同時に送信することによって，通信回数の削減を行う．Aggressive Exchange を図 A.19に示す．

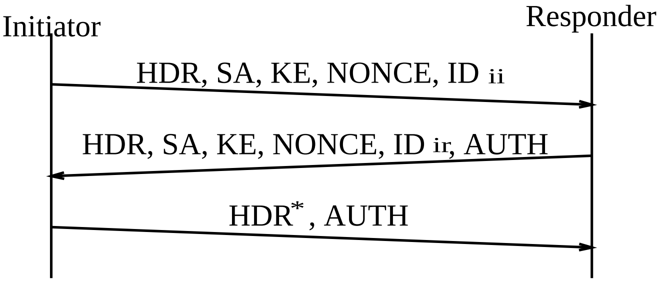


図 A.19: Aggressive Exchange

A.2.5 Informational Exchange

一方向の通知を行う．エラー通知などに利用される．Informational Exchange を図 A.20に示す．

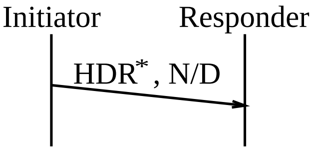


図 A.20: Informational Exchange

表 A.4: Notify Message – Error Types

Errors	Value
INVALID–PAYLOAD–TYPE	1
DOI–NOT–SUPPORTED	2
SITUATION–NOT–SUPPORTED	3
INVALID–COOKIE	4
INVALID–MAJOR–VERSION	5
INVALID–MINOR–VERSION	6
INVALID–EXCHANGE–TYPE	7
INVALID–FLAGS	8
INVALID–MESSAGE–ID	9
INVALID–PROTOCOL–ID	10
INVALID–SPI	11
INVALID–TRANSFORM–ID	12
ATTRIBUTES–NOT–SUPPORTED	13
NO–PROPOSAL–CHOSEN	14
BAD–PROPOSAL–SYNTAX	15
PAYLOAD–MALFORMED	16
INVALID–KEY–INFORMATION	17
INVALID–ID–INFORMATION	18
INVALID–CERT–ENCODING	19
INVALID–CERTIFICATE	20
BAD–CERT–REQUEST–SYNTAX	21
INVALID–CERT–AUTHORITY	22
INVALID–HASH–INFORMATION	23
AUTHENTICATION–FAILED	24
INVALID–SIGNATURE	25
ADDRESS–NOTIFICATION	26
RESERVED (Future Use)	27 – 8191
Private Use	8192 – 16383

表 A.5: Notify Message – Status Types

Status	Value
CONNECTED	16384
RESERVED (Future Use)	16385 – 24575
Private Use	24576 – 32767

Appendix B

PF_KEY の仕様

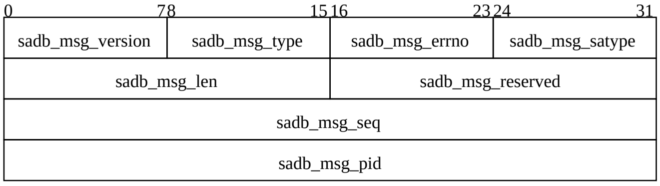
PF_KEY(**PF_KEY key management application programming interface**) verion 2[22] は，BSD Socket をベースとした鍵交換デーモンとの通信プロトコルを定義している．また，PF_KEY は SA を必要とするあらゆるアプリケーションに対して画一的なインターフェイスを提供している．従って，SA を必要とするアプリケーションは，PF_KEY を用いることによって，容易に鍵交換デーモンとに SA の確立を要求する事が可能となる．ただし，PF_KEY は，IETF ipsec working group によって提案されたものではなく，Sun Microsystems の D. L. McDonald によって提案されたものであり，未だ世界標準規格に至っていない．

B.1　メッセージフォーマット

本節では，PF_KEY で定義されているメッセージのフォーマットについて説明する．PF_KEY では必ず Base Message Header がメッセージに含まれ，付加情報は Base Message Header に連なった形で送信される．

B.1.1　Base Message Header Format

PF_KEY では，ベースとなるメッセージヘッダが決まっている．SA に関する情報は，このメッセージヘッダの後に連なる．Bse Message Header のフォーマットを図 B.1に示す．



0 … 7	8 … 15	16 … 23	24 … 31
sadb_msg_version	sadb_msg_type	sadb_msg_errno	sadb_msg_satype
sadb_msg_len		sadb_msg_reserved	
sadb_msg_seq			
sadb_msg_pid			

図 B.1: Base Message Header

sadb_msg_version (1 Byte) PF_KEY のバージョン識別子．バージョン識別子を表 B.1に示す．

sadb_msg_type (1 Byte) メッセージの種類．メッセージの種類を表 B.2に示す．

sadb_msg_errno (1 Byte) エラーが起こった際のエラーコード．

sadb_msg_satype (1 Byte) SA の種類．SA の種類を表 B.3に示す．

sadb_msg_len (2 Bytes) メッセージの全長．

sadb_msg_reserved (2 Bytes) 未使用．0 でなければならない．

sadb_msg_seq (4 Bytes) メッセージの通し番号．sadb_msg_pid と組み合わせることにより，メッセージを識別する．

sadb_msg_pid (4 Bytes) メッセージの送信元やメッセージの送信先．sadb_msg_seq と組み合わせることにより，メッセージを識別する．

表 B.1: PF_KEY Version

Version	Value
PF_KEY_V2	2

表 B.2: PF_KEY Message Type

Message Type	Value
SADB_RESERVED	0
SADB_GETSPI	1
SADB_UPDATE	2
SADB_ADD	3
SADB_DELETE	4
SADB_GET	5
SADB_ACQUIRE	6
SADB_REGISTER	7
SADB_EXPIRE	8
SADB_FLUSH	9
SADB_DUMP	10

表 B.3: PF_KEY SA Type

SA Type	Value
SADB_SATYPE_UNSPEC	0
SADB_SATYPE_AH	1
SADB_SATYPE_ESP	2
SADB_SATYPE_RSVP	3
SADB_SATYPE_OSPFV2	4
SADB_SATYPE_RIPV2	5
SADB_SATYPE_MIP	6

B.1.2 Additional Message Fields

Additional Message Fields は，B.1.3小節から B.1.11小節で定義される PF_KEY メッセージヘッダの共通部分である．Additional Messge Fields を図 B.2に示す．

sadb_ext_len (2 Bytes) 付加情報の長さ．

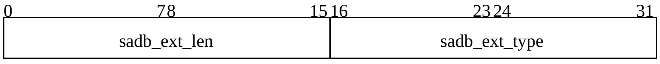


図 B.2: Additional Message Fields

sadb_ext_type (2 Bytes) 現付加情報の種類．付加情報の種類を表 B.4に示す．

表 B.4: PF_KEY Extension Type

Extension Type	Value
SADB_EXT_RESERVED	0
SADB_EXT_SA	1
SADB_EXT_LIFETIME_CURRENT	2
SADB_EXT_LIFETIME_HARD	3
SADB_EXT_LIFETIME_SOFT	4
SADB_EXT_ADDRESS_SRC	5
SADB_EXT_ADDRESS_DST	6
SADB_EXT_ADDRESS_PROXY	7
SADB_EXT_KEY_AUTH	8
SADB_EXT_KEY_ENCRYPT	9
SADB_EXT_IDENTITY_SRC	10
SADB_EXT_IDENTITY_DST	11
SADB_EXT_SENSITIVITY	12
SADB_EXT_PROPOSAL	13
SADB_EXT_SUPPORTED_AUTH	14
SADB_EXT_SUPPORTED_ENCRYPT	15
SADB_EXT_SPIRANGE	16

B.1.3 Association Extension

Association Extension は，SA に関する詳細情報を示すために使用される．Association Extension のフォーマットを図 B.3に示す．

sadb_sa_len (2 Bytes) Association Extension のメッセージ長．

sadb_sa_exttype (2 Bytes) Association Extension を示す識別子 (1)．

sadb_sa_spi (4 Bytes) SA の SPI．

sadb_sa_replay (1 Byte) replay window の大きさ．0 の時には replay window を使用しない事を意味する．

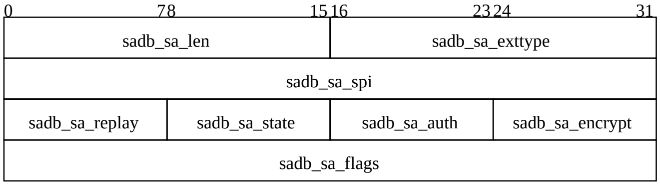


図 B.3: Association Extension

sadb_sa_state (1 Byte) SA の状態を示す識別子．識別子を表 B.5に示す．

sadb_sa_auth (1 Byte) SA で使用される認証アルゴリズムを示す識別子．認証アルゴリズムの識別子を表 B.6に示す．

sadb_sa_encrypt (1 Byte) SA で使用される暗号アルゴリズムを示す識別子．暗号アルゴリズムの識別子を表 B.7に示す．

sadb_sa_flags (4 Bytes) SA の性質を示すフラグ．フラグを表 B.8に示す．

表 B.5: PF_KEY SA Stats

State	Value
SADB_SASTATE_LARVAL	0
SADB_SASTATE_MATURE	1
SADB_SASTATE_DYING	2
SADB_SASTATE_DEAD	3

表 B.6: PF_KEY Authentication Algorithm

Authentication Algorithm	Value
SADB_AALG_NONE	0
SADB_AALG_MD5HMAC	1
SADB_AALG_SAH1HMAC	2

B.1.4 Lifetime Extension

Lifetime Extension は，SA の有効期限に関する情報などを示すために使用される．Lifetime Extension のフォーマットを図 B.4に示す．さらに，Lifetime Extension には以下の 3 種類が定義されている．

CURRENT SA の現在の状態．

表 B.7: PF_KEY Encryption Algorithm

Encryption Algorithm	Value
SADB_EALG_NONE	0
SADB_EALG_DESCBC	1
SADB_EALG_3DESCBC	2

表 B.8: PF_KEY SA Falgs

Flag	Value
SADB_SAFLAGS_PFS	1

HARD SA の厳密な有効期限．

SOFT SA の緩い有効期限．

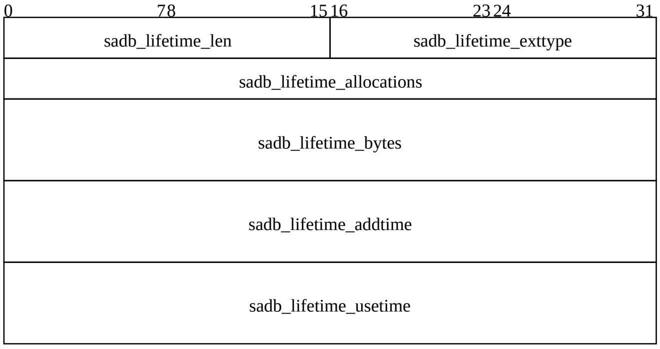


図 B.4: Lifetime Extension

sadb_lifetime_len (2 Bytes) Lifetime Extension のメッセージ長．

sadb_lifetime_exttype (2 Bytes) Lifetime Extension を示す識別子 (2，3，4)．

sadb_lifetime_allocations (4 Bytes) CURRENT の時には，現在 SA が割り当てられているコネクションやエンドポイントやフローの数．また HARD や SOFT の時には，SA が有効期限が切れるまでに割り当てられ得るコネクションやエンドポイントやフローの数．

sadb_lifetime_bytes (8 Bytes) CURRENT の時には，これまでに SA が処理したバイト数．また HARD や SOFT の時には，SA が有効期限が切れるまでに処理し得るバイト数．

sadb_lifetime_addtime (8 Bytes) CURRENT の時には，SA が確立された時間．また HARD や SOFT の時には，SA が確立されてから，有効期限が切れるまでの時間．

sadb_lifetime_usetime (8 Bytes) CURRENT の時には，SA が最初に使用された時間．また HARD や SOFT の時には，SA が最初に使用されてから，有効期限が切れるまでの時間．

B.1.5 Address Extension

Address Extension は，アドレスを示すために使用される．Address Extension のフォーマットを図 B.5 に示す．さらに，Address Extension には以下の 3 種類が定義されている．

SRC 送信元のアドレス．

DST 送信先のアドレス．

PROXY プロキシのアドレス．

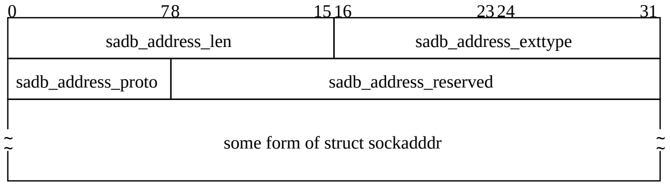


図 B.5: Address Extension

sadb_address_len (2 Bytes) Address Extension のメッセージ長．

sadb_address_exttype (2 Bytes) Address Extension を示す識別子 (5, 6, 7)．

sadb_address_proto (1 Byte) トランスポートプロトコルの識別子．

sadb_address_reserved (3 Bytes) 未使用．0 でなければならない．

B.1.6 Key Extension

Key Extension は，SA で使用される鍵を示すために使用される．Key Extension のフォーマットを図 B.6に示す．さらに，Key Extension には以下の 2 種類が定義されている．

AUTH 認証用の鍵用．

ENCRYPT 暗号用の鍵用．

sadb_key_len (2 Bytes) Key Extension のメッセージ長．

sadb_key_exttype (2 Bytes) Key Extension を示す識別子 (8，9)．

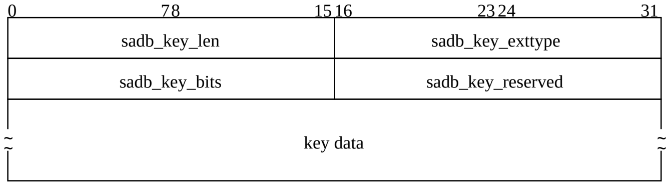


図 B.6: Key Extension

sadb_key_bits (2 Bytes) key data の長さ．

sadb_key_reserved (2 Bytes) 未使用．0 でなければならない．

B.1.7 Identity Extension

Identity Extension は，エンドポイントの身元情報を示すために使用される．Identity Extension のフォーマットを図 B.7に示す．さらに，Identity Extension には以下の 2 種類が定義されている．

SRC 送信元の身元情報．

DST 送信先の身元情報．

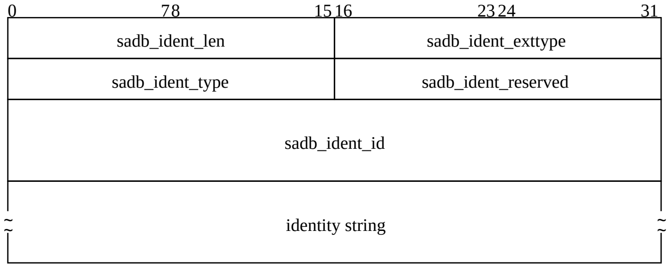


図 B.7: Identity Extension

sadb_ident_len (2 Bytes) Identity Extension のメッセージ長．

sadb_ident_exttype (2 Bytes) Identity Extension を示す識別子 (10，11)．

sadb_ident_type (2 Bytes) 身元情報の種類を示す識別子．識別子を表 B.9に示す．

sadb_ident_reserved (2 Bytes) 未使用．0 でなければならない．

sadb_ident_id (8 Bytes) identity string を補助するために使用される身元情報．

表 B.9: PF_KEY Identity Type

Identity Type	Value
SADB_IDENTTYPE_RESERVED	0
SADB_IDENTTYPE_PREFIX	1
SADB_IDENTTYPE_FQDN	2
SADB_IDENTTYPE_USERFQDN	3

B.1.8 Sensitivity Extension

Senxitivity Extension は，SA のセキュリティレベルを示すために使用される．Sensitivity Extension のフォーマットを図 B.8に示す．

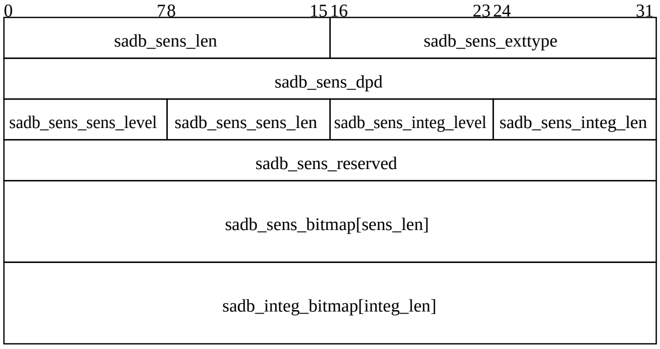


図 B.8: Sensitivity Extension

sadb_sens_len (2 Bytes) Sensitivity Extension のメッセージ長．

sadb_sens_exttype (2 Bytes) Sensitivity Extension を示す識別子 (12)．

sadb_sens_dpd (4 Bytes) 保護領域．

sadb_sens_sens_level (1 Byte) センシティビティレベル．

sadb_sens_sens_len (1 Byte) sadb_sens_bitmap の長さ．

sadb_sens_integ_level (1 Byte) インテグリティレベル．

sadb_sens_integ_len (1 Byte) sadb_integ_bitmap の長さ．

sadb_sens_reserved (4 Bytes) 未使用．0 でなければならない．

B.1.9 Proposal Extension

Proposal Extension は，SA の確立環境を示すために使用される．Proposal Extension を図 B.9に示す．

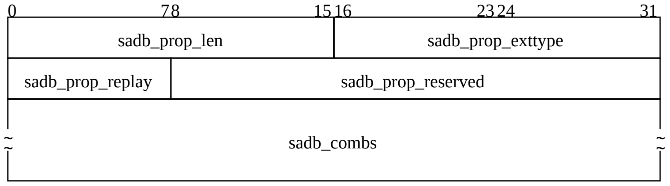


図 B.9: Proposal Extension

sadb_prop_len (2 Bytes) Proposal Extension のメッセージ長．

sadb_prop_exttype (2 Bytes) Proposal Extension を示す識別子 (13)．

sadb_prop_replay (1 Byte) replay window の大きさ．0 の時には，replay window を使用しない事を意味する．

sadb_prop_reserved (3 Bytes) 未使用．0 でなければならない．

sadb_combs (任意長) SA の確立環境を示す構造体．図 B.10に示す構造体からなり，SA の確立環境として好ましい順に並べられる．

sadb_comb_auth (1 Byte) SA で使用される認証アルゴリズムを示す識別子．認証アルゴリズムの識別子を表 B.6に示す．

sadb_comb_encrypt (1 Byte) SA で使用される暗号アルゴリズムを示す識別子．暗号アルゴリズムの識別子を表 B.7に示す．

sadb_comb_flags (2 Bytes) SA の性質を示すフラグ．フラグを表 B.8に示す．

sadb_comb_auth_minbits (2 Bytes) 認証に用いる鍵長の最短．

sadb_comb_auth_maxbits (2 Bytes) 認証に用いる鍵長の最長．

sadb_comb_encrypt_minbits (2 Bytes) 暗号に用いる鍵長の最短．

sadb_comb_encrypt_maxbits (2 Bytes) 暗号に用いる鍵長の最長．

sadb_comb_reserved (4 Bytes) 未使用．0 でなければならない．

<table>
<tr><th>0 7</th><th>8 15</th><th>16 23</th><th>24 31</th></tr>
<tr><td>sadb_comb_auth</td><td>sadb_comb_encrypt</td><td colspan="2">sadb_comb_flags</td></tr>
<tr><td colspan="2">sadb_comb_auth_minbits</td><td colspan="2">sadb_comb_auth_maxbits</td></tr>
<tr><td colspan="2">sadb_comb_encrypt_minbits</td><td colspan="2">sadb_comb_encrypt_maxbits</td></tr>
<tr><td colspan="4">sadb_comb_reserved</td></tr>
<tr><td colspan="4">sadb_comb_soft_allocations</td></tr>
<tr><td colspan="4">sadb_comb_hard_allocations</td></tr>
<tr><td colspan="4">sadb_comb_soft_bytes</td></tr>
<tr><td colspan="4">sadb_comb_hard_bytes</td></tr>
<tr><td colspan="4">sadb_comb_soft_addtime</td></tr>
<tr><td colspan="4">sadb_comb_hard_addtime</td></tr>
<tr><td colspan="4">sadb_comb_soft_usetime</td></tr>
<tr><td colspan="4">sadb_comb_hard_usetime</td></tr>
</table>

図 B.10: Parameter Combinations

sadb_comb_soft_allocations (4 Bytes) SA が有効期限が切れるまでに割り当てられ得るコネクションやエンドポイントやフローの数の緩い制限．

sadb_comb_hard_allocations (4 Bytes) SA が有効期限が切れるまでに割り当てられ得るコネクションやエンドポイントやフローの数の厳密な制限．

sadb_comb_soft_bytes (8 Bytes) SA が有効期限が切れるまでに処理し得るバイト数の緩い制限．

sadb_comb_hard_bytes (8 Bytes) SA が有効期限が切れるまでに処理し得るバイト数の厳密な制限．

sadb_comb_soft_addtime (8 Bytes) SA が確立されてから，有効期限が切れるまでの緩い時間制限．

sadb_comb_hard_addtime (8 Bytes) SA が確立されてから，有効期限が切れるまでの厳密な時間制限．

sadb_comb_soft_usetime (8 Bytes) SA が最初に使用されてから，有効期限が切れるまでの緩い時間制限．

sadb_comb_hard_usetime (8 Bytes) SA が最初に使用されてから，有効期限が切れるまでの厳密な時間制限．

B.1.10 Supported Algorithms Extension

Supported Algorithms Extension は，システムがサポートしているアルゴリズム全てを示すために使用される．Supported Algorithms Extension のフォーマットを図 B.11に示す．さらに，Supported Algorithms Extension には以下の 2 種類が定義されている．

AUTH 認証アルゴリズム用．

ENCRYPT 暗号アルゴリズム用．

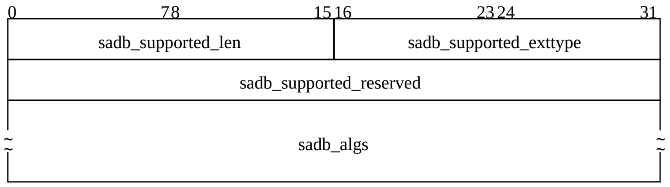


図 B.11: Supported Algorithms Extension

sadb_supported_len (2 Bytes) Supported Algorithms Extension のメッセージ長．

sadb_supported_exttype (2 Bytes) Supported Algorithms Extension を示す識別子 (14，15)．

sadb_supported_reserved (4 Bytes) 未使用．0 でなければならない．

sadb_algs (任意長) システムがサポートしているアルゴリズムを示す構造体．構造体を図 B.12に示す．

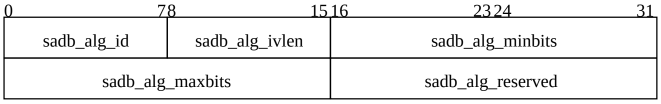


図 B.12: Algorithm Description

sadb_alg_id (1 Byte) アルゴリズムを示す識別子．識別子を表 B.6，表 B.7に示す．

sadb_alg_ivlen (1 Byte) IV (Initialization Vector) の長さ．

sadb_alg_minbits (2 Bytes) 鍵長の最短．

sadb_alg_maxbits (2 Bytes) 鍵長の最長．

sadb_alg_reserved (2 Bytes) 未使用．0 でなければならない．

B.1.11 SPI Range Extension

SPI Range Extension は，SPI 値の許容範囲を示すために使用される．SPI Range Extension のフォーマットを図 B.13に示す．

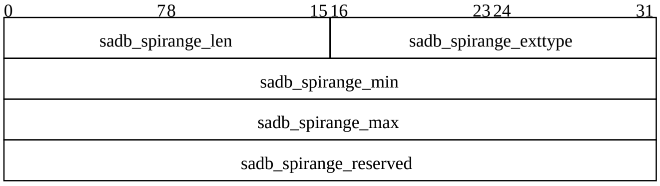


図 B.13: SPI Range Extension

sadb_spirange_len (2 Bytes) SPI Range Extension のメッセージ長．

sadb_spirange_exttype (2 Bytes) SPI Range Extension の識別子 (16)．

sadb_spirange_min (4 Bytes) SPI の最小許容値．

sadb_spirange_max (4 Bytes) SPI の最大許容値．

sadb_spirange_reserved (4 Bytes) 未使用．0 でなければならない．

B.2 メッセージタイプ

本節では，PF_KEY で定義されているメッセージタイプとその振舞いについて説明する．その前に，以下で使用する表記法について説明する．

- “base” は，Base Message Header を表す．
- “SA” は，Association Extension を表す．
- “SA(*)” は，Association Extension の sadb_sa_spi 部だけに値が入っている事を表す．
- “lifetime” は，Lifetime Extension を表す．
- “lifetime” の後に括弧で括られて表示される “H”，“S”，“C” は，それぞれ HARD，SOFT，CURRENT を意味する．
- “address” は，Address Extension を表す．
- “address” の後に括弧で括られて表示される “S”，“D”，“P” は，それぞれ SRC，DST，PROXY を意味する．

- “key” は，Key Extension を表す．
- “key” の後に括弧で括られて表示される “A”，“E” は，それぞれ AUTH，ENCRYPT を意味する．
- “identity” は，Identity Extension を表す．
- “identity” の後に括弧で括られて表示される “S”，“D” は，それぞれ SRC，DST を意味する．
- “sensitivity” は，Sensitivity Extension を表す．
- “proposal” は，Proposal Extension を表す．
- “supported” は，Supported Algorithms Extension を表す．
- “supported” の後に括弧で括られて表示される “A”，“E” は，それぞれ AUTH，ENCRYPT を意味する．
- “spirange” は，SPI Range Extension を表す．

また，エラー発生時には Base Message Header のみが，sadb_msg_errorno に値を含んで送信される．エラーメッセージは，各メッセージタイプの説明時に挙げる物や，以下に挙げる物の中から対応する 1 つが送信される．

EINVAL 不適当なメッセージが送信された時．

ENOMEM メモリが足りない時．

ENOBUFS バッファが足りない時．

EMSGSIZ メッセージの長さが不当に長い時．

B.2.1 SADB_GETSPI

SADB_GETSPI は，プロセスが SPI を使用するに当たって，カーネルから得るために使用される．SADB_GETSPI の振舞いを以下に示す．

user process → kernel

<base, address, SPI, range>

kernel → all listening processes

<base, SA(*), address(SD)>

Error の種類

EEXIST 要求した SPI が使用不可もしくは既に使用されている時

B.2.2 SADB_UPDATE

SADB_UPDATE は，プロセスが既に存在している SA の情報を更新するために使用される．SADB_UPDATE の振舞いを以下に示す．

user process → kernel

<base, SA, (lifetime(HSC),) address(SD), (address(P),) key(AE), (identity(SD),) (sensitivity)>

kernel → all listening processes

<base, SA, (lifetime(HSC),) address(SD), (address(P),) (identity(SD),) (sensitivity)>

Error の種類

ESRCH 更新対象の SPI が存在しない．

EINVAL 更新内容が不適当．

EACCES 更新要求元が不正である．

B.2.3 SADB_ADD

SADB_ADD は，プロセスが SA を登録するために使用される．SADB_ADD の振舞いを以下に示す．

user process → kernel

<base, SA, (lifetime(HS),) address(SD), (address(P),) key(AE), (identity(SD),) (sensitivity)>

kernel → all listening processes

<base, SA, (lifetime(HS),) address(SD), (identity(SD),) (sensitivity)>

Error の種類

EEXIST 登録しようとしている SA が既に存在している．

EINVAL 登録内容が不適当．

B.2.4 SADB_DELETE

SADB_DELETE は，プロセスがカーネル中の SA Database から特定の SA を削除するために使用される．SADB_DELETE の振舞いを以下に示す．

user process → kernel

<base, SA(*), address(SD)>

kernel → all listening processes

<base, SA(*), address(SD)>

B.2.5 SADB_GET

SADB_GET は，プロセスがカーネル中の SA Database から特定の SA を検索するために使用される．SADB_GET の振舞いを以下に示す．

user prcess → kernel

<base, SA(*), address(SD)>

kernel → SADB_GET を送信した process

<base, SA, (lifetime(HSC),) address(SD), (address(P),) key(AE), (identity(SD),) (sensitivity)>

Error の種類

ESRCH 目的の SA が存在していない．

B.2.6 SADB_ACQUIRE

SADB_ACQUIRE は，カーネルが SADB_REGISTER メッセージによってカーネルに登録されている鍵交換デーモンに対して SA の確立要求を送信するために使用される．また，ユーザプロセスが，カーネルを経由して鍵交換デーモンに SA の確立要求を送信するためにも使用される．SADB_ACQUIRE の振舞いを以下に示す訳だが，SADB_ACQUIRE には 3 種類の振舞いがある．

カーネルが鍵交換デーモンに SA の確立を要求

kernel → 鍵交換デーモン

<base, address(SD), (identity(SD),) (sensitivity,) proposal>

鍵交換デーモンがカーネルに **SA** 確立の失敗を通知

鍵交換デーモン → **kernel**

<base>

ユーザプロセスが，カーネルを経由して鍵交換デーモンに **SA** の確立を要求

user process → **kernel**

<base, address(SD), (identity(SD),) (sensitivity,) proposal>

kernel → 鍵交換デーモン

<base, address(SD), (identity(SD),) (sensitivity,) proposal>

Error の種類

EINVAL 不正な要求である．

EPROTONOSUPPORT 適当な鍵交換デーモンが登録されていないために，SA の確立が不可能．

B.2.7 SADB_REGISTER

SADB_REGISTER は，鍵交換デーモンが存在をカーネルに登録するために使用される．SADB_REGISTER の振舞いを以下に示す．

鍵交換デーモン → **kernel**

<base>

kernel → 登録されている全鍵交換デーモン

<base, supported>

B.2.8 SADB_EXPIRE

SADB_EXPIRE は，カーネルが SA の有効期限切れを通知するために使用される．SADB_EXPIRE の振舞いを以下に示す．

kernel → 登録されている全鍵交換デーモン

<base, SA, lifetime(HSC), address(SD)>

B.2.9 SADB_FLUSH

SADB_FLUSH は，プロセスが sadb_msg_satype で指定した種類の SA をカーネル中の SA Database から全て消去するために使用される．

user process → kernel

<base>

kernle → all listening processes

<base>

B.2.10 SADB_DUMP

SADB_DUMP は，プロセスがカーネル中の SA Database に登録されている SA のダンプを取るために使用される．アプリケーションの開発を補助するためのメッセージであり，普段は使用されない．SADB_DUMP の振舞いを以下に示す．

user process → kernel

<base>

kernel → SADB_DUMP を送信した process

<base, SA, (lifetime(HSC),) address(SD), (address(P),) key(AE), (identity(SD),) (sensitivity)>

B.3 実例

いかに PF_KEY が使用されるかを実例を挙げて説明する．ここでは，IPsec を例に説明する．IPsec のシステム構成を図 B.14に示しておく．

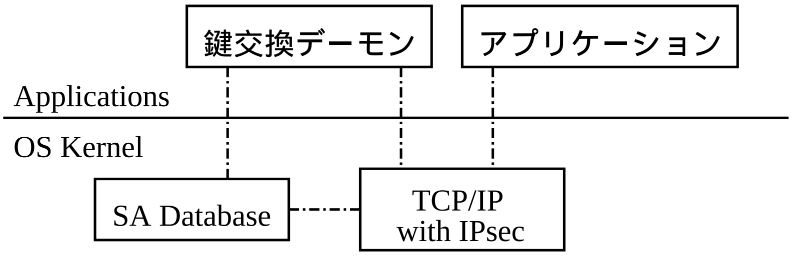


図 B.14: PF_KEY 実例 (IPsec)

1. 鍵交換デーモンは起動すると，カーネルに鍵交換デーモンの存在と鍵交換デーモンが確立する事のできるSAの種類を登録する．カーネルからの返答として，カーネルがサポートしているアルゴリズムのリストが返される．

鍵交換デーモン → カーネル	SADB_REGISTER for ESP
カーネル → 鍵交換デーモン	SADB_REGISTER for ESP, Supported Algorithms
鍵交換デーモン → カーネル	SADB_REGISTER for AH
カーネル → 鍵交換デーモン	SADB_REGISTER for AH, Supported Algorithms

2. アプリケーションがIPsecを用いて通信を行おうとした場合に，通信先とのSAが確立されていない時，カーネルから鍵交換デーモンにSAの確立要求が送信される．今，仮にAHのSAのみが必要であるとする．

カーネル → 鍵交換デーモン	SADB_ACQUIRE for AH, address, identity, sensitivity, proposals

3. SADB_ACQUIREを受信した鍵交換デーモンは，SAの確立を開始する前にカーネルに対して確立すべきSAのSPIを尋ねる．この要求に対して，カーネルはSPIの値をカーネルに接続している全てのプロセスに対して送信する．この時，SA DatabaseにはSPIに対するSAテーブルが割り当てられる．

鍵交換デーモン → カーネル	SADB_GETSPI for AH, address, SPI range
カーネル → 全プロセス	SADB_GETSPI for AH, association, address

4. SAを確立した鍵交換デーモンは，カーネルにSAを登録する．ここで登録されるSAは，自ホストから通信先のホストへのSAである．つまり，先ほどのSADB_GETSPIメッセージによってSA DatabaseにSAテーブルが割り当てられている．

鍵交換デーモン → カーネル	SADB_UPDATE for AH, association, address, keys, etc.
カーネル → 全プロセス	SADB_UPDATE for AH, association, address, etc.

ただし，鍵交換デーモンがSAの確立に失敗した時には，カーネルにエラーが伝えられる．

鍵交換デーモン → カーネル	SADB_UPDATE for AH, association (with an error)
カーネル → 全プロセス	SADB_UPDATE for AH, association (with the error)

5. 通信先のホストから自ホストへの SA を SA Database に登録する．

鍵交換デーモン → カーネル	SADB_ADD for AH, association, address, keys, etc.
カーネル → 全プロセス	SADB_ADD for AH, association, address, etc.

6. SA Database に登録されている SA の有効期限が切れるとカーネルは，全プロセスに対してその旨を送信する．

カーネル → 全プロセス	SADB_EXPIRE for AH, association, address, Hard or Soft or Current, etc.

Appendix C

Simple API の仕様

[21] において，IPsec を用いたソケットを利用するための API が提供されている．Simple API を用いる事によって，ソケット毎に IPsec の種類の指定やセキュリティレベルの設定を行う事ができる．ただし，Simple API は，IETF ipsec working group によって提案されたものではなく，Sun Microsystems の D. L. McDonald によって提案されたものであり，未だ標準仕様に至っていない．

C.1 IPsec の種類

Simple API では以下の 3 種類に対してセキュリティレベルの設定を行う事が可能である．

- AH
- ESP(トランスポートモード)
- ESP(トンネルモード)

C.2 セキュリティレベル

Simple API では以下の 5 つのセキュリティレベルに設定する事が可能である．

None (Default) 送信パケットに対して一切セキュリティ処理を施さない．またセキュリティ処理が施されていようと施されていまいといかなるパケットも受信する．

If available SA が確立されており利用可能であれば，送信パケットに対してセキュリティ処理を施す．SA が利用可能でなければ，None の時と同様である．ただし，SA が利用可能でない時には，SA の確立を要求する．

Use SA が確立されており，利用可能でなければパケットを送信しない．SA が利用できない状態の時には，SA の確立を待ってからセキュリティ処理を施して送信する．受信パケットに関しては，セキュリティ処理が施されていようと施されていまいといかなるパケットも受信する．

Require SA が確立されており，利用可能でなければパケットの送信も受信も行わない．

Require unique Require と同様に SA が確立されており，利用可能でなければパケットの送信も受信も行わない．しかも，送信に利用した SA は，そのセッションでのみ利用可能でなければならない．つまり，セッション毎に送信に利用される SA は異なっていなければならない．

C.3 デフォルトのセキュリティレベル

Simple API を用いる事によって，ソケット毎に IPsec の種類を指定したり，セキュリティレベルを設定したりすることが可能である．しかし，OS はグローバルに設定されたセキュリティレベルや，ユーザがソケットに設定したセキュリティレベルの中で最も安全な設定に従おうとする．つまり，Simple API を利用しても，グローバルに設定されたセキュリティレベルより低いセキュリティレベルに設定する事ができない．

しかし，鍵交換デーモンなど幾つかのアプリケーションは，自由にセキュリティレベルを設定できなくては困る．そこで，Simple API では，幾つかの特殊なアプリケーションのために，全てのセキュリティ処理を省略する **Bypass** というセキュリティレベルが用意されている．

Bypass 全てのセキュリティ処理を無視する．このセキュリティレベルは特殊なアプリケーションに対してのみ有効でなければならない．

C.4 Simple API の書式

```
#include <netinet/in.h>

...

int s, error;
int seclevel;

s = socket(AF_INET{,6}, SOCK_{DGRAM,STREAM}, 0);

seclevel = <security level>;
error = setsockopt(s, IPPROTO_IP{,V6}, <category>, seclevel,
  sizeof(int));
```

<category>には，IPsec の種類を示す以下の値が入る．

```
IP{,V6}_AUTH_LEVEL          /* AH */
IP{,V6}_ESP_TRANS_LEVEL     /* ESP(トランスポートモード) */
IP{,V6}_ESP_NETWORK_LEVEL   /* ESP(トンネルモード) */
```

<security level>には，セキュリティレベルを示す以下の値が入る．

```
IPSEC_LEVEL_BYPASS         /* Bypass */
IPSEC_LEVEL_NONE           /* None (Default) */
IPSEC_LEVEL_DEFAULT        /* 同上 */
IPSEC_LEVEL_AVAIL          /* If available */
IPSEC_LEVEL_USE            /* Use */
IPSEC_LEVEL_REAUIRE        /* Require */
IPSEC_LEVEL_UNIQUE         /* Require unique */
```

参考文献

[1] 伊藤　純一郎，祢宜　知孝．IPv4/v6 における IP security の実装．日本ソフトウェア科学会第 *14* 回大会，169–172，September　1997．

[2] 濱野　淳史．マルチキャスト通信のための鍵交換デーモンの拡張. 卒業論文，慶應義塾大学理工学部電気工学科，神奈川県横浜市港北区日吉 3–14–1，February　1998．

[3] ISC and RSA Announce Security Extensions Implementation for DNS. *http://www.isc.org/rsa_announcement.html.*

[4] R. Atkinson. *IP Authentication Header.* Technical Report, *Request for Comments 1826*, August 1995.

[5] R. Atkinson. *IP Encapsulating Security Payload (ESP).* Technical Report, *Request for Comments 1827*, August 1995.

[6] R. Atkinson. *Security Architecture for the Internet Protocol.* Technical Report, *Request for Comments 1825*, August 1995.

[7] Ashar Aziz, Tom Markson, and Hemma Prafullchandra. *Simple Key–Management For Internet Protocols (SKIP) <draft–ietf–ipsec–skip–07.txt>.* Technical Report, *IPSEC Working Group INTERNET–DRAFT*, August 1996. work in progress.

[8] Ashar Aziz and Martin Patterson. Simple Key–Management for Internet Protocols(SKIP). *Proceedings INET '95*, 1995.

[9] D. Carrel D. Harkins. *The resolution of ISAKMP with Oakley <draft–ietf–ipsec–isakmp–oakley–04.txt>.* Technical Report, *IPSEC Working Group INTERNET–DRAFT*, July 1997. work in progress.

[10] Tim Dierks. *The TLS Protocol Version 1.0 <Draft–ietf–tls–protocol–03.txt>.* Technical Report, *INTERNET–DRAFT*, May 1997.

[11] Simson Garfinkel. *PGP: PRETTY GOOD PRIVACY.* O'Reilly & Associates, 1995.

[12] Torbjorn Granlund and TMG Datakonsult. *The GNU Multiple Precision Arithmetic Library.* Free Software Foundation, 59 Temple Place – Suite 330 Boston, Ma 02111–1307, USA, 2.0.2 edition, June 1996.

[13] Christian Huitema. セキュリティ機能の導入. In 村井　純，編集者, *IPv6* 次世代インターネット・プロトコル, chapter 5, pages 99–122, 株式会社プレンティスホール出版, 東京都新宿区西新宿 8–14–24 西新宿 KF ビル 602, January 1997.

[14] P. Karn, P. Metzger, and W. Simpson. *The ESP DES–CBC Transform.* Technical Report, *Request for Comments 1829*, August 1995.

[15] P Karn and W A Simpson. *Photuris: Session Key Management Protocol <draft–simpson-photuris–13.txt>.* Technical Report, *Network Working Group Internet Draft*, July 1997. work in progress.

[16] Charlie Kaufman, Raida Perlman, and Mike Speciner. *NETWORK SECURITY – PRIVATE Communication in a PUBLIC World*, chapter 10 KERBEROS V4, pages 265 – 294. *Prentice Hall PTR*, 113 Sylvan Avenue Englewood Cliffs, NJ07632, 1995.

[17] Stephen Kent and Randall Atkinson. *IP Authentication Header <draft–ietf–ipsec–auth–header–01.txt>.* Technical Report, *Network Working Group Internet Draft*, July 1997. work in progress.

[18] Stephen Kent and Randall Atkinson. *IP Encapsulating Security Payload <draft–ietf–ipsec–esp–v2–01.txt>.* Technical Report, *Network Working Group Internet Draft*, October 1997. work in progress.

[19] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed–Hashing for Message Authentication.* Technical Report, *Request for Comments 2104*, February 1997.

[20] Douglas Maughan, Mark Schertler, Mark Schneider, and Jeff Turner. *Internet Security Association and Key Managament Protocol(ISAKMP) <draft–ietf–ipsec–isakmp–08.txt>.* Technical Report, *IPSEC Working Group INTERNET–DRAFT*, July 1997. work in progress.

[21] D. L. McDonald. *A Simple IP Security API Extension to BSD Sockets <draft–mcdonald–simple–ipsec–api–01.txt>.* Technical Report, *Network Working Group Internet Draft*, March 1997. work in progress.

[22] D. L. McDonald, C. W. Metz, and B. G. Phan. *PF_KEY Key Management API, Version 2 <draft–mcdonald–pf–key–v2–02.txt>.* Technical Report, *Network Working Group Internet Draft*, April 1997. work in progress.

[23] P. Metzger and W. Simpson. *IP Authentication using Keyed MD5.* Technical Report, *Request for Comments 1828*, August 1995.

[24] P. Metzger and W. Simpson. *IP Authentication using Keyed SHA.* Technical Report, *Request for Comments 1852*, September 1995.

[25] M. Ohta. *Simple Secure DNS <draft–ohta–simple–dns–02.txt>.* Technical Report, *INTERNET DRAFT*, November 1996. work in progress.

[26] H. K. Orman. *The OAKLEY Key Determination Protocol <draft–ietf–ipsec–oakley–01.txt>.* Technical Report, *IPSEC Working Group INTERNET–DRAFT*, May 1996. work in progress.

[27] Derrell Piper. *The Internet IP Security Domain of Interpretation for ISAKMP <draft–ietf–ipsec–ipsec–doi–02.txt>.* Technical Report, *Network Working Group Internet Draft*, February 1997. work in progress.

[28] Bruce Schneier. *Applied Cryptography – Protocols, Algorithms, and Source Code in C –*, chapter 22.1 Diffie–Hellman, pages 513 – 516. *John Wiley & Sons, Inc.*, 605 Third Avenue, New York, N.Y. 10158–0012, second edition, 1996.

[29] Bruce Schneier. *Applied Cryptography – Protocols, Algorithms, and Source Code in C —*, chapter 20 Public–Key Digital Signature Algorithms, pages 483 – 502. *John Wiley & Sons, Inc.*, 605 Third Avenue, New York, N.Y. 10158–0012, second edition, 1996.

[30] William Stallings. *NETWORK AND INTERNETWORK SECURITY – PRINCIPLES AND PRACTICE –*, chapter 8.3 Diffie–Hellman Key Exchange, pages 340 – 343. *PRENTICE HALL*, Englewood Cliffs, New Jersey 07632, international edition, 1995.

[31] William Stallings. *NETWORK AND INTERNETWORK SECURITY – PRINCIPLES AND PRACTICE –*, chapter 8.4 Digital Signature Standard, pages 343 – 346. *PRENTICE HALL*, Englewood Cliffs, New Jersey 07632, 1995.

[32] William Stallings. *NETWORK AND INTERNETWORK SECURITY – PRINCIPLES AND PRACTICE –*, chapter 1.2 Security Attacks, pages 7 – 10. *PRENTICE HALL*, Englewood Cliffs, New Jersey 07632, 1995.

[33] R. Thayer, N. Doraswamy, and R. Glenn. *IP security Document Roadmap <draft–ietf–ipsec–doc–readmap–01.txt>.* Technical Report, *Network Working Group Internet Draft*, July 1997. work in progress.

[34] T. Ylonen. *Ths SSH (Secure Shell) Remote Login Protocol <draft–ylonen–ssh–protocol–00.txt>*. Technical Report, *INTERNET–DRAFT*, November 1995. work in progress.