卒業論文

題目

暗号化アルゴリズムの動的選択に関する研究

指導教授

所 真理雄 教授

慶應義塾大学

理工学部 電気工学科

1995年度

69208149

祢宜 知孝

論文要旨

学科	電気工学	学籍番号	69208149	氏名	祢宜 知孝

(論文題目)

暗号化アルゴリズムの動的選択に関する研究

現在の暗号通信方式では, 受信者は暗号化に使用されたアルゴリズムに対応した復号化のアルゴリズムを必ず所持している必要がある. データの送信者・受信者はあらかじめ暗号通信に用いる暗号化アルゴリズムを決める必要があるので, そのアルゴリズム以外の暗号化アルゴリズムを用いて暗号通信を行うことは不可能であった. すなわち, 従来の暗号通信は, 送信者が, 送信時に受信者が持っていない暗号化アルゴリズムも含めて任意に選択できるような柔軟性を有していなかった.

以上の問題を解決するために, 本論文では送信者が受信者の保持していない暗号化アルゴリズムを用いて暗号文を送信した際も, 受信者による復号が可能なシステムの構築方法を提案する. このシステムでは, はじめに受信者が所持している暗号化アルゴリズムに関する情報を入手するために, ネゴシエーションを行い, 暗号化したデータと, 復号化アルゴリズムをまとめて送信する方法を採択している.

本論文では, 電子メール配送システムに, 提案するシステムを組み込む. インターフェイスとして Emacs 上で動作するメッセージインターフェイスを選択し, 本論文で提案するシステムに対応するよう拡張する.

また, 現在のマルチメディア電子メールの標準規格である MIME には, 暗号化アルゴリズムと暗号化されたデータが一つになった形態が定義されていないので, 本論文で新しい形態を提案する.

慶應義塾大学理工学部

◎ 内容の要旨は約600字

Summary of Bachelor's Thesis

Course	Student No.	SURNAME, Firstname
Electrical Engineering	69208149	Tomonori Negi

Title

Dynamic selection of encryption algorithm

Cryptographic message exchange used today assumes that the sender and the receiver agree on the cipher algorithm they use, and they have proper encryption/decryption program. However, it is questionable that those two parties can agree on the algorithm beforehand. Before, there is no way to select and use the cipher algorithm receiver does not have.

In this paper, we propose a protocol and system for enabling dynamic selection of the cipher algorithm.

The mailers used by the sender and the receiver automatically negotiate on the cipher algorithm they use, using the negotiation protocol proposed in the paper. The ciphertext generated by the sender is sent to the receiver together with the proper decryption program. In proposed way it is not necessery for both parties to have a program code for encryption/decription beforehand, and negotiation will take place automatically.

The paper shows the implementation based on our model *EMAAS* on the top of e-mail system. The system is implemented on Emacs lisp language provided by Emacs editor, as an extention to a mail user interface, Mew. It is confirmed to work sufficiently efficient.

The current Internet standard for Multipurpose Internet Mail Extensions (MIME) does not have a spec for encapsulating program code in a message together with the ciphertext. Therefore, in this paper we propose a new content type of MIME for it, too.

Faculty of Science and Technology, KEIO UNIVERSITY

目 次

図目次

表 目 次

第 1 章

序文

1.1 はじめに

日本において, 1995 年はインターネット元年と言われる位, インターネットが注目され, 急速に普及した年であった. 新聞の紙面にインターネットと言う言葉が幾つも踊り, テレビのニュースでも当り前のように使われるようになった.

インターネットは, 誰でも簡単に利用する事ができる, 世界中に張り巡らされたコンピュータネットワークである. ユーザはインターネットを利用して, 電子メールの交換や, データベースへのアクセス等を行っている. インターネット上で商業活動を始める人も出て来た.

現在のネットワークにはセキュリティの問題が存在する. つまりネットワーク上では, 情報の盗聴や, 情報の改竄, なりすまし等ネットワークを通じて不正が行われることがあり, これらに対して防衛策を講じなくてなくてはならない. 不正行為としては次のようなものがあげられる.

- 情報の盗聴 : 通信が行われている経路上で, 通信の内容を盗み見ること.
- トラフィック解析 : 情報がどこからどこへ流れているのか, また, どのような種類の情報が流れているのかを解析すること.
- 情報の改竄 : 通信が行われている経路上で, 通信内容を故意に変更すること.
- 他の計算機へのなりすまし : 他の計算機の IP アドレスを用いて他の計算機になりすますこと.
- 他の計算機・ネットワークへの不要なメッセージ発信による運用妨害 : 不必要なメッセージを送り, ネットワークや計算機に過剰な負荷を与えること.
- 計算機への不正なアクセス : 本来はアクセスする権限がない計算機へアクセスし, 資源を利用したり, 情報を入手すること.
- ネットワークの制御情報の改竄・不正発信 : ネットワークを制御している情報を改竄, あるいは不正に発信し, 正常な運営を妨害すること.

このようなネットワークの不正行為に対する防衛手段として, さまざまな方法が提供されている. かつては, これらの防衛手段は, 軍隊などの特殊な組織だけのものであったが , ネットワークが世界中に広まった今, 決して軍隊などの特殊な組織だけのものではなくなってきている.

具体的なセキュリティ向上の方法として, 情報の盗聴に対してはデータの暗号化, 情報の改竄に対しては一方向ハッシュ関数を用いた電子署名, なりすましに対しては認証などが考えられている. 暗号化の方法としては DES, IDEA 等多数の方法が提供されている. また電子署名の方法, 認証の方法に対しても多数の方法が提供されている. これを組み合わせたシステム PGP, PEM 等も提供されている.

暗号化の考えは, 古くは古代エジプトのパロ時代から存在する. ジュリアス・シーザが文字を循環シフトして通信をした記録が残っている. この循環シフトをする暗号化をジュリアス・シーザにちなんでシーザ暗号と呼んでいる [crypt]. その後, 電子計算機が誕生するまでは, いくつかの比較的簡単な暗号化方法が使われてきた. 単文字換字式暗号, 多表式暗号, 転置式暗号がそれである.

しかし以上のような簡単な暗号化は, もはや役にたたなくなってしまった. 現在では, 電子計算機の計算力に頼り, 電子計算機が誕生するまでに使われた暗号方法よりはるかに計算量の多い暗号方法が数多く存在している.

しかし, 現在さまざまなセキュリティの方法が提供されているにもかかわらず, ユーザはこれらをあまり利用せず, データをそのままネットワーク上に流している. 現状を放置すれば, 必ずトラブルが表面化してくると考えられる. 現にトラブルは生じているのである. ただ, ユーザはこれらのトラブルを特殊な場合のように考え, 自分にはトラブルが降り掛かって来ないと思っている.

不正行為に対するユーザの認識が甘いという点もあるが, それ以上に, 現在提供されているセキュリティ確保の方法は, (1) 不透明で分かりにくい, (2) 複雑な操作が必要であり使いにくい等の問題が存在する. このためユーザの多くはセキュリティ対策をしていない.

また, 提供されているセキュリティの方法を利用しているユーザは, もっと柔軟性に富んだ使いやすい方法を要求している.

そこで, 本論文では, 軍隊など特殊な組織のためのセキュリティシステムではなく, 一般ユーザを対象とした柔軟性に富んだ誰もが簡単に使えるセキュリティシステムを提案し, これを実装する.

1.2 本論文の構成

本論文では, 2 章で既存の暗号化アルゴリズム (DES や SAFER 等) 及び既存のセキュリティシステム (PGP や PEM 等) について詳しく述べる. そしてこれらの問題点を指摘し, まとめる.

3 章では, 2 章で挙げた問題点を再度まとめあげ, 各問題点に対する解決方法を述べる.

そして, これらの問題点を解決すべく, 新たなシステムを提案し, 新たなシステムのモデルについて説明する.

4 章では, 本論文で提案するシステムの実装方針, 実装方法について述べる.

そして, 5 章でシステムの評価 ・検討を行い, 6 章で今後の課題について述べる. 最後 7 章で結論を述べる.

第 2 章

背景

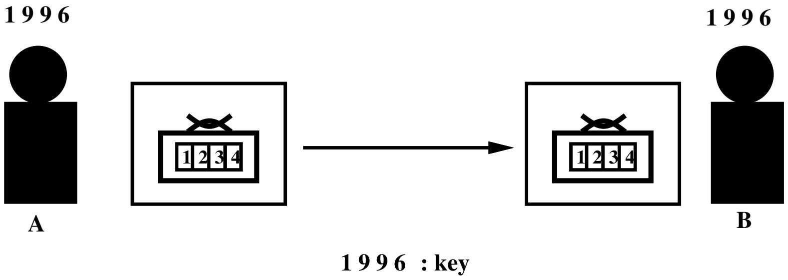


図 2.1: 秘密鍵暗号系

本章では, 暗号化アルゴリズム, 暗号化アルゴリズムを用いた暗号システム, Mew, MIME について概説し, 暗号化アルゴリズム ・暗号システムの問題点を述べる.

2.1 暗号化

暗号化のアルゴリズムは以下の 2 種類に分類される.

秘密鍵暗号系

別名, 対称暗号化アルゴリズムと呼ばれる. 秘密鍵暗号系は, 暗号化に用いる鍵が復号化に用いる鍵から計算できるアルゴリズムである. 普通送信者の鍵 Kは, 受信者の鍵 K'と同じかもしくは受信者の鍵から非常に簡単に計算できる. 図 2.1に秘密鍵暗号系の概念図を示す. 秘密鍵暗号系は, 自転車などに使われる番号合わせのチェーン鍵に似ている. 秘密鍵暗号系は, 送信者は, メッセージをチェーン鍵でロックした箱に入れて送り, 受信者は番号を合わせてチェーン鍵を開け, メッセージを読むイメージである. 秘密鍵暗号系では, 送信者 ・受信者はあらかじめ二者間で使用する鍵を決めておかなければならない.

秘密鍵暗号系はさらに 2 つの暗号方法に分類される. 一つはストリーム (逐次) 暗号であり, 平文[*]の列を bit 毎に暗号化する暗号方式である. ストリーム暗号化アルゴリズムとして, David Wheeler によって考案された **WAKE**, Ron Rivest によって考案された **RC4**, Phil Rogaway と Don Coppersmith によって考案された **SEAL** 等がある. 他方はブロック暗号であり, 平文の列をまず語に区切ってから語毎に暗号化する暗号方式である. 語は普通 block と呼ばれている. ブロック暗号化アルゴリズムとして, NBS(National Bureau of Standards) によって公示された **DES**, Uames Massery によって考案された **SAFER**, Xuejia Lai と James Massey によって提案された **IDEA**, Bruce Scheneier によって開発された **BLOWFISH** がある.

---

[*] 暗号化されていない文章

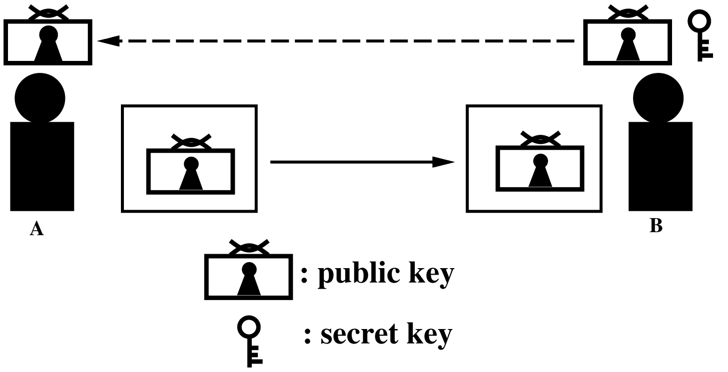


図 2.2: 公開鍵暗号系

公開鍵暗号系

別名, 非対称暗号化アルゴリズムと呼ばれる. 公開鍵暗号系では, 秘密鍵暗号系のアルゴリズムに対して, 暗号化に用いる鍵が復号化に用いる鍵と異なる. 公開鍵暗号系は復号化に用いる鍵が暗号化に用いる鍵からは計算できないアルゴリズムである. 図 2.2に公開鍵暗号系の概念図を示す. 公開鍵暗号系は鍵と錠前に似ている. 送信者は, メッセージを錠前でロックした箱に入れて送り, 受信者は鍵で錠前を開け, メッセージを読むイメージである. 他人に対して公開する公開鍵 (**public key**) と, 決して他人に知られてはならない秘密鍵 (**secret key**) という 2 つの鍵が使用される.

公開鍵暗号としては, Ron Rivest と Adi Shamir と Len Adelman によって考案された **RSA**, Robert McEliece によって開発された **McEliece**, Diffie と Hellman によって開発された **Diffie-Hellman** 等がある. 以上の既存の暗号化アルゴリズムをまとめると, 表 2.1 のようになる.

現在, 秘密鍵暗号系 ・公開鍵暗号系のアルゴリズムを用いたシステムとして, 第三者認証システムである **Kerberos** や, 電子メールの秘密通信方式である **PEM** や **PGP** 等が存在する. Kerberos は MIT で開発された第三者認証システムであり, サーバとクライアント間の認証機構を提供する. Kerberos には秘密鍵暗号系である DES が使われている. PEM は暗号, 認証, 電子署名, 鍵管理を提供するシステムであり, 秘密鍵暗号系である DES と公開鍵暗号系である RSA が用いられている. PGP は, PEM と同様の機能を提供する一方で, 暗号化には秘密鍵暗号系である IDEA と公開鍵暗号系である RSA が用いられている. 以上, 既存の暗号化システムをまとめると表 2.2になる.

表 2.1: 既存のアルゴリズム

分類		アルゴリズム名
秘密鍵暗号系	ストリーム暗号	WAKE
		RC4
		SEAL
	ブロック暗号	DES
		SAFER
		IDEA
		BLOWFISH
公開鍵暗号系		RSA
		McEliece
		Diffie-Hellman

表 2.2: 既存のシステム

システム名	アルゴリズム
Kerberos	DES
PEM	DES + RSA
PGP	IDEA + RSA

2.2 攻撃

暗号解読者が, 暗号化されたデータを暗号解読することを攻撃と呼ぶ. 現在さまざまな攻撃があり, 幾つかを以下に示す.

暗号文のみによる攻撃 (Ciphertext-only attack) 暗号解読者が, 暗号文のみを用いて攻撃する方法である. 全鍵検索が, この方法であり, 暗号解読者は, 取り得る鍵全てに対し, 鍵を一つずつ用いて暗号文を復号化して, 特定の鍵を見つけ出す. 全鍵検索は, 相当数の暗号文と, 長い時間が必要である.

既知の平文による攻撃 (Known-plaintext attack) 暗号解読者が, 相当数の平文とそれに対応する暗号文を用いて攻撃する方法である.

選んだ平文による攻撃 (Chosen-plaintext attack) 暗号解読者が, 暗号解読者の選択によって平文とそれに対応する暗号文を用いて攻撃する方法である. Eli Biham と Adi Shamir によって 1990 年に考案された Differential Cryptanalysis は,Chosen-plaintext attack の一種である. Differential Cryptanalysis は, 2 つの非常に似通った平文 X, X' とそれぞれに対応する暗号文 Y, Y' を選び, X と X' の差分,Y と Y' の差分, そして暗号化の方法より鍵を求める方法であり, 現在最も優れた暗号解読の方法である.

2.3 一方向ハッシュ関数

ハッシュ関数とは, 任意の大きさの値を, ある一定の範囲内の値に変換する関数であり, 一方向ハッシュ関数は, x から yを計算するのは簡単だが, yから x を導くのは非常に困難であるハッシュ関数である. すなわち,

$$y = f(x)$$

は高速に計算できるが,yから

$$y = f(x')$$

となるような x'を計算するための逆関数 f^{-1}は存在せず,x'をみつけるには可能性を全て調べなければならないハッシュ関数である.

一方向ハッシュ関数として MD2, MD4, MD5 などがある.

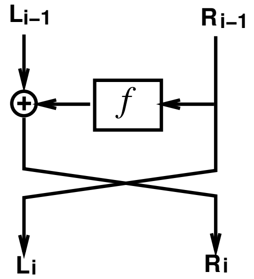


図 2.3: Feistel network

2.4 暗号化アルゴリズム

2.4.1 DES

概要

DES (Data Encryption Standard) は1975年8月1日にアメリカ商務省 **(NBS (National Bureau of Standards))** の名で公示された. NBSは1977年1月15日には政府機関の電算機はDESを用いてデータを暗号化することを義務付け, 民間もこれに従うように指示した.[DES]

DESは後述する **Feistel network** 方式であり, 64-bit長のblockと56bit長の鍵を使用し, 2つの基本的な暗号化技術であるconfusionとdiffusionの組み合わせから成るラウンドを16回繰り返す.

Feistel network 長さnのblockを$n/2$の長さのblock L_{i-1}とR_{i-1}に2分し, i段階目の出力L_iとR_iが前段階の出力から

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

によって決定される暗号方式である. (図2.3)

ここで, K_iはi段階目における鍵であり, 関数fは任意の関数である.

暗号化

図2.4にDESの暗号化過程を示す. 第一に入力に対して置換IPを行った後, 64 bitのblockを左半分の32 bitのblock (L_0) と右半分の32 bitのblock (R_0) に2分する. 第二

に, 鍵から生成された 48 bit 乱数 K_1と R_0から合成式 $f(R_0, K1)$ によって 32 bit の乱数 R''_0を作る. その後, R”$_0$と L_0の XOR[†]を計算して R_1を作り R_0を L_1とする. 第二段と同様の操作を 16 回繰り返し, 最後に求まった L_{16}と R_{16}を結合し, IPの逆転置 IP^{-1}をする. ここで $f(R, K)$ は図 2.5 のようになる.Rの 32 bit を拡大転置を行い 48 bit とし, 48 bit の Kとの XOR を計算し, 6 bit 宛の 8 組に分ける. 各 6 bit を圧縮変換して 4 bit にし, できた 32 bit に転置を行い R” を出力する.

また, 図 2.4の右側の図は, K_1 ~ K_{16}を生成する手順を示している. パリティービットを 8 bit 含んだ 64 bit から転置 (PC-1) を行い 56 bit の K_0を作り, これを 2 分して左半分の C_0と右半分の D_0を作る. 次に C_0, D_0を 1 bit 左シフト (LS1) して C_1, D_1を作り, 結合させた 56 bit を圧縮転置 (PC-2) を行って 48 bit の K_1を作る. これを繰り返し K_{16}まで生成する.

復号化

DES は対称暗号アルゴリズムであるので、復号化は暗号化の逆順に操作すればよい.

DES のモード

ブロック暗号には, ノイズの多いチャネルで送信する場合や, ファイルの一部だけを復号化しなければならない場合のために, ブロック暗号を利用してストリーム暗号を行うモードがある. DES では次のモードがある.

ECB モード **Electronic CodeBook mode**

入力の各 block を鍵を使って暗号化し, 出力を block として生成する. この方式は, メッセージを一度に 1 block 宛暗号化する簡単なものである. この方式で暗号化されたメッセージは, block の途中が挿入されたか, あるいは削除されてもわからないことがある. 一部の bit が変更されても 1 つの 64 bit 長の block が駄目になるだけですむので, ノイズの多いチャネルでのデータ転送に適している.

[†] 排他的論理和

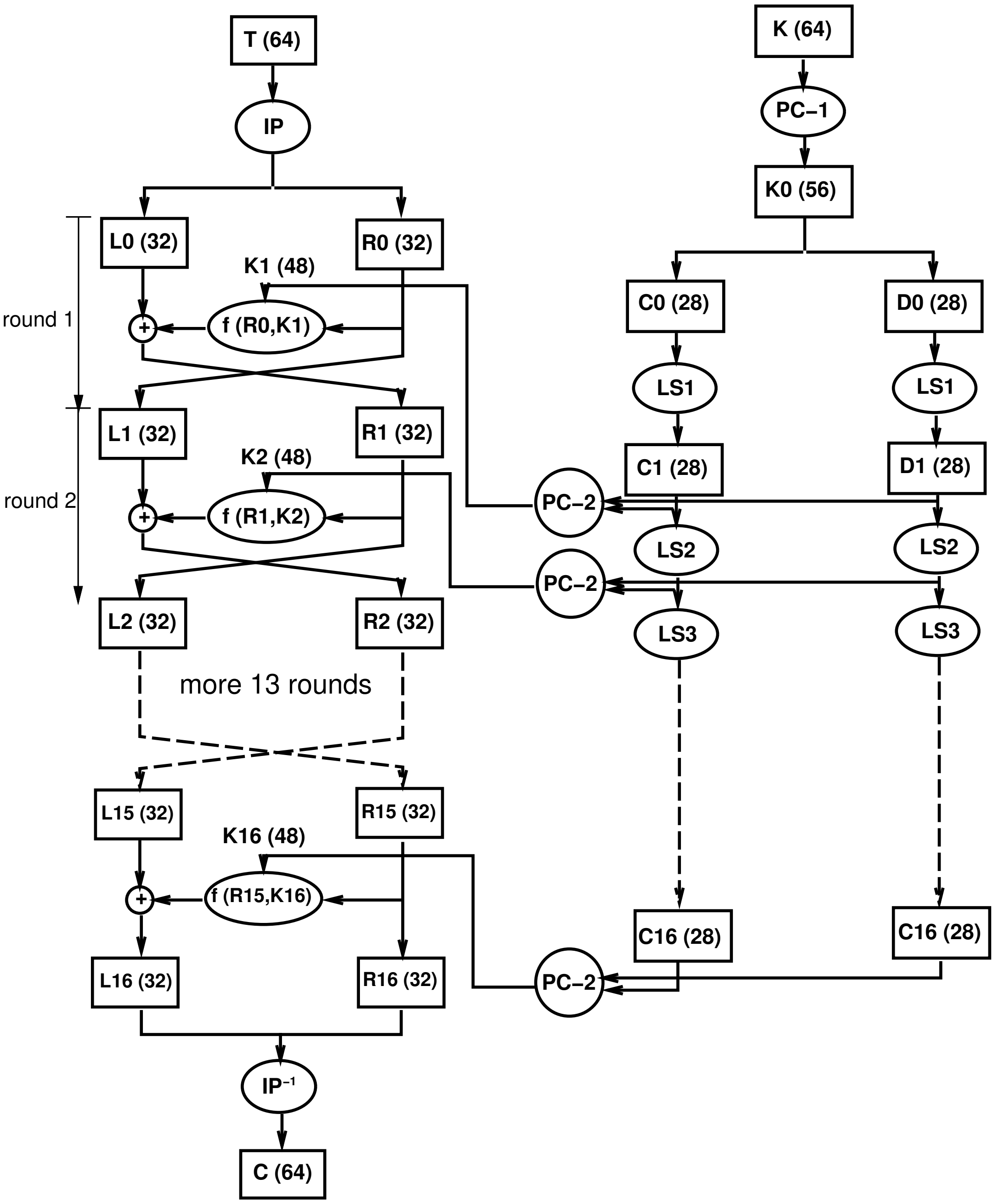


図 2.4: DES の暗号過程

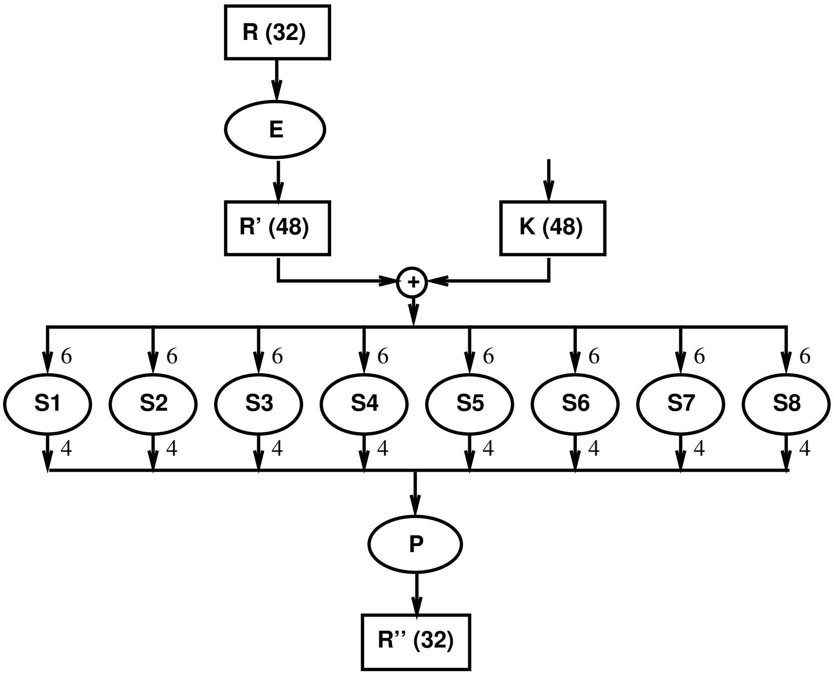


図 2.5: DES 関数 $f(R, K)$

CBC モード	**Cipher Block Chaining mode** 平文と前の block の暗号化された値との間で XOR を行う. 最初の block にはなんらかの既知の値を使用する. 次に, XOR の結果を鍵を用いて暗号化する.(図 2.6) メッセージのどの bit も暗号全体に伝搬するため, 最後の block を暗号テキストが変更されていないことをチェックするためのチェックサム/シグネチャとして使用できる. また, 平文に同じ文字がいくつ続いても, 暗号化すると, 平文では同じ文字がいくつも続いていることがわからなくなる.
CFB モード	**Cipher-FeedBack mode** まず何らかの値を ECB モードで暗号化する. その結果と data block の XOR を計算し, 出力する. そして, その出力を暗号化の入力とする. (図 2.7) この方式は自己同期性を持っている. たとえば, 暗号化された大型データベースの一部分だけを復号化したい場合には, その部分の前にある, 特定の長さの block を用いることで復号化することが可能である.

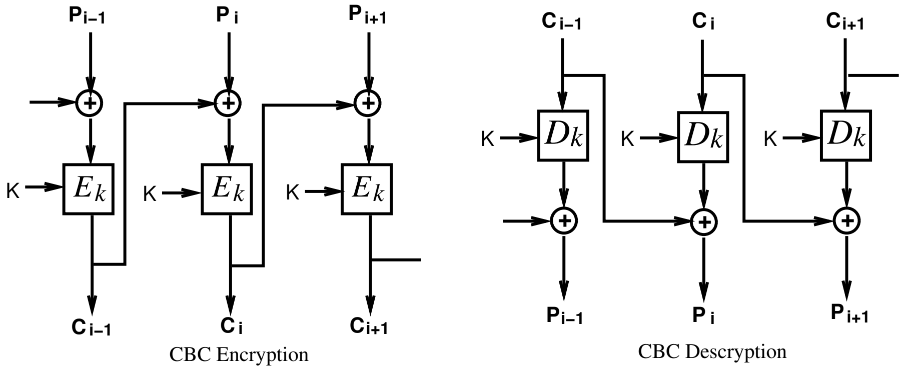


図 2.6: CBC モード

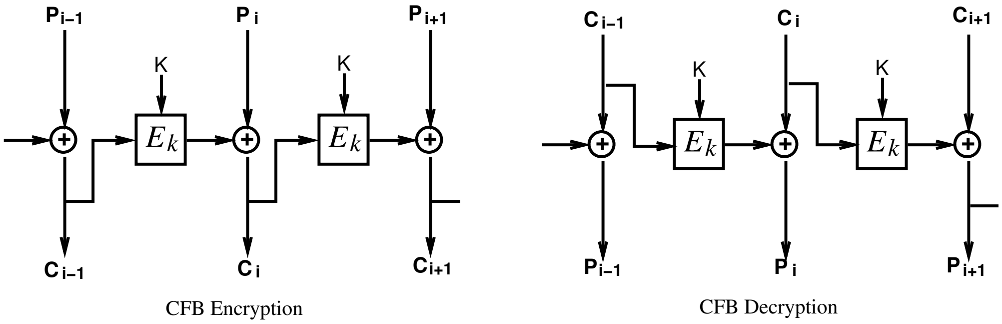
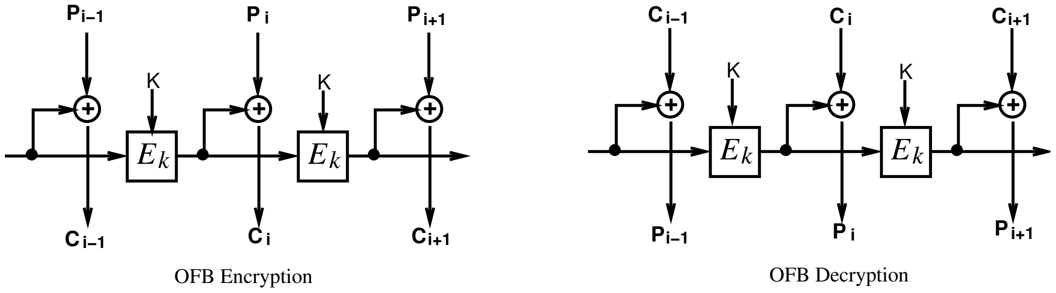


図 2.7: CFB モード



OFB Encryption　OFB Decryption

図 2.8: OFB モード

表 2.3: DES の弱点鍵

0101	0101	0101	0101
1F1F	1F1F	0E0E	0E0E
E0E0	E0E0	F1F1	F1F1
FEFE	FEFE	FEFE	FEFE

OFB モード	**Output-FeedBack mode** まず何らかの値を暗号化し, この結果と data block の XOR を計算し, 出力すると同時に, 暗号化した結果は, 次の暗号化の入力となる (図 2.8).

3-DES

DES の鍵長は不十分であり, そこで 3-DES が考案された. 3-DES は,2 つの異なった鍵で 3 回 DES の操作を行う暗号化であり, DES の 2 倍の暗号強度がある. 3-DES は, 現在金融機関で使用されている.

安全性

今日では, DES は必ずしも強い暗号化アルゴリズムではなくなり, さらに時間の経過と共に CPU の性能が向上し, 弱くなる傾向にある.

DES には幾つかの弱点鍵が存在する. 弱点鍵を用いた場合, 各ラウンドで使用される鍵が全て同じになってしまい, 結果として暗号強度が非常に弱くなってしまう. しかし, 弱点となる鍵は非常に少ないので問題ではないが, ユーザはこのような鍵を最初に生成しないように気を付けなければならない. 弱点鍵の例としては表 2.3のようなものがあげられる.

また, DES の鍵の長さが不十分である, という指摘が数多くなされている. 1981 年には Diffie が, DES 解読専用並列計算機を用いれば全鍵検索をしても 5000 万ドルの費用で 2 日の内に解読できる事を指摘している. さらに, 1993 年には Michael Wiener の設計した 100 万ドルの計算機でわずか 3.5 時間で解読できるようになってしまった [DES 93].

DES の暗号強度について様々な議論が行われてきた. そして **differential cryptanalysis** よって, DES の鍵の長に対する議論は一応の解決がついている. DES は, いかなる鍵生成方法を用いても鍵の長さが 56 bit である限り, より強くする事は不可能である [DES 91]. DES の 2 倍の強度がある 3-DES でさえ, 現在では暗号強度は, 弱くなってしまっている.

2.4.2 SAFER

概要

SAFER K-64(Secure And Fast Encryption Routine with a Key of length 64 bits) は **James Massery** によって考案された [SAFER][SAFER K-64 94].

アルゴリズムとしてはDESで用いていたFeistel network方式ではなく, **iterated block cipher** 方式を採用しており, 64 bit 長の block と 64 bit 長の鍵を使用する. SAFER は普通 iterated block cipher を 6 ラウンド行う. 各ラウンドにおいて, ユーザが決めた 64 bit の鍵から生成した 64 bit の subkey が 2 つ使用される.

iterated block cipher iterated block cipher 方式とは, 排他的論理和等の簡単な基本的な関数を複数回繰り返す暗号化の方式である.

暗号化

図 2.9に SAFER K-64 の暗号化過程を示す. 暗号文の生成方法は, 排他的論理和等の基本的な計算を組み合わせからなる変換[‡] (Encrypting Round) を平文に対し r 回同様に施し, 最後に出力用の変換 (Output Transformation) を施す. (一般的には r=6 であるが, 必要であれば r=10 まで可能である.) 各ラウンドは 2 つの 64 bit 長の subkey を使用し, Output Transformation は 1 つの 64 bit 長の subkey を使用する. Output Transformation では, 入力の 64 bit を Byte ごとに 8 つに分けた内の 1, 4, 5, 8 番目と, 最後の subkey K_{2r+1}を Byte ごとに 8 つに分けたうちの 1, 4, 5, 8 番目の XOR を計算し, 入力の 2, 3, 6, 7 番目と, K_{2r+1}の 2, 3, 6, 7 番目の加算を計算する. そして求まった結果が出力される.

[‡] iterated block cipher

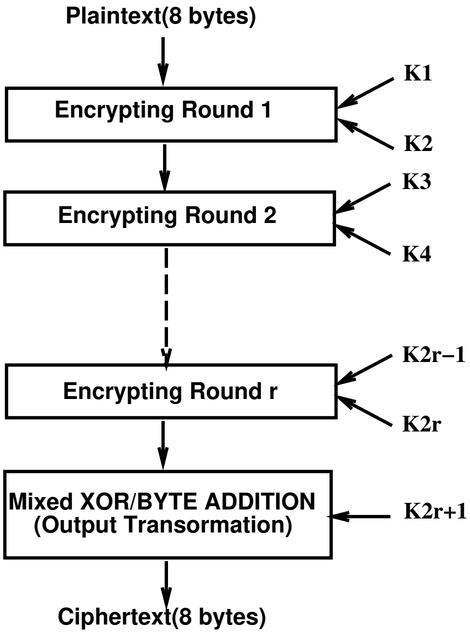


図 2.9: SAFER K-64 の暗号化過程

Encrypting Round では図 2.10に示した計算が行われる．この時, 64 bit の入力は, 8 個の 1 Byte 長の sub-block に分割される．変換アルゴリズムは, i ラウンドの第一段階として, 8 個の sub-block と subkeyK_{2i-1}の XOR または加算を計算する．第二段階として, sub-block に以下の 2 つの非線形変換のどちらか一方を適用する．

$$y = 45^x \bmod 257$$

$$y = \log_{45} x$$

第三段階として, 再び subkey K_{2i}と第二段の出力 yの XOR または加算を計算する．第四段階として, 3 回の線形変換 **PHT(Psudo-Hadamard Transform)** を施す．PHT は 2 つの入力 a_1, a_2に対して

$$b_1 = (2a_1 + a_2) \bmod 256$$

$$b_2 = (a_1 + a_2) \bmod 256$$

を計算して b_1, b_2を出力する．

subkey の生成方法を図 2.11に示す．subkey K_1はユーザが決定した鍵である．K_1以後の subkey は

$$K_{i+1} = (K_1 << 3 * i) + B_i$$

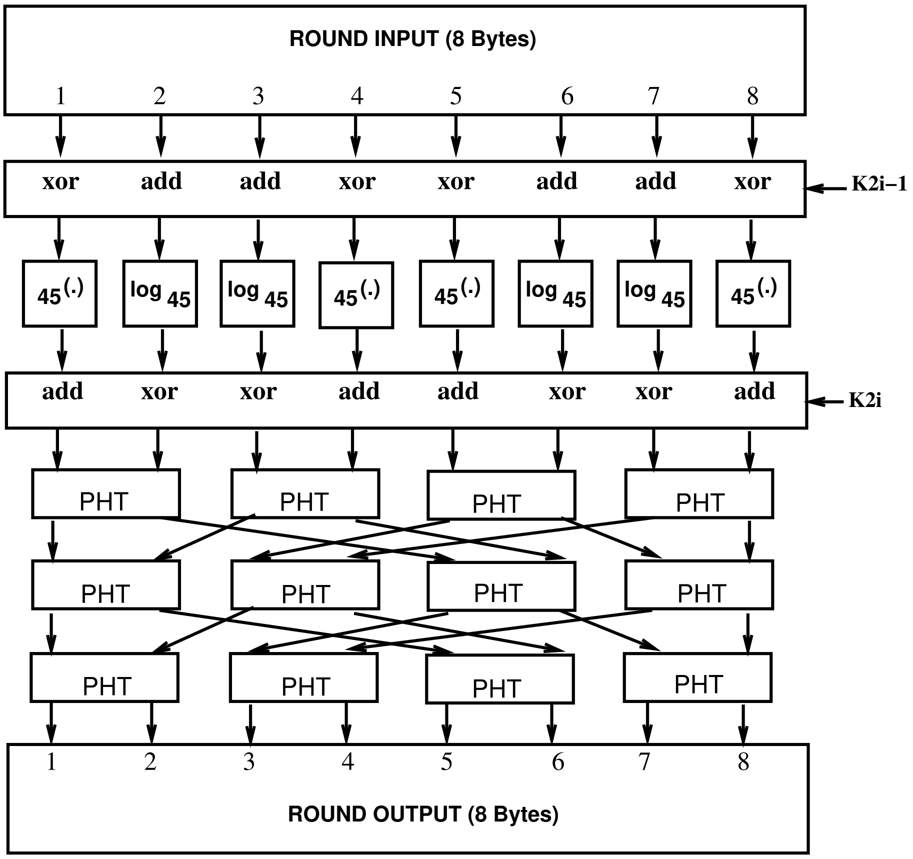


図 2.10: 1 Encrypting Round

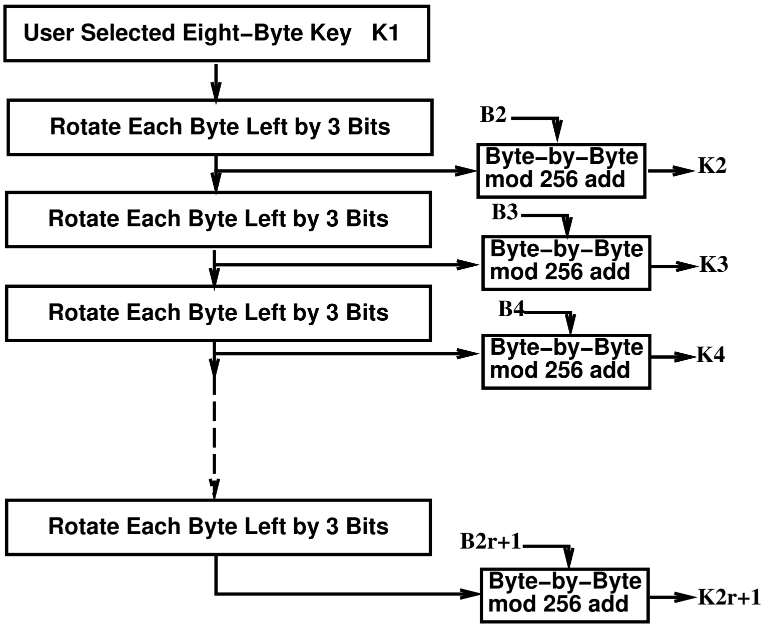


図 2.11: SAFER K-64 の鍵生成

で求められる. ここで B_iはラウンド定数と呼ばれるものであり, B_iの j番目の Byte は次の式によって求められる.

$$B[i,j] = 45^{45^{(9*i+j) \bmod 256} \bmod 257} \bmod 257$$

復号化

SAFER は対称暗号系であるので, 復号化は暗号化の逆を行えば良い.

安全性

r> 8 の時には Differential Cryptanalysis に対してさえ安全であり, r=6 の時には多くの暗号解析に対して安全であると言われている.

SAFER を一方向ハッシュ関数として用いる場合, SAFER は他の暗号化アルゴリズムよりはるかに弱い. これは, SAFER の鍵生成方法が原因である. SAFER では, 幾つかの異なった平文を同じ暗号文に暗号化する鍵が計 2 つ以上 (最悪の場合計 10 以上) 存在する.

SAFER K-128

SAFERには64 bitの鍵を使うSAFER-K64の他に128 bitの鍵を使うSAFER-K128もある．SAFER-K128は, 鍵の長さが128 bitであること以外, 全てSAFER K-64と同じである．SAFER-K128では64 bit長の2つの鍵Ka, Kbを用いる．鍵の生成の際には, 2つのsubkeyを並行に生成し, その後, 鍵を入れ替える.

2.4.3 IDEA

概要

IDEA(International Data Encryption Algorithm)は, 1990年に**Xuejia Lai**と**James Massey**によって**PES(Proposed Encryption Standard)**と言う名で発表された. その後, PESは, 補強され**IPES(Improved Proposed Encryption Standard)**となり, そしてIPESは, 1992年に改良され, 同時にIDEAと改名された. 現在, IDEAはもっとも強いブロック暗号化アルゴリズムと考えられている[IDEA].

IDEAは, 64 bit長のblockと128 bit長の鍵を使用し, 16 bitのsub-blockを一単位として処理をする[IDEA 93a][IDEA 93b].

暗号化

図2.12にIDEAのアルゴリズムを示す．64 bitのデータblockは4つの16 bitのsub-block, X_1, X_2, X_3, X_4に分割される．4つのsub-blockはアルゴリズムの最初のラウンドの入力となる．IDEAでは最終的に合計8ラウンド行なう.

1ラウンド毎に次のように処理が行なわれる.

1. sub-block X_1と最初のsubkeyを掛ける.
2. sub-block X_2と2番目のsubkeyを足す.
3. sub-block X_3と3番目のsubkeyを足す.
4. sub-block X_4と4番目のsubkeyを掛ける.
5. (1)と(3)の結果のXORを計算する.
6. (2)と(4)の結果のXORを計算する.
7. (5)の結果と5番目のsubkeyを掛ける.

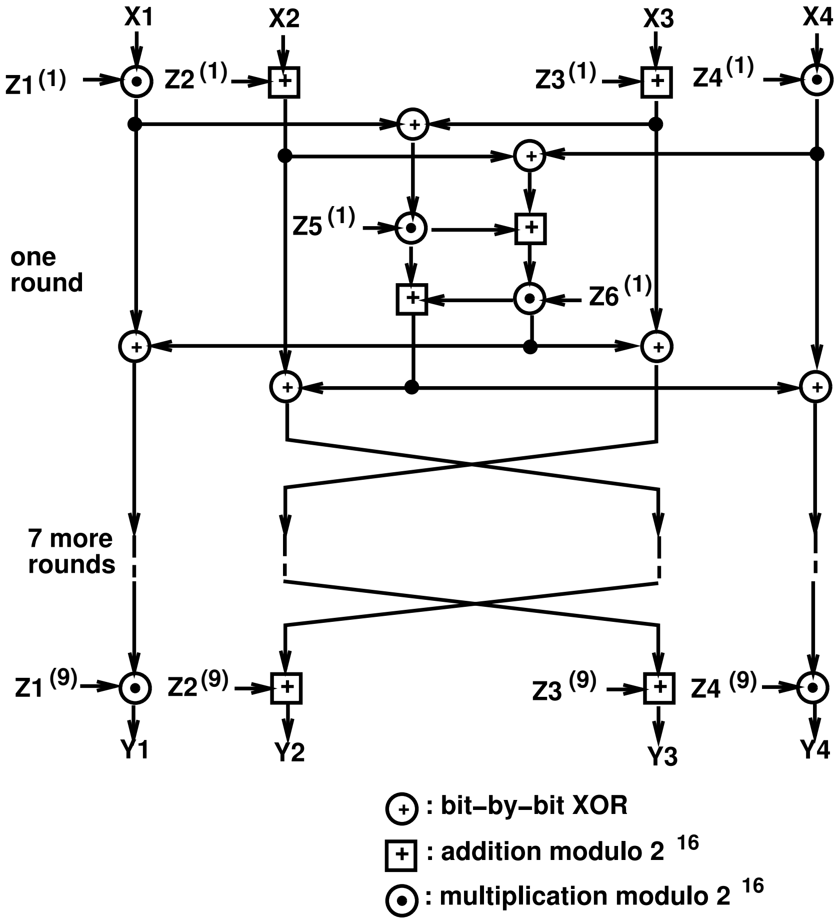


図 2.12: IDEA 暗号過程

8. (6) と (7) の結果を足す.
9. (8) の結果と 6 番目の subkey を掛ける.
10. (7) と (9) の結果を足す.
11. (1) と (9) の結果の XOR を計算する.
12. (3) と (9) の結果の XOR を計算する.
13. (2) と (10) の結果の XOR を計算する.
14. (4) と (10) の結果の XOR を計算する.

(11), (12), (13), (14) の結果がラウンドの出力であり, (12), (13) の結果が入れ換えられ次のラウンドの入力となる.

8 ラウンド終了後に出力に対して次のような変換が行なわれる.

1. X_1と最初の subkey を掛ける.
2. X_2と 2 番目の subkey を足す.
3. X_3と 3 番目の subkey を足す.
4. X_4と 4 番目の subkey を掛ける.

IDEA が使用する 16 bit の 52 個の subkey は, 次の要領で 128 bit の鍵から生成される. 128 bit の鍵は 16 bit 毎に 8 つに分けられる. これらが, 最初の 8 つの subkey になる. 次に, 128 bit の鍵は 25 bit 循環シフトされ, 再び 16 bit ずつ 8 つに分けられる. これらが次の 8 つの subkey となる. この手順を繰り返す.

復号化

復号化は, subkey が逆順で使用され, かつ subkey が多少異なる事を除いて暗号化と同じアルゴリズムで行われる. 復号化時に使われる subkey は, 暗号化時に用いた鍵の符号反転した物あるいは, 逆数である. 暗号化時の subkey と復号化時の subkey の関係を表 2.4に示す.

表 2.4: IDEA の subkey

ラウンド	暗号化時	復号化時
1	$Z_1^{(1)} Z_2^{(1)} Z_3^{(1)} Z_4^{(1)} Z_5^{(1)} Z_6^{(1)}$	$Z_1^{(9)-1} - Z_2^{(9)} - Z_3^{(9)} Z_4^{(9)-1} Z_5^{(8)} Z_6^{(8)}$
2	$Z_1^{(2)} Z_2^{(2)} Z_3^{(2)} Z_4^{(2)} Z_5^{(2)} Z_6^{(2)}$	$Z_1^{(8)-1} - Z_2^{(8)} - Z_3^{(8)} Z_4^{(8)-1} Z_5^{(7)} Z_6^{(7)}$
3	$Z_1^{(3)} Z_2^{(3)} Z_3^{(3)} Z_4^{(3)} Z_5^{(3)} Z_6^{(3)}$	$Z_1^{(7)-1} - Z_2^{(7)} - Z_3^{(7)} Z_4^{(7)-1} Z_5^{(6)} Z_6^{(6)}$
4	$Z_1^{(4)} Z_2^{(4)} Z_3^{(4)} Z_4^{(4)} Z_5^{(4)} Z_6^{(4)}$	$Z_1^{(6)-1} - Z_2^{(6)} - Z_3^{(6)} Z_4^{(6)-1} Z_5^{(5)} Z_6^{(5)}$
5	$Z_1^{(5)} Z_2^{(5)} Z_3^{(5)} Z_4^{(5)} Z_5^{(5)} Z_6^{(5)}$	$Z_1^{(5)-1} - Z_2^{(5)} - Z_3^{(5)} Z_4^{(5)-1} Z_5^{(4)} Z_6^{(4)}$
6	$Z_1^{(6)} Z_2^{(6)} Z_3^{(6)} Z_4^{(6)} Z_5^{(6)} Z_6^{(6)}$	$Z_1^{(4)-1} - Z_2^{(4)} - Z_3^{(4)} Z_4^{(4)-1} Z_5^{(3)} Z_6^{(3)}$
7	$Z_1^{(7)} Z_2^{(7)} Z_3^{(7)} Z_4^{(7)} Z_5^{(7)} Z_6^{(7)}$	$Z_1^{(3)-1} - Z_2^{(3)} - Z_3^{(3)} Z_4^{(3)-1} Z_5^{(2)} Z_6^{(2)}$
8	$Z_1^{(8)} Z_2^{(8)} Z_3^{(8)} Z_4^{(8)} Z_5^{(8)} Z_6^{(8)}$	$Z_1^{(2)-1} - Z_2^{(2)} - Z_3^{(2)} Z_4^{(2)-1} Z_5^{(1)} Z_6^{(1)}$
output	$Z_1^{(9)} Z_2^{(9)} Z_3^{(9)} Z_4^{(9)}$	$Z_1^{(1)-1} - Z_2^{(1)} - Z_3^{(1)} Z_4^{(1)-1}$

安全性

現在, 秘密鍵暗号系の中では IDEA が最も安全であると考えられている. 全鍵検索を行った場合, 1 秒間に 10 億個の鍵について調べる計算機を用いたとしても, 10^{13}年かかってしまう. IDEA のラウンドを, わずか 4 回行っただけで, Differential Cryptanalysis に対しても安全な暗号系となる.

2.4.4 FEAL

概要

FEAL(Fast data Encipherment ALgorithm) は NTT の清水, 宮口ら [FEAL 87a] によって開発された. FEAL は 64 bit 長の block と 64 bit 長の鍵を使用する [FEAL 87b] [FEAL].

暗号化

図 2.13に FEAL のアルゴリズムを示す.

はじめに, 入力された 64 bit のデータ block と 64 bit の鍵の XOR を計算し, 求まった結果を左半分の 32 bit と右半分の 32 bit に分割する. 左半分と右半分の XOR を計算した結果が新しい右半分の値となり, 左半分と新しく求まった右半分が, それ以降 n 回繰り返されるラウンドへの入力となる.

各ラウンドでは, 入力の右半分と 16 bit の鍵を引数とした関数 fの値を計算し, これによって求まった結果と左半分の XOR を計算した値を新しい左半分の値とする.

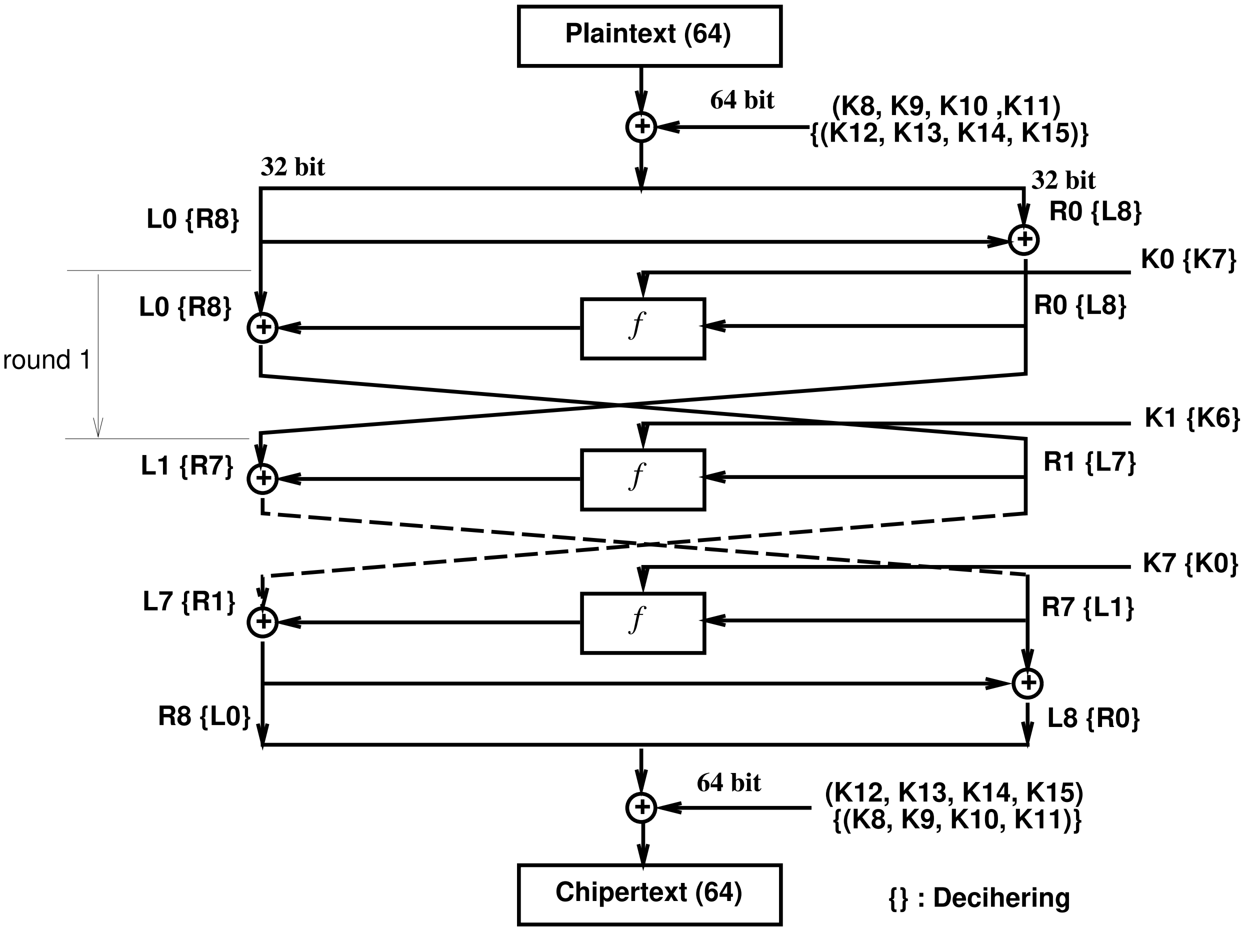


図 2.13: FEAL の暗号過程

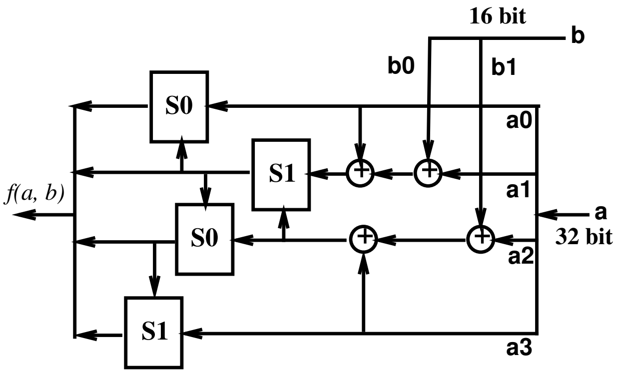


図 2.14: FEAL の関数 f

図 2.14に関数 fのアルゴリズムを示す. 最初に, 32 bit の入力データは 4 つの 8 bit の塊に分けらてから処理される. 二つの関数 S_0, S_1は次のように定義される.

$$S_0(a,b) = rotate\ left\ 2\ bits((a+b) \bmod 256)$$

$$S_1(a,b) = rotate\ left\ 2\ bits((a+b+1) \bmod 256)$$

その後, 左半分と右半分を入れ換える. ただし n ラウンド目では, 入れ換えは行なわれない. n ラウンド終了後に, 左半分と右半分の XOR を計算し, 新たに右半分の値とする. 最後に左半分と右半分を結合し, その後 64 bit の鍵と XOR を計算する.

図 2.15に FEAL が暗号化の際に用いる subkey の生成方法を示す. subkey を生成する手順は以下の通りである.

1. 64 bit の入力を 32 bit 毎に左半分の A_0, 右半分の B_0に分ける.
2. 左半分 A_0, 右半分 B_0を引数として, 関数 f_kを計算する.
3. (2) の結果を 16 bit 毎に分け, 鍵 K_0, K_1とする.
4. (1) の右半分を次の段の左半分 A_1にし,(2) の結果を次の段の右半分 B_1とする.
5. 右半分 B_1と前段の左半分 A_0の XOR を計算する
6. 左半分 A_1と (5) の結果を引数として, 関数 f_kを計算する.
7. (6) の結果を 16 bit 毎に分け, 鍵 K_2, K_3とする.
8. (4), (5), (6), (7) と同様に操作を以降 6 回繰り返し, 鍵 K_{14}, K_{15}まで求める

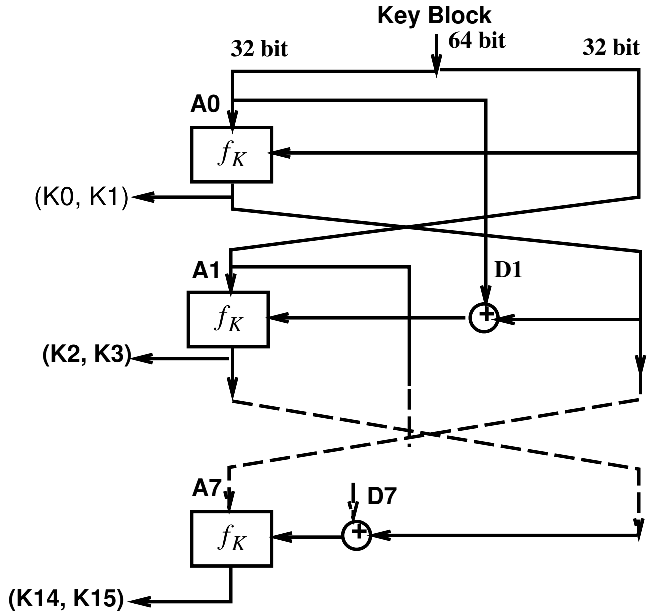


図 2.15: FEAL の鍵生成

また, subkey を生成する際に使用する関数 f_kを図 2.16 に示す. 関数 f_kの計算手順は以下の通りである.

1. 2 つの 32 bit の入力はそれぞれ 8 bit 毎 4 つに分けられ, a0, a1, a2, a3 と b0, b1, b2, b3 になる
2. a0 と a1 の XOR を計算する
3. a2 と a3 の XOR を計算する
4. (3) の結果と b0 の XOR を計算する.
5. (2) と (4) の結果を引数として S_1を計算する.
6. (5) と b1 の XOR を計算する.
7. (3) と (6) を引数として S_0を計算する.
8. (5) の結果と b2 の XOR を計算する.
9. a0 と (8) の結果を引数として S_0を計算する.
10. (7) の結果と b3 の XOR を計算する.

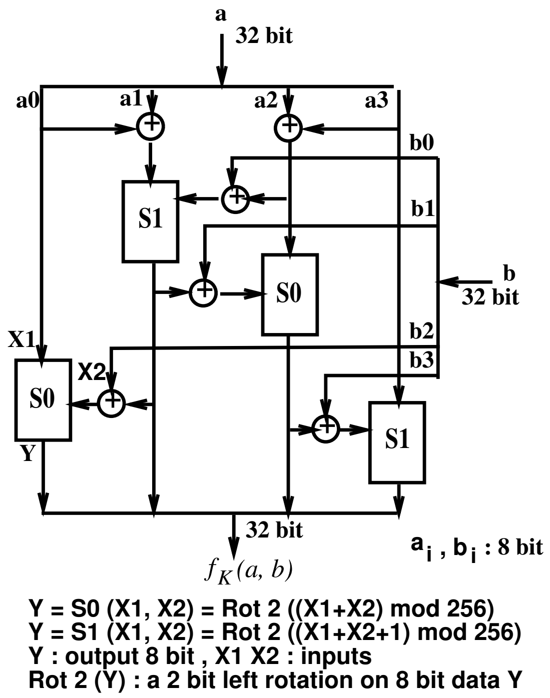


図 2.16: FEAL の鍵生成関数 f_K

11. (10) の結果と a3 を引数として S_1を計算する.
12. (9), (5), (7), (11) の結果を結合して出力する.

関数 S_0, S_1は既に述べた通りである.

安全性

現在, FEAL はで生成した暗号文は, 32 以下のラウンド数では簡単に解読されてしまう. また, 鍵の長さを 128 bit にした **FEAL-NX** でも, 動的スワップ関数を用いた **FEAL-NXS** でも同様に簡単に解読されてしまう. このように, 現在では FEAL は相対的に弱い暗号化アルゴリズムとなっている.

2.4.5 GOST

概要

GOST(Gosudarstvennyi Standard) は, 旧ソビエト連邦で開発されたブロック暗号化アルゴリズムである [GOST].

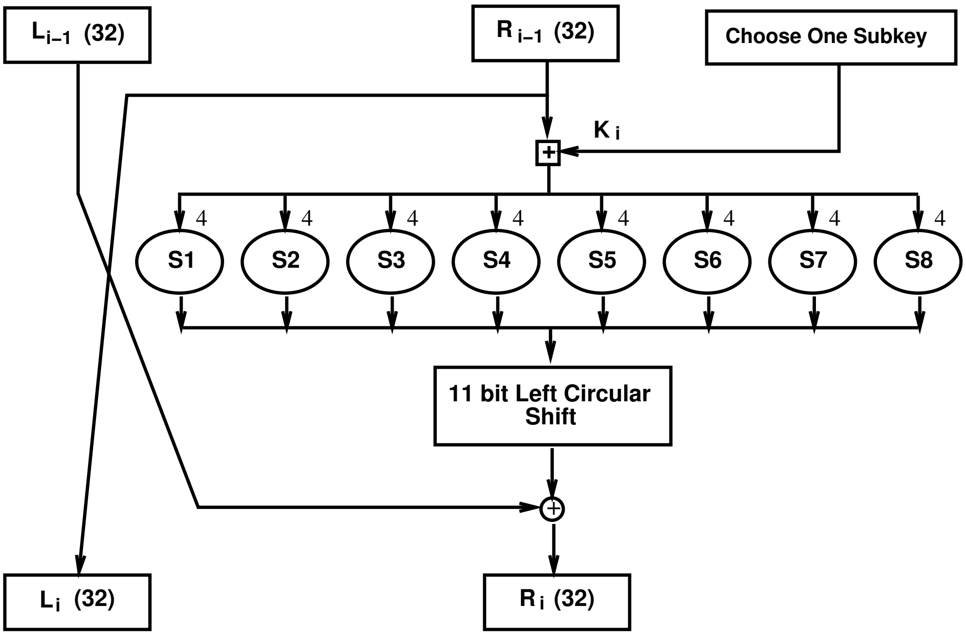


図 2.17: GOST 暗号過程 (1 ラウンド)

GOST は 64 bit 長の block と 256 bit 長の鍵を使用し, 単純な暗号化操作 (ラウンド) を 32 回繰り返す [GOST 94].

暗号化

暗号化手順としては, 最初に, 64 bit の入力を左半分の L と右半分の R に 2 分する. 次に簡単な暗号化 (ラウンド) を 32 回繰り返し, 最後に, 求まった左半分と右半分を結合して終りである.

この 32 回繰り返すラウンドは以下の通りである. 1 ラウンド毎に図 2.17に示す操作を行う. i ラウンド目の操作は, まず右半分の R_{i-1} が i 番目の subkey と s^{32} を基数とした加算を行い, 求まった結果を 4 bit 毎の 8 個の塊にわけ, それぞれに対して異なった置換する. そのようにして求まった 32 bit を 11 bit 左に循環シフトし, L_{i-1} と XOR を行い, その結果を R_i とする. また R_{i-1} をそのまま L_i とする.

また, 各ラウンドで使用する subkey は次のように決定される.

256 bit の鍵から 32 bit 毎に 8 つの subkey $k_1, k_2, ..., k_8$ に分割され, 表 2.5に従って使用される subkey が決定される.

表 2.5: GOST subkey の使用表

Round:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Subkey:	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Round:	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Subkey:	1	2	3	4	5	6	7	8	8	7	6	5	4	3	2	1

安全性

GOST は DES より強い暗号化アルゴリズムであると考えられている. GOST の安全性は, 256 bit から成る鍵の長さと, ラウンドを 32 回繰り返すということによって保証されている.

2.4.6 BLOWFISH

概要

BLOWFISH は **Schneier** によって開発された暗号化アルゴリズムであり, 鍵の長さの決っていない 64 bit の block を使用するブロック暗号方式である [BLOWFISH].

BLOWFISH のアルゴリズムは, 2 つの部分から構成されている. 一つは, 鍵の拡張部分であり, もう一つはデータを暗号化する部分である. 鍵拡張部分は最長 448 bit の鍵を合計 4168 Byte の subkey の配列に変換する. データを暗号化する部分は, Feistel network 方式を 16 回繰り返す方法を採用している.

暗号化

BLOWFISH は, 多くの subkey を使用し, これらの subkey は暗号化・復号化の前に計算されていなければならない. subkey は以下のように分類される.

- 32 bit の subkey を 18 個有する P 配列:
  $P_1, P_2, ..., P_{18}$
- 32 bit の要素を 256 個持つ 4 つの S-box:
  $S_{1,0}, S_{1,1}, ..., S_{1,255}$
  $S_{2,0}, S_{2,1}, ..., S_{2,255}$
  $S_{3,0}, S_{3,1}, ..., S_{3,255}$
  $S_{4,0}, S_{4,1}, ..., S_{4,255}$

BLOWFISH の暗号化過程は図 2.18のようになる. 最初に, 入力された 64 bit のデータ

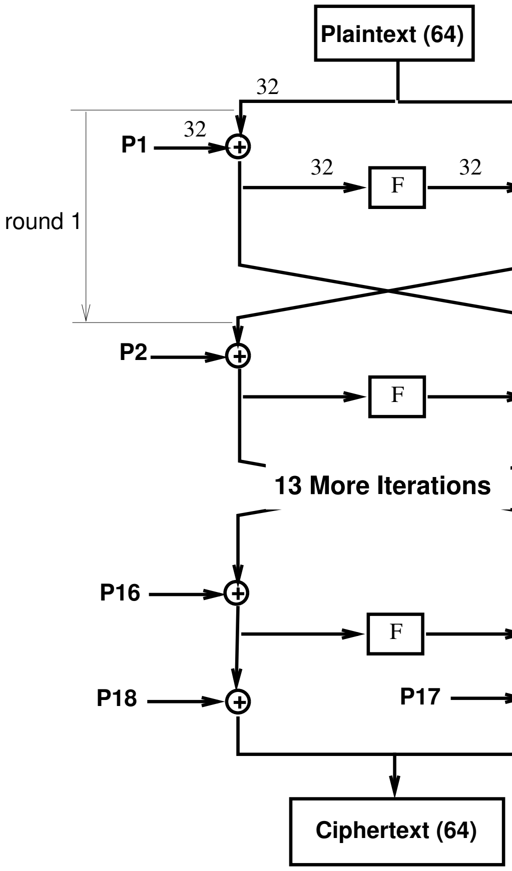


図 2.18: BLOWFISH 暗号過程

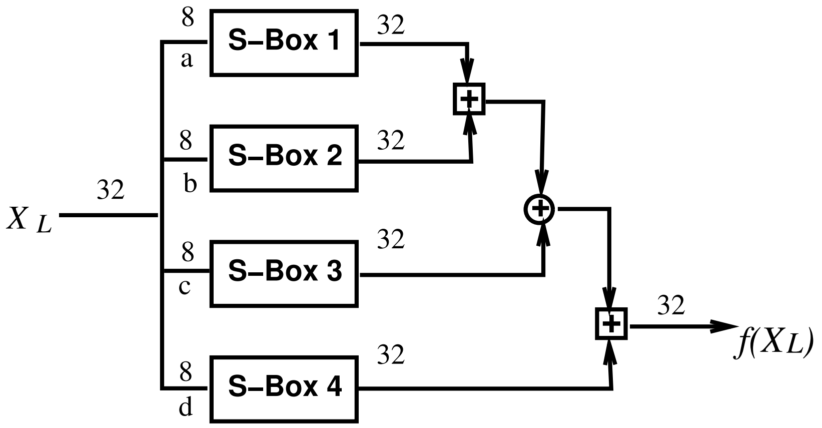


図 2.19: BLOWFISH 関数 f

Xは 32 bit 毎の X_Lと X_Rに 2 分される．第二段として，16 回以下のような簡単な計算を行う．

$$X_L = X_L \oplus P_i$$

$$X_R = F(X_L) \oplus X_R$$

$$X_L \leftrightarrow X_R$$

ただし，16 回目は最後の X_Lと X_Rの入れ換えは行わない．第三段として，

$$X_R = X_R \oplus P_{17}$$

$$X_L = X_L \oplus P_{18}$$

を行う．

最後に X_Lと X_Rを結合して終りである．

また，関数 f のアルゴリズムは図 2.19に示された通りであり，関数 f は，次のように定義される．

$$F(X_L) = ((S_{1,a} + S_{2,b} \bmod 2^{32}) \oplus S_{3,c}) + S_{4,d} \bmod 2^{32}$$

a, b, c, d は X_Lを 8bit 毎 4 つに分割した各要素である．

一方，BLOWFISH で使用される鍵は，次のアルゴリズムから求められる．

1. 最初に，πを 16 進表示したストリングから P 配列を決定し，次に S-Box を決定する．

2. P_1と鍵の先頭 32bit の XOR を計算した結果を P_1とし，次に P_2と鍵の次の 32bit の XOR を計算した結果を P_2とするといった手順で次々と P_{18}まで求める．鍵は P_{18}が求まるまで繰り返し使われる．

3. (1), (2) から求まった subkey を用いて, 全てが 0 であるストリングを BLOWFISH の暗号化アルゴリズムで暗号化する.
4. P_1 と P_2 に (3) の出力を代入する.
5. (3) の出力を新しい subkey を用いて BLOWFISH で暗号化する.
6. P_3 と P_4 に (5) の出力を代入する.
7. 上記の処理を繰り返し P 配列および S-Box を求める.

安全性

differential cryptanalysis を用いるとしても, r ラウンド暗号化を行う BLOWFISH に対して 2^{2r+1} 個の chosen plaintext(暗号解読者が選択して入手した平文とそれに対応する暗号文) を必要とする. 現在のところ, 繰り返しの回数を減らさない限り, differential cryptanalysis でも BLOWFISH を破ることは困難である.

2.4.7 RSA

概要

RSA は, 1976 年に **Ron Rivest**, **Adi Shamir**, **Len Adelman** によって考案された公開鍵暗号系の暗号化アルゴリズムである. RSA の暗号強度は, 非常に大きい数を因数分解するのは非常に困難であるという数学的性質により確保されている [RSA1][RSA2].

アルゴリズム

RSA は, 剰余演算と整数が持つ数論特性に基づいている. RSA が使用する数論特性は,「n に対して素である i を $\phi(n)$ 乗し, $\bmod n$ をとると 1 となる 」ことである. $\phi(n)$ はオイラーのファイ関数と呼ばれるものである. ある数に対する $\phi(n)$ は, n より小さく, n に対して 「互いに素である 」整数の個数と定義される.

各利用者はそれぞれ自分用の非常に大きい整数 n と 2 つの整数 e, d を以下に述べる方法で選ぶ. 利用者が公開する鍵は (n, e) の組であり, 対応する秘密にしておく鍵は (n, d) の組である. 通信文 M を暗号化する算法 E と暗号文 C を復号する算法 D は

$$E(M) = M^e \pmod n$$

$$D(M) = C^d \pmod n$$

である.

整数 e と d は 0 から $n-1$ までのすべての整数 X に対して

$$(X^e)^d = X^{ed} = X \pmod{n}$$

となるように選ぶ. したがって先の範囲のすべての X について

$$E(D(X)) = D(E(X)) = X \pmod{n}$$

となる.

大きな数を素因数分解することの難しさを次のようにして生じさせる. 各利用者は無作為抽出によって, 2 つの大きな素数 p と q を選び, p と q の積を n とする. 次に $\phi(n) = (p-1)(q-1)$ と互いに素で, これよりも小さな任意の整数 $d > max(p,q)$ を選ぶ. 最後に $\phi(n)$ と d から, $\phi(n)$ を法とした d の乗算の逆元 e を計算する. つまり,

$$ed = 1 \pmod{\phi(n)}$$

である.

安全性

n と e または d の値が外部に洩れても, 暗号が解読されるおそれはない. RSA で暗号化したメッセージを解読するためには, $\phi(n)$ をみつけ, n を素因数分解しなければならず, 大きな数の素因数分解を効率的に計算する方法は考案されていないからである. 大きな数は, 200 桁以上の 10 進数でなければいけない.

2.4.8 全体的な問題点

以上, 様々な暗号化アルゴリズムについて, 概要を述べた. ここで, これらの全ての暗号化アルゴリズムに共通する問題を提起する.

現在提供されている暗号化アルゴリズムを用いて暗号通信を行う際には, 送信者と受信者の間で, 使用する暗号化アルゴリズムに関する合意が取れている必要がある. このため利用者は, メッセージの重要性とマシンの性能にあった暗号化アルゴリズムを送信時に自由に選択することができない.

また, 以上のアルゴリズムの強度は, “鍵の取り得る値の範囲の広さ” と, “復号化にかかる計算量” によって定められている. 暗号の安全性が計算量に裏付けられている以上, 時が経ち計算機の計算パワーが増大すると共に, 暗号化アルゴリズムが弱くなって行くとい

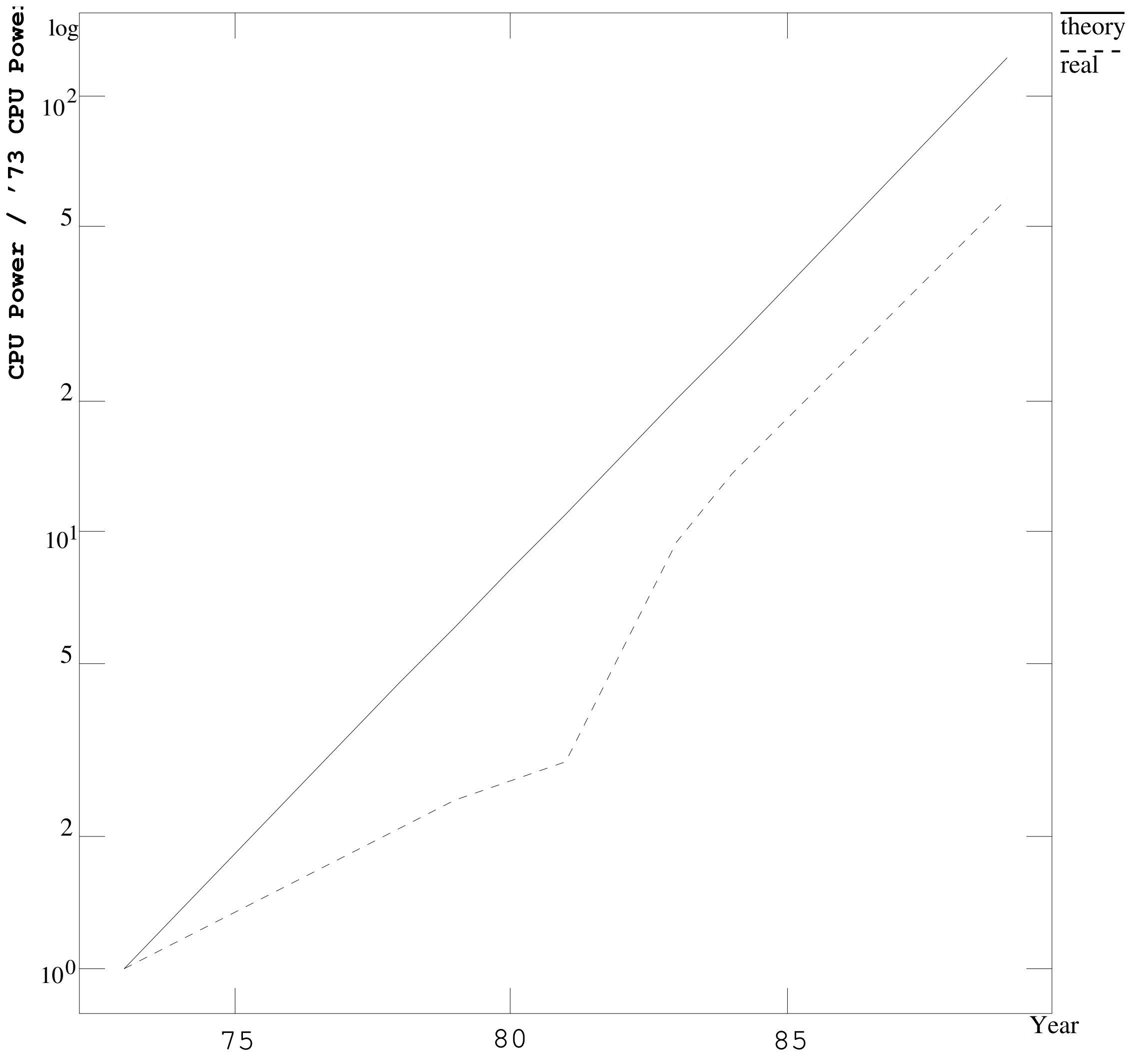


図 2.20: CPU 性能の移り変わり

う非常に大きな問題が存在する. CPU の性能は, 年率平均 35 %ずつ向上すると言われている [Computer Architecture]. 縦軸に 73 年 (Intel が 8088 を開発した年) を 1 とし横軸を年代とした時の CPU 性能値の理論値と DES の暗号化にかかる時間から計算した実測値を図 2.20を示す. これらの値から, DES の暗号解析にかかるであろう時間を理論値, 実測値から求めたものを図 2.21に示す. 縦軸に 73 年を 1 とした DES の全鍵検索による暗号解析にかかる時間, 横軸に年代をとった.

図 2.20, 図 2.21からわかるように, CPU 性能の向上が非常に早く, わずか 10 年で DES の暗号解析にかかる時間が 1/10 以下になってしまった. 1995 年には, $1,000,000 のマシンを用いれば, わずか 3.5 時間で DES の解析ができてしまうようになった [DES 91]. これでは, もはや DES が安全であるとはいえない.

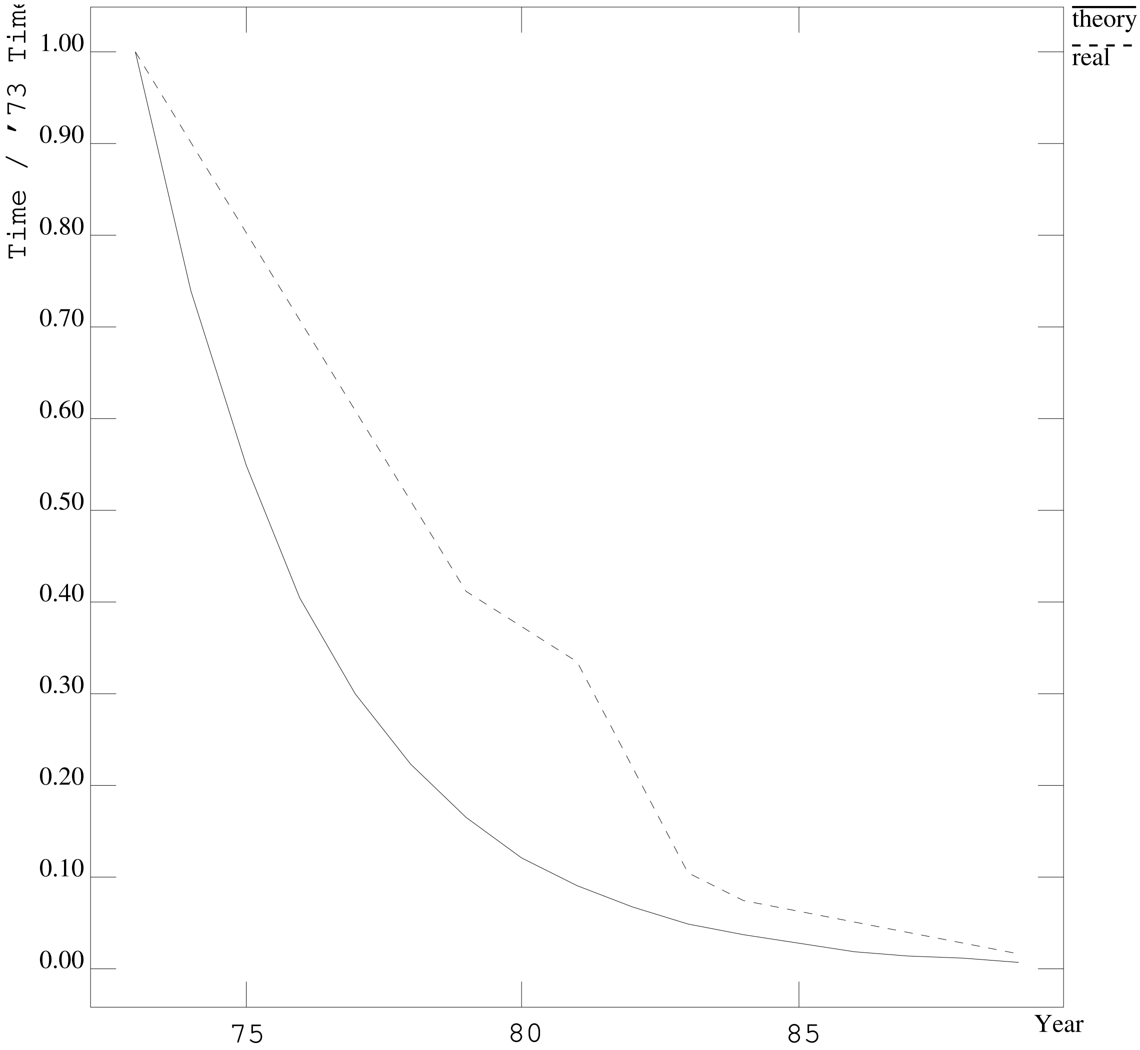


図 2.21: DES の解析時間の移り変わり

2.5 暗号システム

本節では, 前節で述べた暗号化アルゴリズムを用いた暗号システムについて述べる.

2.5.1 PGP

PGP(Pretty Good Privacy) はファイルの暗号化システム兼電子メールセキュリティシステムであり, 電子メールセキュリティシステムとして利用する際には, E-mail の本文の暗号化, 電子署名による改竄の防止とユーザの認証, そして鍵配送の機能が提供されている. 以降, 電子メールセキュリティシステムについて説明する. メッセージの暗号化には **IDEA** と **RSA** を利用し, 電子署名には一方向ハッシュ関数の **MD5** と RSA を利用している. メッセージの暗号化と電子署名はそれぞれ単独で使え, 組み合わせて使うことも可能である [PGP a][PGP b].

PGP の暗号化

PGP では, 暗号通信を行う際, 始めに IDEA の鍵をランダムに生成し, 生成された鍵を用いて E-mail の本文を IDEA で暗号化する. 暗号化を行った後, 使用した鍵を受信者の公開鍵を用いて RSA で暗号化し, IDEA で暗号化した本文と RSA で暗号化した IDEA の鍵を一緒に送信する. PGP を用いて暗号化されたメールを受け取った受信者は, 秘密鍵を用いて RSA で暗号化された IDEA の鍵を復号化し, IDEA の鍵を手にいれる. そして, 手にいれた IDEA の鍵を用いて IDEA で暗号化された本文を復号化する.

PGP の電子署名

PGP での電子署名では, 送信者は本文の MD5 をとり, MD5 の結果を秘密鍵を用いて RSA で暗号化し, E-mail の本文に添付した後, 送信する.

電子署名付きのメールを受け取った受信者は, まず RSA で暗号化されたハッシュ値を送信者の公開鍵で復号化し, 本文に対して MD5 をとり比較する. 同じであれば, メールは改竄されていない.

PGP の電子署名&暗号化

送信者は, まず本文の MD5 を計算し, MD5 の結果を秘密鍵を用いて RSA で暗号化し, 本文に添付する. 次に IDEA の鍵をランダムに生成し, 本文と電子署名を IDEA で暗号化する. 最後に, IDEA の鍵を受信者の公開鍵を用いて RSA で暗号化し, 送信する.

電子署名と暗号化が施されたメールを受け取った受信者は, まず秘密鍵を用いて RSA で暗号化された IDEA の鍵を復号化して, IDEA の鍵を取り出す. 取り出した IDEA の鍵を用いて, IDEA で暗号化された本文と電子署名を復号化する. 最後に送信者の公開鍵を用いて電子署名を RSA で復号化し, 取り出した MD5 と本文から計算した MD5 を比較する.

鍵配送

公開鍵自体が送信される際に, 改竄される恐れがある. そこで, 安全な鍵配送が必要となる.

PGP では, 公開鍵の指紋というアイディアで安全な鍵配送を実現している. 公開鍵の指紋は, 公開鍵に対して MD5 を施した 16 Byte の数値である. ある 2 人 (A, B) が公開鍵を交換する時, A と B は電話等であらかじめ公開鍵の指紋を交換しておき, 公開鍵自体を E-mail で交換する. A, B はそれぞれ, E-mail で送られてきた公開鍵の MD5 を計算し, 改竄されていないことを確認する.

PGP の問題点

PGP では, 盗聴, 改竄, なりすましは防止できるが, いやがらせ目的の電子メールの送信だけは防ぐ事ができない.

2.5.2 PEM

PEM(Privacy Enhanced Mail) は電子メールセキュリティシステムであり, DES と RSA による本文の暗号化, RSA と MD5 による電子署名, 公開鍵証明書によるユーザ認証を提供している [PEM 95][PEM].

公開鍵証明書は, 信頼できる発行局によって電子署名された個人情報と公開鍵からなりたっている. 証明書の中の情報は誰でも読むことはできるが, 発行局の秘密鍵が盗まれない限り偽造は不可能である.

PEM は, 鍵配送以外は PGP と同じである.

PEM の鍵配送

PEM では, ユーザはあらかじめ公開鍵証明書を発行する発行局に公開鍵を登録しておかなければならない.

PEM では, 公開鍵は, 必ず公開鍵証明書の形式で取り扱われる. PEM は, 公開鍵証明書の形式で公開鍵を取り扱う事により, 正規の公開鍵を偽物とすり換えて他人になりすます不正行為を防止している.

メッセージの送信者は, 自分の公開鍵と受信者の公開鍵に対する要求を発行局に要求する. 発行局は, 送信者の公開鍵証明書を送信者に送信し, 配布局に送信者に受信者の公開鍵証明書を送信する要求をだす. 要求を受け取った配布局は, 受信者の公開鍵証明書を送信者に送信する.

PEM の問題点

PEM では, 利用者は予め発行局に公開鍵を登録しなければならないが, 鍵の発行局や鍵の配布局のスケーラビリティが問題となっている.

また, PEM は PGP と同様に盗聴, 改竄, なりすましは防止できるが, いやがらせ目的の電子メールの送信は防ぐことができない.

2.5.3 全体的な問題点

以上のように, 暗号化アルゴリズムを用いたさまざまなシステムが提供されてきている. しかし, 利用者であるはずのユーザのセキュリティに対する関心が弱いため, これらのシステムは有効に活用されていない. 現在のネットワークは盗聴やトラフィック解析など, 不正行為が当たり前のように行われており, 非常に危険である. したがって, ユーザは, ネットワークを頻繁に利用し, ネットワーク上で商業活動を行ったりする以上, セキュリティ確保に対して注意し, 積極的に暗号システムを活用しなければならない.

しかし, 現在提供されている暗号システムを用いて通信を行う際には, 送信者と受信者の間で, 暗号方式に関する合意が取れている必要がある. 以上に説明した, PGP にしても PME にしても, 送信者, 受信者共に PGP なり PEM を所持しており, 公開鍵を公開しておかなければならない. したがって, 利用する暗号システムを送信時に自由に選ぶといった柔軟性はない.

また, 暗号システムは, いくつかの暗号化アルゴリズムの組み合わせにより成り立っているため, 暗号化アルゴリズムが弱くなるに従って暗号システムも弱くなってしまう.

2.6 MIME

MIME (Multipurpose Internet Mail Extensions) [mime I] [mime II] [mime IV] [mime V] は, 1992 年に RFC1341 において規定された電子メールにおけるデータ処理の

RFC 822	MIME
文字集合が明確に規定されていない. (US-ASCII の定義があいまい)	文字集合が明確に規定されている.
ヘッダに非アルファベット文字の挿入が不可能.	非アルファベット文字を別の文字列に変換してヘッダに挿入する.
多国語を考慮していない. (US-ASCII の使用を仮定. または送受信者間の合意が必要)	MIME-Version と Content-Type により多国語の使用が可能.
マルチメディア (音声/画像) 電子メールは不可能.	MIME-Version と Content-Type によりマルチメディア電子メールが可能. バイナリデータには符号化を行う.
マルチパートが不可能.	MIME-Version と Content-Type によりマルチパートが可能.

表 2.6: RFC822 規格と MIME 規格

統一仕様である. MIME が規定されるまでは, 1982 年に定義された RFC822 [RFC 822] が統一仕様であった. RFC822 メールは, 文字情報の交換の手段として世界的に普及していた. しかし, RFC822 には以下のような問題点があった.

- US-ASCII という文字集合の使用を仮定していたが, 文字集合の定義が曖昧だった.
- ヘッダに非アルファベット文字を挿入することが不可能.
- 多国語の使用が不可能.
- 音声, 画像などマルチメディアのデータ送信が不可能.

MIME は, 以上の問題点を解決するために新たに考案された規定である. (表 2.6)

RFC822 メールはヘッダと本文を空行で区切るという単純な形式であり, ヘッダに非アルファベット文字を挿入する事はできなかった. MIME では, 日本語等, US-ASCII 文字に定義されていない非アルファベット文字を安全に送ることが可能な文字列に変換し, 挿入することにより, 非アルファベット文字をヘッダに挿入する事が可能となった.

また, RFC822 メールでは, 本文が US-ASCII 文字以外の文字列から成り立つことができず, US-ASCII 文字以外を使用する際にはあらかじめ送受信者間の合意が必要であった. MIME では, 使用する文字コードを明記することで多国語に対応している.

RFC822 メールでは, 本文に文字情報以外の情報を含むことができなかった. MIME では, RFC822 メールとの互換性を持ったまま, 音声や画像などの文字情報以外の情報も本

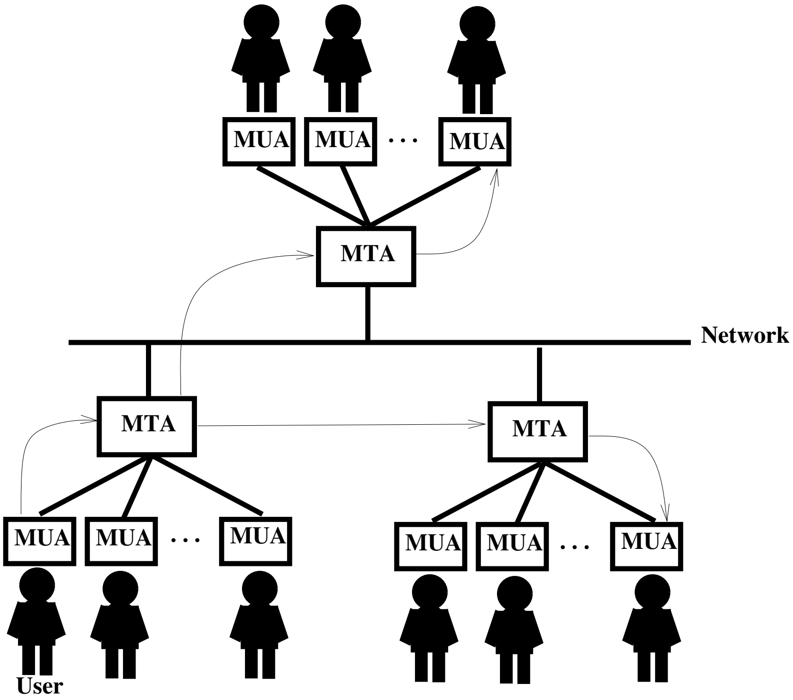


図 2.22: MUA と MTA

文に含むことができるようになっている. MIME では, 境界となる文字列で本文をパートと呼ばれる単位に区切り, 各パート毎に音声や画像等のメディアを取り込む事が可能である. その構造は階層的に定義されており, 各パートが, さらに複数のパートから成立する事も可能である.

MIME では, RFC822 で定められたメッセージヘッダに内容の種別を表す以下のようなフィールドを追加し, **MUA (Mail User Agent)** に複雑なメッセージの扱いを任せることによって, 文字情報以外の情報等が扱えるようにしている. MUA とは, ユーザが E-mail の読み出し, 応答, 作成, 処理をするために実行するプログラムであり, 呼び出された MUA は, **MTA (Mail Transport Agent)** にメールを渡したり, MTA からメールを受け取ったりする. MTA は, 多数のユーザのメール配信を処理し, マシン間で E-mail を転送する. MUA ・MTA の概念図を図 2.22に示す.

MIME-Version : 電子メールにラベルを付け, MIME 仕様に準拠したメールであることを示す時に使われる. 受信側のメッセージを処理する MUA は, この情報にもとづいてメッセージを処理する.

Content-Type : メッセージに含まれるオブジェクトのタイプやサブタイプの種別を

表すために使われる. サブタイプでは, オブジェクトの性格をより詳細に伝えるパラメータも指定できる. これによって受信側の MUA はメッセージの内容を容易に識別できる.

MIME の仕様では, 以下の 7 つの特別な Content-Type があらかじめ世界標準として定義されている.

Text : 受信者が, 送信者が送った書式と同様に表示された電子メールを読むことができるようにするために使われる. これは Content-Type のデフォルトである. サブタイプはデフォルトとして plain(プレインテキスト) が設定されている. 重要なパラメータは charset で, そのテキストの文字セットを指定する.

Multipart : 異なるコンテントタイプやサブタイプを含め, 本文が幾つかに分かれている電子メールを送る時に使われる. これのおかげで, 受信者側の MUA は各部分を正しく復元処理できる. サブタイプには次のようなものがある.

- Mixed : 各部分が順番に処理されるようにするために使われる.
- Parallel : MUA によって各部分が同時に処理されるようにするために使われる.(音声と動画の同時再生など)
- Alternative : 同一のデータを複数の異なる形式で表すために使われる.
- Digest : 各部分が message 型であることを表すために使われる.

Message : カプセル化されたメッセージを指定するために使われる. サブタイプには次のものがある.

- rfc822 : RFC822 に準拠したメッセージの全体をカプセル化する場合に使われる.
- partial : 非常に大きなメッセージを送る場合に使われる. Message/partial と指定すると, 非常に大きなメッセージを小さく分割できる. 分割されたメッセージは受信側の MUA が一つにまとめて復元する.
- external-body : ローカルまたはネットワーク上の別のサーバのファイルを MUA が参照できるようにするために使われる.

Image : FAX やプリンタ, また画像表示が可能なウィンドウ・システムなどを必要とする画像データを送る場合に使われる. 次のサブタイプがある.

- jpeg : ISO/IEC 10918 圧縮レンダリングを用いたフルカラーやグレースケールの画像の場合に使われる.

- gif : CompuServe の GIF(Graphical Interchange Format) 形式を用いた画像の場合に使われる.

Video : 専用のハードウェアやソフトウェアを用いて再生する動画を送る時に使われる.

- mpeg : ISO/IEC 11172 に基づく動画の場合に使われる.

Audio : 音声データを送る時に使われる. サブタイプは basic が使われる

Application : MUA が直接解釈できないバイナリデータである時に使われる. 次のようなサブタイプがある.

- octet stream : これ以外のコンテントタイプに該当しないバイナリデータをカプセル化する時に使われる.
- postscript : Adobe の編集不可能な文書フォーマットの時に使われる

新たにコンテントタイプを追加したい場合は **IANA(Internet Assigned Numbers Authority)** に登録する必要がある. ”X-サブタイプ名”なるヘッダを利用すれば独自のサブタイプが追加できる. 以上 Contenty-Type とサブタイプをまとめると表 2.7になる.

Content-Transfer-Encoding : コンテントタイプの多くは, 8 bit の文字またはバイナリデータで表現される. しかし, インターネットで利用されている **SMTP(Simple Mail Transfer Protocol)** では 7 bit のデータ (US-ASCII) で, 1 行 1000 文字以内という制限があり, コンテントタイプ・データは使えない. そこで, MIME では 8 bit データやバイナリデータを 1 行 1000 文字以内の 7 bit データに符号化して, MUA でコンテントタイプを読む時点で元に戻している. このプロセスで最も重要となるのが, Content-Transfer-Encoding という符号化手法を表す MIME ヘッダである.

Content-Transfer-Encoding の値は次のものに限られている.

quoted-printable : US-ASCII 文字セットのうち印字可能な文字から構成されるデータを示す. quoted-printable 符号化により, 特定の ASCII 文字を含む ASCII データも利用できるようになる.

base64 : base64 符号化は, バイナリデータの表現にもっとも適している. 独自の base64 符号を用いて 3 Byte 毎に印字可能な 4 つの ASCII 文字に変換する.

7 bit : RFC821 で規定された US-ASCII 文字のみからなるデータであることを示す.

8 bit : US-ASCII 以外の文字を含む可能性があることを示す.

binary : US-ASCII 以外の文字が含まれたメッセージで, 1 行が 1000 文字を越える可能性があり, SMTP 転送では支障をきたすかもしれないことを示す.

x-token : 非標準の Content-Transfer-Encoding を利用できるようにする.

符号化されるのは, quoted-printable と base64 だけである. 7 bit と 8 bit, binary はメッセージに含まれるオブジェクト内にどのようなデータがあるか, またはゲートウェイでどういう符号化が必要かを示すために利用される.

Content-ID : Content-ID について述べる前に, MIME の外部参照機能 (Content-Type: Message/External-Body) について述べる.

外部参照機能 ”Content-Type: Message/External-Body” による外部参照機能は, MIME の最も注目すべき強力な特徴であり, これによって MUA を強力な情報アクセス/検索のための中枢機能として利用できるようになっている.

Content-Type: Message/External-Body のパラメータには次のものがある.

access-type : 外部参照ファイルの検索に用いるアクセス・メカニズム (ftp, anon-ftp 等) を示す.

expiration : RFC822 で規定された形式で記された日時を示す. 外部参照ファイルは, ここで指定された日まで参照先にあることが保証されるが, 指定日を過ぎると, ファイルは消去される可能性がある.

size : データバイト数を示す.

permission : 受信者にデータの読み書きの権限があるかどうかを示す.

name : 外部参照ファイルの名前を示す.

site : ホスト名だけでなく, ドメイン名も含んだ **FQDN (Fully-Qualified Domain Name)** で記述された外部参照ファイルの在処を示す.

directory : 指定された外部参照ファイルが置いてあるディレクトリを示す.

mode : 外部参照ファイルの検索時に用いられるモード (ASCII, NETASCII 等) を示す.

表 2.7: MIME の Content-Type とサブタイプ

Content-Type	サブタイプ	パラメータ	符号化法
Text	plain	charset	quoted-printable
Multipart	mixed,alternative digest,parallel	boundary	使用不可
Message	rfc822,partial external-body	id,number total,size directory server 等	使用不可
Application	octet stream post-script	type padding	base64
Image	jpeg,gif	なし	base64 を推奨
Audio	basic	なし	base64 を推奨
Video	mpeg	なし	base64 を推奨

以上の外部参照機能を用いた際, メールサーバを通じて検索した情報を含む電子メールは, 要求後しばらく経ってから届く可能性がある. したがって, 送られてきた電子メールがどの MIME メッセージによる検索結果であるかを識別する必要があり, ここで Content-ID が使われる.

Content-Description : ”Content-Type : Image”の情報を”Content- Type : Text”の情報と関連を持たせるために使われる.

2.6.1 MIME-PEM ・MIME-PGP

最初, MIME と PEM, MIME と PGP はそれぞれ独自に構造を持つメールを規定していた. したがって, 両者を共存させることが不可能であった. そこで, 現在 PEM では Content-Type:の値として application/pem-1421 を使い, PEM のヘッダである Content-Domain:の値として新たに MIME を定義し MIME と PEM の統合を図っている [mew-PEM] [mew-PGP]. 将来的には次の方法を採用することで MIME と PEM の統合を図る.

- Content-Type:として Multipart/Security を定義する
- 署名用の Content-Type:として Application/Signature を定義し, Multipart /Security の中で用いる

- 暗号用の Content-Type:として Application/Keys を定義し, Multipart/ Security の中で用いる

以上の方法を採用することにより, より MIME との親和性が高く,PEM の機能が高くなる.

また, 現在 PGP では Content-Type:の値として application/pgp を定義し, 更に, 復号後の処理をパラメータで与えている.

2.7 Mew

Mew (Message interface to Emacs Window) は MIME や PEM, PGP のように本文に構造を持つメールにたいして, Emacs に特化した操作を提供するメッセージインターフェイスであり, 山本ら [mew 94b] によって開発されたもである. Mew は次のような特徴を持っている.

- 粒度の細かい MIME 操作の提供
- ディレクトリ構造を multipart へ変換する構成器の機能を提供
- PEM の自動復号, および暗号化と署名手続きの提供
- PGP の自動復号, および暗号化と署名手続きの提供
- PGP/MIME の統合
- LRU アルゴリズムに基づいたメッセージのキャッシュ
- 非同期の inc と scan
- 高水準な refile folder 推測機能
- alias の補完と展開
- pick の使いやすいインターフェイス
- さまざまなマーク機能

2.8 暗号化の問題点のまとめ

現在, 提供されている暗号化アルゴリズム, 暗号システムを用いて通信を行う際には, 送信者と受信者の間で, 使用する暗号化アルゴリズム, 暗号システムに関する合意が必要である. したがって, 通信内容の重要性とマシンの性能に合った暗号化アルゴリズム, 暗号システムを送信時に自由に選択し, 使えるといった柔軟性はない.

また, 現在提供されている暗号化アルゴリズム, 暗号システムは, 時が経ち, CPU の性能の向上に伴い, 安全強度が弱くなっていく. 現在提供されている暗号システムは, 使用する暗号化アルゴリズムが固定されており, 暗号化アルゴリズムの安全強度が弱くなると, 暗号システムも使い物にならなくなる.

さらに, 暗号化アルゴリズムには, 鍵配送の方法が提供されていない. 暗号化アルゴリズムを利用して暗号通信を行うためには, 送信者・受信者間であらかじめ鍵を共有していなければならない.

PGP・PEM に関しては Mew 等のインターフェイスが提供されているが, 他の暗号化アルゴリズム, 暗号システムには使いやすく, 便利なユーザインターフェイスが提供されていない.

第 3 章

モデル

前章で, 現在提供されている暗号化アルゴリズム及び暗号システムの機構と問題点について述べてきた. 本章では, これらの問題に対する解決方法, そして, 本論文で採用する方法, モデルについて述べる.

暗号化アルゴリズムおよび暗号システムの問題点は次の 4 つが挙げられた.

問題点 1 送信者・受信者間で, 使用する暗号化アルゴリズム・暗号システムに関する合意が必要.

問題点 2 暗号化アルゴリズム・暗号システムは, CPU の性能向上に伴い, 時と共に弱くなっていく.

問題点 3 多くの暗号化アルゴリズム・暗号システムには, 便利なユーザインターフェイスが提供されていない.

問題点 4 暗号化アルゴリズムには, 鍵配送の方式が提供されていない.

3.1 問題に対するアプローチ

ここでは, セキュリティおよび暗号化アルゴリズムに関する問題点に対処する方法を挙げ, その長短について議論する.

問題点 **1**

送信者と受信者の間で使用する暗号化アルゴリズム・暗号システムに関する合意が必要であるという問題に対して次のアプローチが考えられる.

1. ネットワークの利用者全員が単一の暗号化アルゴリズム・暗号システムを利用するということで合意する.
2. ネットワークの利用者全員にいくつかの暗号化アルゴリズム・暗号システムを所持することを義務付ける
3. 暗号化に使用した暗号化アルゴリズムに対応する復号化アルゴリズムを, 暗号化したメッセージと一緒に受信者に送る.

1 の方法には, 次のような問題がある.

- 時が経つと共に安全強度が弱くなる.
- 最も強い安全強度を必要とする分野に合わせなければならないので, 計算コストがかかる.

単一の暗号化アルゴリズム・暗号システムを使用していると, 使用しているアルゴリズムが時とともに, CPU の性能が向上し, 安全強度が弱くなってしまう. 時が経つと共に安全強度が弱くなることに対応するためには, 安全強度が弱くなった暗号化アルゴリズム・暗号システムを次の新たなる安全強度が強いものに交換していかなければならないが, ネットワークの利用者全員が一斉に新しい暗号化アルゴリズム・暗号システムに交換することは非常に困難である. また, 使用する暗号化アルゴリズム・暗号システムを 1 つに決めてしまう時には, 最も安全強度の強い暗号化アルゴリズム・暗号システムにせざるを得ず, 計算コストがかかることに耐えなければならない.

2 の方法には, 次のような問題がある.

- 非常に多くのネットワークの利用者に対し, 暗号化アルゴリズム・暗号システムの所持を義務付け, これを徹底させることは, 非常に困難
- ユーザは全暗号化アルゴリズム・暗号システムを所持していなければならず, 管理が大変である.

この方法は, 使用する暗号化方式に関して送信者と受信者の合意が必要なく, データの重要性などに合わせて使用する暗号化アルゴリズムが選べ, 柔軟性にも優れている. しかし, ネットワークの利用者が, 急速に増加した今日では, 暗号化アルゴリズム・暗号システムの所持を義務付け, これを徹底することは, 非常に難しく, 現実問題として不可能である.

またネットワークのユーザ全員に全ての暗号化アルゴリズム・暗号システムを所持することを義務付けると, 幾つもある暗号化アルゴリズム・暗号システムを管理・維持していかなければならない. また, 常に新たな暗号化アルゴリズム・暗号システムの情報に注意をし, 新たな暗号化アルゴリズム・暗号システムが開発された際には, 手に入れ, 使えるようにしなければならない. 暗号化アルゴリズム・暗号システムを管理・維持・運用していくことはユーザに多大なる負荷をかける.

3 の方法では, 暗号化アルゴリズムを送信することによるトラフィックの増大が問題となるが, 3 の方法を利用すれば, 使用する暗号化アルゴリズムに関する受信者との合意なしに, 送信者はデータの重要性等に合わせた暗号化アルゴリズムを使用する事ができる. しかも, 受信者は使用する暗号化アルゴリズムに関して合意をしていないにも関わらず確実にデータを復号化できる. 3 の方法は, データの重要性等に合わせた暗号化が行えるため, 柔軟性に優れ, 2 の方法より現実的である.

アプローチ 1 ~ 3 の中でアプローチ 3 が, 最も優れ適当であると思われる. そこで, 本論文では問題 1 の解決策として 3 番目の方法を採択する.

問題点 **2**

暗号化アルゴリズム・暗号システムは, 時と共に弱くなっていくという問題に対して次のアプローチが考えられる.

- 安全強度の弱くなってしまった, 暗号化アルゴリズム・暗号システムの使用をあきらめ, 新たな安全強度の強い暗号化アルゴリズムを使用する.

現在提供されている暗号化アルゴリズム・暗号システムは, ”鍵の取り得る値の範囲” と”復号化にかかる計算量” によって安全強度を保っている. 安全強度が弱くなってしまった暗号化アルゴリズム・暗号システムは, 安全強度を強くするために以下のいずれかの方法を取らなくてはならない.

- 鍵の長さを長くする.
- 暗号化の処理を複雑にする.

上記のいずれの方法も, 新たな暗号化アルゴリズム・暗号システムを構築し直す事である. つまり, 安全強度が弱くなってしまった時には, 新たな暗号化アルゴリズム・暗号システムを使用するしか, 解決策はない.

安全強度を保持するためには, 新たな暗号化アルゴリズム・暗号システムを導入しなければならないが, 新たな暗号化アルゴリズム・暗号システムを導入する際に問題点 1 が関係してくる. たとえ, 送信者が安全確保のために新たな暗号化アルゴリズム・暗号システムを導入しても, 受信者との合意がなければ, 送信者は使用することができない. 送信者は, 新たな暗号化アルゴリズム・暗号システムを使用することの合意を取ってから新たな暗号化アルゴリズム・暗号システムを使用するか, もしくは今まで使用していた暗号化アルゴリズム・暗号システムを使用するしかない.

従って, 本論文では, 暗号化アルゴリズム・暗号システムが時と共に弱くなっていく事に対応するために, 新たな暗号化アルゴリズム・暗号システムを導入し, 送信者は, 暗号化されたデータと共に暗号化アルゴリズムを送信する方法を採択する.

問題点 **3**

PGP や PEM 等, 一部の暗号システムにはインターフェイスが提供されているが, 多くの暗号化アルゴリズム・暗号システムには便利で使いやすいインターフェイスが提供されておらず, ユーザはコマンドラインからの操作を余儀なくされている. 既に提供されているインターフェイスの一つに Mew がある. Mew は, 使いやすく, 便利な MIME のイン

ターフェイスとして開発され, PGP と PEM が MIME と統合されることにより, Mew が PEM と PGP のインターフェイスとして機能している.

本論文では, 現在の使いやすく, 便利なユーザインターフェイスを利用し, 様々な暗号化アルゴリズムを使用できるように拡張することにより, 問題点 3 を解決する.

問題点 4

現在, 幾つかの鍵配送システムが提供されているが, いずれの鍵配送システムも公開鍵暗号系の暗号化アルゴリズムを用いており, 送信者・受信者はあらかじめ使用する鍵配送システムに関して合意していなければならない. あらかじめ使用する鍵配送システムに関して合意をせずに鍵配送システムを使用することは, 非常に困難であり, 今後更なる研究が必要である.

本論文の時点では, 送信者・受信者は既に鍵を共有しているものと仮定する.

3.2 EMAAS モデル

3.2.1 概要

前節で各問題点に対するアプローチについて考察した. そこで, 本論文で提案するシステムは, 前節で考えられたアプローチをふまえ次のような機能を持つようにする.

- システムは, 様々な暗号化アルゴリズムを利用することが可能な, 便利で使いやすいインターフェイスを有する
- システムは, 送信者がデータを送る度に送信者にデータの暗号化を促す. システムが送信者にデータの暗号化を促す際には, 送信者がデータの重要性, 送信者のマシン性能, 受信者のマシンの性能を考えて, 使用する暗号化アルゴリズムを自由に選べるようにする
- システムは, 選択された暗号化アルゴリズムを用いて自動的にデータを暗号化する
- 従来は暗号化したデータだけを送信していたところを, 暗号化したデータと, 使用された暗号化アルゴリズムに対応する復号化アルゴリズムを一緒に送信する.(図 3.1)
- 受信者側のシステムは, ユーザが送られて来たデータを利用する際に, 送られて来た復号化アルゴリズムを評価し, 暗号化されたデータを自動的に復号化する

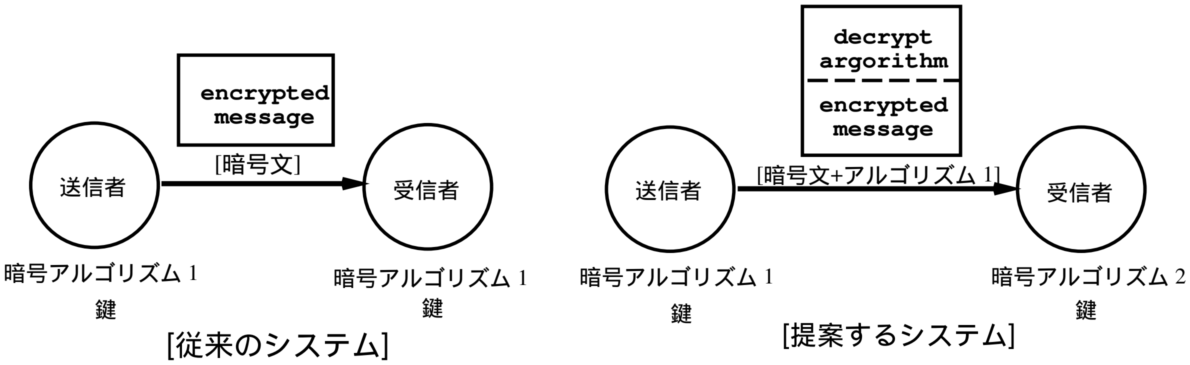


図 3.1: 従来のシステムと提案するシステム

本論文が取る方針の概略図は, 図 3.2となる. 送信者側において, データは暗号化され, 使用した暗号化アルゴリズムに対応する復号化アルゴリズムと共に送られる. 受信者側では送られて来たデータを復号化アルゴリズムの部分と暗号化されたデータに分割し, 送られて来た復号化アルゴリズムを使ってデータを復号化する.

本論文が提案する方法を用いると, ユーザはデータを暗号化する際に, 相手が使用するべき復号化アルゴリズムを持っているかどうかを考えずに, 暗号化を行う事ができる.

従来は, 文書の重要性に合わせた暗号化を行うのは困難であったが, 本論文が提案する方法では文章の重要性, 受信者のマシンの性能, 送信者のマシンの性能を考えた上で適切な暗号化を行う事が可能である.

3.2.2 基本方針

本論文では新しいシステム EMAAS(Encrypted Message And Algorithm System) を提案する. 本論文で提案するシステムの時間経過は図 3.3のようになる.

送信者・受信者は幾つかの暗号化アルゴリズムを所有しており, また送信者・受信者は各自が使用するマシンの性能に関する情報を所有してる.

送信者が, 受信者にデータないしはメッセージを暗号化して送る際, 次の手順で送信され, 受信側で復号化される.

1. ネゴシエーションを行い, 受信者のマシンの性能, 受信者の持っている暗号化アルゴリズム・復号化アルゴリズムに関する情報を入手する. ネゴシエーションの手順は次小節で述べる.
2. 送信者は, 送信するデータの重要性とネゴシエーションによって得た情報より, 使用する暗号化アルゴリズムを選択する.

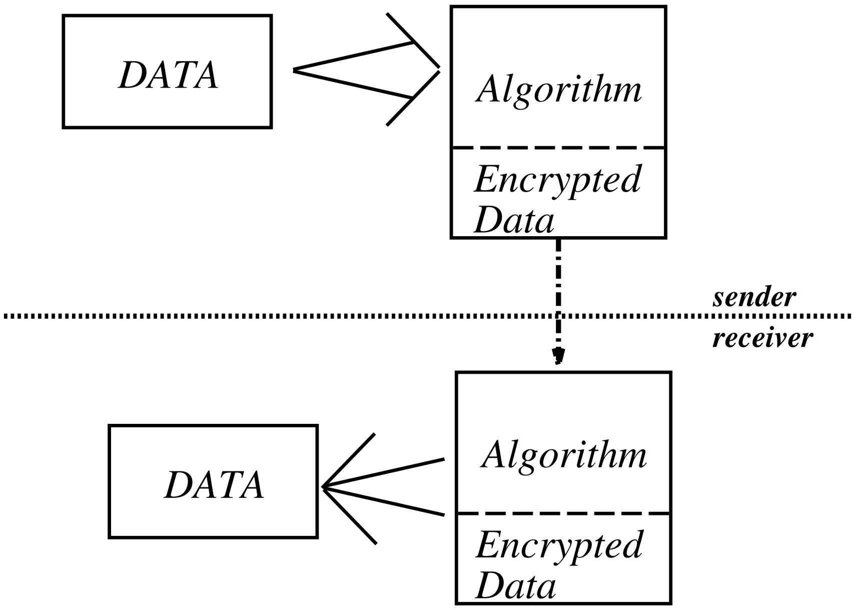


図 3.2: 提案するモデル

3. 送信者は, 選択した暗号化アルゴリズムでデータを暗号化する.

4. 暗号化したデータと選択した暗号化アルゴリズムに対応する復号化アルゴリズムを受信者に送信する. ただし, ネゴシエーションの結果, 使用する暗号化アルゴリズムに対応する復号化アルゴリズムを受信者が所持している場合には復号化アルゴリズムを送信しない.

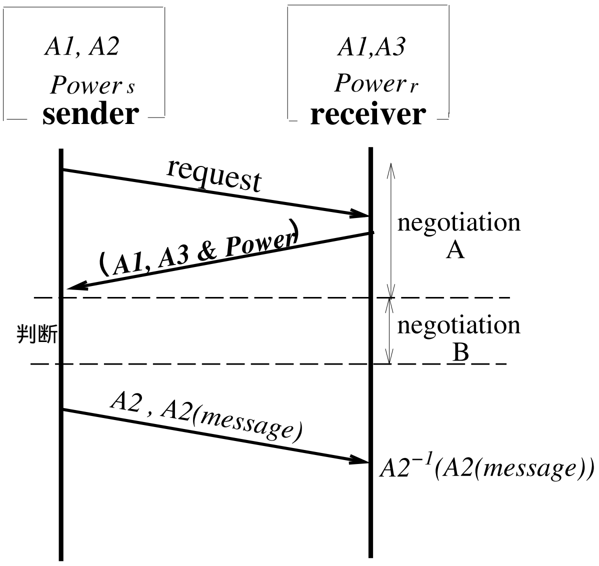
5. 受信者は, 復号化アルゴリズムと暗号化されたデータが送信されてきた場合, 復号化アルゴリズムと暗号化されたデータに分ける.

6. データの暗号化に使用された暗号化アルゴリズムに対応する復号化アルゴリズムを実行し, 暗号化されたデータを復号化する.

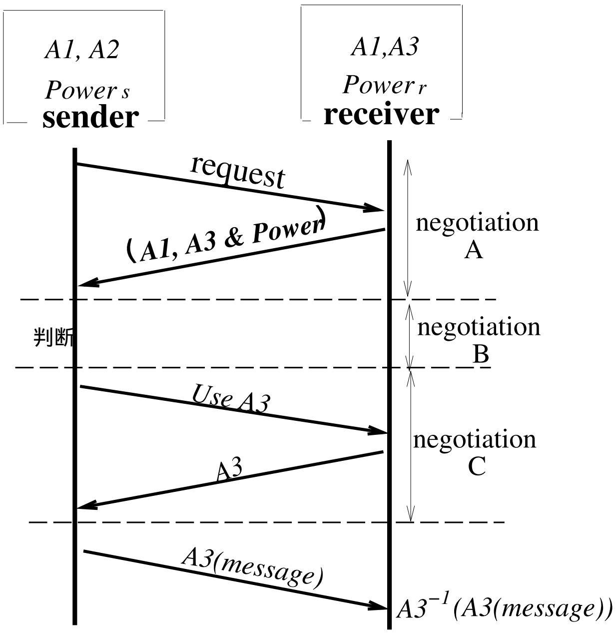
データの暗号化・復号化, 暗号化されたデータと復号化アルゴリズムの結合・分離は全てシステム EMAAS によって自動的に処理される.

3.2.3 ネゴシエーション

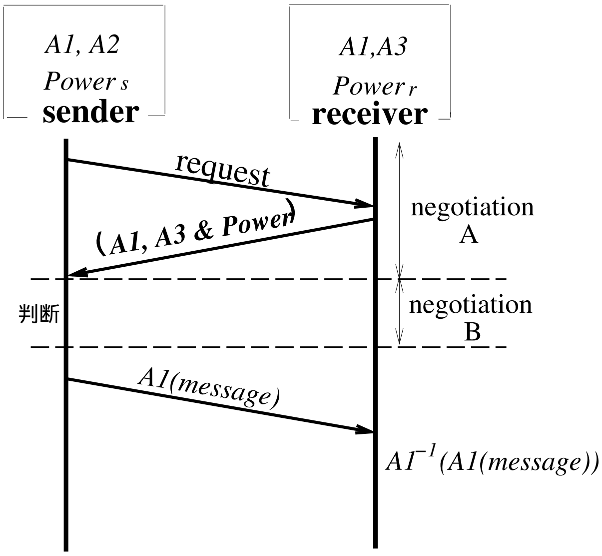
送信者は, 受信者とネゴシエーションをする事により, 受信者が持っている暗号化アルゴリズム・復号化アルゴリズム, そして受信者のマシンの性能に関する情報を入手する.



送信者の持っているアルゴリズムを利用
(pattern 1)



受信者の持っているアルゴリズムを利用
(pattern 2)



両者の持っているアルゴリズムを利用
(pattern 3)

図 3.3: システムの時間経過

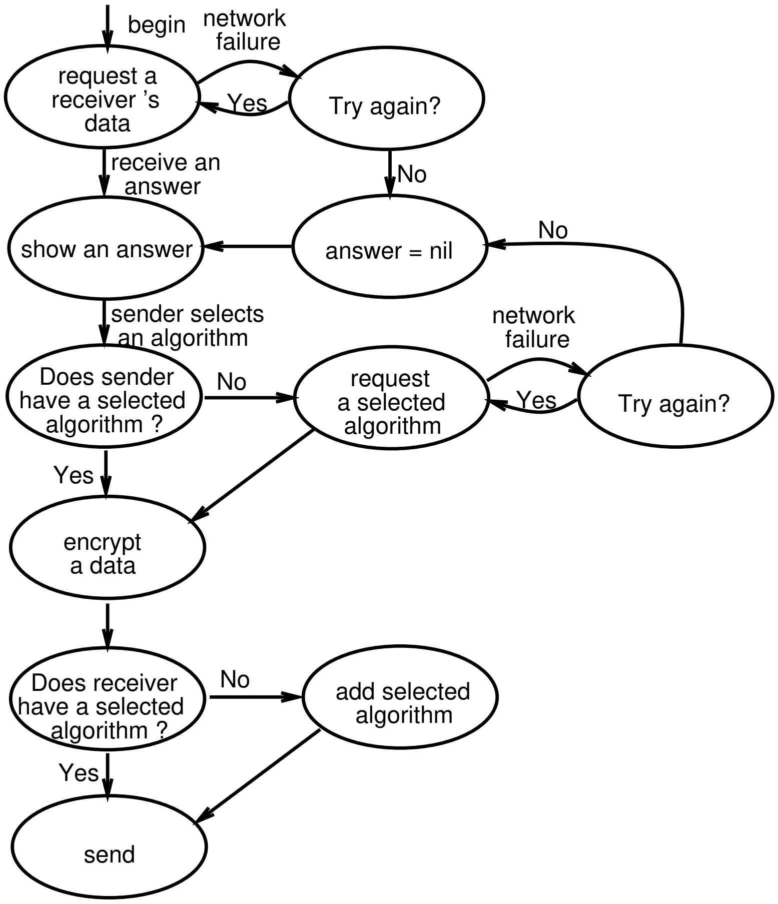


図 3.4: 送信者側の negotiation 状態遷移図

ネゴシエーションを経て, 送信者が, データを受信者に送るまでの状態遷移は図 3.4となり, 実際の手順は以下のようになる.

1. 送信者は, 受信者が持っている暗号化・復号化アルゴリズムとマシン性能の情報に対する要求を受信者に出す.(図 3.3 negotiation A In pattern1, pattern2, pattern3)

2. 要求を受け取った受信者は, 受信者が持っている暗号化・復号化アルゴリズムとマシンの性能について調べ, 送信者に返答としてこれらの情報を返す.(図 3.3 negotiation A In pattern1, pattern2, pattern3)

3. 返答として, 受信者が持っている暗号化アルゴリズムとマシン性能の情報を受け取った送信者は, 送信するデータの重要性, 送信者のマシン性能, 受信者のマシン性能, 使用できる暗号化アルゴリズムから, データの暗号化に使用する暗号化アルゴリズムを選ぶ.(図 3.3 negotiation B In pattern1, pattern2, pattern3)

4. 送信者が選んだ暗号化アルゴリズムが, 受信者しか所有していない暗号化アルゴリズムであれば, 送信者は受信者に使用する暗号化アルゴリズムを送信者に送るように要求を出す.(図 3.3 negotiation C In pattern2)
5. アルゴリズムを送るように要求を受けた送信者は, 返答として, 要求された暗号化アルゴリズムを送る.(図 3.3 negotiation C In pattern2)
6. 送信者は選んだ暗号化アルゴリズムを用いて. データを暗号化して送信者に送る. この時, 受信者が持っていない暗号化アルゴリズムを使用した場合には暗号化アルゴリズムを一緒に送る.(after negotiation In pattern1)

受信者に要求を出した時に, ネットワーク障害が起こり, ある一定時間が経っても返答が返って来なかった時には, 送信者は再度要求の送信を試みるか, 受信者の情報を得ることを諦めるかを選択する. 送信者が, 受信者の情報を得ることを諦める場合は, 送信者が持っている暗号化アルゴリズムの中から使用する暗号化アルゴリズムを選択し, データを暗号化した後, 使用した暗号化アルゴリズムと一緒に送信する.

マシン性能

マシンの性能値としては, MIPS 値, CPU クロック値等さまざまなものが考えられる. マシンの性能値として MIPS 値や CPU クロック値を使用する際には, ユーザは自分の使用するマシンの MIPS 値や CPU クロック値を把握して, 手動で登録しなければならない. しかし, ユーザ全てが自分のマシンの MIPS 値や CPU クロックを把握しているわけではない.

MIPS 値, CPU 値を把握していないユーザも多くいるため, マシンの性能値としては, マシンが性能値を自動計測して自動的に登録できるような値が望ましい. そこで本論文では, EMAAS が起動される度に, 簡単なベンチマークを計測し, この計測値の平均をマシンの性能値とする. 簡単なベンチマークの平均値をマシンの性能値として採用すると, 次のような利点がある. ワークステーションのような分散環境を考えた時, ニュースサーバやプリンタサーバ等の役割をもったマシンも存在する. 何かしらの役割を担っているマシンは, ユーザにマシン本来の性能全てを提供できない. したがって, MIPS 値や CPU 値をマシンの性能値とした時には, マシン本来の性能値がマシンの性能値として登録され, 実際にユーザに提供されるマシンの性能値から遠くかけ離れた値として登録されてしまう. ところが, 簡単なベンチマークの平均値をマシンの性能値とした時には, 実際にユーザに提供されているマシンの性能値に近い値が登録される. システムの使い始めは正確な

マシンの性能値となりにくいが, 使用し続けるうちに値が平均化され, 非常に正確な値に近い値となると考えられる.

第 4 章

実装

本章では, 本論文で行った実装について述べる.

4.1 E-Mail 配送システムにおける EMAAS の実装

前章で, 便利で使いやすいインターフェイスを有した, 暗号化されたデータと共に, 使用した暗号化アルゴリズムに対応する復号化アルゴリズムを, 送るシステム EMAAS を提案した. この EMAAS は, 以下の機能を有している.

- システムは, 送信者がデータを送る度に送信者にデータの暗号化を促す.
- システムが送信者にデータの暗号化を促す際には, 送信者がデータの重要性, 送信者のマシンの性能, 受信者のマシンの性能を考えて, 使用する暗号化アルゴリズムを選べる.
- システムは, 選択された暗号化アルゴリズムを用いて自動的にデータを暗号化する.
- 選択された暗号化アルゴリズムに対応する復号化アルゴリズムを, 暗号化したデータと一緒に送信する.
- 受信者側では, 復号化アルゴリズムと暗号化されたデータを分け, 復号化アルゴリズムを用いて, データを自動的に復号化する.

本論文では, EMAAS を E-Mail 配送システムにおける MUA 上に実装する. インターフェイスには既存の Mew を利用し, 以上の 5 つの機能を有した EMAAS を組み込んで拡張する. Mew は, 機能性にすぐれた, 便利なメッセージインターフェイスであるので, 本論文では, インターフェイスとして Mew を利用し, 拡張することにした.

E-Mail 配送における EMAAS の概念図を図 4.1に示す. 送信者側の EMAAS の実行手順は以下の通りである.

1. 送信者は, Mew のメール送信モードを選び, 受信者のメールアドレスを入力する.
2. EMAAS は, 受信者が所持している暗号化・復号化アルゴリズムと受信者のマシンの性能に関する情報を入手するために, 受信者とネゴシエーションを行う. ネゴシエーションについては, 次節で説明する.
3. 送信者は, EMAAS がネゴシエーションを行っている間に, 受信者に送るメッセージを記入する.

4. 送信者は, 使用する暗号化アルゴリズムを選択する.
5. 送信者が, メッセージを書き終わると, EMAAS は送信者の記入したメッセージを, 選択された暗号化アルゴリズムを用いて暗号化する.
6. 暗号化したメッセージと使用した暗号化アルゴリズムに対応する復号化アルゴリズムを結合する. ただし, 受信者が使用した暗号化アルゴリズムに対応する復号化アルゴリズムを持っている時には, 復号化アルゴリズムを結合させない.
7. 暗号化されたメッセージと復号化アルゴリズムを結合させたものにメールヘッダをつけて送信する.

以上の手順を見てわかる通り, EMAAS が, 受信者とネゴシエーションを行い, メッセージを暗号化し, 必要であれば復号化アルゴリズムを加えて送信するので, 送信者は, 受信者のメールアドレスを入力し, メッセージを書いて, 使用する暗号化アルゴリズムを選ぶだけである.

受信者側の EMAAS の実行手順は以下の通りである.

1. 暗号化されたメッセージと復号化アルゴリズムが一つになったメールを受け取ると, 初めに EMAAS が, 暗号化されたメッセージ部と復号化アルゴリズム部に分ける.
2. 次に EMAAS は復号化アルゴリズムを実行し, 暗号化されたメッセージを復号化する.
3. 最後に復号化されたメッセージを表示する.

以上に示した通り, EMAAS が復号化アルゴリズムと暗号化されたデータの分離, データの復号化を行い, 結果を表示するので, 受信者は読むメールを選択するだけである. 受信者には暗号化されたデータ, そして復号化アルゴリズムのソースは一切見えず, あたかも平文でメッセージが送られて来たかのように見える.

データの暗号化・復号化, 暗号化されたデータと復号化アルゴリズムの結合・分離は, 全て EMAAS が自動的に行う.

E-mail を利用する際には, 使用する E-mail の配送方式を決定しなければならない. そこで本論文では, E-mail の配送方式として MIME を選び, 新たな MIME の Content-Type を提案する. E-Mail の配送方式に MIME を採択した理由として以下の点が挙げられる.

- インターフェスとして選択した Mew が MIME のインターフェイスであり, MIME との適合性が良い.

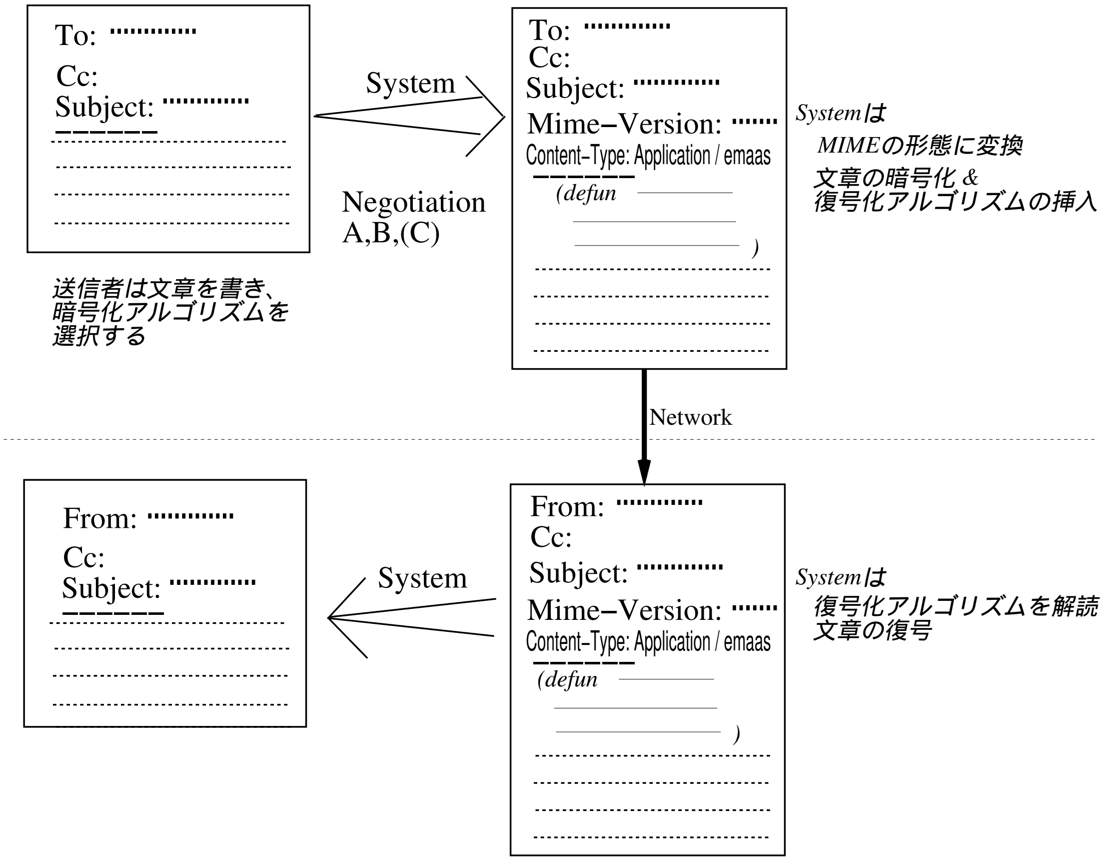
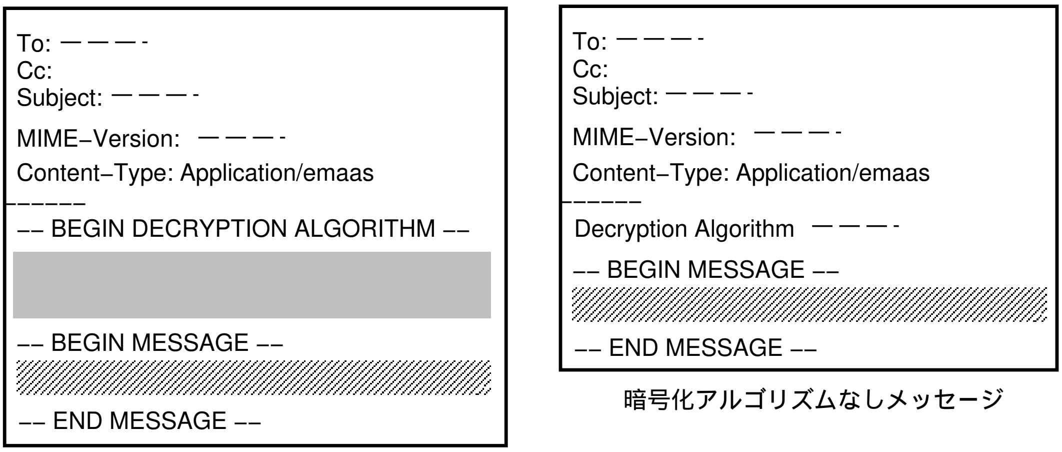


図 4.1: E-mail 配送モデル



To: ———-
Cc:
Subject: ———-
MIME−Version: ———-
Content−Type: Application/emaas

–– BEGIN DECRYPTION ALGORITHM ––
–– BEGIN MESSAGE ––
–– END MESSAGE ––

暗号化アルゴリズム付きメッセージ

To: ———-
Cc:
Subject: ———-
MIME−Version: ———-
Content−Type: Application/emaas

Decryption Algorithm ———-
–– BEGIN MESSAGE ––
–– END MESSAGE ––

暗号化アルゴリズムなしメッセージ

図 4.2: Application/emaas

- MIME は, RFC822 の上位互換で世界統一の E-Mail 配送方式である.
- 今後, MIME が全世界で使用されるようになると考えられる.
- IANA に登録すれば, 新たな MIME の Content-Type を世界統一規格とすることが可能である.

EMAAS のメールは暗号化されたデータと復号化アルゴリズムからなっており, 今までにない, 新しい E-Mail 形式を使用するので, 新たに Content-Type を提案し, この Content-Type を用いて E-Mail 配送を行う必要がある. 新たに提案する Content-Type は, 既に統一規格となっている"Application"を拡張した"Application/emaas"である. EMAAS の Content-Type として, "Application"の拡張を用いた理由は, PGP と PEM の Content-Type が"Application"の拡張として定義されており, 暗号化したメールの Content-Type として"Application"に統一した方が良いと考えたためである. 復号化アルゴリズムは, 復号化アルゴリズムのソースである場合と, 暗号化に使用された暗号化アルゴリズムを示す文字列である場合がある.(図 4.2)

EMAAS は暗号化されたメッセージと復号化アルゴリズムを送る時に, メールのヘッダフィールドに"Content-Type:Application/emaas"と MIME-Version 等を挿入する. また,EMAAS がメールのヘッダフィールドに"Content-Type:Application/emaas" を含んだメールを受信すると, EMAAS は, そのメールが暗号化されたメッセージと復号化アルゴリズムからなっていることを判断する. EMAAS が暗号化されたメッセージと復号化ア

ルゴリズムからなるメールであることを判断すると, 上記で説明した受信者側の実行手順が実行される.

4.2 E-Mail 配送におけるネゴシエーション

本節ではE-Mail配送におけるネゴシエーションプロトコルについて詳述する.

送信者は, 受信者の情報を入手するためにネゴシエーションを行う. そこで, 送信者がネゴシエーションを行った際に, 正確な情報が送信者に伝わるように, 受信者の情報を管理する必要がある. そこでE-Mail配送上に実装するEMAASでは, ユーザが持っている暗号化アルゴリズムや, マシンの性能の情報は, ユーザが属するメールサーバが管理することとする. メールの受信者の中には, 普段はネットワークに接続しておらず, ダイヤルアップIP接続やuucp接続でメールを読む者もいる. 受信者がダイヤルアップIP接続等をしてメールを読む際は, メールサーバが受信者に関する情報を管理していなければならない. さもなくば, 送信者は, 受信者が接続していない時には, 受信者に関する情報が入手不可能となる. したがって, 本論文では, ユーザに関する情報は, メールサーバが管理するようにした.

送信者側において, EMAASがネゴシエーションを始めてから, メールを送信するまでの状態遷移を図4.3に示す.

送信者側においてネゴシエーションを始めてから, メールを送信するまでの手順は以下のようになる.

1 EMAASは, 受信者宛に受信者に関する情報の要求メールを送る. この時, ヘッダに受信者に関する要求のメールである事を示すヘッダを挿入する.

2 メールが届くか, ある一定時間が経過し, Time Outをするか, 送信者がメッセージを書き終わりメールの送信を試みるまでEMAASは待機する.

- メールが届いたのであれば, 3に進む.
- Time Outした時には, 送信者に再度要求メールを受信者に送るかどうかを尋ね, 送信者が再度要求メールを送信することを選んだ時には, 1にもどる. 再度送信しない時には, 受信者からの返答がnilであるとして, 4に飛ぶ.
- 送信者がメールの送信を試みている時には, 送信者に返答が返ってくるまで待つかどうか尋ねる. 送信者が待つと答えた時には, 2に戻り再び待機する. 待たない時には, 受信者からの返答がnilであるとして, 4に飛ぶ.

3 届いたメールが要求に対する返答メールでなければ2へ戻り, 再び待機する.

4 送信者に, 送信者の情報と受信者の情報を表示し, 使用する暗号化アルゴリズムを選ばせる.

5 選ばれた暗号化アルゴリズムを送信者が所持しているかどうかを調べる. 送信者が所持している場合には, 6へ進む. 送信者が所持していない場合には以下の手順を行う.

5.1 受信者に選択された暗号化アルゴリズムを送信者に送るように要求メールを送信する. この時, 暗号化アルゴリズムに対する要求メールであることを示すヘッダを挿入する.

5.2 メールが届くか, ある一定時間が経過し, Time Outをするか, 送信者がメッセージを書き終わりメールの送信を試みるまで待機する.

- メールが届いたのであれば, 5.3に進む.
- Time Outした時には, 送信者に再度要求メールを受信者に送信するか尋ねる. 再度送信する時には, 5.1から再び実行する. 再度送信しない時には, 受信者からの返答がnilであるとして4に戻る.
- 送信者がメールの送信を試みている時には, 送信者に返答が返ってくるまで待つかどうかを尋ねる. 送信者が待つと答えた時には, 再び5.2に戻り待機する. 待たない時には, 送信者からの返答がnilであるとして4に戻る.

5.3 届いたメールが要求に対する返答メールでなければ, 5.2に戻り, 再び待機する.

5.4 送られた暗号化アルゴリズムでメッセージを暗号化する.

6 送信者がメッセージを書き終わるまで待機する.

7 選ばれた暗号化アルゴリズムでメッセージを暗号化する.

8 選ばれた暗号化アルゴリズムに対する復号化アルゴリズムを受信者が所持しているかどうかを調べる.

- 受信者が, 選ばれた暗号化アルゴリズムに対する復号化アルゴリズムを所持していない時には, 復号化アルゴリズムと暗号化されたデータを結合する.

9 メッセージを受信者に送信する.

また, 受信者側におけるメールホストの状態遷移を図4.4に示す. 受信者側においてメールホストがメールを受け取った際の実行手順は以下のようになる.

図 4.3: E-Mail 配送上のネゴシエーション (送信者側)

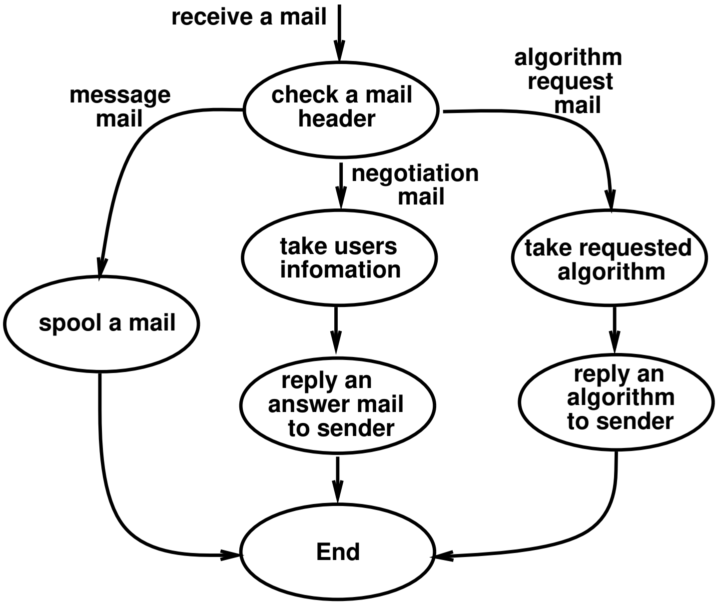


図 4.4: E-Mail 配送上のネゴシエーション (受信者側)

- 最初に受け取ったメールを調べる.
- 受け取ったメールが通常のメッセージメールである時には, 受け取ったメールをスプールする.
- 受け取ったメールが受信者の情報に対する要求メールである時には, 以下の手順を実行する.
    1. メールアドレスを調べ, メールアドレス先のユーザの情報を調べる.
    2. 調べた情報を要求に対する返答メールであることを示すヘッダをつけて送信者に送信する.
- 受け取ったメールが暗号化アルゴリズムに対する要求メールである時には, 以下の手順を実行する.
    1. 要求された暗号化アルゴリズムを取ってくる.
    2. 取ってきた暗号化アルゴリズムを, 要求に対する返答メールであることを示すヘッダをつけて送信者に送信する.

本論文で提案するネゴシエーションの方法の問題点として, 次のことが挙げられる. 提案するネゴシエーションの方法は, 既存の E-Mail 配送システムを利用した方法であり, E-Mail 配送プロトコルは End-to-End でコネクションを張るプロトコルではないため, ネゴシエーションに非常に時間がかかってしまうことがあげられる. ネゴシエーションに非常に時間がかかってしまうため, 本論文で提案するネゴシエーションの方法では, ネゴシエーションの途中で送信者がメッセージを書き終えてメールの送信を行おうとする場合が多くなると考えられる. 将来的にはネゴシエーションの方法として End-to-End のコネクション指向を持った方法も必要となるであろうが, 本論文では, 既存の E-Mail 配送システムを用いたネゴシエションの方法のみを提案する.

ネゴシエーションプロトコルを実装する際には, Time out の値を決定しなければならない. Time out の値は E-Mail 配送システムの事を考慮すると, 大きな値にする必要がある.

また, ネゴシエーションプロトコルを実装する際には, 次のようなメールサーバの拡張が必要である.

- 各ユーザの持っている暗号化アルゴリズムとマシン性能を管理する機能.
- ヘッダを見て要求メッセージである事を認識する機能.
- 受信者のメールホストが要求メッセージをスプールしない機能.
- 要求メッセージに対する返答メッセージの自動配送機能.

ネゴシエーションを行う際には, 暗号化アルゴリズムの名前をやり取りするため, 異なった暗号化アルゴリズムに同一の名前がついていると混乱のもととなる. そこで, 暗号化アルゴリズムの名付け方に統一された方法が必要となる. 本論文では, EMAAS のために Emacs-Lisp で記述された暗号化アルゴリズムには, 暗号化アルゴリズムの名前と記述者の名前と製作日を連結させた名前をつける事とする. 名前のフォーマットは, ”暗号化アルゴリズムの名前_記述者の名前_製作年_製作月” とする. 例えば本論文で提供する Emacs-Lisp で記述した DES は”DES_negi_96_2” となる.

4.3 暗号化アルゴリズム

本論文で実装するシステムでは, 暗号化アルゴリズム記述言語として Emacs-Lisp を用いている.

Emacs-Lisp は, エディタ (Emacs, Mule) 内で使用することを前提として設計された Lisp 言語である. ファイルや, バッファ, ディスプレイ, サブプロセスなどを扱う機能だけでなく, テキストを検索したり, 構文解析したりするための特別な機能も持っている [GNU Emacs Lisp 18] [GNU Emacs Lisp 19.29].

EMAAS が, 暗号化アルゴリズム記述言語として Emacs-Lisp を用いるのは次の理由からである.

- Emacs-Lisp の互換性の高さ
- 記述性の良さ
- EMAAS が, インターフェイスとして Emacs, Mule 上で動く Mew を使用するので Emacs, Mule 内の使用を前提に設計された Emacs-Lisp は EMAAS と相性が良い

Emacs-Lisp は, 扱える整数の範囲などがマシンによって異なりはするが, C 言語等のようにマシンによって異なるソースプログラムを用意する必要はない. このように Emacs-Lisp はマシン依存の少ない, 互換性が高いプログラミング言語である.

Emacs-Lisp は, 実行に時間がかかるという欠点はあるが,EMAAS のように, ソースプログラムを送る時には, 互換性の高い Emacs-Lisp は非常に都合が良い. 受信者側のマシンタイプが送信者のマシンタイプと異なると, C 言語で書いたソースコードは, 実行出来ない事があるが, Emacs-Lisp で書いたソースでは, そのようなことは少ない. 送られて来た暗号化アルゴリズムが実行できなければ, 暗号化されたメッセージを読む事ができなくなってしまうことを考えると, 本論文で提案するシステム EMAAS では, Emacs-Lisp は, 6章の 5 で述べるような問題があるが, 非常に有効な言語である.

しかし, Emacs-Lisp は Emacs のバージョンによって提供されている関数の種類が異なり, また扱える数値の範囲も異なる. 従って暗号化アルゴリズムを記述する際には, 基本的な Emacs-Lisp の関数だけを用い, かつ扱える数値範囲に依存しないようにすることで, さまざまなバージョンの Emacs 上で使えるようにしなければならない. 本論文では秘密鍵暗号系の DES と SAFER を Emacs-Lisp で記述して提供する.

4.4 実装する際の前提

本論文では E-Mail 上に EMAAS を実装する際に, 次の事を前提とした.

前提 1 メッセージの送信者または受信者は, Emacs-Lisp で記述された幾つかの秘密鍵暗号アルゴリズムを所持しているとする. これらの秘密鍵暗号アルゴリズムは送信者・

受信者のどちらか一方が所持していれば良く, 送信者・受信者両者が所持している必要はない.

前提 2 送信者・受信者は, あらかじめメッセージの暗号化・復号化に用いる鍵を共有しているものとする. 本論文では, 送信者・受信者はパスワードを共有しており, 送信者・受信者はメッセージの暗号化・復号化の度に, パスワードから鍵を一方向ハッシュ関数を用いて生成するものとする. いかなる一方向ハッシュ関数を用いて, どのように鍵を生成するかは暗号化アルゴリズムに依存する.

一方, 表 4.1に示したように, 暗号化は鍵の長さが長ければ長いほど安全であるので, 将来さらに長い鍵を使用する秘密鍵暗号系アルゴリズムが出てくるであろう. パスワードとして使用することが可能な文字数は, 小文字 26 文字, 大文字 26 文字, 数字 10 文字, の計 62 文字である. 少なくとも, 512 bit の鍵に対応するには, 86 文字 ($62^{85} < 2^{512} < 62^{86}$) のパスワード (パスフレーズと呼ぶ方がふさわしい) が必要となる. したがって, 本論文で提案するシステム EMAAS は 86 文字以上のパスワードが使用できるようになっていることが望ましい.

また, 今後さらに長い鍵が必要となり, 多くの人の秘密鍵を, 非常に長いパスワードとして所持する事は困難であるので, 秘密鍵を格納したファイルを暗号化しておき, そのファイルの暗号化鍵を所持しておく方法も考えられ, 現実的である.

前提 3 さまざまな不正行為が存在する中で, 本論文では盗聴の防止, なりすましの防止を目的とし, 伝送路上での改竄に関しては一切考えない物とする. 本システムにおいて, メッセージは暗号化して配送される. したがって, 改竄を試みる者が, 鍵を知らない以上, メッセージの改竄は行えないこととなる. 改竄を考えた時, 最も問題となる点が, 配送されるメール中の復号化アルゴリズムである. メール中の復号化アルゴリズムが改竄され, 入力したパスワードが受信者の知らない間に改竄者に送られると言う事が起こり得る. 不正行為を行う者にパスワードが知られてしまっては, どのような暗号化アルゴリズムも意味を成さなくなってしまう. 伝送路上での書き換え対策に関しては今後の課題とする.

前提 4 本論文において, これまでにネゴシエーションのプロトコルに関して, 説明を行ってきたが, 本論文の実装ではネゴシエーションの過程は既に終了しているということを前提とする. 本論文では, ネゴシエーション以降の, 以下の部分を実装する.

1. 送信者が選んだ暗号化アルゴリズムでメッセージを暗号化する
2. 暗号化されたメッセージと暗号化アルゴリズムを結合して, 受信者に送信する

表 4.1: 1995 年の DES における鍵長と解析時間とコスト

	Length of Key in bits					
Cost	40	56	64	80	112	128
$100K	2seconds	35hours	1year	70,000years	10^{14}years	10^{19}years
$1M	.2seconds	3.5hours	37days	7000years	10^{13}years	10^{18}years
$10M	.02seconds	21minutes	4days	700years	10^{12}years	10^{17}years
$100M	2milliseconds	2minutes	9hours	70years	10^{11}years	10^{16}years
$1G	.2milliseconds	13seconds	1hours	7years	10^{10}years	10^{15}years
$10G	.02milliseconds	1seconds	5.4minutes	245days	10^{9}years	10^{14}years
$100G	2microseconds	.1seconds	32seconds	24days	10^{8}years	10^{13}years
$1T	.2microseconds	.01seconds	3seconds	2.4days	10^{7}years	10^{12}years
$10T	.02microseconds	1milliseconds	.3seconds	6hours	10^{6}years	10^{11}years

3. 受信者側で暗号化されたメッセージと暗号化アルゴリズムを分離する
4. メッセージを復号化し, 表示する

EMAAS の実装目的は以下の問題点を解決することである.

- 使用する暗号化アルゴリズムに関する合意が必要である.
- 暗号化アルゴリズムは時と共に弱くなっていく.

以上の問題点を解決するには, メッセージと共に復号化アルゴリズムを送ることが大切であり, ネゴシエーションは柔軟性をより増すためにある. したがって, 本論文の実装ではネゴシエーションを今後の課題とした.

4.5 システムの実装

本節では, 実装を行った暗号化アルゴリズムとシステムについて説明する.

はじめに Emacs-Lisp で記述した CBC モードの DES 及び CBC モードの SAFER について述べる. CBC モードは, 前述した通り, 平文と前の block を暗号化した際の結果の XOR を計算した後,DES の操作を行うモードである. (2.4.1小節参照)

DES は暗号化の処理に 32 bit 操作を必要とする. ところが Emacs-Lisp は 24 bit までの値しか扱えない (マシンや Emacs-Lisp のバージョンによって 28 bit もしくはそれ以上を扱える) ので, リストを用い, 32 bit を上 8 bit と下 24 bit に分け, 32 bit 操作を実現した (図 4.5). この方式は, マシンや Emacs-Lisp のバージョンが異なった時, 正常に動作し

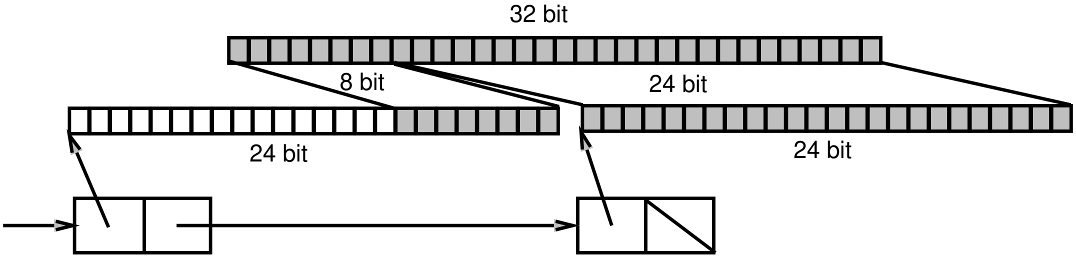


図 4.5: 24 bit - 32 bit 拡張 1

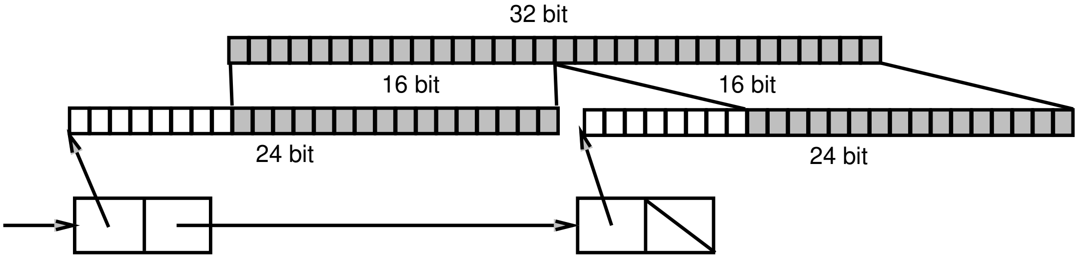


図 4.6: 24 bit - 32 bit 拡張 2

ない可能性が存在するので, 上16 bitと下16 bitに分ける方式 (図4.6) に変更することが望ましい. DESでは32 bitの基本操作として, 以下のような関数を実装し, 使用している.

- 左へn bitシフトする “llsh”.
- 右へn bitシフトする “lrsh”.
- 32 bitと32 bitの論理積を計算する “llogand”.
- 32 bitと32 bitの論理和を計算する “llogior”.
- 32 bitと32bitの排他的論理和を計算する “llogxor”.
- 32 bitと32 bitの和を計算する “lplus”.

同様にSAFERにおいても一部32 bit操作を必要とし,DESと同様リストを用いて32bit操作を実現した. ただし, SAFERでは32 bitを上16 bit, 下16 bitに分けた (図4.6). SAFERでは32 bitの基本操作として, 以下のような関数を実装し, 使用している.

- 左へn bitシフトする “llsh”.

- 右へ n bit シフトする “lrsh”.
- 32 bit の否定を計算する “llognot”.
- 32 bit と 32 bit の論理積を計算する “llogand”.
- 32 bit と 32 bit の論理和を計算する “llogior”.
- 32 bit と 32 bit の排他論理和を計算する “llogxor”.
- 32 bit と 32 bit の和を計算する “lplus”.
- 32bit と 32 bit の差を計算する “lminus”.
- 32 bit と 32bit の大小関係を調べる “leq”(= “=”), “lt”(= “>”), “st”(= “<”).

次に EMASS の実装について述べる. 暗号化アルゴリズムと暗号化されたデータを扱えるように, Mew の一部を改良した. 基本操作は, 既存の Mew のままである.

送信者が, Mew 上でメールを送る命令を選び送信先を入力すると, EMAAS は受信者とネゴシエーションを始める. EMAAS は, ネゴシエーションの途中で送信者に対し使用できる暗号化アルゴリズム等の情報を示し, 送信者は使用する暗号化アルゴリズムを選択する. ただし本論文では, ネゴシエーションが終了していると仮定するため, 以上の過程は省略される. 送信者は, システムがネゴシエーションを行っている間メッセージを記入する. 送信者は, メッセージの記入が終了した時点で, メールの送信を命令する. 送信者が行う操作はここまでである. ただし, この時点でネゴシエーションが終了していない時には, システムがネゴシエーションの終了を待つか, 送信者の持っている暗号化アルゴリズムで暗号化を行うかを送信者に質問する.

送信命令を受け取ったシステムは, 以下のように実行する.

1. メッセージ部をバッファに切り出す.
2. 選択された暗号化アルゴリズムを実行し, メッセージを暗号化する.
3. base64 で符号化を行う.
4. ヘッダ部に”`Content-Type:  Application/emaas`”等の MIME のヘッダを挿入する.
5. ヘッダの次の行に
”`------ BEGIN DECODE ALGORITHM ------`”
という復号化アルゴリズムの始まりを示す文字列を挿入する.

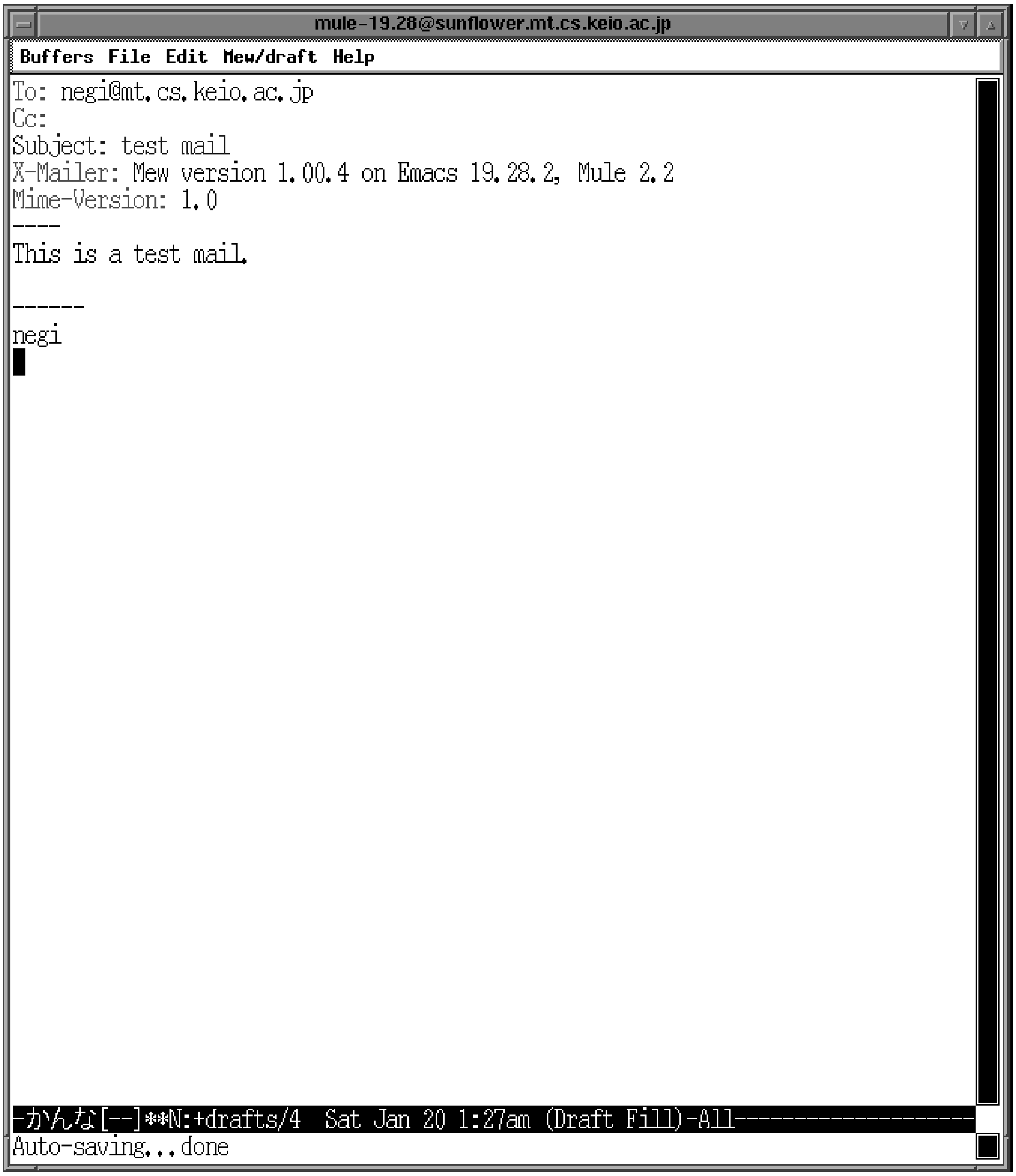
6. 上記の文字列の後, 使用した暗号化アルゴリズムに対応する復号化アルゴリズムを挿入する.
7. 復号化アルゴリズムの後にメッセージの開始を示す文字列
   ”`------ BEGIN MESSAGE ------`”
   を挿入する.
8. 最後に暗号化されたメッセージとメッセージの終了を示す文字列
   ”`------ END MESSAGE ------`”
   を挿入して, 送信する.

一方, 受信者が, メールを読む時, EMAAS は以下のように実行する.

1. Content-Type が”`Application/emaas`”となっている事を検出する.
2. メールをヘッダ部, 復号化アルゴリズム部, 暗号化されたメッセージ部に分割する.
3. 復号化アルゴリズム部, メッセージ部をそれぞれ別のバッファに切り出す.
4. 復号化アルゴリズムを評価する.
5. 暗号化されたメッセージ部を base64 で復号化する.
6. base64 で復号化したメッセージを復号化アルゴリズムを用いて, 復号化する.
7. 最後に, ヘッダ部と復号化したデータを結合して, 表示する.

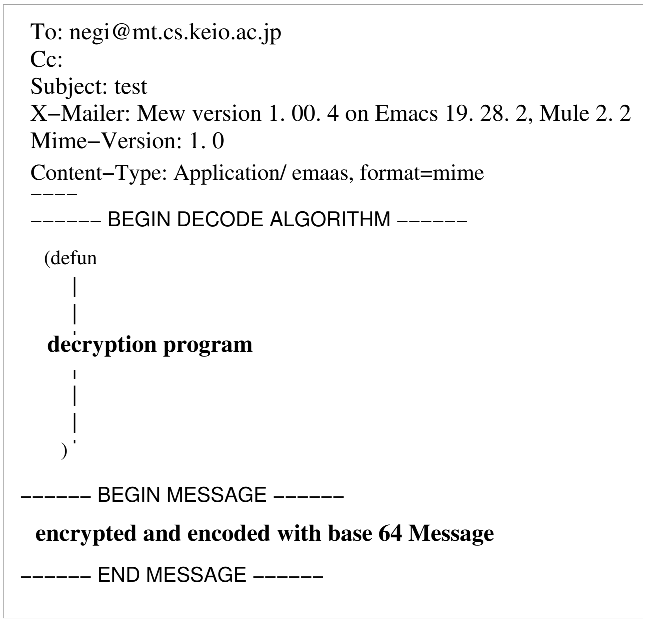
4.6 動作例

本論文で行った実装の動作例を示す。Mew では, メールを記述するためのドラフトモードにするためには”w”とキー入力を行う. 本論文ではネゴシエーションが終了したあとの作業を実現しているので, (step 1) まず送信者はメールを記述する (図 4.7). (step 2) EMAAS が,Content-Type が application/emaas であり, 復号化アルゴリズムと, メッセージを暗号化し base64 で符号化したものを含んだメールに変換する (図 4.8). (step 3) EMAAS は, 変換後受信者宛にメールを送信する (図 4.9). 図 4.9の詳細は Appendix A に載せる. (step 4) メールを受け取った受信者側の EMAAS は, メッセージの部分を base64 で復号化した後, 送られてきた復号化アルゴリズムで復号化し, 復号化したメッセージを表示する (図 4.10).



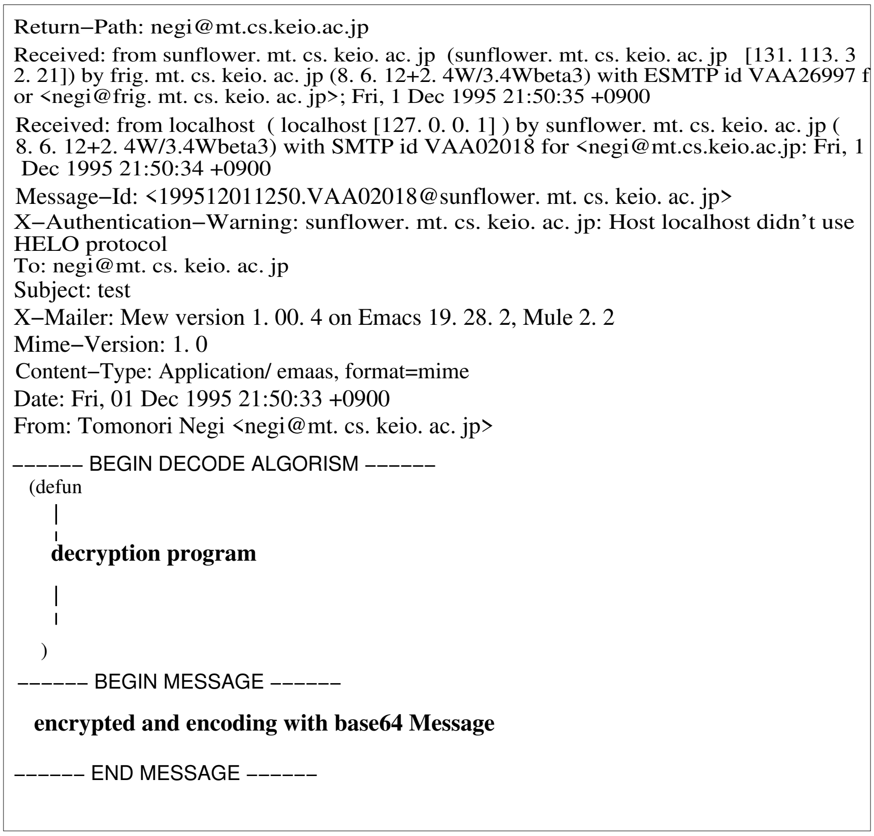
```
To: negi@mt.cs.keio.ac.jp
Cc:
Subject: test mail
X-Mailer: Mew version 1.00.4 on Emacs 19.28.2, Mule 2.2
Mime-Version: 1.0
----
This is a test mail.

-------
negi

-かんな[--]**N:+drafts/4  Sat Jan 20 1:27am (Draft Fill)-All--------------------
Auto-saving...done
```

図 4.7: 実行例 step1



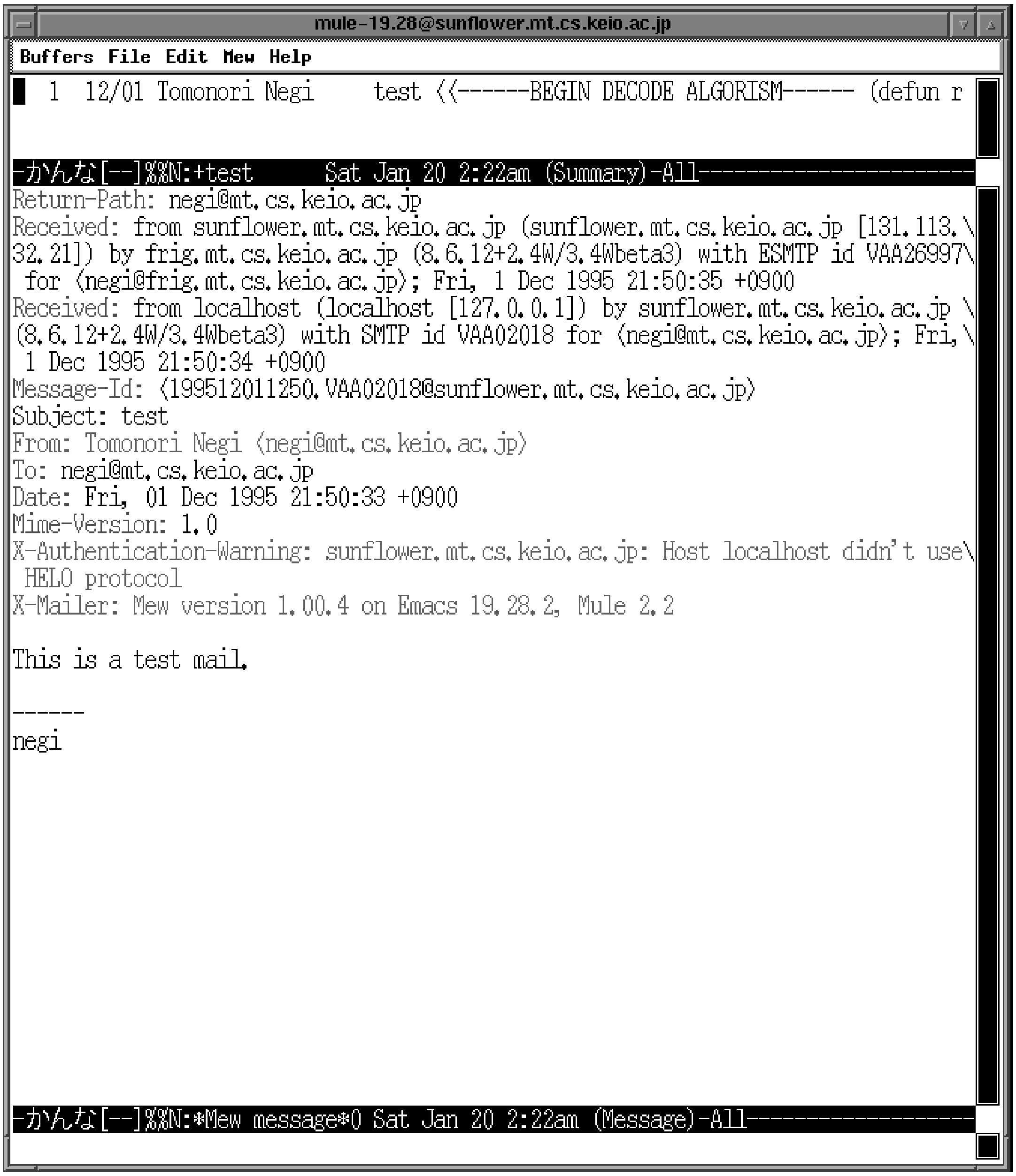
```
To: negi@mt.cs.keio.ac.jp
Cc:
Subject: test
X–Mailer: Mew version 1. 00. 4 on Emacs 19. 28. 2, Mule 2. 2
Mime–Version: 1. 0
Content–Type: Application/ emaas, format=mime
––––
–––––– BEGIN DECODE ALGORITHM ––––––
  (defun
     |
     |
  decryption program
     |
     |
     |
    )
–––––– BEGIN MESSAGE ––––––
 encrypted and encoded with base 64 Message
–––––– END MESSAGE ––––––
```

図 4.8: 実行例 step2



```
Return–Path: negi@mt.cs.keio.ac.jp
Received: from sunflower. mt. cs. keio. ac. jp  (sunflower. mt. cs. keio. ac. jp   [131. 113. 3
2. 21]) by frig. mt. cs. keio. ac. jp (8. 6. 12+2. 4W/3.4Wbeta3) with ESMTP id VAA26997 f
or <negi@frig. mt. cs. keio. ac. jp>; Fri, 1 Dec 1995 21:50:35 +0900
Received: from localhost  ( localhost [127. 0. 0. 1] ) by sunflower. mt. cs. keio. ac. jp (
8. 6. 12+2. 4W/3.4Wbeta3) with SMTP id VAA02018 for <negi@mt.cs.keio.ac.jp: Fri, 1
 Dec 1995 21:50:34 +0900
Message–Id: <199512011250.VAA02018@sunflower. mt. cs. keio. ac. jp>
X–Authentication–Warning: sunflower. mt. cs. keio. ac. jp: Host localhost didn’t use
HELO protocol
To: negi@mt. cs. keio. ac. jp
Subject: test
X–Mailer: Mew version 1. 00. 4 on Emacs 19. 28. 2, Mule 2. 2
Mime–Version: 1. 0
Content–Type: Application/ emaas, format=mime
Date: Fri, 01 Dec 1995 21:50:33 +0900
From: Tomonori Negi <negi@mt. cs. keio. ac. jp>
–––––– BEGIN DECODE ALGORISM ––––––
  (defun
     |
     |
   decryption program
     |
     |
   )
–––––– BEGIN MESSAGE ––––––
  encrypted and encoding with base64 Message
–––––– END MESSAGE ––––––
```

図 4.9: 実行例 step3 (Appendix A)



```
█ 1  12/01 Tomonori Negi     test <<------BEGIN DECODE ALGORISM------ (defun r


-かんな[--]%%N:+test       Sat Jan 20 2:22am (Summary)-All-----------------------
Return-Path: negi@mt.cs.keio.ac.jp
Received: from sunflower.mt.cs.keio.ac.jp (sunflower.mt.cs.keio.ac.jp [131.113.\
32.21]) by frig.mt.cs.keio.ac.jp (8.6.12+2.4W/3.4Wbeta3) with ESMTP id VAA26997\
 for <negi@frig.mt.cs.keio.ac.jp>; Fri, 1 Dec 1995 21:50:35 +0900
Received: from localhost (localhost [127.0.0.1]) by sunflower.mt.cs.keio.ac.jp \
(8.6.12+2.4W/3.4Wbeta3) with SMTP id VAA02018 for <negi@mt.cs.keio.ac.jp>; Fri,\
 1 Dec 1995 21:50:34 +0900
Message-Id: <199512011250.VAA02018@sunflower.mt.cs.keio.ac.jp>
Subject: test
From: Tomonori Negi <negi@mt.cs.keio.ac.jp>
To: negi@mt.cs.keio.ac.jp
Date: Fri, 01 Dec 1995 21:50:33 +0900
Mime-Version: 1.0
X-Authentication-Warning: sunflower.mt.cs.keio.ac.jp: Host localhost didn't use\
 HELO protocol
X-Mailer: Mew version 1.00.4 on Emacs 19.28.2, Mule 2.2

This is a test mail.

------
negi
















-かんな[--]%%N:*Mew message*0 Sat Jan 20 2:22am (Message)-All-------------------
```

図 4.10: 実行例 step4

第 5 章

評価・検討

本章では, 本論文で提供した, Emacs-Lisp で記述した DES と SAFER について評価した後, Emacs-Lisp で記述した DES と SAFER の実行時間を C で記述した DES や SAFER の実行時間と比較する. また, 本論文で実装したシステムの使いやすさ・柔軟性について評価・検討する.

5.1 Emacs-Lisp で記述された DES と SAFER

本論文で提案したシステム EMAAS では, 復号化アルゴリズムをメッセージと共に送信するため, トラフィックの増大が心配される. そこで, 復号化アルゴリズムを圧縮して送信する方法が考えられる. 本節では Emacs-Lisp で記述した DES と SAFER のプログラムの大きさ, とこれらを gzip で圧縮した時の大きさについて評価・検討をし, gzip の有効性について考える.

また, Emacs-Lisp で記述した DES と SAFER の性能を調べるために, 暗号化・復号化にかかる実行時間を調べ, C で記述した DES と SAFER と比較する.

5.1.1 DES・SAFER のプログラムサイズ

表 5.1に DES の暗号化・復号化と SAFER の暗号化・復号化, それぞれのプログラムコードの大きさを gzip で圧縮する前と圧縮した後で比較し, 同時に圧縮率を示す.

DES は, 暗号化する過程でつねに 32 bit の複雑な操作を必要とする. これに対して SAFER は, 16 bit 操作と 8 bit 操作が中心であり, ごくまれに 32 bit の非常に単純な操作のみを必要とする. したがって, DES のソースは SAFER のソースと比較して, 非常に大きくなってしまう. また別の原因としては, 実装する際に DES を先に記述したため, SAFER を記述した時に比べ Emacs-Lisp に不慣れであり, DES は非常に統一の取れていない無駄の多いプログラムになっているということが, 考えられる. しかし, 最適化を進めると, 少なくとも 500 Byte は小さくなると考えられる. SAFER も, さらに最適化を行いプログラムの大きさを小さくすることが可能であると考えられる.

表 5.1に示したように, gzip を用いると Emacs-Lisp で記述されたソースを 1/5 程度に圧縮可能である. EMAAS では, 毎回復号化アルゴリズムを送信するわけではないが, 復号化アルゴリズムを圧縮せずに送信すると, 非常に大きなメールを送る事となってしまう. そこで, gzip を用いるとプログラムを小さくすることが可能なので, 将来的には復号化アルゴリズムを gzip を用いて圧縮をし, base64 で符号化を行った後に送信するのが適当と思われる. しかし, base64 は既に Mew に組み込まれているが, gzip は自分で実装しなければならない. 現在, ユーザ全員が gzip を所持しているわけではない. そこで, 以下の

表 5.1: DES・SAFER の大きさ

ソース	gzip 前 (Byte)	gzip 後 (Byte)	圧縮率
DES(暗号化)	30904	5833	$\frac{1}{5.3}$
DES(復号化)	30405	5748	$\frac{1}{5.3}$
SAFER(暗号化)	12169	2532	$\frac{1}{4.8}$
SAFER(復号化)	14798	3077	$\frac{1}{4.8}$

2 つの方法が考えられる.

- ネゴシエーションの際に, 送信者・受信者が, gzip に限らず何らかの圧縮ツールを持っているかどうかを調べ, 同じ圧縮ツールを所持している際にはその圧縮ツールを用いる. 同じ圧縮ツールを所持していない時には, 圧縮せずに送る.
- gzip をシステム EMAAS に組み込み, 利用者全員が必ず gzip を所持するようにする.

gzip は, 非常に有効な圧縮ツールであり, システムの利用者全員が持っていることが望ましく, システム EMAAS に gzip を組み込む方が適当であると考えられる.

5.1.2 実行時間

本節では,Emacs-Lisp で記述した DES・SAFER の実行時間と C で記述した DES・SAFER の実行時間を比較・検討する.

それぞれの暗号化アルゴリズムを, SPARC station 20 上に起動した Mule-19.28 上で実行し, 実行時間を測定した.

256 Byte, 512 Byte, 1024 Byte, 2048 Byte のデータに対して DES のプログラムを実行した際, 暗号化・復号化に要した時間を表 5.2に示す. また, それぞれ同じサイズのデータに対して SAFER のプログラムを実行した際, 暗号化・復号化に要した時間を表 5.3に示す. いずれも, 暗号化・復号化の処理に要した時間のみを測定し, データの読み込み, データの書き出し, パスワードからの鍵の生成, プログラムの初期化等の処理時間は含まれていない.

結果として見てみると, Emacs-Lisp で記述した DES・SAFER は, 実行時間において C で記述したものに比べ 1000 倍以上劣っている. しかも, データのサイズが大きくなるに従って実行時間の格差が大きくなっている. これは, データが大きくなるほど, Emacs-Lisp の命令実行回数が増えることによる. DES において, 実行時間の際は顕著である. これは, DES では常に 32 bit 操作を行い, その度にリスト操作を行うことに起因している.

表 5.2: DES の実行時間: Emacs-Lisp VS C

データサイズ		Emacs-Lisp [sec]	C [msec]
256 Byte	暗号化	6.70	0.42
	復号化	6.73	0.42
512 Byte	暗号化	13.33	0.83
	復号化	13.36	0.83
1024 Byte	暗号化	26.69	1.64
	復号化	26.74	1.65
2048 Byte	暗号化	53.39	3.26
	復号化	53.46	3.26

表 5.3: SAFER の実行時間: Emacs-Lisp VS C

データサイズ		Emacs-Lisp [sec]	C [msec]
256 Byte	暗号化	0.63	0.22
	復号化	0.58	0.22
512 Byte	暗号化	1.21	0.44
	復号化	1.20	0.44
1024 Byte	暗号化	2.40	0.86
	復号化	2.41	0.87
2048 Byte	暗号化	4.73	1.73
	復号化	4.70	1.70

Emacs-Lisp で記述した DES・SAFER の実行時間を早くするためには, 32 ビット以上の値を扱え, しかも実行速度の速い Emacs-Lisp が必要である.

次に, 32 bit 操作を実現するためのリスト操作を使わずに, 16 bit 操作までを用いて DES を Emacs-Lisp で記述した時の実行時間について検討する. 本論文で提供した DES は 32 bit 操作を必要とするプログラムだが, 16 bit 操作までしか使わずに DES のアルゴリズムと等価なプログラムを作る事は可能である.

横軸にデータのサイズ, 縦軸に Emacs-Lisp で記述した DES と SAFER を用いた時の暗号化・復号化にかかる時間を取ったグラフを図 5.1 に示す. そして, 横軸にデータのサイズ, 縦軸に C で記述した DES と SAFER を用いた時の暗号化・復号化にかかる時間の関係を図 5.2に示す.C で記述した DES と SAFER の関係 (図 5.2) と Emacs-Lisp で記述した SAFER の実行時間 (図 5.1) から, 16 bit 操作までを使用した DES のプログラムの実行時間を以下の手順で概算すると図 5.3のようになる.

1. 図 5.2より DES と SAFER の暗号化・復号化の傾きと y 切片 (データサイズが 0 Byte の時の値) を求める.
2. 1 で求めた, DES と SAFER の傾きの比と, DES と SAFER の y 切片の比を求める.
3. 図 5.1より SAFER の傾きと y 切片を求める.
4. 2 で求めた比と 3 で求めた傾きと y 切片から, DES の y 切片と傾きを求める.

図 5.3に示した値は, Emacs-Lisp で記述した SAFER の測定値, および C で記述した DES と SAFER の測定値より求めたが, Emacs-Lisp で記述した SAFER も数回 32 bit 操作を必要とするため, 多少大きい実行時間が求まっている. しかしながら, 図 5.1と図 5.3 を比較した時, 2 つの処理速度の違いは明らかである. Emacs-Lisp で記述した DES も 16 bit までの操作のみを用いて記述すれば, 現在の Emacs-Lisp で記述した DES よりはるかに速く実行できる.

5.2 使いやすさ・柔軟性

使いやすさ

本論文では, インターフェイスとして, 既存の Mew を拡張して提供した. Mew は, MIME の便利なインターフェイスを目的として作られたインターフェイスである. 既存の Mew の便利で優れた機能を失うことなく, システム EMAAS を Mew に組み込んだ. ユーザは従来の Mew と同様の操作を行うだけで, システム EMAAS を使用することが可能であ

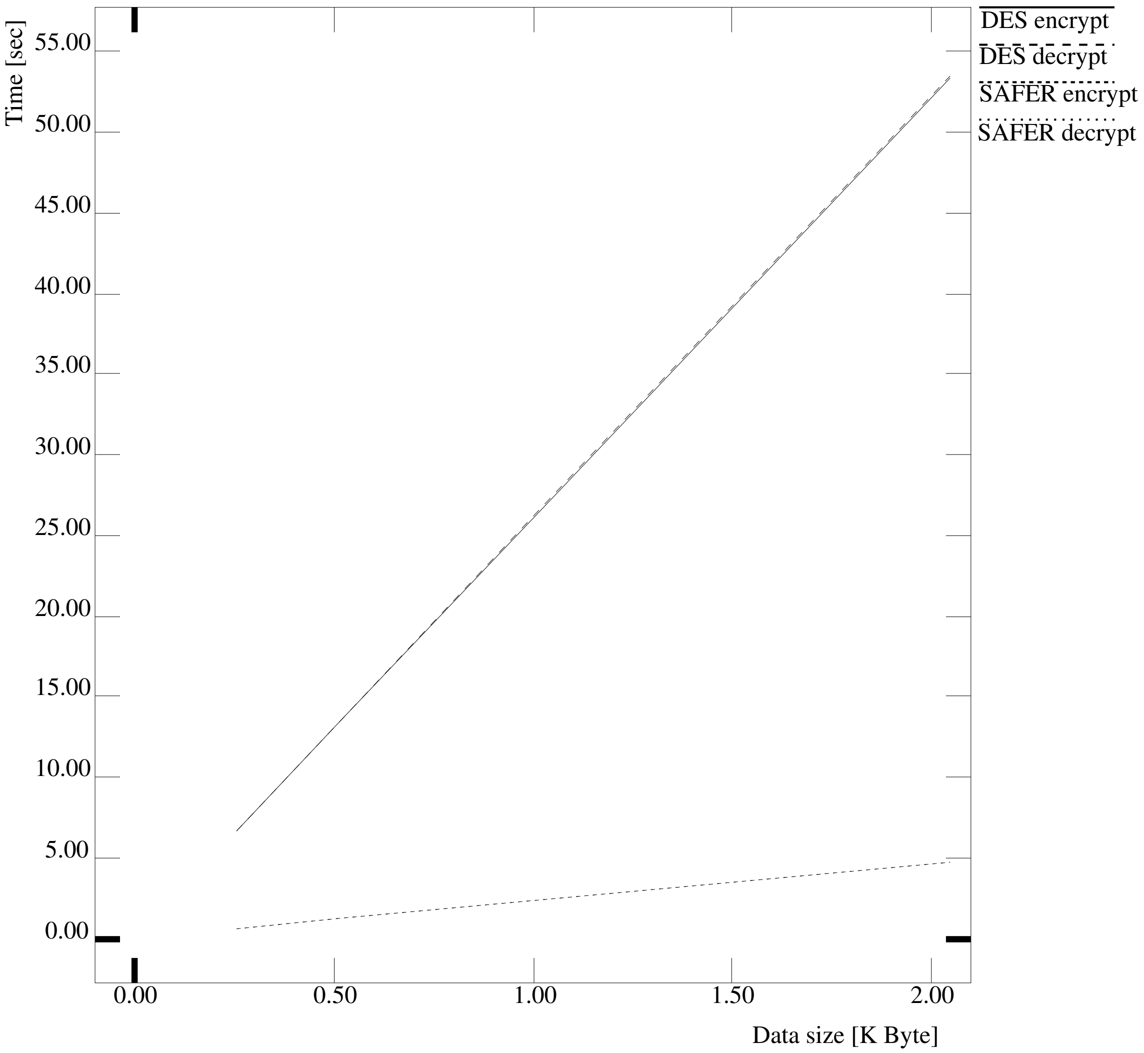


図 5.1: E-Lisp による DES と E-Lisp による SAFER の実行時間

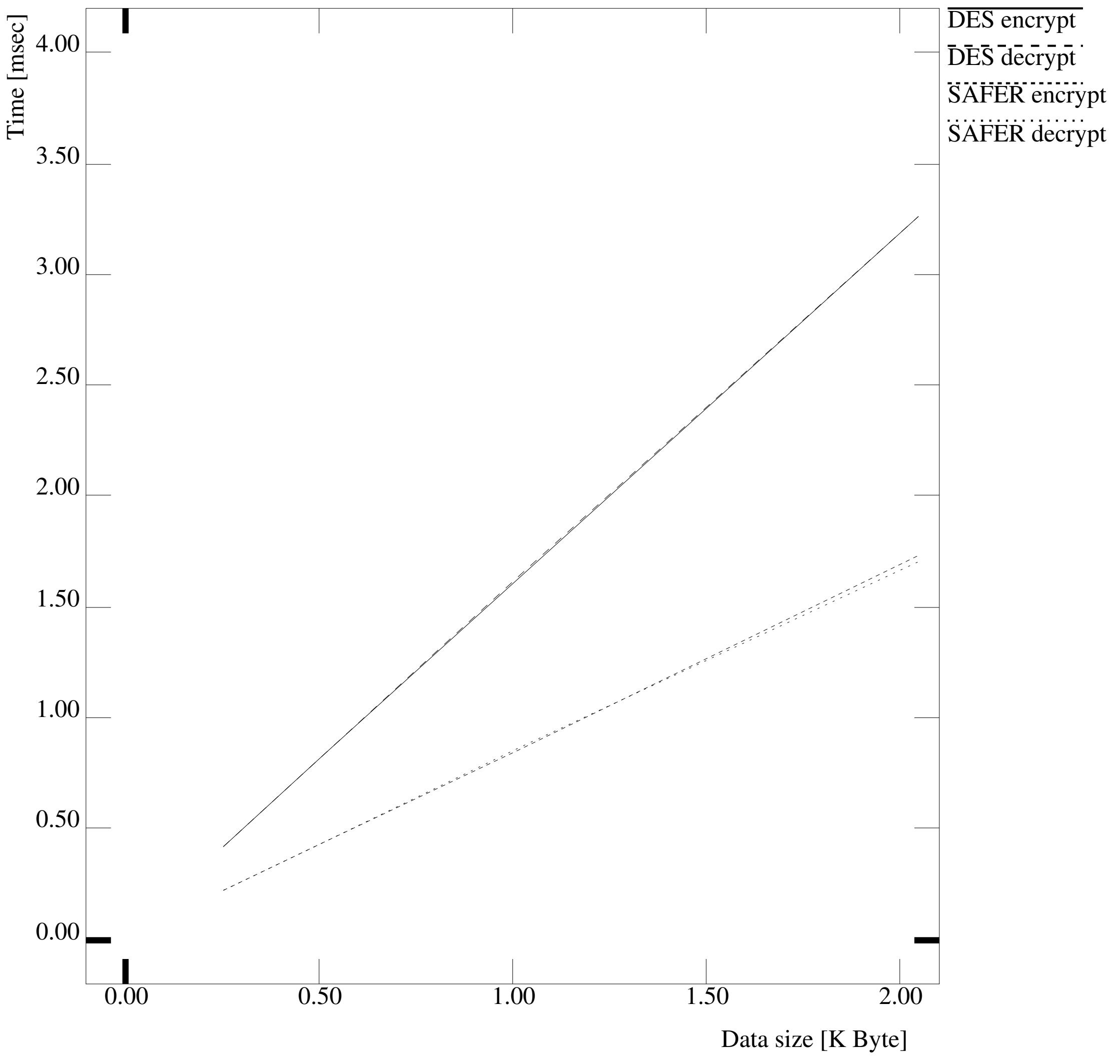


図 5.2: C による DES と C による SAFER の実行時間

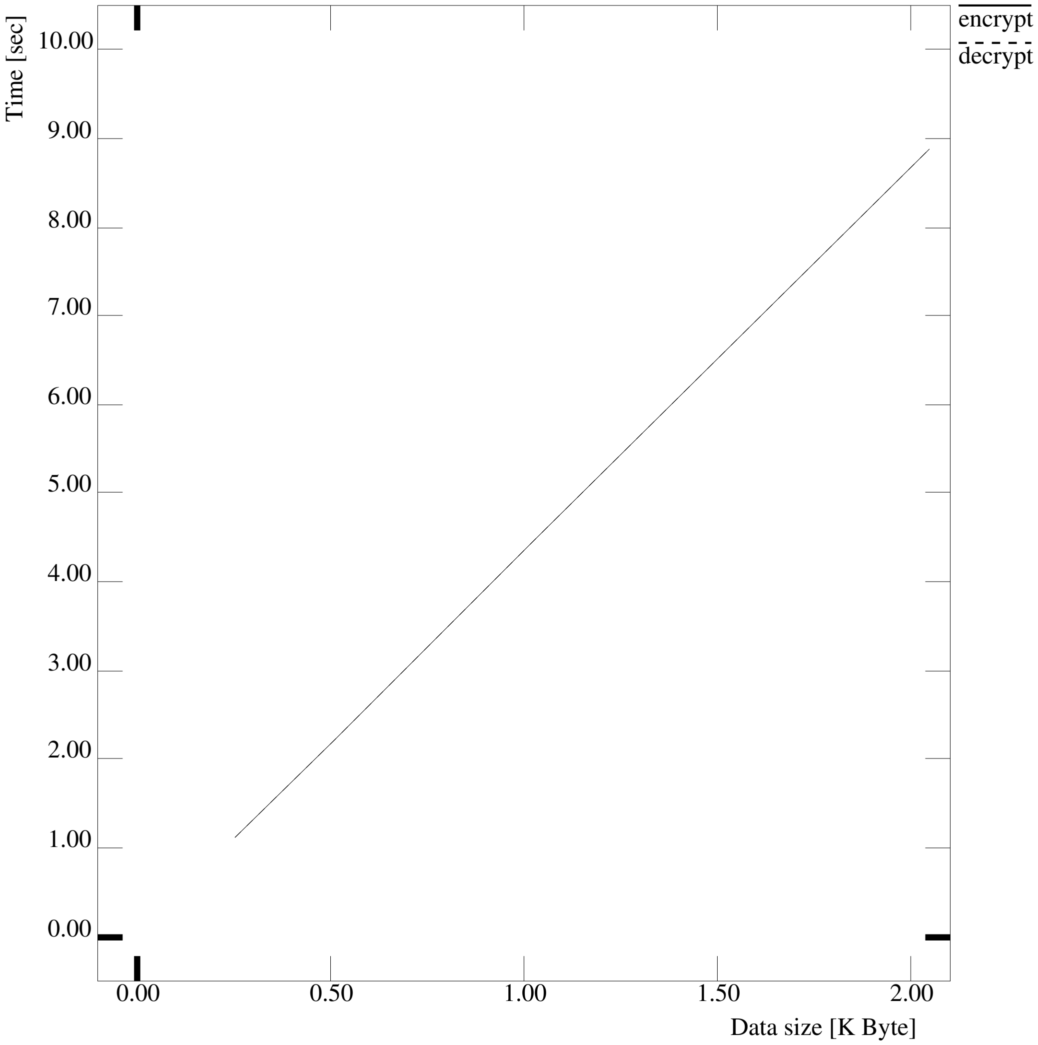


図 5.3: 32 bit のリスト操作をなくした Emacs-Lisp の DES 予想値

表 5.4: 従来のシステムと EMAAS の柔軟性

暗号化アルゴリズムの有無		通信可能	
送信者	受信者	従来	EMAAS
○	○	◎	◎
○	×	×	◎
×	○	×	◎
×	×	×	×

り, EMAAS は, 非常に使いやすいシステムである. ただし, Mew は Emacs・Mule 上で動作するため, 処理速度が遅くなるという欠点が存在する.

柔軟性

表 5.4に示したように, 従来は送信者・受信者両者が同一の暗号化アルゴリズムを所持している時のみ, メッセージを暗号化して送信することが可能であった. しかし, 本論文で提案した EMAAS は, 送信者・受信者両者が同一の暗号化アルゴリズムを所持している時のみならず, 送信者・受信者のいずれか一方が暗号化アルゴリズムを所持していれば, メッセージを暗号化して送信することが可能である. 送信者・受信者のいずれか一方が所持している暗号化アルゴリズムを暗号化に使用可能とすることで, データの重要性, マシンの性能に合わせた暗号化アルゴリズムの使用が可能となり, しかも柔軟性に富んだ暗号化が可能となる.

また, 従来は, 送信者・受信者の間で使用する暗号化アルゴリズムに関して, 合意をした上で, 暗号化アルゴリズムを使用していた. そこで, 使用していた暗号化アルゴリズムの安全強度が弱くなっても, 再び両者で合意に達するまで, 新たな安全強度の強い暗号化アルゴリズムを使用することは不可能であった. しかし, EMAAS では送信者・受信者のどちらか一方が暗号化アルゴリズムを所持していれば, 送信者と受信者の間で合意なしに暗号化アルゴリズムを使用可能としたため, いち早く安全強度の強い新たな暗号化アルゴリズムを使用することが可能となる.

第 6 章

今後の課題

本章では, 今後の課題について述べる. 今後の課題としては以下の事が考えられる.

1. EMAAS に圧縮ツール (gzip) を組み込む.
2. ネゴシエーションプロトコルの実装.
3. 鍵配送方式の提案と実装.
4. 伝送路上における改竄対策.
5. 新たな暗号化アルゴリズム記述言語の設計.
6. 本論文で提案した MIME の Content-Type の世界標準化.
7. 暗号化アルゴリズム ・復号化アルゴリズムの追加採用
8. Mew に関する評価.
9. EMAAS の応用.
10. IP V6 のサーベイ.

以下では, これらの課題についてより詳しく論ずる.

1. 本論文の実装では復号化アルゴリズムを圧縮せず, ソースをそのまま送信するようにした. しかし, この方法では, 非常に大きなデータを送信することになってしまう. また, 第 5.1.1小節で述べた通り, gzip は Emacs-Lisp のソースコードを約 1/5 程度までデータを圧縮することが可能であるので, 復号化アルゴリズムを送信する際に非常に有効なツールであり, EMAAS の利用者全員が所持することが望ましいと考えられる. そこで, 今後は,DEWDA に圧縮ツール (gzip) を組み込み, 暗号化アルゴリズムを圧縮して送信し, 受信側で暗号化アルゴリズムを解凍した後に実行するよう改良する.

2. 本論文では, ネゴシエーションは終了しているという前提のもと, 実装を行った. したがって, 今後, 本論文で提案したネゴシエーションのプロトコルを実装する必要がある. タイムアウトの時間設定, メールサーバーの拡張等を行い, ネゴシエーションプロトコルの実装を行う. ネゴシエーションにかかる時間を計測する. ネゴシエーションに時間がかかりすぎ, 利用する際に問題があると判断した際には, コネクションを End-to-End で張るネットワークプロトコルをも利用可能なネゴシエーションプロトコルを考える. 送信者は, ネゴシエーションを受信者側のメールホストと行うが, メールアドレスからはホスト

名がわからないため, コネクションを End-to-End で張るようなネットワークプロトコルを利用する方法は非常に困難であると考えられる. そこで送信者が, 受信者のメールホスト名を知っている時には, コネクションを End-to-End で張るネットワークプロトコルを利用し, それ以外の時は, メール配送システムを利用してネゴシエーションを行う方式が適当であると考える.

3. 今回は, 実装を行う際には鍵を共有してることを前提とした. しかし, 送信者は, 通信相手と鍵を共有していない時には, 何らかの方法で共有鍵を決めなければならない. 本論文で提案したシステムには, 鍵を安全に配送する方法が組み込まれていないため, ユーザは電話等別の通信手段を用いなければ鍵を共有できない. そこで今後, 鍵を安全に配送するプロトコルを考え, EMAAS に組み込む. いかなる暗号化アルゴリズムも使わず, 鍵も使わずに鍵を安全に配送することは非常に困難であり, 現在ではまだこのような方法は存在しない. PGP や PEM は, メッセージの暗号化・復号化に利用する IDEA, DES の鍵を受信者に安全に配送する機能を所持しているが, これらは RSA と公開鍵・秘密鍵を利用して行っている. PGP や PEM 以外にも **EKE(Encrypted Key Exchange)** など鍵を安全に配送するシステムは提案されているが, いずれも暗号化アルゴリズムと鍵を用いている. RSA 等の公開鍵暗号系アルゴリズムを EMAAS に組み込むことによって安全に鍵を配送することが可能となるが, これらの公開鍵暗号系アルゴリズムも時が経つと共に弱くなることは明らかである. アルゴリズムが EMAAS に組み込まれている場合には, 利用している公開鍵暗号系アルゴリズムが弱くなってしまった時に, アルゴリズムを容易には変更することができず, 本論文の目的である,”暗号化アルゴリズムは, 時が経つと共に弱くなるので, この問題点を解決する” と相反してしまう. したがって, 安全な鍵配送のために特定の公開鍵暗号系アルゴリズムを EMAAS に組み込むこんでしまうことは好ましくない.

4. 本論文では, 伝送路上での改竄が起こらないことを前提とした. 伝送路上での改竄は簡単に行うことができるため, 改竄を考えずに使い続けることは不可能である. 今後, 伝送上の改竄対策を考える.

現在, 一般には改竄防止対策として, MD5 等の一方向ハッシュ関数が用いられている. EMAAS でも, MD5 を用いて改竄を発見することは可能である. 具体的には, 暗号化すべきメッセージに送信する暗号化アルゴリズムの MD5 を付け加え, メッセージと MD5 の結果を一緒に, 暗号化してしまう方法が考えられる. 暗号化されたメッセージが改竄された時には, 鍵が改竄者に知られていなければ, 改竄を復号時に発見できる. しかし, 復号化

アルゴリズムを改竄された時には, 復号化を行った後にしか改竄されたことがわからず, 最悪の場合, 改竄者にパスワードが知られてしまう可能性が存在する. したがって, 改竄されたことが判明した後にどのように改竄されたかを調べ, 改竄に対する後始末をしなければならない. このように MD5 によって改竄を発見することは可能ではあるが, 改竄をチェックする前に, 送られて来たプログラムを実行してしまうので, 不正な命令が挿入されている際にはその命令も実行してしまい危険である. 従って, プログラムを実行する前に改竄を発見する方法を考えなければならない. また, 改竄されたとしても, 被害を最小限度に抑えられるよう, 暗号化アルゴリズムを記述する言語の関数を制限し, 受信者にとって致命的な不正行為を行えないようにする必要がある. 言語に関する今後の方針は次に述べる.

5. 本論文では, 暗号化アルゴリズム記述言語に Emacs-Lisp を採用した. これは, Emacs-Lisp は Emacs や Mule が動作するマシン上であれば, マシンに依存することなく, 必ず実行することができるためである. しかし,Emacs-Lisp にも以下のような問題点がある.

1. Emacs-Lisp は実行に時間がかかる.
2. 扱える整数値が 24 bit 幅と小さい.
3. プログラムが改竄されても発見できない.
4. ファイルの改変等を禁止できない.

問題点 3, 4 に関しては Emacs-Lisp に限ったことではなく, 今日存在するありとあらゆる言語に共通して言えることである. 今後, Emacs-Lisp の上位互換で, 以上の問題を解決した, 暗号化アルゴリズム記述言語を考える必要がある.

6. 本論文で MIME の新しい Content-Type ”Application/emaas”を提案したが, この Content-Type は世界統一規格となっていない. 今後, ”Application/emaas”を新たな Content-Type として IANA に登録し, 世界統一規格にする.

7. 本論文では, 暗号化アルゴリズム ・復号化アルゴリズムのどちらか一方のみが, 送受信されたため, ユーザは送信されて来たアルゴリズムを取り込み, 新たにアルゴリズムの種類を増やす事が不可能であった. 今後, 暗号化アルゴリズム ・復号化アルゴリズムが常に一組として, 送受信されるようにし, ユーザが, 送信されて来たアルゴリズムを取り込み, 新たにアルゴリズムの種類を増やせるようにする.

8.　本論文ではインターフェイスとして, Emacs や Mule 上で動く Mew を用い, Mew にシステム EMAAS を組み込んだ. Mew は非常に優れたインタフェースであるが, Emacs や Mule 上で動作するため, 実行速度が遅く, Emacs や Mule がないと利用できないという問題点も存在する. 今後, Mew の利便性, Mew と EMAAS の相性について評価をし, Mew では不十分である際には, より使いやすく, 便利な EMAAS 対応のインターフェイスを提供する必要がある.

8.　本論文では E-Mail 配送システム上に EMAAS を実装したが, EMAAS は E-Mail に限らず, さまざまなネットワークプロトコル上にも同様に実装することが可能である. 今後 telnet 等他のネットワークプロトコルにも EMAAS を実装する.

また EMAAS は、暗号化プログラムを送信するのみならず, 画像処理プログラムと画像データを送り, 受信先で表示させるなどの応用も可能である. 現在 MIME でも, 画像データを送り, 受信先で画像を表示することは可能である. しかし, MIME では受信者は画像データを処理するためのプログラムを所持していなければならず, 所持していなければ表示はできない. EMAAS では, 受信者が画像データを処理するプログラムを持っていなくても, プログラムを送信することによって受信者が画像データを表示することが可能となる. 例としては, 送信者が受信者に MPEG の動画像データを送信する際には, 最初にネゴシエーションを行い, 受信者が MPEG プレイヤを所持していなければ, 送信者が MPEG プレイヤのプログラムと, MPEG の動画像データを一緒に送信する. 受信者は, 送られて来た MPEG プレイヤのプログラムを実行して, MPEG の動画像を見ることが可能となる.EMAAS は動画像データに限らず様々な用途に応用可能であるから, 今後は EMAAS の応用方法についてより一層考える必要がある.

9.　新たなインターネットプロトコルとして, 現在 **IP V6 (Internet Protocol Version 6)** が開発されている.IP V6 にはセキュリティ機能が含まれており, セキュリティの機能の 1 つとして認証機能がある. IPV6 では, この認証機能によって, IPV6 パケットの内容が改竄されたり, アドレスを書換えようとしたかを見極めることができる.IPV6 の認証機能を用いれば, 安全な鍵配送を行える可能性が存在する. しかし, IPV6 は現在開発段階であり, まだ実用化されていないので, IPV6 についてサーベイをし, 安全な鍵配送の可能性について考える必要がある.

第 7 章

結論

本論文では, ネゴシエーションを行った後暗号化したデータと一緒に暗号化アルゴリズムを送る新しいシステム EMAAS を提案し, 実装した. このシステムは, 柔軟性に富み, 誰もが簡単に使えるシステムである.

本論文で提案した新しいシステムは次のような特徴を持つ.

- ユーザは通信相手がどのような暗号化アルゴリズムを所持しているかを考慮せずに任意の暗号化アルゴリズムを用いて暗号通信を行える
- 暗号通信だけでなく, 様々な通信方法に応用できる, 非常に将来性をもったシステムである. 復号化アルゴリズムの送信に限らず, 他のプログラムを送信する事も可能である.
- 現在は, 実行に時間がかかる, 鍵配送の機能が提供されていない, ネットワーク上での改竄対策が施されていないなど, まだ問題は残っているが, 今後の研究によってこれらの問題は解決される.

EMAAS は, まだ数多くの問題を抱えているが, これらの問題が解決された時, EMAAS は一般ユーザにはなくてはならないものとなるであろう.

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教授 所 真理雄博士に感謝致します.

また, 本研究に関し多大なる御助言を頂きました, 本研究室 後期博士課程の伊藤 純一郎氏に, お忙しい中論文の査読をして頂きました, 本研究室 修士課程 1 年の蓜 秀久氏, 猿渡 隆介氏に感謝致します.

また, 本研究に関し貴重な御意見を頂戴いたしました本研究室 m^3 グループの皆様に感謝致します.

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に感謝致します。

1996 年 2 月

Appendix A

システムの実行例

```
Return-Path: negi@mt.cs.keio.ac.jp
Received: from sunflower.mt.cs.keio.ac.jp (sunflower.mt.cs.keio.ac.jp [131.113.3
2.21]) by frig.mt.cs.keio.ac.jp (8.6.12+2.4W/3.4Wbeta3) with ESMTP id VAA26997 f
or <negi@frig.mt.cs.keio.ac.jp>; Fri, 1 Dec 1995 21:50:35 +0900
Received: from localhost (localhost [127.0.0.1]) by sunflower.mt.cs.keio.ac.jp (
8.6.12+2.4W/3.4Wbeta3) with SMTP id VAA02018 for <negi@mt.cs.keio.ac.jp>; Fri, 1
 Dec 1995 21:50:34 +0900
Message-Id: <199512011250.VAA02018@sunflower.mt.cs.keio.ac.jp>
X-Authentication-Warning: sunflower.mt.cs.keio.ac.jp: Host localhost didn't use
HELO protocol
To: negi@mt.cs.keio.ac.jp
Subject: test
X-Mailer: Mew version 1.00.4 on Emacs 19.28.2, Mule 2.2
Mime-Version: 1.0
Content-Type: Application/emaas; format=mime
Date: Fri, 01 Dec 1995 21:50:33 +0900
From: Tomonori Negi <negi@mt.cs.keio.ac.jp>

------ BEGIN DECODE ALGORITHM ------
(defun des-dec ()
  (interactive)
  (let ((KEYSIZB 1024)
        (KEYSIZ 8))
    (setq key (substring (des-dec-read-pw) 0 8))
    (dodecryption key)
))

(defun dodecryption (key)
  "accept password"
  (interactive)
  (let ((size 8)
        (kk nil)
        (all 0)
        (ex 1)
        (buf ""))
    (setq decbuffer (generate-new-buffer "decbuffer"))
    (setq buffername (current-buffer))
    (setq i 0)
    (while (< i size)
      (setq l 0)
      (setq k (aref key i))
      (setq j 0)
      (while (< j 8)
        (if (= (logand k 1) 1)
            (setq l (1+ l)))
        (setq k (lsh k -1))
        (setq j (1+ j)))
      (if (= (logand l 1) 1)
```

```
      (setq kk (append kk (list (list 0 (logand (aref key i) 127)))))
    (setq kk (append kk (list (list 0 (logior (aref key i) 128))))))
  (setq i (1+ i)))
(setq ks (des-set-key kk))
(setq kk nil)
(setq foriv (list 0 0))
(setq iv (list foriv foriv foriv foriv foriv foriv foriv foriv))
(setq l 1)
(setq rem 0)
(setq bsiz (buffer-size))
(catch 'dec
  (while t
    (if (= ex 1)
        (progn
          (setq buf (fread buf rem (* 1024 8)))
          (setq start (nth 0 buf))
          (setq end (nth 1 buf))
          (setq l (nth 2 buf))
          (setq buf (nth 3 buf))
          (setq all (+ all l))
          (setq ex 0)
          (setq rem (% l 8))
          (setq l (- l rem))))
    (setq obuf (des-cbc-decrypt buf l ks iv))
    (if (>= l 8)
        (setq iv (list (list 0 (aref buf (- l 8)))
                       (list 0 (aref buf (- l 7)))
                       (list 0 (aref buf (- l 6)))
                       (list 0 (aref buf (- l 5)))
                       (list 0 (aref buf (- l 4)))
                       (list 0 (aref buf (- l 3)))
                       (list 0 (aref buf (- l 2)))
                       (list 0 (aref buf (- l 1))))))
    (setq buf (fread buf rem (* 1024 8)))
    (setq start (nth 0 buf))
    (setq end (nth 1 buf))
    (setq ll (nth 2 buf))
    (setq buf (nth 3 buf))
    (setq all (+ all ll))
    (setq ll (+ ll rem))
    (setq rem (% ll 8))
    (setq ll (- ll rem))
    (if (and (>= all bsiz) (= ll 0))
        (progn
          (setq last (aref obuf (1- l)))
          (setq l (+ (- l 8) last))))
    (setq obuf (substring obuf 0 l))
    (setq l ll)
```

```
        (set-buffer decbuffer)
        (insert obuf)
        (set-buffer buffername)
        (if (and (>= all bsiz) (= l 0))
            (throw 'dec nil))))
    (kill-region (point-min) (point-max))
    (insert-buffer decbuffer)
    (kill-buffer decbuffer)
))

(defun des-cbc-decrypt (in l schedule iv)
  (interactive)
  (let ((n 0)
        (m 0)
        (tin0 (list))
        (tin1 (list))
        (out ""))
    (setq xory (c2l iv n))
    (setq n (nth 0 xory))
    (setq xor0 (nth 1 xory))
    (setq xorz (c2l iv n))
    (setq n (nth 0 xorz))
    (setq xor1 (nth 1 xorz))
    (while (> l 0)
      (setq tiny (s2l in m))
      (setq m (nth 0 tiny))
      (setq tin0 (nth 1 tiny))
      (setq tinz (s2l in m))
      (setq m (nth 0 tinz))
      (setq tin1 (nth 1 tinz))
      (setq tin (list tin0 tin1))
      (setq tout (des-decrypt tin schedule))
      (setq tout0 (llogxor (nth 0 tout) xor0))
      (setq tout1 (llogxor (nth 1 tout) xor1))
      (if (>= l 8)
          (progn
            (setq out (l2s tout0 out))
            (setq out (l2s tout1 out)))
        (setq out (l2sn tout0 tout1 out l)))
      (setq xor0 tin0)
      (setq xor1 tin1)
      (setq l (- l 8)))
    out))


(defun des-decrypt (input ks)
  (interactive)
  (let ((l (list 0 0))
```

```
      (r (list 0 0))
      (u (list 0 0))
      (i)
      (v (list 0 0)))
(setq l (nth 0 input))
(setq r (nth 1 input))
(setq u (ip l r))
(setq l (nth 0 u))
(setq r (nth 1 u))
(setq u (nth 2 u))
(setq u (llogior (llsh r 1) (lrsh r 31)))
(setq r (llogior (llsh l 1) (lrsh l 31)))
(setq l u)
(setq l (llogand l (list 255 -1)))
(setq r (llogand r (list 255 -1)))
(setq i 30)
(while (> i 0)
  (setq v (llogxor r (nth i ks)))
  (setq u (llogxor r (nth (1+ i) ks)))
  (setq u (lplus (lrsh u 4) (llsh u 28)))
  (setq l (llogxor
           l
           (llogior (nth (logand (nth 1 u) 63) des-SPtrans1)
                    (llogior
                     (nth (logand (nth 1 (lrsh u 8)) 63) des-SPtrans3)
                     (llogior
                      (nth (logand (nth 1 (lrsh u 16)) 63) des-SPtrans5)
                      (llogior
                       (nth (logand (nth 1 (lrsh u 24)) 63) des-SPtrans7)
                       (llogior
                        (nth (logand (nth 1 v) 63) des-SPtrans0)
                        (llogior
                         (nth (logand (nth 1 (lrsh v 8)) 63) des-SPtrans2)
                         (llogior
                          (nth (logand (nth 1 (lrsh v 16)) 63) des-SPtrans4)
                          (nth (logand (nth 1 (lrsh v 24)) 63) des-SPtrans6)
                          )))))))))
  (setq v (llogxor l (nth (- i 2) ks)))
  (setq u (llogxor l (nth (- i 1) ks)))
  (setq u (lplus (lrsh u 4) (llsh u 28)))
  (setq r (llogxor
           r
           (llogior (nth (logand (nth 1 u) 63) des-SPtrans1)
                    (llogior
                     (nth (logand (nth 1 (lrsh u 8)) 63) des-SPtrans3)
                     (llogior
                      (nth (logand (nth 1 (lrsh u 16)) 63) des-SPtrans5)
                      (llogior
```

```
                         (nth (logand (nth 1 (lrsh u 24)) 63) des-SPtrans7)
                         (llogior
                          (nth (logand (nth 1 v) 63) des-SPtrans0)
                          (llogior
                           (nth (logand (nth 1 (lrsh v 8)) 63) des-SPtrans2)
                           (llogior
                            (nth (logand (nth 1 (lrsh v 16)) 63) des-SPtrans4)
                            (nth (logand (nth 1 (lrsh v 24)) 63) des-SPtrans6)
                            )))))))))
    (setq i (- i 4)))
  (setq l (llogior (lrsh l 1) (llsh l 31)))
  (setq r (llogior (lrsh r 1) (llsh r 31)))
  (setq l (llogand l (list 255 -1)))
  (setq r (llogand r (list 255 -1)))
  (setq u (fp r l))
  (setq r (nth 0 u))
  (setq l (nth 1 u))
  (setq u (nth 2 u))
  (list l r)))

(defun fp (l r)
  "return (l r u)"
  (interactive)
  (setq u (llogand (llogxor (lrsh l 1) r) (list 85 5592405)))
  (setq r (llogxor r u))
  (setq l (llogxor l (llsh u 1)))
  (setq u (llogand (llogxor (lrsh r 8) l) (list 0 -65281)))
  (setq l (llogxor l u))
  (setq r (llogxor r (llsh u 8)))
  (setq u (llogand (llogxor (lrsh l 2) r) (list 51 3355443)))
  (setq r (llogxor r u))
  (setq l (llogxor l (llsh u 2)))
  (setq u (llogand (llogxor (lrsh r 16) l) (list 0 65535)))
  (setq l (llogxor l u))
  (setq r (llogxor r (llsh u 16)))
  (setq u (llogand (llogxor (lrsh l 4) r) (list 15 986895)))
  (setq r (llogxor r u))
  (setq l (llogxor l (llsh u 4)))
  (list l r u))

(defun ip (l r)
  "return (l r u)"
  (interactive)
  (let ((u))
  (setq u (llogand (llogxor (lrsh r 4) l) (list 15 986895)))
  (setq l (llogxor l u))
  (setq r (llogxor r (llsh u 4)))
  (setq u (llogand (llogxor (lrsh l 16) r) (list 0 65535)))
```

```
  (setq r (llogxor r u))
  (setq l (llogxor l (llsh u 16)))
  (setq u (llogand (llogxor (lrsh r 2) l) (list 51 3355443)))
  (setq l (llogxor l u))
  (setq r (llogxor r (llsh u 2)))
  (setq u (llogand (llogxor (lrsh l 8) r) (list 0 -65281)))
  (setq r (llogxor r u))
  (setq l (llogxor l (llsh u 8)))
  (setq u (llogand (llogxor (lrsh r 1) l) (list 85 5592405)))
  (setq l (llogxor l u))
  (setq r (llogxor r (llsh u 1)))
  (list l r u)))

(defun fread (buf rem count)
  "read count char from buffer and return (start end l string)"
  (interactive)
  (let ((l))
    (setq start (point))
    (goto-char (+ start count))
    (setq str (buffer-substring start (point)))
    (setq l (length str))
    (setq str (concat (substring buf 0 rem) str))
    (list start (point) l str)))

(defconst shifts2 [0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0])

(defun des-set-key (in)
  "change password to key  return ((a b) (c d) ...)"
  (interactive)
  (let ((n 0)
        (k nil)
        (i))
    (setq y (c2l in n))
    (setq n (car y))
    (setq c (car (cdr y)))
    (setq z (c2l in n))
    (setq n (car z))
    (setq d (car (cdr z)))
    (setq u (llogand (llogxor (lrsh d 4) c) (list 15 986895)))
    (setq c (llogxor c u))
    (setq d (llogxor d (llsh u 4)))
    (setq u (llogand (llogxor (llsh c 18) c) (list 204 -3407872)))
    (setq c (llogxor (llogxor c u) (lrsh u 18)))
    (setq u (llogand (llogxor (llsh d 18) d) (list 204 -3407872)))
    (setq d (llogxor (llogxor d u) (lrsh u 18)))
    (setq u (llogand (llogxor (lrsh d 1) c) (list 85 5592405)))
    (setq c (llogxor c u))
    (setq d (llogxor d (llsh u 1)))
```

```
(setq u (llogand (llogxor (lrsh c 8) d) (list 0 -65281)))
(setq d (llogxor d u))
(setq c (llogxor c (llsh u 8)))
(setq u (llogand (llogxor (lrsh d 1) c) (list 85 5592405)))
(setq c (llogxor c u))
(setq d (llogxor d (llsh u 1)))
(setq d (llogior
         (llogior
          (llogior (llsh (llogand d (list 0 255)) 16)
                   (llogand d (list 0 65280)))
          (lrsh (llogand d (list 0 -65536)) 16))
         (lrsh (llogand c (list 240 0)) 4)))
(setq c (llogand c (list 15 -1)))
(setq i 0)
(while (< i 16)
  (if (= (aref shifts2 i) 1)
      (progn
        (setq c (llogior (lrsh c 2) (llsh c 26)))
        (setq d (llogior (lrsh d 2) (llsh d 26))))
    (setq c (llogior (lrsh c 1) (llsh c 27)))
    (setq d (llogior (lrsh d 1) (llsh d 27))))
  (setq c (llogand c (list 15 -1)))
  (setq d (llogand d (list 15 -1)))
  (setq s (llogior
           (nth (nth 1 (llogand c (list 0 63))) des-skb0)
           (llogior
            (nth (nth 1 (llogior (llogand (lrsh c 6) (list 0 3))
                                 (llogand (lrsh c 7) (list 0 60))))
                 des-skb1)
            (llogior
             (nth (nth 1 (llogior (llogand (lrsh c 13) (list 0 15))
                                  (llogand (lrsh c 14) (list 0 48))))
                  des-skb2)
             (nth (nth 1 (llogior (llogand (lrsh c 20) (list 0 1))
                                  (llogior (llogand (lrsh c 21) (list 0 6))
                                           (llogand (lrsh c 22)
                                                    (list 0 56)))))
                  des-skb3)))))
  (setq u (llogior
           (nth (nth 1 (llogand d (list 0 63))) des-skb4)
           (llogior
            (nth (nth 1 (llogior (llogand (lrsh d 7) (list 0 3))
                                 (llogand (lrsh d 8) (list 0 60))))
                 des-skb5)
            (llogior
             (nth (nth 1 (llogand (lrsh d 15) (list 0 63)))
                  des-skb6)
             (nth (nth 1 (llogior (llogand (lrsh d 21) (list 0 15))
```

```
                                      (llogand (lrsh d 22) (list 0 48))))
                      des-skb7)))))
      (setq k (append k (list (llogand (llogior (llsh u 16)
                                                (llogand s (list 0 65535)))
                                       (list 255 -1)))))
      (setq s (llogior (lrsh s 16) (llogand u (list 255 -65536))))
      (setq s (llogior (llsh s 4) (lrsh s 28)))
      (setq k (append k (list (llogand s (list 255 -1)))))
      (setq i (1+ i)))
    k))

(defun l2s (l s)
  "in (hoge hogehoge) and out abcdefg"
  (interactive)
    (setq s (concat s (char-to-string (logand (nth 1 l) 255))))
    (setq s (concat s (char-to-string (logand (nth 1 (lrsh l 8)) 255))))
    (setq s (concat s (char-to-string (logand (nth 1 (lrsh l 16)) 255))))
    (setq s (concat s (char-to-string (logand (nth 1 (lrsh l 24)) 255)))))

(defun l2sn (l1 l2 s n)
  (interactive)
  (setq no (char-to-string 0))
  (if (= n 8)
      (setq s (concat s no no no no no no no
                      (char-to-string (logand (nth 1 (lrsh l2 24)) 255)))))
  (if (= n 7)
      (setq s (concat s no no no no no no
                      (char-to-string (logand (nth 1 (lrsh l2 16)) 255)))))
  (if (= n 6)
      (setq s (concat s no no no no no
                      (char-to-string (logand (nth 1 (lrsh l2 8)) 255)))))
  (if (= n 5)
      (setq s (concat s no no no no
                      (char-to-string (logand (nth 1 l2) 255)))))
  (if (= n 4)
      (setq s (concat s no no no
                      (char-to-string (logand (nth 1 (lrsh l1 24)) 255)))))
  (if (= n 3)
      (setq s (concat s no no
                      (char-to-string (logand (nth l (lrsh l1 16)) 255)))))
  (if (= n 2)
      (setq s (concat s no
                      (char-to-string (logand (nth 1 (lrsh l1 8)) 255)))))
  (if (= n 1)
      (setq s (concat s (char-to-string (logand (nth 1 l1) 255))))))

(defun c2l (c n)
  "in ((a b) (c d) (e f) ...) and out (n l)"
```

```
  (interactive)
  (let ((l (list 0 0)))
    (setq l (nth n c))
    (setq n (1+ n))
    (setq l (llogior l (llsh (nth n c) 8)))
    (setq n (1+ n))
    (setq l (llogior l (llsh (nth n c) 16)))
    (setq n (1+ n))
    (setq l (llogior l (llsh (nth n c) 24)))
    (setq n (1+ n))
    (list n l)))

(defun s2l (s n)
  "in abcdefg and out (n l)"
  (interactive)
  (let ((l (list 0 0)))
    (setq l (list 0 (aref s n)))
    (setq n (1+ n))
    (setq l (llogior l (llsh (list 0 (aref s n)) 8)))
    (setq n (1+ n))
    (setq l (llogior l (llsh (list 0 (aref s n)) 16)))
    (setq n (1+ n))
    (setq l (llogior l (llsh (list 0 (aref s n)) 24)))
    (setq n (1+ n))
    (list n l)))

(defun des-dec-read-pw ()
  "read pass word"
  (interactive)
  (let ((count 0)
        (size 1025)
        (ok 0))
    (catch 'ok
      (while t
        (setq key (read-pw size "ENTER password: "))
        (setq ver-key (read-pw size "Verifying password: "))
        (if (and (string= key ver-key) (>= (length key) 8))
            (progn
              (setq ok 1)
              (throw 'ok nil))
          (if (>= count 3)
              (progn
                (setq ok 0)
                (setq key nil)
                (throw 'ok nil)))
          )))
    (setq ver-key nil)
    key))
```

```
(defun read-pw (size prompt)
  (interactive)
  (let ((pass)
        (c 0)
        (echo-keystrokes 0)
        (ociea cursor-in-echo-area))
    (unwind-protect
        (progn
          (setq cursor-in-echo-area 1)
          (while (and (/= c ?\r) (/= c ?\n) (/= c ?\e) (/= c 7))
            (message "%s%s"
                     prompt
                     (make-string (length pass) ?.))
            (setq c (read-char))
            (if (= c ?\C-u)
                (setq pass "")
              (if (and (/= c ?\b) (/= c ?\177))
                  (setq pass (concat pass (char-to-string c)))
                (if (> (size pass) 0)
                    (setq pass (substring pass 0 -1))))))
          (setq cursor-in-echo-area -1)
          )
      (setq cursor-in-echo-area ociea)
      nil)
    (message "")
    (sit-for 0)
    (substring pass 0 -1)))

(defconst des-skb0
  (list (list 0 0) (list 0 16) (list 32 0) (list 32 16)
        (list 0 65536) (list 0 65552) (list 32 65536) (list 32 65552)
        (list 0 2048) (list 0 2064) (list 32 2048) (list 32 2064)
        (list 0 67584) (list 0 67600) (list 32 67584) (list 32 67600)
        (list 0 32) (list 0 48) (list 32 32) (list 32 48)
        (list 0 65568) (list 0 65584) (list 32 65568) (list 32 65584)
        (list 0 2080) (list 0 2096) (list 32 2080) (list 32 2096)
        (list 0 67616) (list 0 67632) (list 32 67616) (list 32 67632)
        (list 0 524288) (list 0 524304) (list 32 524288) (list 32 524303)
        (list 0 589824) (list 0 589840) (list 32 589824) (list 32 589840)
        (list 0 526336) (list 0 526352) (list 32 526336) (list 32 526352)
        (list 0 591872) (list 0 591888) (list 32 591872) (list 32 591888)
        (list 0 524320) (list 0 524336) (list 32 524320) (list 32 524336)
        (list 0 589856) (list 0 589872) (list 32 589856) (list 32 589872)
        (list 0 526368) (list 0 526384) (list 32 526368) (list 32 526384)
        (list 0 591904) (list 0 591920) (list 32 591904) (list 32 591920)))

(defconst des-skb1
```

```
  (list (list 0 0) (list 2 0) (list 0 8192) (list 2 8192)
        (list 0 2097152) (list 2 2097152) (list 0 2105344) (list 2 2105344)
        (list 0 4) (list 2 4) (list 0 8196) (list 2 8196)
        (list 0 2097156) (list 2 2097156) (list 0 2105348) (list 2 2105348)
        (list 0 1024) (list 2 1024) (list 0 9216) (list 2 9216)
        (list 0 2098176) (list 2 2098176) (list 0 2106368) (list 2 2106368)
        (list 0 1028) (list 2 1028) (list 0 9220) (list 2 9220)
        (list 0 2098180) (list 2 2098180) (list 0 2106372) (list 2 2106372)
        (list 16 0) (list 18 0)  (list 16 8192) (list 18 8192)
        (list 16 2097152) (list 18 2097152) (list 16 2105344) (list 18 2105344)
        (list 16 4) (list 18 4) (list 16 8196) (list 18 8196)
        (list 16 2097156) (list 18 2097156) (list 16 2105348) (list 18 2105348)
        (list 16 1024) (list 18 1024) (list 16 9216) (list 18 9216)
        (list 16 2098176) (list 18 2098176) (list 16 2106368) (list 18 2106368)
        (list 16 1028) (list 18 1028) (list 16 9220) (list 18 9220)
        (list 16 2098180) (list 18 2098180) (list 16 2106372) (list 18 2106372)
))

(defconst des-skb2
  (list (list 0 0) (list 0 1) (list 0 262144) (list 0 262145)
        (list 1 0) (list 1 1) (list 1 262144) (list 1 262145)
        (list 0 2) (list 0 3) (list 0 262146) (list 0 262147)
        (list 1 2) (list 1 3) (list 1 262146) (list 1 262147)
        (list 0 512) (list 0 513) (list 0 262656) (list 0 262657)
        (list 1 512) (list 1 513) (list 1 262656) (list 1 262657)
        (list 0 514) (list 0 515) (list 0 262658) (list 0 262659)
        (list 1 514) (list 1 515) (list 1 262658) (list 1 262659)
        (list 8 0) (list 8 1) (list 8 262144) (list 8 262145)
        (list 9 0) (list 9 1) (list 9 262144) (list 9 262145)
        (list 8 2) (list 8 3) (list 8 262146) (list 8 262147)
        (list 9 2) (list 9 3) (list 9 262146) (list 9 262147)
        (list 8 512) (list 8 513) (list 8 262656) (list 8 262657)
        (list 9 512) (list 9 513) (list 9 262656) (list 9 262657)
        (list 8 514) (list 8 515) (list 8 262658) (list 8 262659)
        (list 9 514) (list 9 515) (list 9 262658) (list 9 262659)))

(defconst des-skb3
  (list (list 0 0) (list 0 1048576) (list 0 256) (list 0 1048832)
        (list 0 8) (list 0 1048584) (list 0 264) (list 0 1048840)
        (list 0 4096) (list 0 1052672) (list 0 4352) (list 0 1052928)
        (list 0 4104) (list 0 1052680) (list 0 4360) (list 0 1052936)
        (list 4 0) (list 4 1048576) (list 4 256) (list 4 1048832)
        (list 4 8) (list 4 1048584) (list 4 264) (list 4 1048840)
        (list 4 4096) (list 4 1052672) (list 4 4352) (list 4 1052928)
        (list 4 4104) (list 4 1052680) (list 4 4360) (list 4 1052936)
        (list 0 131072) (list 0 1179648) (list 0 131328) (list 0 1179904)
        (list 0 131080) (list 0 1179656) (list 0 131336) (list 0 1179912)
        (list 0 135168) (list 0 1183744) (list 0 135424) (list 0 1184000)
```

```
        (list 0 135176) (list 0 1183752) (list 0 135432) (list 0 1184008)
        (list 4 131072) (list 4 1179648) (list 4 131328) (list 4 1179904)
        (list 4 131080) (list 4 1179656) (list 4 131336) (list 4 1179912)
        (list 4 135168) (list 4 1183744) (list 4 135424) (list 4 1184000)
        (list 4 135176) (list 4 1183752) (list 4 135432) (list 4 1184008)))

(defconst des-skb4
  (list (list 0 0) (list 16 0) (list 0 65536) (list 16 65536)
        (list 0 4) (list 16 4) (list 0 65540) (list 16 65540)
        (list 32 0) (list 48 0) (list 32 65536) (list 48 65536)
        (list 32 4) (list 48 4) (list 32 65540) (list 48 65540)
        (list 0 1048576) (list 16 1048576) (list 0 1114112) (list 16 1114112)
        (list 0 1048580) (list 16 1048580) (list 0 1114116) (list 16 1114116)
        (list 32 1048576) (list 48 1048576) (list 32 1114112) (list 48 1114112)
        (list 32 1048580) (list 48 1048580) (list 32 1114116) (list 48 1114116)
        (list 0 4096) (list 16 4096) (list 0 69632) (list 16 69632)
        (list 0 4100) (list 16 4100) (list 0 69636) (list 16 69636)
        (list 32 4096) (list 48 4096) (list 32 69632) (list 48 69632)
        (list 32 4100) (list 48 4100) (list 32 69632) (list 48 69636)
        (list 0 1052672) (list 16 1052672) (list 0 1118208) (list 16 1118208)
        (list 0 1052676) (list 16 1052676) (list 0 1118212) (list 16 1118212)
        (list 32 1052672) (list 48 1052672) (list 32 1118208) (list 48 1118208)
        (list 32 1052676) (list 48 1052676) (list 32 1118212) (list 48 1118212)
))

(defconst des-skb5
  (list (list 0 0) (list 8 0) (list 0 8) (list 8 8)
        (list 0 1024) (list 8 1024) (list 0 1032) (list 8 1032)
        (list 0 131072) (list 8 131072) (list 0 131080) (list 8 131080)
        (list 0 132096) (list 8 132096) (list 0 132104) (list 8 132104)
        (list 0 1) (list 8 1) (list 0 9) (list 8 9)
        (list 0 1025) (list 8 1025) (list 0 1033) (list 8 1033)
        (list 0 131073) (list 8 131073) (list 0 131081) (list 8 131081)
        (list 0 132097) (list 8 132097) (list 0 132105) (list 8 132105)
        (list 2 0) (list 10 0) (list 2 8) (list 10 8)
        (list 2 1024) (list 10 1024) (list 2 1032) (list 10 1032)
        (list 2 131072) (list 10 131072) (list 2 131080) (list 10 131080)
        (list 2 132096) (list 10 132096) (list 2 132104) (list 10 132104)
        (list 2 1) (list 10 1) (list 2 9) (list 10 9)
        (list 2 1025) (list 10 1025) (list 2 1033) (list 10 1033)
        (list 2 131073) (list 10 131073) (list 2 131081) (list 10 131081)
        (list 2 132097) (list 10 132097) (list 2 132105) (list 10 132105)))

(defconst des-skb6
  (list (list 0 0) (list 0 256) (list 0 524288) (list 0 524544)
        (list 1 0) (list 1 256) (list 1 524288) (list 1 524544)
        (list 0 16) (list 0 272) (list 0 524304) (list 0 524560)
        (list 1 16) (list 1 272) (list 1 524304) (list 1 524560)
```

```
        (list 0 2097152) (list 0 2097408) (list 0 2621440) (list 0 2621696)
        (list 1 2097152) (list 1 2097408) (list 1 2621440) (list 1 2621696)
        (list 0 2097168) (list 0 2097424) (list 0 2621456) (list 0 2621712)
        (list 1 2097168) (list 1 2097424) (list 1 2621456) (list 1 2621712)
        (list 0 512) (list 0 768) (list 0 524800) (list 0 525056)
        (list 1 512) (list 1 768) (list 1 524800) (list 1 525056)
        (list 0 528) (list 0 784) (list 0 524816) (list 0 525072)
        (list 1 528) (list 1 784) (list 1 524816) (list 1 525072)
        (list 0 2097664) (list 0 2097920) (list 0 2621952) (list 0 2622208)
        (list 1 2097664) (list 1 2097920) (list 1 2621952) (list 1 2622208)
        (list 0 2097680) (list 0 2097936) (list 0 2621968) (list 0 2622224)
        (list 1 2097680) (list 1 2097936) (list 1 2621968) (list 1 2622224)))

(defconst des-skb7
  (list (list 0 0) (list 4 0) (list 0 262144) (list 4 262144)
        (list 0 2) (list 4 2) (list 0 262146) (list 4 262146)
        (list 0 8192) (list 4 8192) (list 0 270336) (list 4 270336)
        (list 0 8194) (list 4 8194) (list 0 270338) (list 4 270338)
        (list 0 32) (list 4 32) (list 0 262176) (list 4 262176)
        (list 0 34) (list 4 34) (list 0 262178) (list 4 262178)
        (list 0 8224) (list 4 8224) (list 0 270368) (list 4 270368)
        (list 0 8226) (list 4 8226) (list 0 270370) (list 4 270370)
        (list 0 2048) (list 4 2048) (list 0 264192) (list 4 264192)
        (list 0 2050) (list 4 2050) (list 0 264194) (list 4 264194)
        (list 0 10240) (list 4 10240) (list 0 272384) (list 4 272384)
        (list 0 10242) (list 4 10242) (list 0 272386) (list 4 272386)
        (list 0 2080) (list 4 2080) (list 0 264224) (list 4 264224)
        (list 0 2082) (list 4 2082) (list 0 264226) (list 4 264226)
        (list 0 10272) (list 4 10272) (list 0 272416) (list 4 272416)
        (list 0 10274) (list 4 10274) (list 0 272418) (list 4 272418)))

(defconst des-SPtrans0
  (list
   (list 0 -8257024) (list 0 131072) (list 128 -8388608) (list 128 -8257024)
   (list 0 -8388608) (list 128 131584) (list 128 131072) (list 128 -8388608)
   (list 128 131584) (list 0 -8257024) (list 0 -8257536) (list 128 512)
   (list 128 -8388096) (list 0 -8388608) (list 0 0) (list 128 131072)
   (list 0 131072) (list 128 0) (list 0 -8388096) (list 0 131584)
   (list 128 -8257024) (list 0 -8257536) (list 128 512) (list 0 -8388096)
   (list 128 0) (list 0 512) (list 0 131584) (list 128 -8257536)
   (list 0 512) (list 128 -8388096) (list 128 -8257536) (list 0 0)
   (list 0 0) (list 128 -8257024) (list 0 -8388096) (list 128 131072)
   (list 0 -8257024) (list 0 131072) (list 128 512) (list 0 -8388096)
   (list 128 -8257536) (list 0 512) (list 0 131584) (list 128 -8388608)
   (list 128 131584) (list 128 0) (list 128 -8388608) (list 0 -8257536)
   (list 128 -8257024) (list 0 131584) (list 0 -8257536) (list 128 -8388096)
   (list 0 -8388608) (list 128 512) (list 128 131072) (list 0 0)
   (list 0 131072) (list 0 -8388608) (list 128 -8388096) (list 0 -8257024)
```

```
   (list 128 0) (list 128 -8257536) (list 0 512) (list 128 131584)))

(defconst des-SPtrans1
  (list
   (list 16 270340) (list 0 0) (list 0 270336) (list 16 262144)
   (list 16 4) (list 0 8196) (list 16 8192) (list 0 270336)
   (list 0 8192) (list 16 262148) (list 0 4) (list 16 8192)
   (list 0 262148) (list 16 270336) (list 16 262144) (list 0 4)
   (list 0 262144) (list 16 8196) (list 16 262148) (list 0 8192)
   (list 0 270340) (list 16 0) (list 0 0) (list 0 262148)
   (list 16 8196) (list 0 270340) (list 16 270336) (list 16 4)
   (list 16 0) (list 0 262144) (list 0 8196) (list 16 270340)
   (list 0 262148) (list 16 270336) (list 16 8192) (list 0 270340)
   (list 16 270340)(list 0 262148) (list 16 4) (list 0 0)
   (list 16 0) (list 0 8196) (list 0 262144) (list 16 262148)
   (list 0 8192) (list 16 0) (list 0 270340) (list 16 8196)
   (list 16 270336) (list 0 8192) (list 0 0) (list 16 4)
   (list 0 4) (list 16 270340) (list 0 270336) (list 16 262144)
   (list 16 262148) (list 0 262144) (list 0 8196) (list 16 8192)
   (list 16 8196) (list 0 4) (list 16 262144) (list 0 270336)))

(defconst des-SPtrans2
  (list
   (list 65 0) (list 1 65600) (list 0 64) (list 65 64)
   (list 64 65536) (list 1 0) (list 65 64) (list 0 65600)
   (list 1 64) (list 0 65536) (list 1 65536) (list 64 0)
   (list 65 65600) (list 64 64) (list 64 0) (list 65 65536)
   (list 0 0) (list 64 65536) (list 1 65600) (list 0 64)
   (list 64 64) (list 65 65600) (list 0 65536) (list 65 0)
   (list 65 65536) (list 1 64) (list 64 65600) (list 1 65536)
   (list 0 65600) (list 0 0) (list 1 0) (list 64 65600)
   (list 1 65600) (list 0 64) (list 64 0) (list 0 65536)
   (list 64 64) (list 64 65536) (list 1 65536) (list 65 64)
   (list 0 0) (list 1 65600) (list 0 65600) (list 65 65536)
   (list 64 65536) (list 1 0) (list 65 65600) (list 64 0)
   (list 64 65600) (list 65 0) (list 1 0) (list 65 65600)
   (list 0 65536) (list 1 64) (list 65 64) (list 0 65600)
   (list 1 64) (list 0 0) (list 65 65536) (list 64 64)
   (list 65 0) (list 64 65600) (list 0 64) (list 1 65536)))

(defconst des-SPtrans3
  (list
   (list 0 1049602) (list 4 1024) (list 0 2) (list 4 1049602)
   (list 0 0) (list 4 1048576) (list 4 1026) (list 0 1048578)
   (list 4 1049600) (list 4 2) (list 4 0) (list 0 1026)
   (list 4 2) (list 0 1049602) (list 0 1048576) (list 4 0)
   (list 4 1048578) (list 0 1049600) (list 0 1024) (list 0 2)
   (list 0 1049600) (list 4 1026) (list 4 1048576) (list 0 1024)
```

```
   (list 0 1026) (list 0 0) (list 0 1048578) (list 4 1049600)
   (list 4 1024) (list 4 1048578) (list 4 1049602) (list 0 1048576)
   (list 4 1048578) (list 0 1026) (list 0 1048576) (list 4 2)
   (list 0 1049600) (list 4 1024) (list 0 2) (list 4 1048576)
   (list 4 1026) (list 0 0) (list 0 1024) (list 0 1048578)
   (list 0 0) (list 4 1048578) (list 4 1049600) (list 0 1024)
   (list 4 0) (list 4 1049602) (list 0 1049602) (list 0 1048576)
   (list 4 1049602) (list 0 2) (list 4 1024) (list 0 1049602)
   (list 0 1048578) (list 0 1049600) (list 4 1048576) (list 4 1026)
   (list 0 1026) (list 4 0) (list 4 2) (list 4 1049600)))

(defconst des-SPtrans4
  (list
   (list 2 0) (list 0 16384) (list 0 256) (list 2 16648)
   (list 2 16392) (list 2 256) (list 0 16648) (list 2 16384)
   (list 0 16384) (list 0 8) (list 2 8) (list 0 16640)
   (list 2 264) (list 2 16392) (list 2 16640) (list 0 0)
   (list 0 16640) (list 2 0) (list 0 16392) (list 0 264)
   (list 2 256) (list 0 16648) (list 0 0) (list 2 8)
   (list 0 8) (list 2 264) (list 2 16648) (list 0 16392)
   (list 2 16384) (list 0 256) (list 0 264) (list 2 16640)
   (list 2 16640) (list 2 264) (list 0 16392) (list 2 16384)
   (list 0 16384) (list 0 8) (list 2 8) (list 2 256)
   (list 2 0) (list 0 16640) (list 2 16648) (list 0 0)
   (list 0 16648) (list 2 0) (list 0 256) (list 0 16392)
   (list 2 264) (list 0 256) (list 0 0) (list 2 16648)
   (list 2 16392) (list 2 16640) (list 0 264) (list 0 16384)
   (list 0 16640) (list 2 16392) (list 2 256) (list 0 264)
   (list 0 8) (list 0 16648) (list 2 16384) (list 2 8)))

(defconst des-SPtrans5
  (list
   (list 32 16) (list 0 524304) (list 0 0) (list 32 526336)
   (list 0 524304) (list 0 2048) (list 32 2064) (list 0 524288)
   (list 0 2064) (list 32 526352) (list 0 526336) (list 32 0)
   (list 32 2048) (list 32 16) (list 32 524288) (list 0 526352)
   (list 0 524288) (list 32 2064) (list 32 524304) (list 0 0)
   (list 0 2048) (list 0 16) (list 32 526336) (list 32 524304)
   (list 32 526352) (list 32 524288) (list 32 0) (list 0 2064)
   (list 0 16) (list 0 526336) (list 0 526352) (list 32 2048)
   (list 0 2064) (list 32 0) (list 32 2048) (list 0 526352)
   (list 32 526336) (list 0 524304) (list 0 0) (list 32 2048)
   (list 32 0) (list 0 2048) (list 32 524304) (list 0 524288)
   (list 0 524304) (list 32 526352) (list 0 526336) (list 0 16)
   (list 32 526352) (list 0 526336) (list 0 524288) (list 32 2064)
   (list 32 16) (list 32 524288) (list 0 526352) (list 0 0)
   (list 0 2048) (list 32 16) (list 32 2064) (list 32 526336)
   (list 32 524288) (list 0 2064) (list 0 16) (list 32 524304)))
```

```
(defconst des-SPtrans6
  (list
   (list 0 4096) (list 0 128) (list 0 4194432) (list 0 4194305)
   (list 0 4198529) (list 0 4097) (list 0 4224) (list 0 0)
   (list 0 4194304) (list 0 4194433) (list 0 129) (list 0 4198400)
   (list 0 1) (list 0 4198528) (list 0 4198400) (list 0 129)
   (list 0 4194433) (list 0 4096) (list 0 4097) (list 0 4198529)
   (list 0 0) (list 0 4194432) (list 0 4194305) (list 0 4224)
   (list 0 4198401) (list 0 4225) (list 0 4198528) (list 0 1)
   (list 0 4225) (list 0 4198401) (list 0 128) (list 0 4194304)
   (list 0 4225) (list 0 4198400) (list 0 4198401) (list 0 129)
   (list 0 4096) (list 0 128) (list 0 4194304) (list 0 4198401)
   (list 0 4194433) (list 0 4225) (list 0 4224) (list 0 0)
   (list 0 128) (list 0 4194305) (list 0 1) (list 0 4194432)
   (list 0 0) (list 0 4194433) (list 0 4194432) (list 0 4224)
   (list 0 129) (list 0 4096) (list 0 4198529) (list 0 4194304)
   (list 0 4198528) (list 0 1) (list 0 4097) (list 0 4198529)
   (list 0 4194305) (list 0 4198528) (list 0 4198400) (list 0 4097)))

(defconst des-SPtrans7
  (list
   (list 8 2097184) (list 8 2129920) (list 0 32800) (list 0 0)
   (list 8 32768) (list 0 2097184) (list 8 2097152) (list 8 2129952)
   (list 0 32) (list 8 0) (list 0 2129920) (list 0 32800)
   (list 0 2129952) (list 8 32800) (list 8 32) (list 8 2097152)
   (list 0 32768) (list 0 2129952) (list 0 2097184) (list 8 32768)
   (list 8 2129952) (list 8 32) (list 0 0) (list 0 2129920)
   (list 8 0) (list 0 2097152) (list 8 32800) (list 8 2097184)
   (list 0 2097152) (list 0 32768) (list 8 2129920) (list 0 32)
   (list 0 2097152) (list 0 32768) (list 8 32) (list 8 2129952)
   (list 0 32800) (list 8 0) (list 0 0) (list 0 2129920)
   (list 8 2097184) (list 8 32800) (list 8 32768) (list 0 2097184)
   (list 8 2129920) (list 0 32) (list 0 2097184) (list 8 32768)
   (list 8 2129952) (list 0 2097152) (list 8 2097152) (list 8 32)
   (list 0 2129920) (list 0 32800) (list 8 32800) (list 8 2097152)
   (list 0 32) (list 8 2129920) (list 0 2129952) (list 0 0)
   (list 8 0) (list 8 2097184) (list 0 32768) (list 0 2129952)))

(defun llsh (list n)
  "32bit left shift (a b)->(hoge hogehoge)"
  (interactive)
  (setq high (car list))
  (setq low (car (cdr list)))
  (while (> n 0)
    (setq high (lsh high 1))
    (if (= (logand -8388608 low) -8388608)
        (setq high (1+ high)))
```

```
    (setq high (logand 255 high))
    (setq low (lsh low 1))
    (setq n (1- n)))
  (list high low))

(defun lrsh (list n)
  "32bit right shift (a b)->(hoge hogehoge)"
  (interactive)
  (setq high (car list))
  (setq low (car (cdr list)))
  (while (> n 0)
    (setq low (lsh low -1))
    (if (= (logand 1 high) 1)
        (setq low (- low 8388608)))
    (setq high (lsh high -1))
    (setq n (1- n)))
  (list high low))

(defun llogand (in1 in2)
  "32bit and (a b)->(hoge hogehoge)"
  (interactive)
  (list (logand (car in1) (car in2)) (logand (car (cdr in1)) (car (cdr in2)))))

(defun llogior (in1 in2)
  "32bit or (a b)->(hoge hogehoge)"
  (interactive)
  (list (logior (car in1) (car in2)) (logior (car (cdr in1)) (car (cdr in2)))))

(defun llogxor (in1 in2)
  "32bit xor (a b)->(hoge hogehoge)"
  (interactive)
  (list (logxor (car in1) (car in2)) (logxor (car (cdr in1)) (car (cdr in2)))))

(defun lplus (in1 in2)
  (interactive)
  (let ((u 0)
        (v 0))
    (if (and (= (nth 0 in2) 0) (= (nth 1 in2) 0))
        (llogand in1 (list 255 -1))
      (setq u (llogxor in1 in2))
      (setq v (llsh (llogand in1 in2) 1))
      (lplus u v))))
```

```
------ BEGIN MESSAGE ------
quxT5bAmGrPWG5Y7+a0WQ2kBvtq8N3u1A5PSlOpvQ4l8+ViO7RQWXw==
------ END MESSAGE ------
```

参考文献

[mime I] Nathaniel Borenstein, Ned Freed Multipurpose Internet Mail Extensions (MIME) Part One: Format of INternet Message Bodies In *Network Working Group Internet Draft RFC1341*. May 1995.

[mime II] Nathaniel Borenstein, Ned Freed Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types In *Network Working Group Internet Draft RFC1522*. May 1995.

[mime IV] Ned Freed, Innosoft, John Postel, ISI Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures In *Network Working Group Internet Draft RFC1424*. August 1995.

[mime V] Nathaniel Borenstein, Ned Freed Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples In *Network Working Group Internet Draft RFC1425*. August 1995.

[RFC 822] David H. Crocker Standard for the format of ARPA Internet text messages In *Network Working Group Internet Draft RFC822*. August 1982.

[mew 94a] 山本 和彦, 櫻井 三子, 乃村 能成 Mew - 構造を持つメールへのインターフェイス In マルチメディア通信と分散処理ワークショップ. Oct, 1994.

[mew 94b] 山本 和彦 Mew における MIME の取り扱いについて In *WIDE* 研究会. Dec, 1994.

[mew-PEM] 山本 和彦, 櫻井 三子 Emacs における MIME と PEM のインターフェイスについて In *WIDE* 研究会. May, 1994.

[mew-PGP] 山本 和彦 PGP MIME Integration In *Internet-Draft*. Oct, 1995.

[crypt] A.Lempel Cryptology in Transition. In *ACM Computing Surveys*. Vol. 11 ,No. 4 ,pp.285-304.

[DES 77] Whitfield Difie and Martin E. Hellman Exhaustive Cryptanalysis of the NBS Data Encryption Standard In *Computer*, v. 10, n. 6, Jun 1977, pp.74-84.

[DES] Bruce Scheneier APPLIED CRYPTOGRAPHY SECOND EDITION 12. In *John Wiley & Sons, Inc*, pp.265-301.

[DES 93] Michael J. Wiener Efficient DES Key Search. In *presented at the rump session of CRYPTO ’93*,Aug. 1993.

[DES 91] Eli Biham and Adi Shamir Differential Cryptanalysis of DES-like Cryptosystems. In *Journal of Cryptology*, v4. n. 1, 1991, pp3-72.

[SAFER K-64 94] James Masscy SAFER K-64 A Byte-Oriented Block-Ciphering Algorithm. In *Fast Software Encryption, Cambridge Security Workshop Proceedings.* Springer Verlag ,1994 ,pp.1-17.

[SAFER] Bruce Scheneier APPLIED CRYPTOGRAPHY SECOND EDITION 14.4 SAFER. In *John Wiley & Sons, Inc*, pp.339-341.

[IDEA 93a] Willi Meier On the Security of the IDEA Block Cipher. In *Advances in Cryptology EUROCRYPT ’93.* Springer Verlag , 1993 ,pp.371-385.

[IDEA 93b] Joan Daemen, Ren’e Govaerts and Vandewalle Weak Keys for IDEA In *Advances in Cryptology EUROCRYPT ’93.* Springer Verlag , 1993 ,pp.224-231.

[IDEA] Bruce Scheneier APPLIED CRYPTOGRAPHY SECOND EDITION 13.9 IDEA. In *John Wiley & Sons, Inc*, pp.319-325.

[FEAL 87a] 清水 明宏, 宮口 庄司高速データ暗号アルゴリズム FEAL In 電子情報通信学会論文誌 *’‘877VOL.J70-D No.7.* 電子情報通信学会, 1987, pp.1413-1423.

[FEAL 87b] Akihiro Shimizu & Shoji Miyaguchi Fast Data Encryption Algorithm FEAL In *Advances in Cryptology EUROCRYPT ’87.* Springer Verlag , 1987 ,pp.267-278.

[FEAL] Bruce Scheneier APPLIED CRYPTOGRAPHY SECOND EDITION 13.4 FEAL. In *John Wiley & Sons, Inc*, pp.308-311.

[BLOWFISH 94] Bruce Scheneier Descryption of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish). In *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer Verlag , 1994 ,pp.191-204.

[BLOWFISH] Bruce Scheneier APPLIED CRYPTOGRAPHY SECOND EDITION 14.3 BLOWFISH. In *John Wiley & Sons, Inc*, pp.336-339.

[GOST 94] C. Charnes, L. O'Connor, J. Pieprzyk, R. Safavi-Naini, Y. Zheng Comments on Soviet Encryption Algorithm In *Advances in Cryptology EUROCRYPT '94*. Springer Verlag , 1994 ,pp.433-438.

[GOST] Bruce Scheneier APPLIED CRYPTOGRAPHY SECOND EDITION 14.1 GOST. In *John Wiley & Sons, Inc*, pp.331-334.

[RSA1] Bruce Scheneier APPLIED CRYPTOGRAPHY SECOND EDITION 19.3 RSA. In *John Wiley & Sons, Inc*, pp.466-474.

[RSA2] アルト・サロマー 公開鍵暗号系. In 東京電機大学出版局.

[PGP a] Simson Garfinkel PGP: PRETTY GOOD PRIVACY In *O'Reilly & Associates, Inc.*.

[PGP b] Bruce Scheneier APPLIED CRYPTOGRAPHY SECOND EDITION 24.12 PGP. In *John Wiley & Sons, Inc*, pp.584-587.

[PEM 95] 菊池 浩明, 黒田 康嗣, 永井 武 プライバシ強化メール PEM における証明書配布局の実装と評価. In 情報処理学会論文誌,Vol. 36 No.8, 1995. Aug.

[PEM] Bruce Scheneier APPLIED CRYPTOGRAPHY SECOND EDITION 24.10 PEM. In *John Wiley & Sons, Inc*, pp.577-584.

[GNU Emacs Lisp 18] Bill Lewis GNU Emacs Lisp Manual -Version 18 2nd DRAFT-. In 株式会社ビレッジセンター出版局

[GNU Emacs Lisp 19.29] Bil Lewis, Dan LaLiberte, and the GNU Manual Group GNU Emacs Lisp リファレンス マニュアル GNU Emacs Version 19.29. In 透土社.

[Computer Architecture] John L.Hennessy, David A.Patterson Conputer Architecture: A Quantitative Approach. In 日経 *BP* 出版センター