ブロック暗号アルゴリズムＭＩＳＴＹ

松井 充

三菱電機株式会社
情報技術総合研究所
〒 247 鎌倉市大船 5-1-1
matsui@iss.isl.melco.co.jp

あらまし　新しい秘密鍵暗号アルゴリズム MISTY1 および MISTY2 を提案する。MISTY1 および MISTY2 は 128 ビットの暗号化鍵をもつ 64 ビットブロック暗号であり、安全性の点では差分解読法と線形解読法に対する証明可能安全性を実現している点が大きな特長である。またソフトウエア・ハードウエアを問わず高速な暗号化処理が実現できるよう設計されており、例えば８段の MISTY1 はソフトウエアでは Pentium 100MHz で 20Mbps，PA7200 120MHz では 40Mbps の暗号化速度を達成している。本稿ではこれら暗号アルゴリズムの設計原理および詳細仕様、ならびにサンプルプログラムを記載する。

和文キーワード　暗号設計，ブロック暗号，差分解読法，線形解読法，証明可能安全性

Block Encryption Algorithm MISTY

Mitsuru MATSUI

Information R&D Center
Mitsubishi Electric Corporation
5-1-1 Ofuna, Kamakura, 247, Japan
matsui@iss.isl.melco.co.jp

Abstract　We propose new secret-key cryptosystems MISTY1 and MISTY2, which are block ciphers with 128-bit key and 64-bit data block. They are provably secure against differential and linear cryptanalysis, and also fast on software implementations as well as on hardware platforms. Our experimental results show that MISTY1 with eight rounds encrypts data stream at the speed of 20Mbps and 40Mbps on Pentium 100MHz and PA7200 120MHz, respectively. In this paper, we show our design principles of MISTY1 and MISTY2 and publish their detailed description with sample programs written in C language.

英文 key words　Design, Block Cipher, Differential Cryptanalysis, Linear Cryptanalysis, Provable Security

1　はじめに

本稿では我々が新しく設計した暗号アルゴリズム MISTY1 および MISTY2 の設計基準並びに詳細仕様を報告する。MISTY1 および MISTY2 はそれぞれ 128 ビットの暗号化鍵をもつ 64 ビットブロック暗号であり、我々はその設計にあたって十分な安全性と、ハードウエア・ソフトウエアを問わずあらゆるプラットフォームでの高速性を両立させるよう工夫を行なった。以下これら二つの暗号アルゴリズムを総称して MISTY と呼ぶ。

MISTY の安全性の根拠の一つは、差分解読法と線形解読法に対する証明可能安全性 (provable security) 構造の採用である。差分解読法と線形解読法は、ブロック暗号の汎用的な解読法として現在最も強力なものとされており、暗号設計者にとってこれらに対する対策は大きな課題となっているが、我々は証明可能安全性の理論に基づいて、線形解読法や差分解読法に対する安全性が数値的に保証されるよう MISTY の設計を行なった。この他にも、サイズの異なる置換表の利用や、鍵に依存して形の変化する線形変換関数の採用など安全性を高める考慮を行なっている。

さらに MISTY は並列処理構造を強く意識して設計されており、このためハードウエアや、あるいは命令の並列処理が可能なプロセッサ上ではソフトウエアでも高い処理能力を発揮することができる。この結果例えば MISTY1 (８段版) の CBC モードでの暗号化速度は、ハードウエアでは 300Mbps （0.5μ gate-array，ソフトウエアシミュレーションによる typical 速度値）、またソフトウエアでは Intel Pentium 100MHz 上で 20Mbps、また HP PA7200 120MHz 上では 40Mbps (アセンブリ言語，それぞれ実測結果) を確認した。

以下では、 MISTY を設計するにあたって我々が設けた基本設計基準ならびにその検討内容について詳述する。なお MISTY1 および MISTY2 の詳細仕様は本稿の付録１に示されており、その構造は読者に完全に transparent である。また読者自身による評価を助けるため、 MISTY1 (８段版) のＣ言語によるサンプルプログラムを付録２に示した。

2　基本設計基準

我々は MISTY を設計するにあたって、十分な安全性と、ハードウエア・ソフトウエアを問わずあらゆるプラットフォームでの高速性を両立させるため、次の３つの基本設計基準を設定した。

1. 安全性に関する何らかの数値的な根拠をもつこと。
2. プロセッサの種類によらずソフトウエアで実用的な性能を達成すること。
3. ハードウエア上で十分な高速性を実現すること。

一般に実用的な暗号とは、固定長の暗号化鍵で多量のデータを暗号化しても、現実的な意味で安全性が損なわれない方式であり、したがって MISTY に無条件の安全性（情報論的安全性）を求めることはもとより不可能である。しかしながら我々は、安全性に関して信頼性のある何らかの数値的根拠は必要と考えた。特にブロック暗号については、差分解読法と線形解読法に対する対策は暗号設計者にとって不可欠でありながら、現実には、ほとんどすべてのブロック暗号で、これらの解読法に対する厳密な安全性は証明されていないのが現状である。そこで我々は差分解読法と線形解読法に対する証明可能安全性を実現することにより、これら解読法に対する安全性を数値的に保証することで最初の基本設計基準にのぞんだ。この詳細については第４節で述べる。

次にソフトウエアでの性能について我々が注目した点は、最近提案される多くのブロック暗号アルゴリズムが、特定の仕様のプロセッサ（例えば 32 ビットＡＬＵをもつ等）で最高の速度を達成するように設計されているため、しばしばそれ以外の仕様のプロセッサでは性能が大きく低下することである。これに対して我々は、特定の仕様のプロセッサで最高の性能を追求することよりも、あらゆるプロセッサで適度な高速性と小型化を実現することを重要視し、マルチプラットフォームに対応できることを選択した。この結果、例えば特定のプロセッサでのみ高速な命令は MISTY には採用しない方針をとった。

さらにハードウエアに関しては、最近提案されるほとんどの暗号アルゴリズムがソフトウエアでの実現を前提としており、このため暗号アルゴリズムによっては、ハードウエアでは極端に規模が大きくなったり、あるいはソフトウエアにくらべて速度向上があまり期待できないことがあることに注目した。これに対して我々は、 MISTY のハードウエアでの性能目標を、ソフトウエアでは実現不可能な数百 Mbps クラスの通信路にも対応できることとした。具体的には例えばテーブル参照命令の場合、ソフトウエアでは一般にそのテーブルの内容と速度の関係は少ないのに対し、ハードウエアでは、場合によってはテーブルの内容を直接論理回路で構成することによって速度を飛躍的に改善できる場合がある。そこで MISTY ではテーブル内容をハードウエア向きに可能な限り最適化するなど、ハードウエアの特性をできる限り生かした構造をとることとした。

3 基本演算要素の検討

次に我々は、基本設計基準を満たすようなブロック暗号アルゴリズムに採用すべき基本的な演算要素を分類し、それぞれの項目について安全性や、ソフトウエア・ハードウエアでの実用性について検討を行なった。本節ではこの検討結果について記述する。

- 論理演算
  AND, OR, XOR のような論理演算（特に XOR）はブロック暗号アルゴリズムの最も一般的な演算要素であり、ソフトウエアでもハードウエアでも高速に実現できるため、実用上大変都合がよい。しかしながら安全性をこれらの演算に求めることは困難である。

- 算術演算
  加算や減算は、すべてのプロセッサで一命令で実行でき、またデータランダム化にも一定の寄与があるため多くの暗号アルゴリズムで採用されている演算要素である。一方これをハードウエアで実現すると、データランダム化の割には速度が低速であるという点と、算術演算は差分解読法と線形解読法に対する証明可能安全性をもつ構造を作りにくいという問題点がある。

- シフト演算
  シフト演算（特に回転シフト）も多くの暗号アルゴリズムで利用されている演算要素である。これは間接的にデータランダム化に寄与し、しかもハードウエアではロジックが不要であるという大きな長所がある。しかしソフトウエアでは対象とするデータの長さによって速度が大きく影響される。例えば 32 ビットデータを対象とする回転シフトを 8 あるいは 16 ビットプロセッサで行う場合、命令数の増加による速度低下はしばしば無視できないことがある。

- テーブル参照
  テーブル参照の効率はソフトウエアで行う場合メモリアクセススピードに大きく依存する。最近のプロセッサはメモリの読み書きが条件つきながら 1 サイクル（あるいはそれ以下）で実行できるため、昔のようなメモリアクセスのペナルティは少なくなっている。一方ハードウエアでのテーブル参照は、 ROM などを使った場合一般に低速であるが、デーブルの内容によっては直接論理回路で組むと大変高速になる場合がある。また安全性の点からは、テーブル参照は一度の実行でランダム性を大きくあげることができるという長所がある。

これらの考察と、ソフトウエアとハードウエアの性能を両立させるという観点から、我々は MISTY アルゴリズムの基本演算要素はハードウエア用に調整されたテーブル参照を基本とし、これに論理演算を組み合わせたものとすることにした。

4 証明可能安全性の理論

本節では、筆者による文献 [4] に基づいて差分解読法と線形解読法に対する証明可能安全性の理論を紹介する。このトピック自身興味深い研究テーマであるが、ここでは紙面の関係で MISTY の安全性に直接関係する部分だけをとり出してまとめることにした。この理論は MISTY の安全性の根幹をなすものである。

4.1 平均差分確率・平均線形確率

一般に n ビットの入力 x と、ℓ ビットの鍵パラメータ k をもつ関数 $F_k(x)$ の平均差分確率 DP^F および平均線形確率 LP^F を次のように定義する。

$$DP^F \stackrel{def}{=} \frac{1}{2^\ell}\sum_k \max_{\Delta x\neq 0,\Delta y} \frac{\#\{x|F_k(x)\oplus F_k(x\oplus\Delta x)=\Delta y\}}{2^n} \tag{1}$$

$$LP^F \stackrel{def}{=} \frac{1}{2^\ell}\sum_k \max_{\Gamma x,\Gamma y\neq 0} \left(2\frac{\#\{x|x\bullet\Gamma x = F_k(x)\bullet\Gamma y\}}{2^n}-1\right)^2 \tag{2}$$

$F_k(x)$ が暗号化関数の場合は、これらの値が差分解読法や線形解読法に対する厳密な安全性の指標となるのである。これらの値は小さいほど差分解読法や線形解読法に対して安全であると言えるため、それが十分小さいことが理論的に証明される時、暗号化関数 $F_k(x)$ は差分解読法と線形解読法に対する証明可能安全性を持つという。なお、以下では鍵パラメータを持たない関数 $F(x)$ に対しても、$\ell = 0$ として上の定義を適用するものとする。

4.2 証明可能安全性の基本定理

次の三つの定理は、「小さい」関数の平均差分確率／平均線形確率とそれを組み合わせた「大きい」関数の平均差分確率／平均線形確率との関係を明らかにするものである。これによって「小さく強い」関数を組み合わせて「大きく強い」関数を組み立てることができる。定理１は、最初は平均差分確率に対して Knudsen と Nyberg が [1]、続いて Nyberg が平均線形確率の場合に証明したものである [2]。

定理 1 図１のアルゴリズムで、すべての内部関数 f_i が全単射で、その平均差分確率／平均線形確率が p 以下である時、３段以上あれば全体の関数の平均差分確率／平均線形確率は p^2 以下となる。

なお正確には、文献 [1] ではこの評価が p^2 ではなく $2p^2$ となっている。しかし最近青木・太田が、より厳密な評価値 p^2 が成り立つことを発見した [3]。

続いて筆者は、定理１の記述が図２のアルゴリズム場合に対しても全く同様に成り立つことを示した [4]。図１のアルゴリズムと図２アルゴリズムの本質的な違いは、図２の場合は内部関数 f_i が並列に処理できるため、より高速に処理できるという点である。

定理 2 図２のアルゴリズムで、すべての内部関数 f_i が全単射で、その平均差分確率／平均線形確率が p 以下である時、３段以上あれば全体の関数の平均差分確率／平均線形確率は p^2 以下である。

さて、以上の定理は入力データが内部で二等分されるというのが前提であったが、筆者は二等分でない場合にも類似の公式が成り立つことを示した [4]。具体的には、図３のアルゴリズムで入力データは左右に n_1 ビットと n_2 ビット（但し $n_1 \geq n_2$）にそれぞれ分割される場合を考えるのである。ここで、奇数段目の内部関数 f_i 終了後の排他的論理和は、n_2 ビットのデータを n_1 ビットにゼロ拡張したのち行なうものとし、さらに偶数段目の内部関数 f_i 終了後の排他的論理和は、n_1 ビットのデータの先頭 $n_1 - n_2$ ビットを切り捨てることによって n_2 ビットにしたのち行なうものとする。この約束のもとで次の一般的な定理が成り立つ。

定理 3 図３のアルゴリズムで、すべての内部関数 f_i が全単射で、その平均差分確率／平均線形確率がそれぞれ p_i 以下である時、３段以上あれば全体の関数の平均差分確率／平均線形確率は次に示す値以下である。

$$max\{p_1p_2,\ p_2p_3,\ 2^{n_1-n_2}p_1p_3\} \tag{3}$$

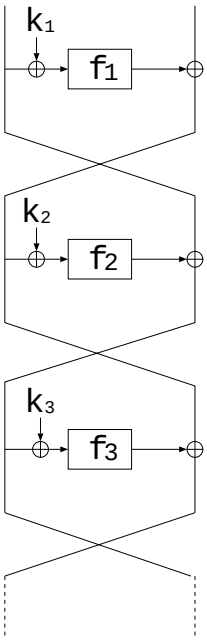

図１

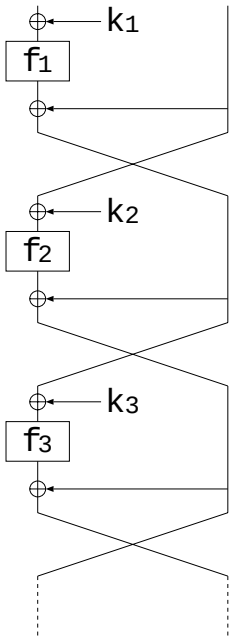

図２

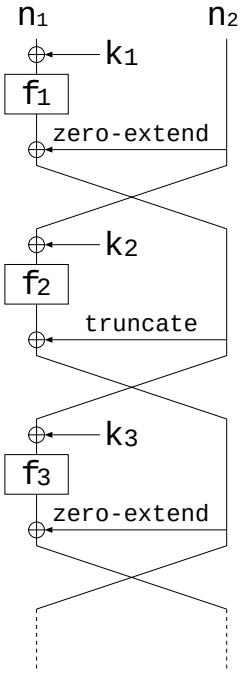

図３

5 MISTY データランダム化部の設計

本節以下では MISTY の具体的な設計についてのべる。なお MISTY1 および MISTY2 アルゴリズムの完全な記述については付録１を参照のこと。

5.1 基本構造

MISTY のデータランダム化部の設計にあたり、我々は差分解読法／線形解読法に対する証明可能安全性の定理を再帰的に用いることによって、サイズの小さいテーブルを組み合わせてアルゴリズム全体を構成するというプロセスをとった。これは例えば、図２のアルゴリズムの内部関数 f_i の構造として再び図２のアルゴリズムを用いるのである。この場合、最も小さい関数の平均差分確率／平均線形確率を p とすれば、定理２によってアルゴリズム全体の確率を p^4 以下にすることができる。

さて、64 ビットブロック暗号に定理１あるいは定理２を適用すると（定理１を適用したものが MISTY1 で、定理２を適用したものが MISTY2 である）、内部関数は 32 ビットの入出力をもつことになる（これを MISTY では FO 関数と呼ぶ）が、この関数にもう一度定理２を適用すると、最も小さい関数（これを MISTY では FI 関数と呼ぶ）の入出力サイズは 16 ビットとなる。しかしこれでは、FI 関数はまだテーブルとしては大きすぎるので再び再帰構造を用いることにした。ここで我々は 16 ビットを 8 ビットと 8 ビットに分割するのではなく、定理３を利用して、16 ビットを 9 ビットと 7 ビットに分割するという方針をとった（図４）。

この理由は、全単射関数の平均差分確率／平均線形確率の最小値は、その関数の入出力サイズが偶数ビットか奇数ビットかで異なっているためである。すなわち n が奇数の時には、n ビット入出力をもつ全単射関数の平均差分確率／平均線形確率の最小値は 2^{-n+1} であるのに対し、n が偶数の場合にはこの値が 2^{-n+2} であると予想されている (未解決問題)。従って 16 ビットを 8 ビットと 8 ビットに分割した場合には、（上の予想が正しいと仮定すれば）暗号アルゴリズム全体の平均差分確率／平均線形確率は $(((2^{-8+2})^2)^2)^2 = 2^{-48}$ 以下しか保証できないのに対して、16 ビットを 9 ビットと 7 ビットに分割した場合には $((2^{-9+1}2^{-7+1})^2)^2 = 2^{-56}$ 以下を保証することができる。

このように奇数ビットへの不等分分割は、差分解読法と線形解読法に対する安全上の利点があるが、その一方で次のような２つの欠点も持っている。そのひとつは等分分割の場合に比べてアルゴリズムの並列度が低下する点であり、もうひとつはソフトウエアで構成した場合に速度が低下する点である。しかしながら、2^{-48} と 2^{-56} の差は大きいことと、同じサイズのテーブルのみを用いるよりも異なる２つのサイズのテーブルを用いる方が安全上望ましいと予想されることなどから不等分分割を最終的に採用することとした。以下、この 9 ビット入出力の全単射関数を S_9，7 ビット入出力の全単射関数を S_7 と記述することにする。

なお、S_9 のテーブルサイズはソフトウエアで普通に構成した場合 1024 バイトが必要なため、ソフトウエア容量の観点から FI 関数の第１段と第３段では同じテーブルを用いることとしている。

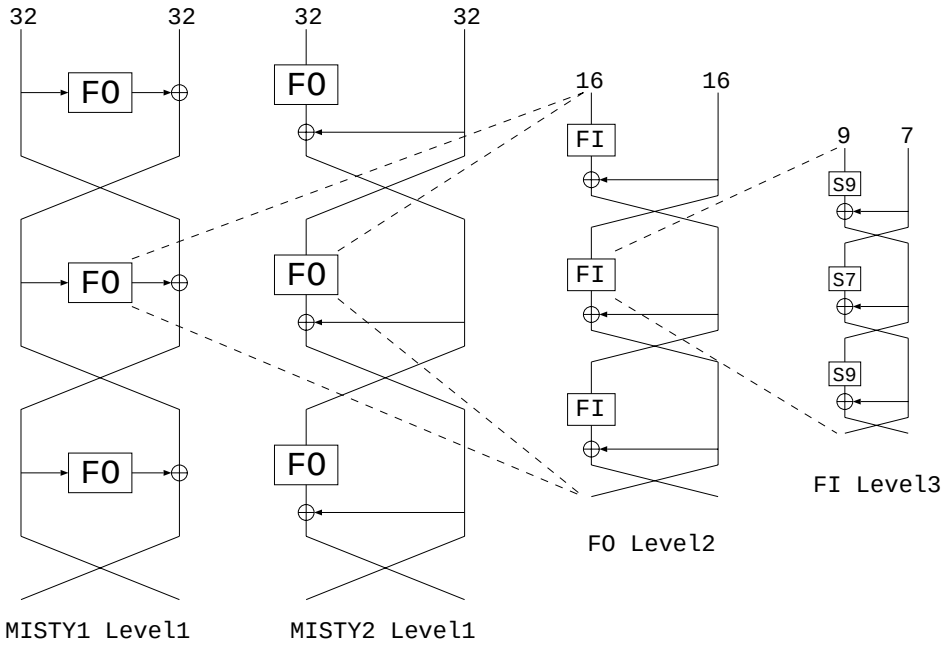

図４． MISTY データランダム化部の基本再帰構造

実際の MISTY1 および MISTY2 では安全性に対する柔軟性を保つため、外側（図４の Level1）の段数（これを MISTY の段数という）は可変の仕様にしているが、FI 関数および FO 関数は３段固定にしている。さてここで MISTY1 と MISTY2 の速度を比較してみよう。並列度を考慮にいれなければ、同じ段数の MISTY1 と MISTY2 の暗号化／復号化に必要な計算量は明らかに同じであるが、並列度を考慮すると、これらの暗号化／復号化時間は同じではない。これは、MISTY1 は FI 関数が２つずつ並列計算できるのに対して、MISTY2（の暗号化）では FI 関数が４つずつ並列計算できるのが基本的な理由である。

今最大に並列処理が行なわれるものとの仮定のもとで、MISTY1 および MISTY2 全体の暗号化／復号化時間がどの程度になるかを計算したのが表１である。ここで段数 n は４の倍数とし、演算時間は S_9 の計算時間何回分に相当するかで評価している。なお、簡単のため S_7 の計算時間は S_9 と同程度と仮定し、また排他的論理和の計算時間は無視している。この表からわかるように MISTY2 は MISTY1 に比べて暗号化は一般に高速であるが ECB および CBC モードでの復号化は MISTY1 の方が高速である（これは MISTY2 の逆演算には並列処理が一切使えないことによる）。従って MISTY2 は OFB や CFB モードに適した方式と言うことができる。

	暗号化 (ECB,CBC,OFB,CFB)	復号化 (ECB,CBC)	復号化 (OFB,CFB)
n 段 MISTY1	$3n$	$3n$	$3n$
n 段 MISTY2	$1.5n$	$9n$	$1.5n$

表１．並列処理を考慮した MISTY1, MISTY2 の暗号化／復号化時間（S_9 の演算回数）

5.2 S_7 および S_9

S_7 および S_9 の選択にあたって我々はまず、以下の３つの条件をこの優先順位で考慮に入れた。

1. 平均差分確率／平均線形確率が最小値をとること。
2. ハードウエア遅延がなるべく小さいこと。
3. 代数次数 (algebraic degree) がなるべく高いこと。

第一の条件である平均差分確率／平均線形確率については、この値が最小（S_9 の場合は 2^{-8}，S_7 の場合は 2^{-6}）となる関数の一般形として現在（恐らくただ一つ）知られているのは有限体上の巾乗関数である。そこで我々は最初 $S_i(x) = A \circ x^{\alpha} \circ B \quad (i = 7, 9)$ の形をとる関数全体について、（以下に示す基準に基づいて）ハードウエア遅延を計算する予定であった。ここで A および B は任意の全単射線形変換であり、α は $(2^i - 1, \alpha) = 1$ を満たす任意の整数（これは巾乗関数が全単射になるための必要十分条件）とする。

しかしながら、実際には上にあげた形のすべての関数についてハードウエア遅延を調べることは計算量的に困難であった。そこで我々は次に、$S_i(x) = A \circ x^{\alpha} \quad (i = 7, 9)$ の形をとる全単射関数全体で、$GF(2)$ 上の基底として、多項式基底と正規基底をもつものすべてを探索範囲とすることとした。すなわち、最初の定義ではあらゆる基底がその探索範囲に含まれるのに対して、我々が行なったのは、入力側の基底に一定の制限をつけた範囲内での探索ということである。

さて、ここで我々の採用したハードウエア遅延の定義が必要であるが、それには若干の準備が必要である。今、一般に i ビットの入出力をもつ関数 $y = f(x)$ に対して、x および y をバイナリ列を用いて $x = (x_0, x_1, x_2, ..., x_{i-1})$, $y = (y_0, y_1, y_2, ..., y_{i-1})$ とあらわす時（すなわち $GF(2)$ 上の適当な基底に関する係数をベクトル表示した時）、次の形の表示を $f(x)$ の出力第 a ビット y_a の AND–XOR 表現と言う。

$$y_a = e^{(0)} + \sum_{0 \le j_1 < n} e^{(1)}_{j_1} x_{j_1} + \sum_{0 \le j_1 < j_2 < n} e^{(2)}_{j_1, j_2} x_{j_1} x_{j_2} + \sum_{0 \le j_1 < j_2 < j_3 < n} e^{(3)}_{j_1, j_2, j_3} x_{j_1} x_{j_2} x_{j_3} + ... \tag{4}$$

ここで $e^{(0)}$,$e^{(1)}_{j_1}$,$e^{(2)}_{j_1,j_2}$, $e^{(3)}_{j_1,j_2,j_3}$ .... は０または１であり、和はビットの排他的論理和をあらわすものとする。ここで我々は y_a のハードウエア遅延を、上式にあらわれる項の総数、すなわち y_a を AND–XOR 表現を用いて論理回路を構成するのに必要な２入力 XOR ゲートの個数 +1 と定義した。さらに関数 f のハードウエア遅延を、f の全ての出力ビットのハードウエア遅延のうちの最大値と定義した。

もちろん、正確なハードウエア速度の見積りには AND ゲートの個数やゲートリダクションの効果を考慮する必要があるが、後に述べるように、我々は結果的に代数次数の小さなものしか扱わなかったことと、有限体上の巾乗関数を対象としてなるべく簡単で効率の良い相対比較指標を目指したため、上のような定義を採用することにした。

y_a の代数次数 (algebraic degree) とは、 AND–XOR 表現に現れる項の最大次数で定義され、さらに関数 f の代数次数とは、 f の全ての出力ビットの代数次数のうちの最大値と定義される。例えば $S_i(x) = A \circ x^\alpha \circ B$ の場合には、代数次数は α のビット重みに一致する。

S_7 の選択

先に示した $S_7(x)$ の候補すべてに対してハードウエア遅延を計算し、次の順序で候補の絞り込みを行なった。まず、代数次数とハードウエア遅延の関係をしらべたところ次の結果が得られた。

- 代数次数が 4 以上の場合は $S_7(x)$ のどの出力ビットのハードウエア遅延も 21 以上になる。
- 代数次数が 3 の場合は $S_7(x)$ のどの出力ビットのハードウエア遅延も 10 以上になる。
- 代数次数が 2 の場合は $S_7(x)$ のどの出力ビットのハードウエア遅延も 7 以上になる。

代数次数が 4 以上の場合は遅延が大き過ぎるとの判断から、我々はこの時点で $S_7(x)$ の候補を代数次数 3 のものに絞り込むこととした。次にこの中から $S_7(x)$ 全体としてのハードウエア遅延が最も小さいものを探索したところ、それは遅延 13 で、しかもそれはビットの順序を除けば 1 つだけであった。そこでこれに定数を加えることで（$S_7(0) \neq 0$ とするため）最終的な S_7 とした。その論理は以下の通りである。

$$
\begin{aligned}
y_0 &= x_0 + x_1x_3 + x_0x_3x_4 + x_1x_5 + x_0x_2x_5 + x_4x_5 + x_0x_1x_6 + x_2x_6 + x_0x_5x_6 + x_3x_5x_6 + 1\\
y_1 &= x_0x_2 + x_0x_4 + x_3x_4 + x_1x_5 + x_2x_4x_5 + x_6 + x_0x_6 + x_3x_6 + x_2x_3x_6 + x_1x_4x_6 + x_0x_5x_6 + 1\\
y_2 &= x_1x_2 + x_0x_2x_3 + x_4 + x_1x_4 + x_0x_1x_4 + x_0x_5 + x_0x_4x_5 + x_3x_4x_5 + x_1x_6 + x_3x_6 + x_0x_3x_6 + x_4x_6 + x_2x_4x_6\\
y_3 &= x_0 + x_1 + x_0x_1x_2 + x_0x_3 + x_2x_4 + x_1x_4x_5 + x_2x_6 + x_1x_3x_6 + x_0x_4x_6 + x_5x_6 + 1\\
y_4 &= x_2x_3 + x_0x_4 + x_1x_3x_4 + x_5 + x_2x_5 + x_1x_2x_5 + x_0x_3x_5 + x_1x_6 + x_1x_5x_6 + x_4x_5x_6 + 1\\
y_5 &= x_0 + x_1 + x_2 + x_0x_1x_2 + x_0x_3 + x_1x_2x_3 + x_1x_4 + x_0x_2x_4 + x_0x_5 + x_0x_1x_5 + x_3x_5 + x_0x_6 + x_2x_5x_6\\
y_6 &= x_0x_1 + x_3 + x_0x_3 + x_2x_3x_4 + x_0x_5 + x_2x_5 + x_3x_5 + x_1x_3x_5 + x_1x_6 + x_1x_2x_6 + x_0x_3x_6 + x_4x_6 + x_2x_5x_6
\end{aligned}
$$

S_9 の選択

同様に $S_9(x)$ について、代数次数とハードウエア遅延の関係をしらべたところ次の結果が得られた。

- 代数次数が 3 以上の場合は $S_9(x)$ のどの出力ビットのハードウエア遅延も 27 以上になる。
- 代数次数が 2 の場合は $S_9(x)$ のどの出力ビットのハードウエア遅延も 9 以上になる。

代数次数が 3 以上の場合は遅延が大き過ぎるとの判断から、我々はこの時点で代数次数 2 のものに絞り込むこととした。次にこの中から $S_9(x)$ 全体としてのハードウエア遅延が最も小さいものを探索したところ、それは遅延 12 で、全部で 9 通り存在した。このなかから 1 つを無作為にえらんで定数を加えることで（$S_9(0) \neq 0$ とするため）最終的な S_9 とした。その論理は以下の通りである。

$$
\begin{aligned}
y_0 &= x_0x_4 + x_0x_5 + x_1x_5 + x_1x_6 + x_2x_6 + x_2x_7 + x_3x_7 + x_3x_8 + x_4x_8 + 1\\
y_1 &= x_0x_2 + x_3 + x_1x_3 + x_2x_3 + x_3x_4 + x_4x_5 + x_0x_6 + x_2x_6 + x_7 + x_0x_8 + x_3x_8 + x_5x_8 + 1\\
y_2 &= x_0x_1 + x_1x_3 + x_4 + x_0x_4 + x_2x_4 + x_3x_4 + x_4x_5 + x_0x_6 + x_5x_6 + x_1x_7 + x_3x_7 + x_8\\
y_3 &= x_0 + x_1x_2 + x_2x_4 + x_5 + x_1x_5 + x_3x_5 + x_4x_5 + x_5x_6 + x_1x_7 + x_6x_7 + x_2x_8 + x_4x_8\\
y_4 &= x_1 + x_0x_3 + x_2x_3 + x_0x_5 + x_3x_5 + x_6 + x_2x_6 + x_4x_6 + x_5x_6 + x_6x_7 + x_2x_8 + x_7x_8\\
y_5 &= x_2 + x_0x_3 + x_1x_4 + x_3x_4 + x_1x_6 + x_4x_6 + x_7 + x_3x_7 + x_5x_7 + x_6x_7 + x_0x_8 + x_7x_8\\
y_6 &= x_0x_1 + x_3 + x_1x_4 + x_2x_5 + x_4x_5 + x_2x_7 + x_5x_7 + x_8 + x_0x_8 + x_4x_8 + x_6x_8 + x_7x_8 + 1\\
y_7 &= x_1 + x_0x_1 + x_1x_2 + x_2x_3 + x_0x_4 + x_5 + x_1x_6 + x_3x_6 + x_0x_7 + x_4x_7 + x_6x_7 + x_1x_8 + 1\\
y_8 &= x_0 + x_0x_1 + x_1x_2 + x_4 + x_0x_5 + x_2x_5 + x_3x_6 + x_5x_6 + x_0x_7 + x_0x_8 + x_3x_8 + x_6x_8 + 1
\end{aligned}
$$

5.3 FL 関数

我々はさらに安全性を高めるべく、証明可能安全性の性質を損なわないように FO 関数および FI 関数以外の関数を付加することを考えた。このため我々は、鍵によって形が変化する線形関数で、簡単に構成できるものという条件のもとで適当な関数を探索した。この結果生まれたのが FL 関数である。

FL 関数は鍵が固定されている限り線形な関数であり、従って証明可能安全性に影響を与えないが、鍵に関しては線形でないので差分解読法や線形解読法以外の解読の可能性を軽減する可能性があると考えた。また FL 関数は、その演算要素として AND,OR,XOR だけを用いているのでハードウエア・ソフトウエアを問わず高速に処理を行なうことができる。

6 MISTY 鍵スケジュール部の設計

鍵の全数探索攻撃に対する十分な安全性を確保するため MISTY の暗号化鍵は 128 ビットとしているが、このほか我々は MISTY の鍵スケジュール部の設計にあたり、安全性と実用性の観点から次の設計基準を設定した。

- 拡大鍵生成時間は暗号化時間に比べて十分短いものにする。
- データランダム化部のすべての段にすべての暗号化鍵のビットを影響させる。
- データランダム化部の各段にできるだけ多くの拡大鍵のビットを影響させる。
- 拡大鍵は 256 ビットとする。

鍵スケジュール部の設計において我々が最も念頭においたことは、実用的な観点からできるだけ軽い構造にすることである。これは、可能な限りランダムな拡大鍵を生成しようとする最近の傾向（最も極端な例として、ハッシュ関数の入力に暗号化鍵をとり、その出力を拡大鍵とする方法がある）と、ある意味では正反対の方向を持ったものである。しかしながら現実には（特にハードウエアで構成した場合には）、鍵スケジュール部の規模および遅延は無視できないこともあり、我々は実用的な観点から鍵交換時のオーバーヘッドをできるだけ軽減するような鍵スケジュール部を設計することとした。この結果 MISTY では鍵スケジュール部の演算として、データランダム化部の FI 関数をそのまま用いている。鍵スケジュール部全体の演算時間は、並列処理を最大限に考慮すれば FI 関数一回の演算時間と同じである。

また、 related-key attack などの鍵スケジュールの構造に着目した暗号解読法（related-key attack は多くの場合現実的な解読法ではないが）に対抗するため、我々はデータランダム化部の各段に暗号化鍵のすべてのビットを影響させるとともに、拡大鍵についてもできるだけ多くのビットを各段に影響させように考慮している。 MISTY では FL 関数一段あたり 32 ビット、また FO 関数一段あたり 112 ビットの拡大鍵が入力されている。

ところで拡大鍵を 256 ビットにしている大きな理由は、 MISTY をソフトウエアで構成する場合に拡大鍵全体をレジスタにのせることで高速性を確保するためである。この技巧は、レジスタ数の少ない Intel 系のプロセッサではもとより実現することは不可能であるが、 MIPS や PA-RISC のような RISC 系プロセッサではレジスタ数が多いので、 256 ビットの拡大鍵なら全体を確実にレジスタにのせることが可能である。これによってメモリアクセス回数が大幅に削減されるのである。

7 おわりに

本稿では新しいブロック暗号アルゴリズム MISTY1 および MISTY2 の設計基準について報告した。仕様上これらの段数は可変であるが、現段階では設計者は、安全性と速度のバランスから MISTY1 は 8 段、 MISTY2 は 12 段で利用することを推奨する。その詳細仕様については付録 1 を参照されたい。

暗号の実用性は多くの第三者によって評価検証されなければならないことはもちろんであり、設計者は MISTY1 および MISTY2 の客観的な評価を歓迎する。付録 2 に、この評価を助けるため設計者自身が C 言語で記述した 8 段仕様 MISTY1 のサンプルプログラムを掲載した。このプログラムは高い移植性が意識されておりおそらくあらゆるプラットフォームとコンパイラで動作するものである。

参考文献

[1] Nyberg, K., Knudsen, L.,: Provable Security against Differential Cryptanalysis. Journal of Cryptology, Vol.8, no.1 (1995)

[2] Nyberg, K.,: Linear Approximation of Block Ciphers. Advances in Cryptology – Eurocrypt'94, Lecture Notes in Computer Science **950**, Springer Verlag (1994)

[3] 青木和麻呂, 太田和夫: 最大平均差分確率および最大平均線形確率のより厳密な評価. Proceedings of SCIS'96, SCIS96-4A (1996)

[4] Matsui, M.,: New Structure of Block Ciphers with Provable Security against Differential and Linear Cryptanalysis. Proceedings of the third international workshop of fast software encryption, Lecture Notes in Computer Science **1039**, Springer Verlag (1996)

Block Cipher Algorithms MISTY1 and MISTY2

Version 1.11 October 2 1996
Mitsubishi Electric Corporation

This document shows a complete description of encryption algorithms MISTY1 and MISTY2, which are secret-key ciphers with 64-bit data block and 128-bit secret key. The number of rounds n of MISTY1 and MISTY2 is variable under the condition that n is a multiple of four.

Data Randomizing Part

- Figures 1 and 2 show data randomizing parts of MISTY1 and MISTY2, respectively. The plaintext is divided into two 32-bit data, which are transformed by bitwise XOR operations denoted by the symbol $\oplus$ and sub-functions FO_i $(1 \le i \le n)$ and FL_i $(1 \le i \le n+2)$. FO_i uses a 64-bit subkey KO_i and a 48-bit subkey KI_i. FL_i uses a 32-bit subkey KL_i.
- Figure 3 shows the structure of FO_i. The input is divided into two 16-bit data, which are transformed by bitwise XOR operations denoted by the symbol $\oplus$ and sub-functions FI_{ij} $(1 \le j \le 3)$, where KO_{ij} $(1 \le j \le 4)$ and KI_{ij} $(1 \le j \le 3)$ are the j-th (from left) 16-bit data of KO_i and KI_i, respectively.
- Figure 4 shows the structure of FI_{ij}. The input is divided into left 9-bit data and right 7-bit data, which are transformed by bitwise XOR operations denoted by the symbol $\oplus$ and substitution tables S_7 and S_9. In the first and third XORs, the 7-bit data is zero-extended to 9 bits, and in the second XOR, the 9-bit data is truncated to 7 bits by discarding its highest two bits. KI_{ij1} and KI_{ij2} are left 7-bit data and right 9-bit data of KI_{ij}, respectively.
- Figure 5 shows the structure of FL_i. The input is divided into two 16-bit data, which are transformed by bitwise XOR operations denoted by the symbol $\oplus$, a bitwise AND operation denoted by the symbol $\cap$ and a bitwise OR operation denoted by the symbol $\cup$, where KL_{ij} $(1 \le j \le 2)$ is the j-th (from left) 16-bit data of KL_i.
- Tables 1 and 2 show decimal representation of the substitution tables S_7 and S_9, respectively.

Key Scheduling Part

- Let K_i $(1 \le i \le 8)$ be the i-th (from left) 16-bit data of the secret key K, and let K'_i $(1 \le i \le 8)$ be the output of FI_{ij} where the input of FI_{ij} is K_i and the key KI_{ij} is K_{i+1}. Also, identify K_9 with K_1.
- The correspondence between the symbols KO_{ij},KI_{ij},KL_{ij} and the actual key is as follows:

Symbol	KO_{i1}	KO_{i2}	KO_{i3}	KO_{i4}	KI_{i1}	KI_{i2}	KI_{i3}	KL_{i1}	KL_{i2}
Key	K_i	K_{i+2}	K_{i+7}	K_{i+4}	K'_{i+5}	K'_{i+1}	K'_{i+3}	$K_{\frac{i+1}{2}}$ (odd i) $K'_{\frac{i}{2}+2}$ (even i)	$K'_{\frac{i+1}{2}+6}$ (odd i) $K_{\frac{i}{2}+4}$ (even i)

where K_i and K'_i are identified with K_{i-8} and K'_{i-8}, respectively, when i exceeds 8.

Test Data

- The following is sample data for MISTY1 with eight rounds in hexadecimal form:

Secret Key (K_1 to K_8)	`00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff`
Plaintext	`01 23 45 67 89 ab cd ef`
Extended Key (K'_1 to K'_8)	`cf 51 8e 7f 5e 29 67 3a cd bc 07 d6 bf 35 5e 11`
Ciphertext	`8b 1d a5 f5 6a b3 d0 7c`

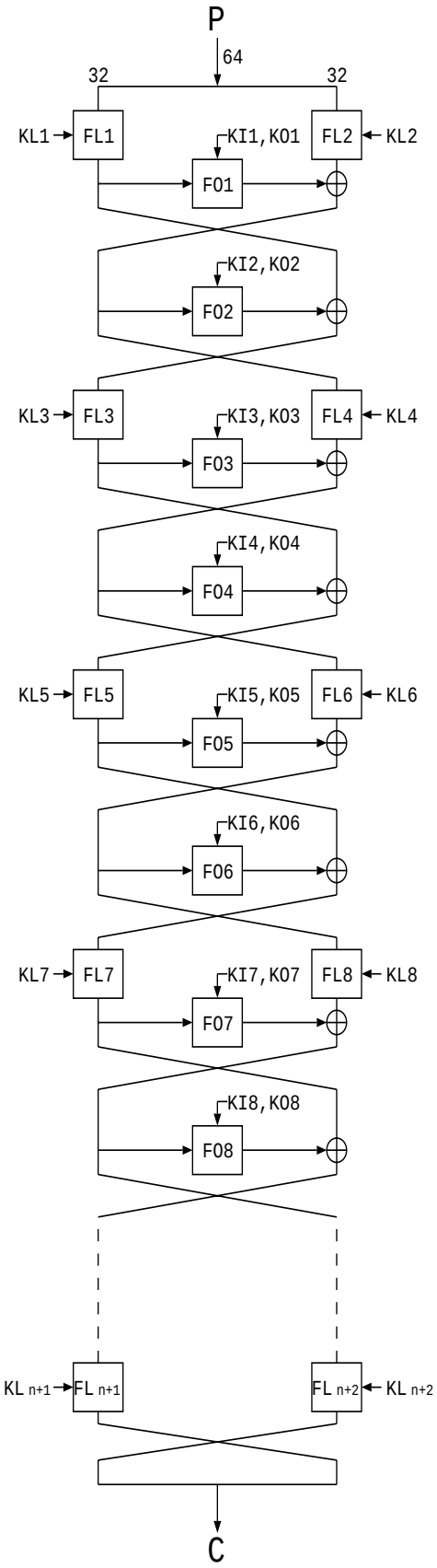

Figure 1: MISTY1

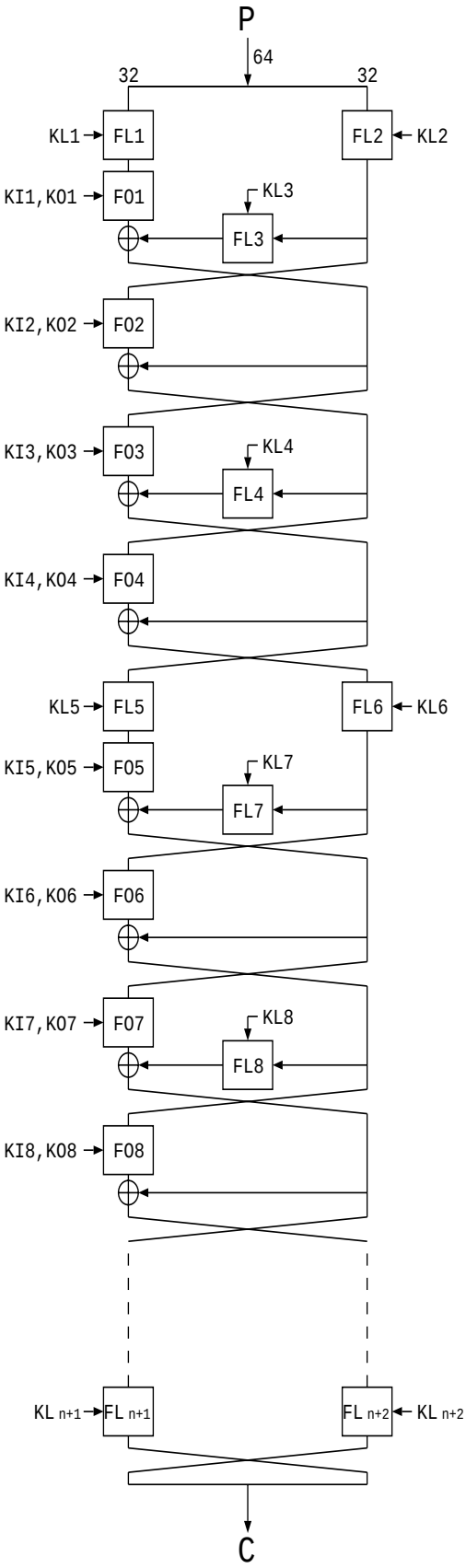

Figure 2: MISTY2

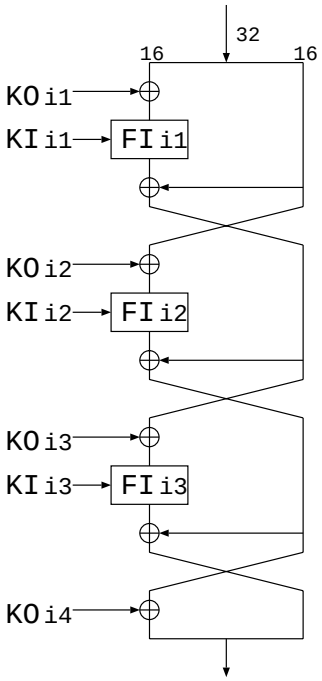

Figure 3: FOi

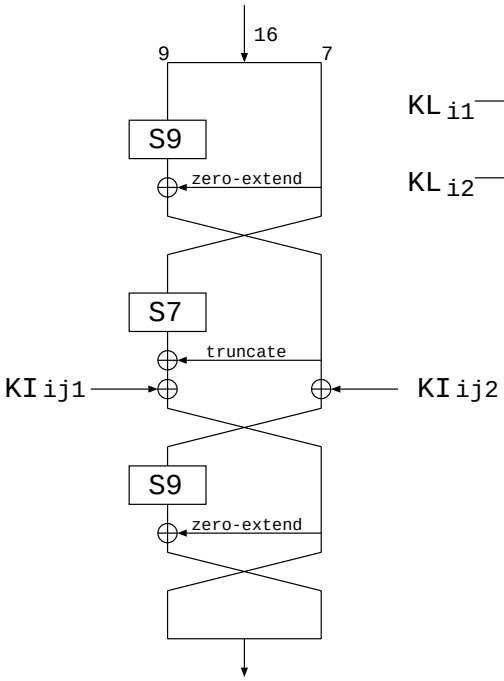

Figure 4: FIij

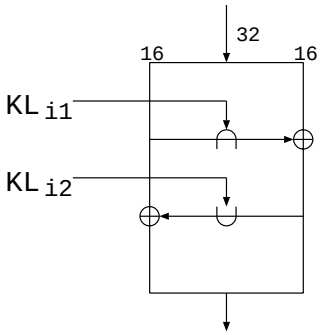

Figure 5: FLi

```
 27, 50, 51, 90, 59, 16, 23, 84, 91, 26,114,115,107, 44,102, 73, 31, 36, 19,108, 55, 46, 63, 74, 93,
 15, 64, 86, 37, 81, 28,  4, 11, 70, 32, 13,123, 53, 68, 66, 43, 30, 65, 20, 75,121, 21,111, 14, 85,
  9, 54,116, 12,103, 83, 40, 10,126, 56,  2,  7, 96, 41, 25, 18,101, 47, 48, 57,  8,104, 95,120, 42,
 76,100, 69,117, 61, 89, 72,  3, 87,124, 79, 98, 60, 29, 33, 94, 39,106,112, 77, 58,  1,109,110, 99,
 24,119, 35,  5, 38,118,  0, 49, 45,122,127, 97, 80, 34, 17,  6, 71, 22, 82, 78,113, 62,105, 67, 52,
 92, 88,125
```

Table 1: The table of S_7.

```
451,203,339,415,483,233,251, 53,385,185,279,491,307,  9, 45,211,199,330, 55,126,235,356,403,472,163,
286, 85, 44, 29,418,355,280,331,338,466, 15, 43, 48,314,229,273,312,398, 99,227,200,500, 27,  1,157,
248,416,365,499, 28,326,125,209,130,490,387,301,244,414,467,221,482,296,480,236, 89,145, 17,303, 38,
220,176,396,271,503,231,364,182,249,216,337,257,332,259,184,340,299,430, 23,113, 12, 71, 88,127,420,
308,297,132,349,413,434,419, 72,124, 81,458, 35,317,423,357, 59, 66,218,402,206,193,107,159,497,300,
388,250,406,481,361,381, 49,384,266,148,474,390,318,284, 96,373,463,103,281,101,104,153,336,  8,  7,
380,183, 36, 25,222,295,219,228,425, 82,265,144,412,449, 40,435,309,362,374,223,485,392,197,366,478,
433,195,479, 54,238,494,240,147, 73,154,438,105,129,293, 11, 94,180,329,455,372, 62,315,439,142,454,
174, 16,149,495, 78,242,509,133,253,246,160,367,131,138,342,155,316,263,359,152,464,489,  3,510,189,
290,137,210,399, 18, 51,106,322,237,368,283,226,335,344,305,327, 93,275,461,121,353,421,377,158,436,
204, 34,306, 26,232,  4,391,493,407, 57,447,471, 39,395,198,156,208,334,108, 52,498,110,202, 37,186,
401,254, 19,262, 47,429,370,475,192,267,470,245,492,269,118,276,427,117,268,484,345, 84,287, 75,196,
446,247, 41,164, 14,496,119, 77,378,134,139,179,369,191,270,260,151,347,352,360,215,187,102,462,252,
146,453,111, 22, 74,161,313,175,241,400, 10,426,323,379, 86,397,358,212,507,333,404,410,135,504,291,
167,440,321, 60,505,320, 42,341,282,417,408,213,294,431, 97,302,343,476,114,394,170,150,277,239, 69,
123,141,325, 83, 95,376,178, 46, 32,469, 63,457,487,428, 68, 56, 20,177,363,171,181, 90,386,456,468,
 24,375,100,207,109,256,409,304,346,  5,288,443,445,224, 79,214,319,452,298, 21,  6,255,411,166, 67,
136, 80,351,488,289,115,382,188,194,201,371,393,501,116,460,486,424,405, 31, 65, 13,442, 50, 61,465,
128,168, 87,441,354,328,217,261, 98,122, 33,511,274,264,448,169,285,432,422,205,243, 92,258, 91,473,
324,502,173,165, 58,459,310,383, 70,225, 30,477,230,311,506,389,140,143, 64,437,190,120,  0,172,272,
350,292,  2,444,162,234,112,508,278,348, 76,450,
```

Table 2: The table of S_9.

Sample Programs of MISTY1 in C Language

Version 1.00 July 22 1996
Mitsubishi Electric Corporation

This document shows two sample programs of MISTY1 with eight rounds. The first one is a main program, which calls the second program. The second one is a core logic of MISTY1 algorithm; It encrypts and decrypts data stream with arbitrary number of blocks in ECB mode using given 128-bit key. Note: due to space constraint, the second program is written in twocolumn format.

```
/*************************************************
 *                                               *
 *    A Sample Main C Program of MISTY1 Algorithm   *
 *                                               *
 *    Language  : Highly Portable C Language        *
 *    Coding by : Mitsuru Matsui / 22 July 1996     *
 *    Copyright : Mitsubishi Electric Corporation   *
 *                                               *
 *************************************************/

typedef unsigned char  uchar;
typedef unsigned short ushort;

extern ushort EXTKEY[4][8];

main()
{
        static uchar text[16] = {0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef,
                                 0xfe,0xdc,0xba,0x98,0x76,0x54,0x32,0x10};
        static uchar key[16]  = {0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,
                                 0x88,0x99,0xaa,0xbb,0xcc,0xdd,0xee,0xff};
        int  i;

        printf( "Secret Key   " );
        for( i=0; i<16; i++ ) printf( "%02x ", key[i] );
        printf( "\n" );

        printf( "Plaintext    " );
        for( i=0; i<16; i++ ) printf( "%02x ", text[i] );
        printf( "\n" );

        misty1( text, key, 2, 0 );

        printf( "Extended Key " );
        for( i=0; i<8; i++ )  printf( "%02x %02x ", (uchar)(EXTKEY[1][i]>>8), (uchar)(EXTKEY[1][i]&0xff) );
        printf( "\n" );

        printf( "Ciphertext   " );
        for( i=0; i<16; i++ ) printf( "%02x ", text[i] );
        printf( "\n" );

        misty1( text, key, 2, 1 );

        printf( "Plaintext    " );
        for( i=0; i<16; i++ ) printf( "%02x ", text[i] );
        printf( "\n" );
}
```

```
/*************************************************
 *                                               *
 *    MISTY1 Block Cipher Algorithm (8-round/ECB)   *
 *                                               *
 *    Language  : Highly Portable C Language        *
 *    Coding by : Mitsuru Matsui / 22 July 1996     *
 *    Copyright : Mitsubishi Electric Coporation    *
 *                                               *
 *************************************************/

typedef unsigned short ushort;
typedef unsigned char  uchar;

ushort EXTKEY[4][8];

static uchar S7[128] = {
 27, 50, 51, 90, 59, 16, 23, 84, 91, 26,114,115,107, 44,102, 73,
 31, 36, 19,108, 55, 46, 63, 74, 93, 15, 64, 86, 37, 81, 28,  4,
 11, 70, 32, 13,123, 53, 68, 66, 43, 30, 65, 20, 75,121, 21,111,
 14, 85,  9, 54,116, 12,103, 83, 40, 10,126, 56,  2,  7, 96, 41,
 25, 18,101, 47, 48, 57,  8,104, 95,120, 42, 76,100, 69,117, 61,
 89, 72,  3, 87,124, 79, 98, 60, 29, 33, 94, 39,106,112, 77, 58,
  1,109,110, 99, 24,119, 35,  5, 38,118,  0, 49, 45,122,127, 97,
 80, 34, 17,  6, 71, 22, 82, 78,113, 62,105, 67, 52, 92, 88,125 };

static ushort S9[512] = {
451,203,339,415,483,233,251, 53,385,185,279,491,307,  9, 45,211,
199,330, 55,126,235,356,403,472,163,286, 85, 44, 29,418,355,280,
331,338,466, 15, 43, 48,314,229,273,312,398, 99,227,200,500, 27,
  1,157,248,416,365,499, 28,326,125,209,130,490,387,301,244,414,
467,221,482,296,480,236, 89,145, 17,303, 38,220,176,396,271,503,
231,364,182,249,216,337,257,332,259,184,340,299,430, 23,113, 12,
 71, 88,127,420,308,297,132,349,413,434,419, 72,124, 81,458, 35,
317,423,357, 59, 66,218,402,206,193,107,159,497,300,388,250,406,
481,361,381, 49,384,266,148,474,390,318,284, 96,373,463,103,281,
101,104,153,336,  8,  7,380,183, 36, 25,222,295,219,228,425, 82,
265,144,412,449, 40,435,309,362,374,223,485,392,197,366,478,433,
195,479, 54,238,494,240,147, 73,154,438,105,129,293, 11, 94,180,
329,455,372, 62,315,439,142,454,174, 16,149,495, 78,242,509,133,
253,246,160,367,131,138,342,155,316,263,359,152,464,489,  3,510,
189,290,137,210,399, 18, 51,106,322,237,368,283,226,335,344,305,
327, 93,275,461,121,353,421,377,158,436,204, 34,306, 26,232,  4,
391,493,407, 57,447,471, 39,395,198,156,208,334,108, 52,498,110,
202, 37,186,401,254, 19,262, 47,429,370,475,192,267,470,245,492,
269,118,276,427,117,268,484,345, 84,287, 75,196,446,247, 41,164,
 14,496,119, 77,378,134,139,179,369,191,270,260,151,347,352,360,
215,187,102,462,252,146,453,111, 22, 74,161,313,175,241,400, 10,
426,323,379, 86,397,358,212,507,333,404,410,135,504,291,167,440,
321, 60,505,320, 42,341,282,417,408,213,294,431, 97,302,343,476,
114,394,170,150,277,239, 69,123,141,325, 83, 95,376,178, 46, 32,
469, 63,457,487,428, 68, 56, 20,177,363,171,181, 90,386,456,468,
 24,375,100,207,109,256,409,304,346,  5,288,443,445,224, 79,214,
319,452,298, 21,  6,255,411,166, 67,136, 80,351,488,289,115,382,
188,194,201,371,393,501,116,460,486,424,405, 31, 65, 13,442, 50,
 61,465,128,168, 87,441,354,328,217,261, 98,122, 33,511,274,264,
448,169,285,432,422,205,243, 92,258, 91,473,324,502,173,165, 58,
459,310,383, 70,225, 30,477,230,311,506,389,140,143, 64,437,190,
120,  0,172,272,350,292,  2,444,162,234,112,508,278,348, 76,450 };
```

```
#define FL_enc( k ){\
r1 ^= r0 & EXTKEY[0][k];\
r3 ^= r2 & EXTKEY[1][(k+2)&7];\
r0 ^= r1 | EXTKEY[1][(k+6)&7];\
r2 ^= r3 | EXTKEY[0][(k+4)&7];\
}

#define FL_dec( k ){\
r0 ^= r1 | EXTKEY[0][(k+4)&7];\
r2 ^= r3 | EXTKEY[1][(k+6)&7];\
r1 ^= r0 & EXTKEY[1][(k+2)&7];\
r3 ^= r2 & EXTKEY[0][k];\
}

#define FI_key( k ){\
r0  = EXTKEY[0][k] >> 7;\
r1  = EXTKEY[0][k] & 0x7f;\
r0  = S9[r0] ^ r1;\
r1  = S7[r1] ^ ( r0 & 0x7f );\
r1 ^= EXTKEY[0][(k+1)&7] >> 9;\
r0 ^= EXTKEY[0][(k+1)&7] & 0x1ff;\
r0  = S9[r0] ^ r1;\
EXTKEY[3][k] = r1;\
EXTKEY[2][k] = r0;\
EXTKEY[1][k] = r1 << 9 ^ r0;\
}

#define FI_txt( a0, a1, k ){\
a1  = a0 >> 7;\
a0 &= 0x7f;\
a1  = S9[a1] ^ a0;\
a0  = S7[a0] ^ a1;\
a1 ^= EXTKEY[2][k];\
a0 ^= EXTKEY[3][k];\
a0 &= 0x7f;\
a1  = S9[a1] ^ a0;\
a1 ^= a0 << 9;\
}

#define FO_txt( a0, a1, a2, a3, k ){\
t0  = a0 ^ EXTKEY[0][k];\
FI_txt( t0, t1, (k+5)&7 );\
t1 ^= a1;\
t2  = a1 ^ EXTKEY[0][(k+2)&7];\
FI_txt( t2, t0, (k+1)&7 );\
t0 ^= t1;\
t1 ^= EXTKEY[0][(k+7)&7];\
FI_txt( t1, t2, (k+3)&7 );\
t2 ^= t0;\
t0 ^= EXTKEY[0][(k+4)&7];\
a2 ^= t0;\
a3 ^= t2;\
}
```

```
/*****************************************
 *                                       *
 *    Encryption/Decryption Subroutine Body    *
 *                                       *
 *       misty1( text, key, block, mode )      *
 *                                       *
 *    text  : plain/ciphertext address  I/O    *
 *    key   : secret-key address         I     *
 *    block : number of text blocks      I     *
 *    mode  : 0:encryption 1:decryption  I     *
 *                                       *
 *****************************************/

misty1( text, key, block, mode )
uchar *text,*key;
int    block,mode;
{
    register ushort t0, t1, t2;
    register ushort r0, r1, r2, r3;

    /*** Key Scheduling ***/

    EXTKEY[0][0] = (ushort)key[0]<<8  ^ (ushort)key[1];
    EXTKEY[0][1] = (ushort)key[2]<<8  ^ (ushort)key[3];
    EXTKEY[0][2] = (ushort)key[4]<<8  ^ (ushort)key[5];
    EXTKEY[0][3] = (ushort)key[6]<<8  ^ (ushort)key[7];
    EXTKEY[0][4] = (ushort)key[8]<<8  ^ (ushort)key[9];
    EXTKEY[0][5] = (ushort)key[10]<<8 ^ (ushort)key[11];
    EXTKEY[0][6] = (ushort)key[12]<<8 ^ (ushort)key[13];
    EXTKEY[0][7] = (ushort)key[14]<<8 ^ (ushort)key[15];

    FI_key( 0 );
    FI_key( 1 );
    FI_key( 2 );
    FI_key( 3 );
    FI_key( 4 );
    FI_key( 5 );
    FI_key( 6 );
    FI_key( 7 );

    /*** Data Randomizing ***/

    if( !(mode & 1) ){

        /*** Encryption ***/

        while( block-- > 0 ){

            r0 = (ushort)text[0]<<8 ^ (ushort)text[1];
            r1 = (ushort)text[2]<<8 ^ (ushort)text[3];
            r2 = (ushort)text[4]<<8 ^ (ushort)text[5];
            r3 = (ushort)text[6]<<8 ^ (ushort)text[7];

            FL_enc( 0 );
            FO_txt( r0, r1, r2, r3, 0 );
            FO_txt( r2, r3, r0, r1, 1 );
            FL_enc( 1 );
            FO_txt( r0, r1, r2, r3, 2 );
            FO_txt( r2, r3, r0, r1, 3 );
            FL_enc( 2 );
            FO_txt( r0, r1, r2, r3, 4 );
            FO_txt( r2, r3, r0, r1, 5 );
            FL_enc( 3 );
            FO_txt( r0, r1, r2, r3, 6 );
            FO_txt( r2, r3, r0, r1, 7 );
            FL_enc( 4 );

            text[0] = r2 >> 8;
            text[1] = r2 & 0xff;
            text[2] = r3 >> 8;
            text[3] = r3 & 0xff;
            text[4] = r0 >> 8;
            text[5] = r0 & 0xff;
            text[6] = r1 >> 8;
            text[7] = r1 & 0xff;

            text += 8;
        }
    }
    else{

        /*** Decryption ***/

        while( block-- > 0 ){

            r0 = (ushort)text[0]<<8 ^ (ushort)text[1];
            r1 = (ushort)text[2]<<8 ^ (ushort)text[3];
            r2 = (ushort)text[4]<<8 ^ (ushort)text[5];
            r3 = (ushort)text[6]<<8 ^ (ushort)text[7];

            FL_dec( 4 );
            FO_txt( r0, r1, r2, r3, 7 );
            FO_txt( r2, r3, r0, r1, 6 );
            FL_dec( 3 );
            FO_txt( r0, r1, r2, r3, 5 );
            FO_txt( r2, r3, r0, r1, 4 );
            FL_dec( 2 );
            FO_txt( r0, r1, r2, r3, 3 );
            FO_txt( r2, r3, r0, r1, 2 );
            FL_dec( 1 );
            FO_txt( r0, r1, r2, r3, 1 );
            FO_txt( r2, r3, r0, r1, 0 );
            FL_dec( 0 );

            text[0] = r2 >> 8;
            text[1] = r2 & 0xff;
            text[2] = r3 >> 8;
            text[3] = r3 & 0xff;
            text[4] = r0 >> 8;
            text[5] = r0 & 0xff;
            text[6] = r1 >> 8;
            text[7] = r1 & 0xff;

            text += 8;
        }
    }
}
```