FEAL-NX のソフトウェア実装

大塚 浩昭　　植田 広樹

NTT 情報流通プラットフォーム研究所
〒239-0847 神奈川県横須賀市光の丘 1-1 NTT R&D センタ Y609A

{ohtsuka,ueda}@isl.ntt.co.jp

あらまし:　Low-end な機器において，例えばスマートカード上での認証手段としては，共通鍵暗号を用いた方式が，速度やメモリの点で有効である．本稿では，FEAL-32X の実装を Z80 上で行なった結果について報告する．FEAL-32X 暗号化処理（データランダム部）は，8 バイトの RAM と 216 バイトの ROM で実装でき，その処理時間は，6185 state/block である．また，本実装結果が，最適実装に近いことを考察する．

キーワード:　共通鍵暗号，ソフトウェア実装，スマートカード，FEAL，Z80

Software implementation of FEAL-NX

Hiroaki Ootsuka　　Hiroki Ueda

NTT Information Sharing Platform Laboratories
1-1, Hikarinooka, Yokosuka-shi, Kanagawa, 239-0847, Japan

{ohtsuka,ueda}@isl.ntt.co.jp

Abstract:　With respect to speed and memory size , mechanisms using symmetric encipherment algorithms are suitable to the implement with low-end devices. One of examples is entity authentication using smart cards. This paper describes and evaluates the performance of software implementation of FEAL-32X on the Z80. It uses 8 bytes of RAM and 216 bytes of ROM, and takes 6185 states/block on the Z80 in the data randomizing part. Furthermore, we discuss how this implementation is optimized by comparing with ideal models.

Keywords:　symmetric encipherment , software implementation , smart cards , FEAL , Z80

1 はじめに

ネットワーク技術や情報関連機器の発展により，社会における情報化が進展している．それに伴い，様々な情報機器がネットワークに接続され，多様なサービスを提供することが可能となる．これによって，従来行なわれてきた社会活動は，ネットワーク上でも行なうことが可能となる．このため，様々な環境で，暗号技術が必要となる．それは秘匿目的に限らない．たとえば本人性の確認（認証）といった，従来では物理的に複製が困難である媒体を対面提示することによって行われてきた行為と同等の行為を，ネットワーク上で表現できる環境が必要となる．

この問題を解決するためには，暗号技術による安全な認証技術と，小型で安価な簡易端末（たとえば，スマートカード）の利用が考えられる．これらの端末に，認証情報を格納させ，利用者は通常この端末を携帯し，必要な場面において端末をネットワーク接続させ，サービスを受けるのである．サービスの例としては，電子マネーや，行政サービスの電子化といったものが考えられる．

端末には，適用モデルによって様々な条件が課せられる．すなわち，導入コストや，端末の使われ方を考慮する（非接触ICカードの利用）といった条件を視野に入れつつ，端末装置を選択することが求められる．これにより，端末が持つ処理性能や，利用可能メモリは大きく影響を受けるが，たとえ低機能（以下，low-end 端末）であったとしても，安全な認証技術を提供することは必須である．

low-end 端末は，処理装置の速度が遅く（8-bit CPUで，クロックが1〜8 MHz 程度），利用出来るメモリサイズが小さい（高々数百バイト）といった特徴を持つ．このような制限下では，公開鍵暗号を用いた認証方法は，処理時間と必要メモリサイズの点から実用的ではない．したがって，共通鍵暗号を用いた認証方法が適切である．暗号技術を用いた認証方法に，ISO 9798-2 [1] で規格化されている two-pass authentication がある．

そこで，low-end 端末でも利用可能な，共通鍵暗号を用いた認証方法の実現について考察することが求められる．共通鍵暗号に関する条件としては，

(1) 安全であること

また主に 8-bit CPU 上で実装したときに，

(2) 処理速度が高速であること

(3) 必要なメモリサイズが少ないこと

が挙げられる．本稿では，安全性について検討されている FEAL-32X を 8-bit CPU で実装し，実装の妥当性の検討と他の共通鍵暗号との比較とを行い，low-end 端末における認証機能の実現方法として FEAL-32X が適当であることを示す．

2 FEAL-NX

2.1 特徴と仕様

FEAL-NX [2] は，NTT から提案されているブロック長 64-bit 鍵長 128-bit の共通鍵暗号である[1]．FEAL-NX では，内部で行う演算の大部分が 8-bit を単位として行われている．このため 8-bitCPU 上でのソフトウェア実装が行いやすいという特徴がある．また FEAL-NX では，S-box を加算演算と循環シフト演算との組み合わせで実現している．このため，DES などに代表されるような S-box をテーブル参照で実現している暗号と比較して，テーブルを必要としない分メモリサイズが少なくて済むという特徴がある．

FEAL-NX の N は，F 関数の繰り返し回数（以下，段数）を意味する．例えば，32 段である FEAL 暗号は，FEAL-32X と表記される．なお FEAL には，FEAL-N と FEAL-NX との 2 種類が存在し，前者は鍵長が 64-bit であり，後者は鍵長が 128-bit である．本稿では，次世代仕様である 128-bit 版の FEAL-NX を対象とする．

2.2 安全性

FEAL-NX の安全性について検討する．差分解読法は，32 段以上の FEAL-NX には適用不能であることが示されている[3]．線形解読法は，26 段以上の FEAL-NX には適用不能であることが示されている[4]．この他にも FEAL-NX の安全性については多くの研究結果が報告されているが，32 段以上の FEAL-NX に対する有効な攻撃法は，現在のところ知られていない[5]．また，鍵の全数探索に対する安全性から，128-bit の鍵長が必要と一般に考えられている．したがって，現在安全な暗号アルゴリズムとして，FEAL-32X は選択肢の一つと判断できる．

3 Z80 上での実装

low-end 端末として，8-bit CPU である Z80 が搭載されたスマートカードを想定する．Z80 を選定した理由は，low-end のスマートカードに搭載される 8-bit CPU としてディファクトスタンダードであり，また昨今カスタム LSI のマクロセルとして標準装備されることが多いからである．

本稿では，1 ブロック当たりの FEAL-32X の暗

---

[1] FEAL-NX の詳しい仕様については，付録２ FEAL-NX 仕様書を，FEAL-NX の C 言語によるサンプルコードについては，付録３ feal.c をそれぞれ参照．

号処理を実装し，処理速度とメモリサイズとに着目して最適化した．なお今回は，拡大鍵を用いて平文を暗号化する機能（以下，データランダム部）のみを暗号処理として評価し，鍵から拡大鍵を生成する機能（以下，鍵拡大部）は評価対象外とした．これは low-end のスマートカードでは，鍵の更新が頻繁に行われることはないと想定したためである．

3.1 前提条件

プログラムを作成するにあたって，以下のプログラム仕様と制限事項とを前提条件とした．また以後に用いる用語は，Z80 アーキテクチャに基く．

3.1.1 プログラム仕様

・サブルーチンコール
　メインルーチンから CALL 命令で呼び出される
・レジスタによるポインタ受け渡し
　平文先頭アドレス，暗号文先頭アドレス，拡大鍵先頭アドレスをレジスタに代入してサブルーチンコールする

3.1.2 制限事項

・プログラムサイズは数百バイト
　スマートカードのメモリサイズを考慮する
・テーブル参照 無し
　テーブル情報の保持により必要メモリサイズが大きくなるのを防ぐ
・プログラムの自己書き換え 無し
　スマートカード上の ROM に存在するため変更不可
・SP 書き換え 無し
　スマートカード上でのアプリケーションのため，SP の変更は行わない
・データランダム部のみを実装し，鍵拡大部を含まない
　鍵の更新(すなわち鍵拡大部)頻度は，暗号文作成(すなわちデータランダム部)頻度と比較して非常に少ない
・メモリアクセスの際のアドレス指定はレジスタ指定のみ
　レジスタによるポインタ受け渡しのため，絶対アドレスによるメモリアクセスは行わない

3.2 最適化の考え方

プログラムを作成するにあたって，以下の考え方に基づき処理速度の最適化を行った．

・レジスタ内での処理を最優先
　メモリアクセスを最少化
　スタックアクセスを最少化
・IX,IY 両レジスタの使用制限
　state 数が大きくかつ 8-bit 毎に分割利用できない IX,IY へのアクセスを最少化
・表裏レジスタの役割分担
　EXX，EX 命令の利用回数を最少化
・ポインタアドレスの制限
　実用上支障が無い程度に，平文，暗号文，拡大鍵のアドレス位置を制限

3.3 実装結果

実際に Z80 アセンブラにて実装した FEAL-32X の 1 ブロック当たりのデータランダム部のプログラム総 state 数，そのプログラムに要した ROM と RAM，および総 state 数から算出した 5MHz CPU 上での処理速度を，表 1 に示す[2]．

表 1　データランダム部 実装結果

総 state 数 (state/block)	ROM (byte)	RAM (byte)	処理速度 (kbit/s)
6185	216	8	51.7 [3]

4　考察

Z80 上での実装に必要とされる総 state 数を算出し，これと実装結果とを比較することで，処理速度の点における実装結果の妥当性を判断する．また，他の共通鍵暗号方式との比較も行う．

4.1 予測値の算出

最初に理想的な 8-bit CPU 上での総 state 数(以下，概算値)を求め，次に Z80 CPU での制限により増加修正された総 state 数(以下，予測値)を求める．

4.1.1 理想的な 8-bit CPU 上での概算値

理想的な 8-bit CPU として以下の仕様を想定し，この理想的な 8-bit CPU 上での概算値を求める．なお以降の state 数の計算には，最小インストラクションが 4 state である通常の Z80 アーキテクチャで規定されている値を用いた．

・レジスタ数が無限
　8-bit レジスタを無限個持つ
・命令形は直交
　8-bit 算術演算命令は，すべてのレジスタ間で直交している

---

[2] 参考として，データランダム部（最適化なし）と，鍵拡大部（最適化なし，あり）の値を付録 1 に示す．

[3] 1.237 ms/block

・算術演算命令は 3 オペランド
　8-bit 算術演算命令のオペランドはソース 2 つディスティネーション 1 つの計 3 オペランドであり，すべてレジスタ指定
・メモリアクセス命令は LD 命令のみ
　メモリアクセスには 8-bit LD 命令を利用

4.1.1.1　F 関数関連

F 関数関連[4]では，以下の処理を行う必要がある．
・拡大鍵(2 バイト)をメモリからレジスタにコピー
・拡大鍵ポインタのインクリメント
・F 関数内での 8-bit 排他的論理和 4 回
・S-box　4 個所
・F 関数の出力結果と L_i (4 バイト)との排他的論理和
これらの処理の実現に必要な命令とその state 数を 表 2 に示す．

表 2　F 関数関連に必要な state 数

命令		回数	@state	合計
LD		2	7	14
INC		2	6	12
XOR		8	4	32
S-box	ADD	4	4	16
	INC	2	4	8
	RLCA	8	4	32
				114

4.1.1.2　前処理・後処理

データランダム部の先頭部分での処理(以下，前処理)と最終部分での処理(以下，後処理)では，以下の処理を行う必要がある．
・平文(8 バイト)をメモリからレジスタにコピー
・拡大鍵 K_N～K_{N+3} (計 8 バイト)をメモリからレジスタにコピー
・平文と拡大鍵 K_N～K_{N+3} 鍵との排他的論理和
・L_0(4 バイト)と R_0(4 バイト)との排他的論理和
・R_N(4 バイト)と L_N(4 バイト)との排他的論理和
・拡大鍵 K_{N+4}～K_{N+7}(計 8 バイト)をメモリからレジスタにコピー
・途中結果(8 バイト)と拡大鍵 K_{N+4}～K_{N+7} との排他的論理和
・暗号結果(8 バイト)をレジスタからメモリにコピー
・平文ポインタ，拡大鍵ポインタ，および暗号文ポインタのインクリメント
これらの処理の実現に必要な命令とその state 数を 表 3 に示す．

[4] F 関数外部にある，F 関数出力結果と L_i との排他的論理和処理や，L_i と R_i との入れ替え処理を含める．

表 3　前処理・後処理に必要な state 数

命令	回数	@state	合計
LD	32	7	224
INC	32	6	192
XOR	24	4	96
			512

さらに F 関数を 32 回繰り返すにあたって，DJNZ 命令を用いるとする．またデータランダム部の最後にはメインルーチンへ戻るための RET 命令が必要である．したがって，理想的な 8-bit CPU 上では以下の値が総 state 数であると予想される．

理想的な 8-bit CPU での総 state 数
= (前処理+後処理) + F 関数 * 32 段
　+ DJNZ ジャンプあり * 31 回
　+ DJNZ ジャンプなし * 1 回
　+ RET
= 512 + (114*32) + 13*31 + 8*1 + 10
概算値 = 4581 state

4.1.2　Z80 CPU による修正

Z80 に固有の制限によって，増加せざるを得ない state 数について考察する．

4.1.2.1　レジスタ数の制限

Z80 は，表レジスタである A,BC,DE,HL と裏レジスタである A’,B’C’,D’E’,H’L’との計 14 バイトを 8-bit レジスタとして持つ．なお IX,IY 両レジスタは 8-bit に分割して使用できないため，考慮外とした．F 関数では，データ入力に 32-bit(4 バイト)，拡大鍵入力に 16-bit(2 バイト)，データ出力に 32-bit(4 バイト)の計 10 バイトを要する．よって F 関数を表レジスタだけで計算することは不可能である．よって F 関数内で裏レジスタを利用する．これにより，F 関数内で最低 1 セット(2 回)は表裏レジスタを切りかえる必要が生じる．これらの処理の実現に必要な命令とその state 数を 表 4 に示す．

表 4　レジスタ数の制限による増加

命令	回数	@state	段数	合計
EXX	2	4	32	256

4.1.2.2　命令系の非直交性

Z80 では，算術演算命令の結果はアキュームレータにしか格納されない．よって ADD 命令を用

いる場合は必ずアキュームレータを経由して結果を格納する必要がある．F 関数関連には，ADD 命令 4 回が含まれている．よってこの場合，LD 命令や EX 命令などを用いて演算結果を他のレジスタに退避させておく必要が生じる．また前処理，後処理それぞれにおいて，拡大鍵との XOR を行う際に，連続して 8 回 XOR 命令を行う必要がある．この場合も LD 命令などを用いて演算結果をアキュームレータ以外に退避させる必要が生じる．さらに F 関数中には 1 つの値が 2 箇所で利用される分岐点が 4 箇所存在する．よってあらかじめ元の値を別のレジスタにコピーしておく必要がある．これらの処理の実現に必要な命令とその state 数を 表 5 に示す．

表 5　命令の非直交性による増加

命令	回数	@state	回数	合計
LD(F 関数)	8	4	32	1024
LD(前後処理)	8	4	2	64
				1088

これらの制限により，Z80CPU 上では以下の値が総 state 数であると予想される．

Z80 CPU 上での総 state 数
= 理想的な 8-bit CPU での総 state 数
+ レジスタ数の制限による増加
+ 命令の非直交性による増加
= 4581 + 256 + 1088
予測値 = 5925 state

4.2 実装結果の妥当性

前章で求められた Z80 CPU 上での総 state 数 5925 state と，実装結果である 6185 state との差は，260 state (全体の約 4%)である．この差について分析する．

前述した予測値との誤差要因として，以下の増加要因と減少要因とが挙げられる．

●増加要因

(1) F 関数内のレジスタ切替命令
F 関数内のレジスタの割り当てによって，レジスタを切り替える回数がさらに増加する．

(2) レジスタ退避
レジスタに保存しきれない値は，スタックやメモリに退避させる．

●減少要因

(1) 複数命令の単一化
理想的な 8-bit CPU では複数命令となる処理を，Z80 に特有の命令を用いることによって 1 命令で代用できる．

(2) 命令の代用
あらかじめ前提条件にて制限した事項によって，概算値算出で用いた命令の代わりに state 数の少ない別命令を用いることが出来る．

次に，最適化の達成度について検討する．例えば，F 関数内で最少単位にあたる 4 state 命令を 1 つ省略できたとすると，全体では 4 state * 32 段 = 128 state の減少になる．よって 260 state の差は，一段あたりに換算すると，予測値より 2 命令だけ多いと見なせる．レジスタ数の制限などを考慮すれば，十分な最適化が行われていると思われる．

4.3 他の共通鍵暗号

今回実装したプログラムについて，64-bit ブロック共通鍵暗号として著名である TripleDES との比較を行う．5 MHz の 8bit MPU が搭載されている ST16CF54B(SGS-Thomson 製)上における SingleDES の暗号処理時間は，10ms/block である[6]．ST16CF54B の 8bit MPU が Z80 と同じアーキテクチャを採用しかつ TripleDES は SingleDES の 3 倍の処理時間がかかると仮定すると，Z80 上における TripleDES の暗号処理時間は 30ms/block (2.13kbit/s) と推測できる．これより本稿のプログラムは，TripleDES と比較して約 24 倍高速であると推測される．

DES に代表されるような S-box にテーブルを用いている共通鍵暗号は，プログラム中にテーブルを保管しておく必要がある．DES では，S-box として(16,4)行列を 8 種類用いている．1 要素につき 1 バイト メモリを利用すると仮定すると，S-box だけで 16*4*8 = 512 バイトを使用することになる．これより本稿のプログラムは，TripleDES と比較して必要とされる ROM のバイト数が，少なくとも 2 分の 1 以下であると推測される．

5　まとめ

low-end 端末における認証方式として適当である共通鍵暗号 FEAL-32X (データランダム部) について，Z80 上で実装を行った．最適化を行った結果，FEAL-32X (データランダム部) は 8 バイトの RAM，216 バイトの ROM，6185 state/block で実装できた．FEAL-32X が，ほぼ予測値に等しい state 数で実装できることを確認した．

参考文献

[1] “Information technology — Security techniques — Entry authentication Part2 (ISO/IEC 9798-2:1999)”,1999.

[2] 宮口・栗原・太田・森田,”FEAL 暗号の拡張”,NTT R&D 39 No.10 pp.1439-1450 , 1990 年.

[3] Kazumaro Aoki, Kunio Kobayashi, and Shiho Moriai. The best differential characteristic search of FEAL. IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences Japan, Vol. E81-A, No. 1, pp. 98--104, 1998. (Japanese preliminary version was presented at ISEC96-31).

[4] Shiho Moriai, Kazumaro Aoki, and Kazuo Ohta. The best linear expression search of FEAL. IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences (Japan), Vol. E79-A, No. 1, pp. 2--11, 1996 (The extended abstract was presented at CRYPTO'95).

[5] 青木, 太田, 盛合, 共通鍵暗号 FEAL の安全性評価, pp. 734-739, NTT R&D Vol. 48, No. 10, 1999 年 10 月.

[6] Helena Handschuh, Pascal Paillier , “Smart Card Crypto-Coprocessors for Public-Key Cryptography” , RSA LABORATORIES', CryptoBytes , Volume4 , Number1 , Summer 1998 , ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto4n1.pdf .

付録 1

表　データランダム部 実装結果（最適化なし）

ROM (byte)	RAM (byte)	処理速度 (kbit/s) [5MHz 時]
278	14	11.4

表　鍵拡大部 実装結果（最適化なし）

ROM (byte)	RAM (byte)	処理時間 (ms) [5MHz 時]
225	35	7.17

表　鍵拡大部 実装結果（最適化あり）

ROM (byte)	RAM (byte)	処理時間 (ms) [5MHz 時]
542	16	1.869