FEAL-NXのソフトウェア実装

NTT　情報流通プラットフォーム研究所

大塚 浩昭　　植田 広樹

概要

8-bit CPU 上での
FEAL-32Xの
速度面での最適化
について報告

背景

- ネットワーク接続された小型簡易端末が普及
  <e.g.> スマートカード
- ネットワーク上でのなりすまし問題
  ↓
- 小型簡易端末を**認証**することが必要

FEAL-NXのソフトウェア実装により実現

小型簡易端末での認証

- 認証プログラムの動作場所
  - 耐タンパ性を備えたチップ上
- チップ
  - 安価: 数十円～数百円で購入可能
  - CPU 低速、メモリ 極小
    <e.g.> low-endのスマートカード

→ 8-bit CPU が中心

想定する実装環境

Low-end のスマートカード

CPUアーキテクチャ … Z80 (8bit-CPU)

8-bit CPUのディファクトスタンダード

CPUクロック … 1〜8 MHz

ROM … 数Kバイト程度

RAM … 256バイト程度

認証プログラムが利用出来るメモリは、さらに少

認証方式

共通鍵暗号による認証を採用

理由：
- 公開鍵暗号
  - 多くのメモリと演算能力が必要
- ハッシュ関数
  - 内部演算が32-bit単位

暗号アルゴリズム

FEAL-32Xを採用

理由:
- 8-bit CPUでの実装に有利
  - 内部演算 … 8-bit単位
  - S-box用テーブルが不必要 → 少メモリ
- 安全性 … 現状では問題なし
  - 全数探索法 → 鍵長 128-bit
  - 差分解読法 → 32段以上のFEALには適用不可
  - 線形解読法 → 26段以上のFEALには適用不可

実装にあたっての前提

前提：　実際にスマートカード上で
利用できるプログラム構成

- プログラムはROM上に配置
  → プログラム自己書き換え なし
- スマートカード上の別アプリケーションと共存
  → SP書き換え なし

実装にあたっての仕様

プログラム仕様

- CALL文で呼び出される
- レジスタによるポインタ受け渡し

実装条件

- プログラムサイズ … 数百バイト
- テーブル参照　なし
- データランダム部のみ実装

データランダム部

- low-endのスマートカードでは鍵更新頻度 低

↓

- あらかじめ拡大鍵を計算しておき保存

最適化への道

レジスタ内での処理を最優先

スタック退避、メモリ退避 … 原則行わない

IX,IYレジスタの使用制限

8-bitに分割不可なレジスタ … なるべく使用しない

表裏レジスタの役割分担

レジスタ切替回数 … 最少化

ポインタアドレスの制限

実用上支障がない程度にアドレス位置を制限

実装結果

実装 総 state 数 …… 6169 states

	ROM (byte)	RAM (byte)	処理速度・処理時間 [5MHz]
データランダム部	212	6	51.8 kbit/s (1.2 ms/block)

- ROM …プログラム規模のバイト数
- RAM …動的に使われるメモリのバイト数
  スタックを含み、平文、暗号文、拡大鍵を除く

考察

本当に最適化されたのか？
(これ以上最適化の余地は無いのか？)

↓

予測値を算出

理想的な 8-bit CPU での 総 state 数を算出

Z80上における修正を加える

↓

予測値と実装結果とを比較考察

理想的な 8-bit CPU とは？

レジスタ数 … 無限

命令形 … 直交

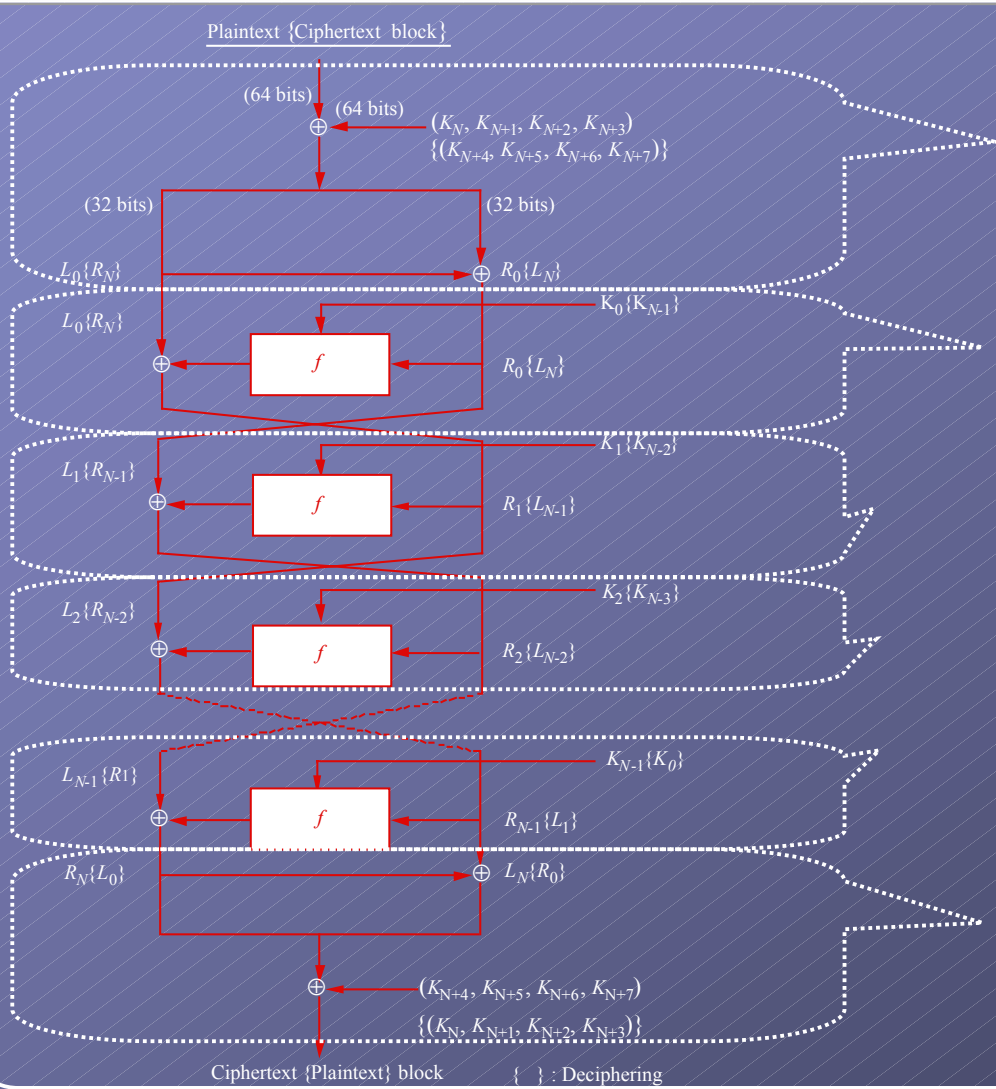
算術演算 … 3オペランド:レジスタ指定

ソース 2

ディスティネーション 1

メモリアクセス … 8-bit LD命令のみ

データランダム部 構造



前処理

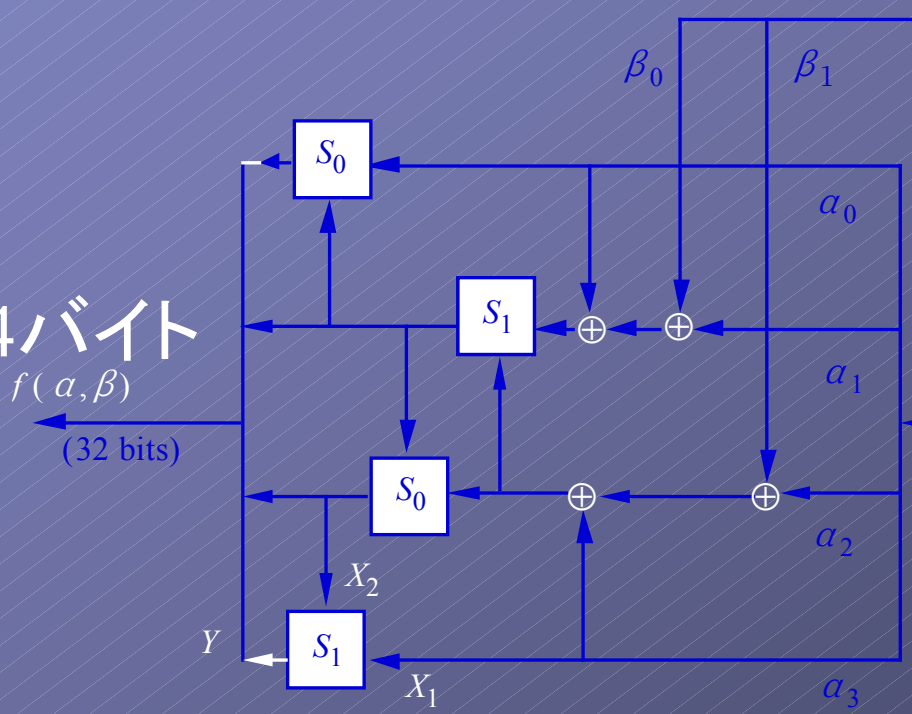
Feistel構造

×

32段

後処理

F関数 構造



拡大鍵 2バイト

出力値 4バイト

$f(\alpha, \beta)$

(32 bits)

入力値 4バイト

α

(32 bits)

α_i, β_i : 8 bits

$Y = S_0(X_1, X_2) = \mathrm{Rot2}((X_1 + X_2) \bmod 256)$

$Y = S_1(X_1, X_2) = \mathrm{Rot2}((X_1 + X_2 + 1) \bmod 256)$

Y : output (8 bits), X_1/ X_2 : inputs (8 bits),

Rot2 (Y) : a 2-bit left rotation on 8-bit data Y

Feistel構造 1段当たり

動作内容		回数	@state	合計
拡大鍵コピー		2	7	14
拡大鍵ポインタ+1		2	6	12
排他的論理和		8	4	32
S-box	加算	4	4	16
	インクリメント	2	4	8
	左ローテート	8	4	32
				114

Feistel構造 1段あたりのstate数 = 114 states

Feistel構造 32段繰り返し

動作内容	回数	@state	合計
Feistel構造 1段	32	114	3468
条件分岐(分岐あり)	31	13	403
条件分岐(分岐なし)	1	8	8
			4059

Feistel構造 32段繰り返し の state 数
= 4059 states

前処理・後処理

動作内容	回数	@state	合計
平文コピー	8	7	56
拡大鍵コピー	16	7	112
暗号文コピー	8	7	56
拡大鍵ポインタ+1	32	6	192
排他的論理和	24	4	96
			512

前処理・後処理での state 数

= 512 states

理想的な 8-bit CPU 上での概算値

理想的な 8-bit CPU上での総 state 数

= Feistel構造 32段繰り返し

+前後処理 +メインルーチンリターン

= 4059 + 512 + 10

= 4581 states

Z80 CPUによる修正

レジスタ数が有限
↓

命令	回数	@state	段数	合計
EXX	2	4	32	256

演算結果は
Aレジのみに格納
↓

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

Z80 CPU上での予測値

＝理想的な 8-bit CPU での概算値
＋ 上記修正項目による増加
＝ 4581 + 256 + 1088

＝ 5925 states

実装結果との比較

	総 state 数	F関数 [総ステート数 ÷ 32]
実装結果	6169	192.8
予測値	5925	185.2
誤差	244	7.6

誤差は、全体の約 4%
→ 十分な最適化が行われていると判断

他の共通鍵暗号との比較

T-DESと比較

処理速度

本稿: 1.2 ms/block

T-DES: 30 ms/block [5MHz,ST16CF54B(8bitCPU)]

出典: RSA LABORATORIES,"CryptoByte"

ROM

本稿: 216バイト

T-DES: 512バイト + α

(S-boxテーブルのみで512バイト)

まとめ

- FEAL-32X(データランダム部)を、Z80上で実装した
  - RAM　6バイト，　ROM　212バイト
  - 6169 states/block (51.8kbit/s [5MHz])
- 十分最適化されていることを示した
- 他の共通鍵暗号と比較して、少ないROMで実装できることを示した

URL

http://info.isl.ntt.co.jp/

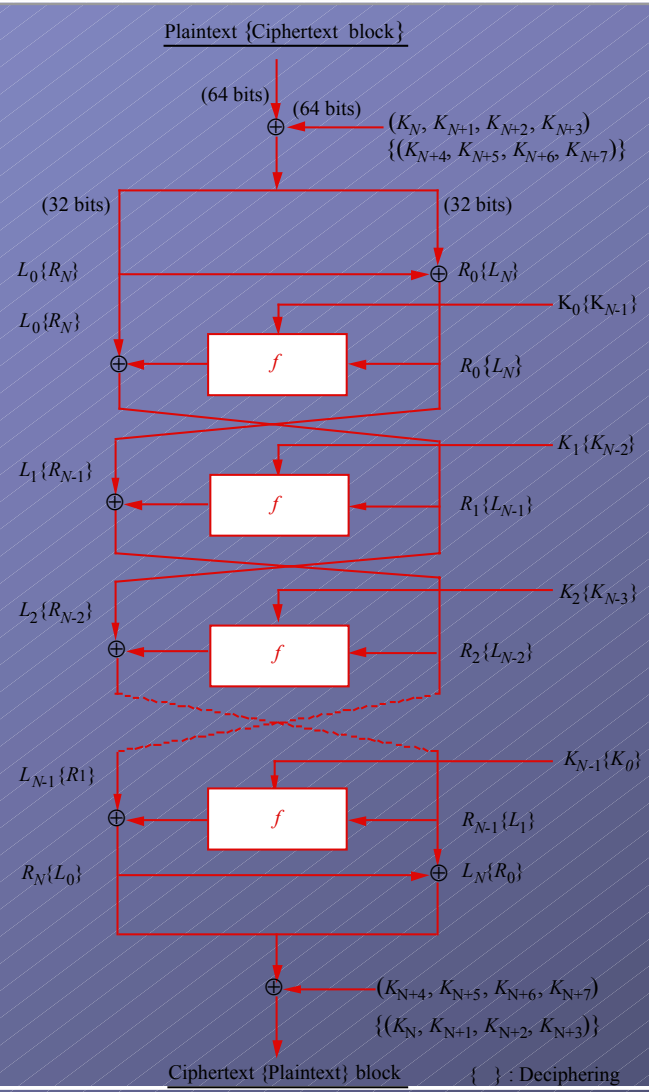
- FEAL-NX
  - 暗号技術概要説明書
  - 仕様書
  - 自己評価書

《参考》 実装結果一覧

		ROM (byte)	RAM (byte)	処理速度／時間 [5MHz]
データランダム部	最適化あり	212	6	51.8 kbit/s
	最適化なし	278	14	11.4 kbit/s
鍵拡大部	最適化あり	465	8	1.67 ms/block
	最適化なし	225	35	7.17 ms/block

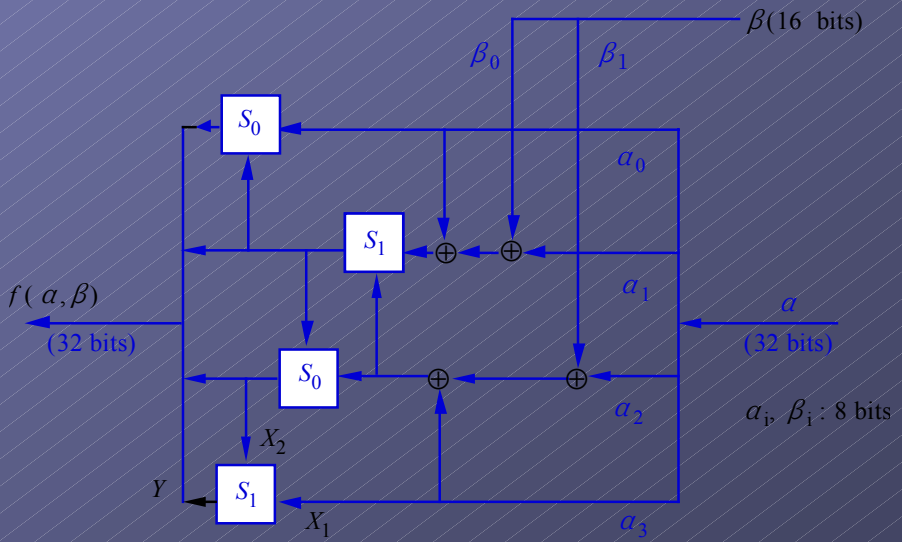
[2000年9月29日現在]

《参考》データランダム部 構造



← データランダム部 構造

↓ F関数 構造



$Y = S_0(X_1, X_2) = \mathrm{Rot2}((X_1+X_2) \bmod 256)$

$Y = S_1(X_1, X_2) = \mathrm{Rot2}((X_1+X_2+1) \bmod 256)$

Y : output (8 bits), X_1/ X_2 : inputs (8 bits),

Rot2 (Y) : a 2-bit left rotation on 8-bit data Y

《参考》増加要因

F関数内のレジスタ切り替え命令

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

レジスタ内容退避

レジスタに保存しきれない値をスタックやメモリに退避

《参考》プログラム呼び出し例

```
LD      HL , plain-text
LD      DE , cipher-text
LD      BC , expanded-key
CALL    encrypt
```