KCipher-2 ソフトウェアの IC カード実装とその評価

宇野 甫† 遠藤 翔† 本間 尚文† 青木 孝文† 仲野 有登‡ 清本 晋作‡ 三宅 優‡

†東北大学大学院情報科学研究科
宮城県仙台市青葉区荒巻字青葉6-6-05
uno@aoki.ecei.tohoku.ac.jp

‡株式会社 KDDI 研究所
埼玉県ふじみ野市大原 2-1-15
kiyomoto@kddilabs.jp

あらまし 本論文では，KCipher-2 ソフトウェアの小型 IC カードへの実装とその評価について述べる. まず, KCipher-2 を 8 ビットの組込みマイコンへ実装するためのコードサイズを削減する手法について述べる. その上で, 乱数マスキングによるサイドチャネル攻撃対策を施した小型実装を提案する. 上記の KCipher-2 ソフトウェアを ATmega163 カードに実装し, その性能評価および安全性評価を行う. 安全性評価では, 未対策実装と対策実装に対してサイドチャネル攻撃標準評価ボード SASEBO-W を用いた電磁波解析実験を行い, 対策の有用性を示す.

Implementation and Evaluation of KCipher-2 Software for Smart Cards

Hajime Uno† Sho Endo† Naofumi Homma Takafumi Aoki†

Yuto Nakano‡ Shinsaku Kiyomoto‡ Yutaka Miyake‡

†Graduate School of Information Science,
Tohoku University.
6-6-05, Aoba, Aramaki, Aoba-ku,
Sendai-shi, Miyagi, JAPAN
uno@aoki.ecei.tohoku.ac.jp

‡KDDI R&D Laboratories, Inc.
2-1-15 Ohara, Fujimino-shi, Saitama,
JAPAN
kiyomoto@kddilabs.jp

Abstract Compact software implementation of KCipher-2 and its evaluation on a smart card are presented in this paper. Our implementation techniques for 8-bit smart card environments reduce the program size of the algorithm and the amount of the memory required for its execution. Furthermore, we propose a compact implementation method with a masking countermeasure against side-channel attacks. The effectiveness of the countermeasure is demonstrated through an experiment of EM analysis. We constructed an experimental platform using an ATmega163 smart card and a Side-channel Attack Standard Evaluation Board SASEBO-W and analyzed performance and security of the above method on the platform.

1 はじめに

KCipher-2[1]は, ワード単位の FSR をベースとした高速な動作を可能としながら, 効率的な動的フィードバック制御機構に起因する高いセキュリティ性を有するストリーム暗号である.

KCipher-2 内部の FSR は，SNOW2.0[2]同様に，32 ビットワード単位の処理を行う．一般的に，FSR のフィードバック関数は原始多項式となるように決定されるが，KCipher-2 は，内部に 2 つの FSR を有し，一方の FSR のレジスタ値を用いて他方の FSR のフィードバック関数の係数を動的に変化させることにより，ワード単位の攪拌に非同期性を付与している．係数を動的選択する制御手法は，縮退を用いたクロック制御機構に比べ，処理速度に与える影響が小さい．

IC カードはキャッシュカードや電子マネーなどに利用される暗号を搭載することが想定された組込み機器である．IC カードに搭載される組込みマイコンは小型化・低消費電力化のために容量の小さいメモリを持つことが多く，コードサイズが比較的大きい KCipher-2 の実装はまだ報告されていない．

本稿では，KCipher-2 ソフトウェアの IC カード向けの小型実装について述べる．まず，プログラムメモリ，データメモリ使用量を削減した 8 ビットマイコンへの実装について述べる．その上で，サイドチャネル攻撃対策を施した小型実装を提案する．本稿では既存の電力・電磁波解析攻撃[3]への対策として，乱数マスキングを適用した手法を示す．上記のソフトウェアを 8 ビットマイコン ATmega163 の IC カードに実装し，その性能評価および安全性評価を行う．安全性評価では，未対策実装，対策実装に対して SASEBO-W を用いた電磁波解析攻撃(CEMA: Correlation Electromagnetic Analysis)実験を行い，対策の効果を実証する．

2 KCipher-2

KCipher-2 は，128 ビットの初期鍵(IK)と，128 ビットの初期ベクトル(IV)の 2 つの独立なパラメータを入力とするストリーム暗号である．**Algorithm1** に KCipher-2 のアルゴリズムの概要を示す．KCipher-2 は，以下の要素で構成される．

- 線形フィードバックシフトレジスタ(FSR-A)および動的フィードバックレジスタ(FSR-B)
- 4 つの内部レジスタ $R1$, $R2$, $L1$, $L2$ を有する Finite State Machine(FSM)

Algorithm1 : KCipher-2

Input: $IK = (IK_3, IK_2, IK_1, IK_0)$, $IV = (IV_3, IV_2, IV_1, IV_0)$

Output: $(z_0^L, z_0^H, z_1^L, z_1^H, ..., z_t^L, z_t^H)$

/*Key setup step*/
for $0 \le i \le 11$ **do**
  if $i \le 3$ **then**
    $K_i = IK_i$
  else if $i = 4n$ **then**
    $K_i = K_{i-4} \oplus Sub((K_{i-1} \ll 8) \oplus (K_{i-1} \gg 24)) \oplus Rcon[i/4 - 1]$
  else
    $K_i = K_{i-4} \oplus K_{i-1}$
  end if
end for

/*Key loading step*/
for $0 \le m \le 4$ **do**
  $A_m = K_{4-m}$
end for
$B_0 = K_{10}, B_1 = K_{11}, B_2 = IV_0, B_3 = IV_1,$
$B_4 = K_8, B_5 = K_9, B_6 = IV_2, B_7 = IV_3,$
$B_8 = K_7, B_9 = K_5, B_{10} = K_6$
/*Internal state update step*/
for $1 \le j \le 24$ **do**
  $R1_{j+1} = Sub(L2_j \boxplus B_{j+9})$
  $R2_{j+1} = Sub(R1_j)$
  $L1_{j+1} = Sub(R2_j \boxplus B_{j+4})$
  $L2_{j+1} = Sub(L1_j)$
  $A_{j+4} = \alpha_0 A_{j-1} \oplus A_{j-2} \oplus z_{j-1}^L$
  $B_{j+10} = (\alpha_1^{cl1_{j-1}} + \alpha_2^{1-cl1_{j-1}} - 1) B_{j-1} \oplus B_j \oplus B_{j+5} \oplus \alpha_3^{cl2_{j-1}} B_{j+7} \oplus z_{j-1}^H$
end for

/*Keystream generation*/
$t \leftarrow 0$
while datalength $< 64 * t$ **do**
  /*Repeat while data length is longer than keystream length*/
  $z_t^L = B_t \boxplus R2_t \oplus R1_t \oplus A_{t+4}$
  $z_t^H = B_{t+10} \boxplus \oplus L2_t \oplus L1_t \oplus A_t$
  $R1_{t+1} = Sub(L2_t \boxplus B_{t+9})$
  $R2_{t+1} = Sub(R1_t)$
  $L1_{t+1} = Sub(R2_t \boxplus B_{t+4})$
  $L2_{t+1} = Sub(L1_t)$
  $A_{t+5} = \alpha_0 A_t \oplus A_{t+3}$
  $B_{t+11} = (\alpha_1^{cl1_t} + \alpha_2^{1-cl1_t} - 1) B_t \oplus B_{t+1} \oplus B_{t+6} \oplus \alpha_3^{cl2_t} B_{t+8}$
  $t \leftarrow t + 1$
end while
return $(z_0^L, z_0^H, z_1^L, z_1^H, ..., z_t^L, z_t^H)$

また, KCipher-2 内部で使用されるレジスタの語長は 32 ビットである.

2.1 構成要素の概要

FSR-A, FSR-B は, それぞれ 5 レジスタ, 11 レジスタで構成される. FSR-A, FSR-B の更新は, 以下の式によって行われる.

$$A_i^{(t+1)} = \begin{cases} A_{i+1}^{(t)} & (i \neq 4) \\ \alpha_0 A_0^{(t)} \oplus A_3^{(t)} & (i = 4) \end{cases} \quad (1)$$

$$B_j^{(t+1)} = \begin{cases} B_{j+1}^{(t)} & (j \neq 10) \\ \alpha_{cl1^{(t)}} B_0^{(t)} \oplus B_1^{(t)} \oplus B_6^{(t)} \oplus \alpha_3^{cl2^{(t)}} B_8^{(t)} & (j = 10) \end{cases} \quad (2)$$

ここで, $\alpha_{(cl1^{(t)})}$は,

$$\alpha_{cl1^{(t)}} = \alpha_1^{cl1^{(t)}} + \alpha_2^{1-cl1^{(t)}} - 1 \quad (3)$$

である. また, 時刻 t の FSR-A, FSR-B の各レジスタの出力を $A_i^{(t)}$, $B_i^{(t)}$とし, i, j をレジスタの番号とする. i, j の範囲は $0 \le i \le 4$, $0 \le j \le 10$ である. 各レジスタの n 番目のビットの値は $A_2^{(t)}[n] \in \{0,1\}$のように表す. n = 31 のとき, レジスタの最上位ビットの値を表す. 式(2)の $cl1$,$cl2$ は, クロック制御ビットであり, FSR-A における A_2 の値を用いてそれぞれ $cl1^{(t)} = A_2^{(t)}[30] \in \{0,1\}$, $cl2^{(t)} = A_2^{(t)}[31] \in \{0,1\}$と与えられる.

一方 FSM は, 整数加算器と置換演算 Sub を含み, FSR-A の 2 つのレジスタ, FSR-B の 4 つのレジスタおよび 4 つの内部状態レジスタの値を入力とし, 1 サイクルごとに 64 ビットの鍵系列を出力する. 置換関数 Sub は, 入力された 32 ビットの値をバイト単位に分割し, 8 ビット入出力の換字処理を各バイトに施す Substitution 処理と, その出力に対し, 32 ビット入出力の Permutation 処理から構成される. これらは, それぞれ共通鍵暗号 AES[4]の S-box 処理, MixColumns 処理と同一の関数である.

2.2 初期化処理

KCipher-2 の処理段階は, 内部状態を初期化する初期化処理と内部状態を更新しながら鍵系列を生成する鍵系列出力処理に分けられる.

初期化処理は, 内部鍵の生成および初期鍵 IK と初期ベクトル IV の読み込み処理, 内部状態の初期化処理で構成される. 以下では, 128 ビットの初期鍵 IK を上位から 32 ビットずつに分割し $IK = (IK_3, IK_2, IK_1, IK_0)$ と表す. また, 128 ビットの初期ベクトル IVも同様に 32 ビットずつに分割し, $IV = (IV_3, IV_2, IV_1, IV_0)$ と表す.

まず, 初期鍵 IK を以下の方法で 12 個の 32 ビット内部鍵 K_k (k = 0,1,...,11) に拡張する.

$$K_k = \begin{cases} IK_{3-k} & (0 \le k \le 3) \\ K_{k-4} \oplus Sub(k_{k-1}) \oplus rc & (k = 4n) \\ K_{k-4} \oplus K_{k-1} & (k \neq 4n) \end{cases} \quad (4)$$

$$k_{k-1} = (K_{k-1} \ll 8) \oplus (K_{k-1} \gg 24) \quad (5)$$

ここで, Sub は Sub 関数の処理を表す. また, rc は定数であり, k = 4 のとき, $rc = (01000000)_{16}$, k = 8 のとき, $rc = (02000000)_{16}$となる.

内部鍵の生成後, 内部鍵と初期ベクトルにより KCipher-2 の内部状態を初期化する. その後, 式(6)~(9)に示すような内部状態の攪拌を 24 回 (t = 0,1,…23) 行う.

$$R1^{(t+1)} = Sub(L2^{(t)} + B_9^{(t)}) \quad (6)$$

$$R2^{(t+1)} = Sub(R1^{(t)}) \quad (7)$$

$$L1^{(t+1)} = Sub(R2^{(t)} + B_4^{(t)}) \quad (8)$$

$$L2^{(t+1)} = Sub(L1^{(t)}) \quad (9)$$

$$A_i^{(t+1)} = \begin{cases} A_{i+1}^{(t)} & (i \neq 4) \\ \alpha_0 A_0^{(t)} \oplus A_3^{(t)} \oplus ZL^{(t)} & (i = 4) \end{cases} \quad (10)$$

$$B_j^{(t+1)} = \begin{cases} B_{j+1}^{(t)} & (j \neq 10) \\ \alpha_{cl1^{(t)}} B_0^{(t)} \oplus B_1^{(t)} \oplus B_6^{(t)} \oplus \alpha_3^{cl2^{(t)}} B_8^{(t)} \oplus ZH^{(t)} & (j = 10) \end{cases} \quad (11)$$

ここで, $\alpha_{(cl1^{(t)})}$ は式(3)と等しい. また, $ZH^{(t)}$と $ZL^{(t)}$は, 時刻 tに出力する64ビットの鍵系列のそれぞれ上位, 下位 32 ビットの値である. 同様に, $R1^{(t)}$, $R2^{(t)}$, $L1^{(t)}$, $L2^{(t)}$は時刻 t の FSM の内部レジスタの値である. 初期化処理中は, 式(1), (2)とは異なり, 鍵系列がフィードバック関数に加え

られる.

2.3 鍵系列出力処理

初期化処理が終わると, 鍵系列出力処理に移行する. 時刻tに出力される64ビットの鍵系列$Z^{(t)}$は, 内部レジスタの値を用いて式(12)および(13)により求められる.

$$ZH^{(t)} = B_{10}^{(t)} \boxplus L2^{(t)} \oplus L1^{(t)} \oplus A_0^{(t)} \quad (12)$$

$$ZL^{(t)} = B_0^{(t)} \boxplus R2^{(t)} \oplus R1^{(t)} \oplus A_4^{(t)} \quad (13)$$

なお鍵系列出力処理中の内部状態の更新は, 式(1), (2), (6)~(9)によって行われる.

3 IC カード向け小型実装

本節ではKCipher-2ソフトウェアのデータメモリおよびプログラムメモリ使用量を削減する方法を述べる. KCipher-2 の実装には, 式(1), (2), (10), (11)で表される FSR-A, FSR-B の値の更新で行われる定数 α_0, α_1, α_2, α_3 と 32 ビット変数の $GF(2^{32})$上の乗算, 非線形関数である Sub 関数が必要であるが, 高速性を重視した単純な実装では上記の演算をすべて参照テーブルで実装することが多い. しかし, IC カードに搭載されるメモリ容量の小さいマイコンでは, 参照テーブルの大きさが制限される. 参照テーブルを全てデータメモリに置く場合, α_jの乗算では 1024 バイトの参照テーブルが 4 つ, Sub 関数では 1024 バイトの参照テーブルが 4 つ必要になり, 計 8192 バイトのメモリが必要となる. そのため, データメモリが 8192 バイト以下のマイコンには実装できない.

3.1 基本アイデア

本稿では, 以下の方針でコードサイズの小さなプログラムを設計する. まず, 参照テーブルを RAM に常駐させず, プログラムメモリのみに置く. テーブル参照の際に必要な部分のみをメモリへロードすることで, アクセス時間が増加する代わりにRAM使用量を削減できる. 次に, 演算の実装方法を適切に選択する. ガロア体上の演算や S-box の実装は, 参照テーブルによる実装の他に, 論理演算による実装が考えられる. ここで, どちらの実装が適するかは演算の複雑さに依存する. 論理演算による実装は単純な式で表される演算に適するが, 複雑な式の場合はプログラムサイズがかえって増加してしまう. 一方, 参照テーブルによる実装は, 処理を表す式の複雑さによらずプログラムサイズは一定となるが, 単純な式の論理演算実装に比べプログラムメモリ使用量が多い.

上記において式の複雑さを考える際には, マイコンにおける論理演算の実装を考える必要がある. マイコンでは 2 つのオペランドの対応するビット同士の論理演算を行う命令が実装されており, オペランドとなるビットが2つの変数の同位置に格納されている場合には効率的にプログラムを実装できるが, そうでない場合には, 1 ビットの論理演算を 1 命令で行うことから, メモリ使用量と実行時間が増大することになる. 一方で, オペランドを整列させるためのコストが大きい場合はその限りではない. 今回は, オペランドを整列する場合としない場合の両方を実装し, 比較することで式に適した実装を行う.

3.2 構成要素の実装方法

本節では, 上記の方針に基づく KCipher-2 の構成要素の実装方法について述べる.

まず, S-box は, 逆元演算とアフィン変換, 定数加算により実現できるが, これらの演算は比較的複雑な論理式で表されることから, 参照テーブルにより実装する. 上節で述べたように, S-box の参照テーブルは RAM に常駐させずにプログラムメモリから読み出す. また, Permutation 処理中の乗算は, 式(14), (15)に示すような8ビット単位のXORとビットシフトのみで記述できる. そのため, ビット単位の演算が適するといえる.

$$y = ((x >> 7) \times (1B)_{16}) \oplus ((x << 1) \& (FF)_{16}) \quad (14)$$

$$y = ((x >> 7) \times (1B)_{16}) \oplus ((x << 1) \& (FF)_{16}) \oplus x \quad (15)$$

一方, 32 ビットの長さを持つ α_j の乗算は, 比較的複雑な論理演算で表させるが, 参照テーブルで実装した場合でも 4096 バイトのメモリを要するため, プログラムサイズが小さくなる実装を判断することは難しい. そこで, 本稿では両方の実装を用意し, 第 5 節で評価する. 参照テーブルを用いる場合には, 全てを参照テーブルとする実装と, 一部の処理を参照テーブルとする実装の 2 通りが考えられる. ここで, 一部を参照テーブルとする場合には, ビット単位の XOR の計算をバイト単位の XOR に変換するための定数をテーブルで実装する. また, 論理演算での実装も複数考えられる. 3.1 節で述べたように, 単純な実装ではプログラムサイズが大きくなるが, α_j の乗算でビット単位の処理が必要となるのは先頭の 8 ビットのみである. したがって, 先頭 8 ビットのみ 1 ビットを 1 バイト変数として扱い, 残り 24 ビットは 8 ビットごとの論理演算で実装することで, プログラムメモリを削減できる. さらに, 論理演算を多項式の乗算により実現することもできる. その場合, α_j の乗算用の参照テーブルを追加する必要がある.

本稿では, 以上で述べた(a)全て参照テーブルによる実装, (b)一部参照テーブルを用いた実装, (c)単純な論理演算実装, (d)先頭 8 ビットのみ 1 ビットを 1 バイトとして扱う実装の 4 つの実装を検討する.

4 サイドチャネル攻撃対策を施した小型実装

4.1 提案手法

本稿では KCipher-2 ハードウェアへの電力解析攻撃対策[3]をソフトウェアとして実現した対策アルゴリズムを提案する. **Algorithm2** に乱数マスキングを用いた KCipher-2 のアルゴリズム (**Algorithm1** からの変更箇所) を示す. KCipher-2 に対するサイドチャネル攻撃では内部状態 $L1$ の値更新処理時の漏洩情報を元に

Algorithm2 : Masked Internal State Update Step

/*Internal state update step*/
$\mathrm{buf0} = 0, \mathrm{buf1} = 0, \mathrm{buf2} = 0, \mathrm{buf3} = 0$
for $1 \le j \le 24$ **do**
  /*Unmasking step*/
  $B_{j+9} \leftarrow B_{j+9} \oplus buf1$
  $L2_j \leftarrow L2j \oplus buf3$
  /*Masking step*/
  $m_{R2}, m_{B4} \leftarrow RandomValue$
  $maskR2 = R2_j \oplus m_{R2}, maskB4 = B_{j+4} \oplus m_{B4}$
  $R1_{j+1} = Sub(L2_j \boxplus B_{j+9})$
  $R2_{j+1} = Sub(R1_j)$
  $b1 = MaskedAdd(maskR2, maskB4, m_{R2}, m_{B1})$
  $L1_{j+1} = MaskedSub(b1, m_{R2} \oplus m_{B4})$
  $L2_{j+1} = MaskedSub(L1_j, buf0)$
  $A_{j+4} = \alpha_0 A_{j-1} \oplus A_{j-2} \oplus z^L_{j-1}$
  $B_{j+10} = (\alpha_1^{cl1_{j-1}} + \alpha_2^{1-cl1_{j-1}} - 1)B_{j-1} \oplus$
    $B_j \oplus B_{j+5} \oplus \alpha_3^{cl2_{j-1}} B_{j+7} \oplus z^H_{j-1}$
  /*Unmask value update step*/
  $m_{add} = m_{R2} \oplus m_{B4}$
  $m_{L1} = Permutation(Affine(m_{add}))$
  $m_{L2} = Permutation(Affine(m_{L1}))$
  buf1 ← buf0,buf3 ← buf2
  buf0 ← m_{L1},buf2 ← m_{L2}
end for

鍵の復元を行う. $L1$ を含む処理は入力側に FSR-B, 内部状態 $R2$ を入力とする整数加算器と Sub 関数があり, 出力側には Sub 関数と内部状態 $L2$ がある. そこで, 整数加算器の入力からマスクし, 内部状態 $L2$ の出力で値をアンマスクすることを考える. 乱数マスキングでは, アンマスク用の値を計算する必要がある. ここで, ある演算を f , 演算器への入力を x, マスク値を m_x とし, 以下の関係式を考える.

$$f(x \oplus m_x) = f(x) \oplus f(m_x) \qquad (16)$$

式(16)が成立する演算は, 演算を 2 回実行もしくは二重化し, 片方でマスクの値を入力することでアンマスクの値を容易に計算できる.

一方で, 式(16)が成立しない演算もある. 鍵復元に利用される $L1$ を含む処理の中では整数加算器と Sub 関数内の Substitution 処理がこれにあたり, これらの処理のハードウェア実装におけるマスキング手法[3]が現在までに提案されている. 以下では, これらのマスキングのソフトウェア実装に適した実現方法を述べる.

4.2 整数加算のマスキング

文献[3]では，整数加算器へのマスキング手法として，Golic によるマスクを考慮した AND 演算(Masked AND)[5]を応用した加算を用いていたが，本手法では，省リソースのソフトウェア実装を実現するため，Karroumi らのマスク加算[6]を用いる．マスクされた加算アルゴリズムを **Algorithm3** に示す．この加算ではマスクされた入力を $\boldsymbol{x} = x \oplus m_x$, $\boldsymbol{y} = y \oplus m_y$とし，$\boldsymbol{s} = (x + y) \oplus (m_x \oplus m_y)$を出力とする．本手法は，1 ビットを 1 バイト変数として扱うことなく計算でき，かつ，アンマスク値の計算が容易であるという特長を有しており，Golic の手法と比べて，効率的なソフトウェア実装を可能にする．

Algorithm3 : Secure addition with blinded operands

Input:$(\boldsymbol{x},\boldsymbol{y}, r_x, r_y, \gamma) \in Z_{2^k}^5$ such that $\boldsymbol{x} = x \oplus r_x$, $\boldsymbol{y} = y \oplus r_y$ and γ a pre-computed random integer

Output:$(\boldsymbol{s}, r_s)$ where $\boldsymbol{s} = (x + y) \oplus r_s \pmod{2^k}$ and $r_s = r_x \oplus r_y$

```
/* Ω0 = (x & y) ⊕ γ */
C ← γ;
T ← x & y ; Ω ← C ⊕ T;
T ← x & ry; Ω ← Ω ⊕ T;
T ← y & rx; Ω ← Ω ⊕ T;
T ← rx & ry; Ω ← Ω ⊕ T;
/* c(1) = 2Ω0 and Ω = 2γ & (x ⊕ y) ⊕ Ω0*/
B ← Ω ≪ 1; C ← C ≪ 1;
A0 ← x ⊕ y; A1 ← rx ⊕ ry;
T ← C & A0; Ω ← Ω ⊕ T;
T ← C & A1; Ω ← Ω ⊕ T;
/*Main loop*/
for i = 2 to k − 1 do
  T ← B & A0; B ← B & A1;
  B ← B ⊕ Ω;
  B ← B ⊕ T;
  B ← B ≪ 1;
end for
/*Aggregation*/
A0 ← A0 ⊕ B;
A0 ← A0 ⊕ C;
return (A0, A1);
```

4.3 Substitution 処理のマスキング

Substitution 処理では，ガロア体上の逆元演算に対するマスキングを実現する必要がある．その方法として，加法的なマスキング手法と乗算的なマスキング手法が考えられる．加法マスキングはビットレベルの処理が必要となるため，プログラムサイズが増大してしまう．一方で，乗法マスキングは $GF(2^8)$の乗算と逆元演算を実装することでビットレベルの計算を行わずに実装でき，プログラムメモリ使用量を抑えることができる．そこで，提案手法では逆元演算のマスクを Akkar らの手法[7]に基づく乗算的なマスキングにより実現する．[7]の手法ではマスク処理を施した逆元演算を図 1 の流れで行う．ここで，逆元演算への入力を $A = a \oplus m$, XOR によるマスクを m, 乗法マスクを y とする．図 1 中に存在するガロア体上の逆元演算は複雑な論理式となることから参照テーブルを用いて実装する．

上記の手法を用いてマスキングを施す場合，対策を施さない実装とは異なる参照テーブルを要する．Substitution 処理のマスキングではガロア体上の逆元演算を参照テーブルにより実装する必要がある．

マスクされた S-box やアンマスク値の計算に用いるアフィン変換は，8 ビットの入力を 1 ビットの配列に変換したのちに XOR を用いて 8 ビットの値を計算することで実装できるが，プログラムサイズ削減のため，提案手法では参照テーブルにより実装する．

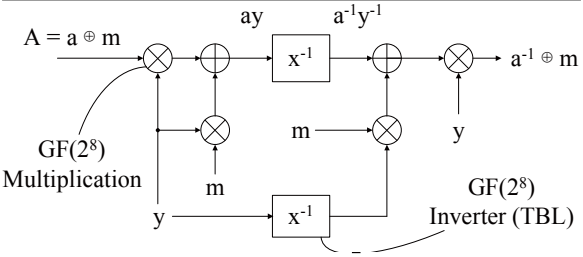

図 1. 乗法的なマスキング手法による逆元演算

5 評価

本節では，提案した KCipher-2 ソフトウェアの性能評価と安全性評価について述べる．性能評価では，プログラムメモリ，データメモリ使用量と実行速度を評価する．安全性評価では，未対策実装・対策実装に対してサイドチャネル標準評価ボード SASEBO-W[8]を用いた電磁波解析攻撃を実行し，対策の有用性を示す．ここでは，α_j の乗算を 3.2 節に示す(a)~(d)の 4 種類で実装し，それらを評価した．

表 1. KCipher-2 ソフトウェアのメモリ使用量と実行時間

	Without countermeasure				With countermeasure	
Implementation No.	1-1	2-1	3-1	4-1	3-2	4-2
Mult. of α_j	(a)	(b)	(c)	(d)	(c)	(d)
Program Memory(bytes)	11834	10824	11392	9728	15532	14186
Data Memory(bytes)	317	314	313	313	829	829
Time(ms)	95.94	143.91	127.98	111.94	768.05	767.93

5.1 性能評価

表 1 に実装した KCipher-2 ソフトウェアのメモリ使用量および実行時間を示す. 実行時間は IC カード上に実装したソフトウェアを SASEBO-W で実行した際の実行時間を SASEBO-W の制御装置で測定した値である.

未対策の場合, プログラムメモリ使用量, データメモリ使用量は 4-1 の実装が最も小さくなっている. 4-1 の実装は参照テーブルを用いずに α_j の乗算をビット単位演算により実装しているためと考えられる. 一方で処理速度については, 1-1 の実装が最も高速である. α_j の乗算を参照テーブルにより実装したことで高速に実装できたと考えられる. 以上の結果より, 提案する設計手法により, 高速な実装と小型な実装を使用する用途により選択できることを確認できる.

対策実装は, 未対策実装と比較すると, プログラムメモリ使用量は約 1.5 倍に, 処理速度は約 6.9 倍となった. 以上より, 対策を施した実装でも, 本稿で対象としている小型 IC カード (ATmega163 マイコン搭載. プログラムメモリ容量 16KB, データメモリ容量 1KB)実装できることが分かる.

5.2 安全性評価

安全性評価では, 作成した未対策実装, 対策実装の KCipher-2 ソフトウェアに対して電磁波解析攻撃を実行し, 秘密鍵推定の可否を評価する. 表 3 に詳細な実験条件を, 図 2 に実験環境概観を示す. 本実験では, 既知の KCipher-2 ハードウェアに対する電磁波解析攻撃[3]をソフトウェア向けに拡張して解析を行った. [9]では, 電磁

表 2. 実験条件

Microcontroller	ATmega163(8bit)
Operation frequency	4MHz
Magnetic field probe	Morita Tech MT-545
Amplifilter	YKC-1000AMP
Oscilloscope	Agilent MSO6104A
Sampling frequency	400MSa/s

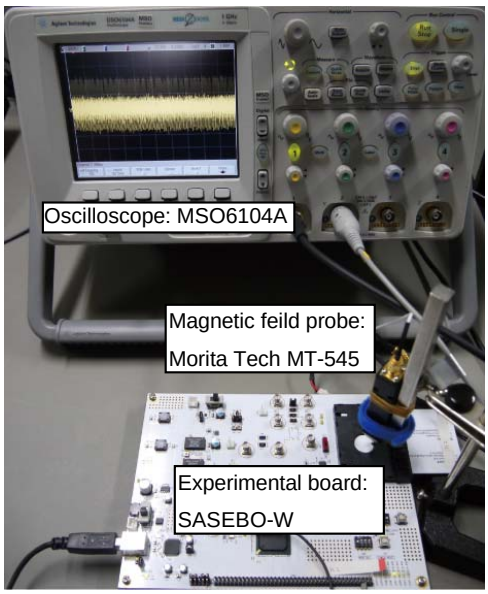

図 2. 実験環境概観

波解析攻撃の中間値を Sub 関数後段の Permutation の出力とし, 平文選択を行うことで上位バイトから鍵推定を行っていた. しかし, ソフトウェア実装では Sub 関数前段の S-box の出力を中間値とした秘密鍵の推定が可能となる. また, 中間値を S-box の出力として下位バイトから鍵の推定を行うことで, 平文選択が不要となる. 以上より, 中間値を S-box の出力とし, 秘密鍵を下位バイトから推定する解析を行った.

図 3 と図 4 にそれぞれに未対策実装と対策実装に対する電磁波解析実験の結果を示す. ここで図 3 は 4-1 の実装に対して, 図 4 は 4-2 の実

装に対して電磁波解析を行った結果である. その他の実装においても, 未対策実装である 1-1, 2-1, 3-1 では図 3 と同様の, 対策実装である 3-2 では図 4 と同様の結果が得られた.

図 3, 4 は部分鍵 K_5 を推定した際の MTD(Measurement to Disclosure)である. MTD は正しい鍵候補と誤った鍵候補の相関値の推移であり, 黒線は正しい鍵候補, 灰線は誤った鍵候補による相関値を表す. また, 横軸は解析に用いた波形数, 縦軸は対応する相関値である. 今回解析に使用した波形は未対策実装, 対策実装ともに 1000 波形である. 図 3, 4 より未対策実装では 50 波形程度で攻撃が成功しているのに対し, 対策実装では 1000 波形用いても秘密鍵を推定できないことが確認できた.

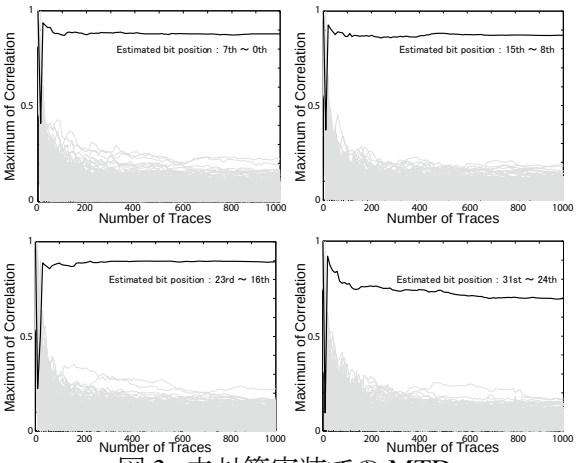

図 3. 未対策実装での MTD

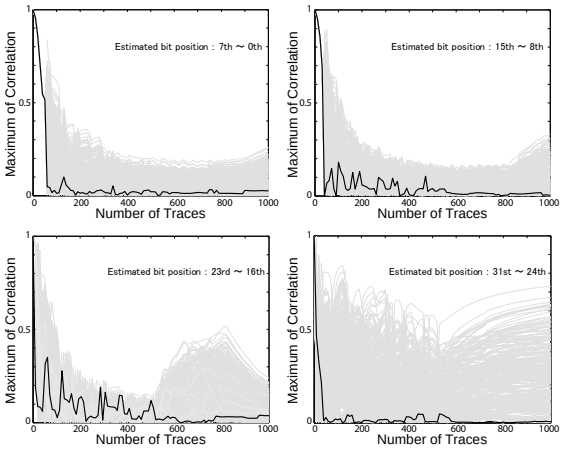

図 4. 対策実装での MTD

6 まとめ

本稿では, KCipher-2 ソフトウェアの小型実装について述べた. 4 種類の実装を提案するとともに, 電磁波解析攻撃に対する対策を施した実装を検討した. それらを ATmega163 搭載の IC カード上に実装し, その性能評価と安全性評価を行った. 性能評価では作成したソフトウェアが小型 IC カード上に実装可能であることを確認した. 電磁波解析攻撃対策を実装することでプログラムメモリ使用量の増加と処理速度の増加をそれぞれ 1.5 倍, 6.9 倍程度に抑えられることを示した. また, 未対策実装, 対策実装に対して SASEBO-W を用いた電磁波解析攻撃を実行し, 対策実装では 1000 波形を用いても鍵の推定が行えないことを示した. 今後の課題としては, 部分鍵 K_9 の推定を行い, 128 ビットの初期鍵すべてを復元することが挙げられる.

参考文献

[1] S. Kiyomoto, T. Tanaka, and K. Sakurai, “K2: A stream cipher algorithm using dynamic feedback control,” SECRYPY, pp.204-213.

[2] P. Ekdahl and T. Johansson, “A new version of the stream cipher snow,” Selected Areas in Cryptography, pp.47-61, Springer, 2003.

[3] T.Hibiki, N.Homma, Y.Nakano, K.Fukushima, S.Kiyomoto, Y.Miyake, and T.Aoki, ”Chosen-IV correlation power analysis on KCipher-2 hardware and a masking-based countermeasure,” IEICE TRANS, vol.E97-A, No.1, pp.157-166, 2014

[4] J. Daemen and V. Rijmen, The design of Rijndael : AES - the advanced encryption standard, Springer Verlag, 2002.

[5] J.Golic, “Techniques for random masking in hardware,” IEEE Trans. Circuits and Systems, vol.54, No.2, pp.291-300, 2007.

[6] M.Karroumi, B.Richard, and M.Joye, “Addition with Blinded Operands” 5th International Workshop on Constructive Side-channel Analysis and Secure Design(COSADE), (Paris, France), Apr. 2014.

[7] M.Akkar and C.Giraud, “An implementation of DES and AES, secure against some attacks,” Proc. CHES, Lecture Notes in Computer Science, vol.2162,pp.309 -318, 2001.

[8] “サイドチャネル攻撃用標準評価ボード SASEBO – SASEBO-W.” http://satoh.cs.uec.ac.jp/SASEBO/ja/board/sasebo-w.html.

[9] 宇野甫, 遠藤翔, 本間尚文, 青木孝文, 仲野有登, 清本晋作, 三宅優, “ZigBee 評価用マイコン上に実装された KCipher-2 に対する相関電磁波解析攻撃の検討, ”電子情報通信学会技術研究報告, Vol. 113, No. 483, pp35--40, 2014