Feistel 構造をもつ軽量暗号の系統的な可逆化

2020SE097 鈴木琳久 2020SE077 山田祐介
指導教員：横山哲郎

1　はじめに

IoT や組込みシステムでは限られた計算能力，メモリサイズ，消費電力，およびオブジェクトコードサイズといった多くのリソース制約がある．軽量暗号 [1] はこのような制約のあるデバイスに向けて設計された暗号技術であり，暗号化における安全性と効率はトレード・オフの関係にある．その安全性について，軽量暗号化アルゴリズムは，タイミング攻撃やメモリ検査などのサイドチャネル攻撃に対して強固である必要がある．つまり，データの暗号化にかかる時間は暗号キーなどの秘密データの値に非依存でなければならない．また，軽量暗号は Feistel 構造のような対称的な構造をとることで実装コストを削減することが可能であり，それによって暗号化と復号のプロセスは基本的に逆になるため，可逆プログラミング言語で軽量暗号アルゴリズムを表すことは自然である．Hermes[2] は軽量暗号アルゴリズムの記述を支援する可逆プログラミング言語であり，秘密値の操作に対して所要時間が不変であるため，タイミング攻撃に対する耐性をもっている．Hermes のような可逆プログラミング言語では可逆な文しか記述できないが，軽量暗号は必ずしも可逆性を有するように設計されていない．加えて，非可逆な軽量暗号アルゴリズムを効率的な Hermes プログラムに手動で可逆化することは煩雑である．そのため，本研究では Feistel 構造の煩雑な可逆化を自動化する手法を提案し，提案した手法の適用範囲を明らかにすることを目的とする．

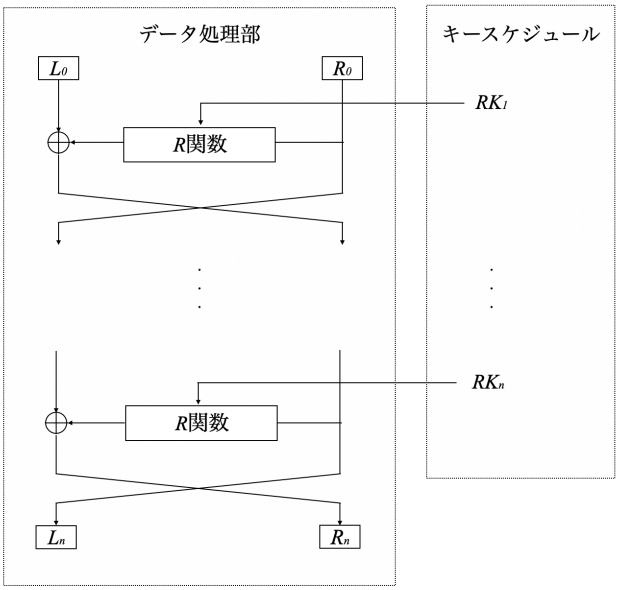

図 1　Feistel 構造

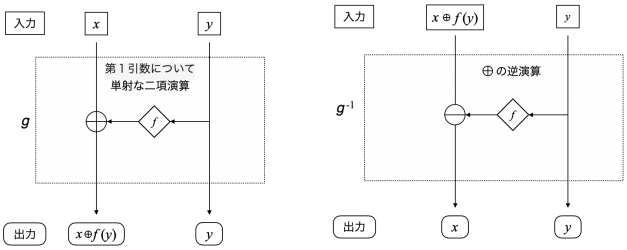

図 2　可逆更新

2　関連研究

2.1　Feistel 構造

Feistel 構造はブロック暗号の構造の 1 種である．入力を左入力 L と右入力 R の 2 つに分割し，右入力 R とキースケジュールによって生成されたラウンドキー RK を引数とする R 関数と左入力 L との排他的論理和をとる．その演算結果と元の右入力 R を左右入れ替えて，それぞれ右入力 R'，左入力 L' を出力とする．この一連の処理が Feistel 構造の 1 ラウンドであり，これを複数回繰り返すことで入力情報を拡散する．ラウンドを n 回繰り返した Feistel 構造を図 1に示す．また，暗号化と復号のプロセスが基本的に逆であることから Feistel 構造の 1 ラウンドは可逆更新の 1 種であると考えられる．

2.2　可逆更新

暗号化と復号のデータ処理はデータの上書きによる破壊的な更新ではなく，可逆更新によってデータを更新する．可逆更新とは，入力 x, y に対して，x と $f(y)$ を第 1 引数について単射な二項演算をした結果 $x \oplus f(y)$ と元入力 y が出力となるような部分関数 g である．逆に，入力 $x \oplus f(y)$, y に対して $\oplus$ の逆演算である $\ominus$ をすると，元入力の x と y が出力となる．可逆更新を図 2に示す．

可逆更新の例として図 3を考える．入力を x, y_1, y_2 とし，y_1, y_2 について非単射な演算 $\otimes$ をした結果 $y_1 \otimes y_2$ と第 1 引数について単射な二項演算 $\oplus$ をした結果 $x \oplus (y_1 \otimes y_2)$ と y_1, y_2 を出力とするとき，逆に入力を $x \oplus (y_1 \otimes y_2)$, y_1, y_2 とし，$\oplus$ の逆演算 $\ominus$ を用いれば，x, y_1, y_2 を出力として得ることができる．このように非単射な演算を用いても，入力と出力の対応を同じにすれば g 全体は単射となる．

3　軽量暗号アルゴリズムの提案

軽量暗号アルゴリズムを以下の手順に沿って可逆化する手法を提案する．

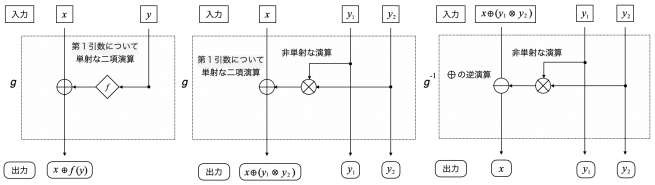

図 3 可逆更新の例

- 仕様の単射化 [3]
  共通鍵暗号では暗号化 *encrypt* と復号 *decrypt* は，共通鍵 k が存在して，任意の平文 t と暗号文 c に対して，

$$encrypt(t,k) = c \tag{1}$$
$$decrypt(c,k) = t \tag{2}$$

  である．これらは明らかに単射ではない．k を余剰な値として返すことで *encrypt* を単射化した

$$encrypt'(t,k) = (c,k) \tag{3}$$

  を得る．*decrypt* の単射化は *encrypt* の単射化である．つまり，$decrypt' = encrypt'^{-1}$ が成り立つ．以降では $encrypt'$ の効率的な実現を行う．
- プログラムの可逆化
  軽量暗号アルゴリズムを記述した A プログラムをクリーン可逆化する．ここで，プログラムの可逆化とはプログラムの各ステップが単射になるようにする変換である．また，クリーン可逆化とは余剰データを出力することがないようにする可逆化である．
- 変数の定義-使用関係を辺とするグラフの作成と順序の変更
  暗号アルゴリズムから有向グラフを作成し，トポロジカルソートによる順序の変更をする．そして，変数の定義-使用関係のグラフを作成する．ここで，グラフの頂点は代入，辺は変数の定義から使用への有向辺である．作成したグラフからクリーン可逆化された暗号アルゴリズムを考える．
- Hermes プログラムの生成
  軽量暗号アルゴリズムを記述する疑似言語から Hermes プログラムへの変換を機械的に行う．

提案した手法の全体図を図 4に示した．

4 軽量暗号アルゴリズムの記述言語

軽量暗号のアルゴリズムの記述に頻繁に使用されるような小さな言語 A を定める．簡単なため，32 ビット符号なし整数型のみの値を考える．

式 e を次のように定める

$$e ::= n \mid x \mid x[e] \mid ROL_n(e) \mid ROR_n(e) \mid e \odot e$$

式は整数値，変数，配列の要素のルックアップ，n ビット左循環シフト ROL_n，n ビット右循環シフト ROR_n，二項

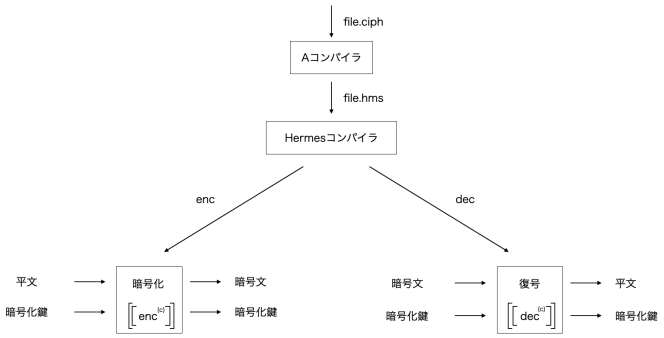

図 4 提案手法の全体図

演算式からなる．ここで，$\odot$ をどちらの引数についても必ずしも単射で無い演算

$$\odot ::= \mathrm{mod} \mid \mathbb{R} \mid \cdots$$

とし，$\mathbb{R}$ を第 1 引数について単射な演算

$$\mathbb{R} ::= \oplus \mid \boxplus$$

とする．ただし，$x \oplus y$ は同じ長さのビット列 x と y のビット毎の排他的論理和である．$x \boxplus y$ は 2^{32} を法とする 32 ビット符号なし整数 x と y の加算である．以上の記法は [4] に倣った．

式のある部分式をホール [] で置き換えた文脈を次のように定める:

$$C ::= [] \mid ROL_n(C) \mid ROR_n(C) \mid C\mathbb{R}e \mid e\mathbb{R}C$$

$C[e]$ は，C 中のホール [] を式 e で置き換えた式を表すこととする．

式 e の値の変数 x への代入は $x \leftarrow e$ と記す．

使用例として RC5[5] に記載されている暗号化アルゴリズムを A で記述したものを下記に示す．

$$\begin{aligned}
&A = A + S[0];\\
&B = B + S[1];\\
&\text{for } i = 1 \text{ to } r \text{ do}\\
&\quad A = ((A \oplus B) \lll B) + S[2 * i];\\
&\quad B = ((B \oplus A) \lll A) + S[2 * i + 1];
\end{aligned}$$

この RC5 の暗号化アルゴリズムを, 言語 A に変換すると以下のようになる．

$$A \leftarrow \mathrm{ROL}_B(A \oplus B) \boxplus S[2 \cdot i] \tag{4}$$
$$B \leftarrow \mathrm{ROL}_A(B \oplus A) \boxplus S[2 \cdot i \boxplus 1] \tag{5}$$

5 Hermes への変換

代入文から Hermes プログラムへの変換は次の通りである:

$$\begin{aligned}
\mathcal{R}[\![x \leftarrow x]\!] &= \\
\mathcal{R}[\![x \leftarrow ROL_n(C[x])]\!] &= \mathcal{R}[\![x \leftarrow C[x]]\!] \\
&\quad\ x \texttt{ <<= } n \\
\mathcal{R}[\![x \leftarrow ROR_n(C[x])]\!] &= \mathcal{R}[\![x \leftarrow C[x]]\!] \\
&\quad\ x \texttt{ >>= } n \\
\mathcal{R}[\![x \leftarrow C[x] \mathbb{R} e]\!] &= \mathcal{R}[\![x \leftarrow C[x]]\!] \\
&\quad\ x\ \ \mathcal{R}_{op}[\![\mathbb{R}\texttt{=}]\!]\ \ \mathcal{T}_{exp}[\![e]\!] \\
\mathcal{R}[\![x \leftarrow e \mathbb{R} C[x]]\!] &= \mathcal{R}[\![x \leftarrow C[x]]\!] \\
&\quad\ x\ \ \mathcal{R}_{op}[\![\mathbb{R}\texttt{=}]\!]\ \ \mathcal{T}_{exp}[\![e]\!] \\
\mathcal{R}[\![e_1 \leftrightarrow e_2]\!] &= \mathcal{T}_{exp}[\![e_1]\!] \leftrightarrow \mathcal{T}_{exp}[\![e_2]\!]
\end{aligned}$$

復号代入演算子の変換は次の通りである:

$$\begin{aligned}
\mathcal{R}_{op}[\![\oplus\texttt{=}]\!] &= \texttt{\^{}=} \\
\mathcal{R}_{op}[\![\boxplus\texttt{=}]\!] &= \texttt{+=}
\end{aligned}$$

式の変換は次の通りである:

$$\begin{aligned}
\mathcal{T}_{exp}[\![x]\!] &= x \\
\mathcal{T}_{exp}[\![n]\!] &= n \\
\mathcal{T}_{exp}[\![x[e]]\!] &= x\texttt{[}\mathcal{T}_{exp}[\![e]\!]\texttt{]} \\
\mathcal{T}_{exp}[\![e_1 \mathbb{R} e_2]\!] &= \mathcal{T}_{exp}[\![e_1]\!]\ \ \mathcal{T}_{op}[\![\mathbb{R}]\!]\ \ \mathcal{T}_{exp}[\![e_2]\!]
\end{aligned}$$

2 項演算子の変換は次の通りである:

$$\begin{aligned}
\mathcal{T}_{op}[\![\oplus]\!] &= \texttt{\^{}} \\
\mathcal{T}_{op}[\![\boxplus]\!] &= \texttt{+} \\
\mathcal{T}_{op}[\![\cdot]\!] &= \texttt{*} \\
\mathcal{T}_{op}[\![\mathrm{mod}]\!] &= \texttt{\%}
\end{aligned}$$

6 SIMON での適用例

軽量暗号 SIMON[6] はハードウェアとソフトウェアの両方での高い実装機能を有しており，特にハードウェアの実装に特化している軽量暗号である．SIMON のブロックサイズはワードサイズ n ビットに対して $2n$ ビットであり，鍵長はワードサイズ n，キーサイズ m に対して mn ビットである．SIMON のビットパラメータは 5である．本研究ではブロックサイズが 128 ビットを考えるが，他の場合でも一部の単純な変更を施すことで同様に扱うことができる．

- SIMON のラウンド関数の単射化
  図 6が単射化した SIMON のラウンド関数である．図 6から出力の K を除いたものが元の SIMON のラウンド関数であり，ラウンドキー K を余剰な値として追加することで単射化した．ここで，SIMON は可逆更新の組み合わせで成り立っている．

block size $2n$	key size mn	word size n	key words n	const seq	rounds T
32	**64**	16	4	z_0	32
48	**72**	24	3	z_0	36
	96		4	z_1	36
64	**96**	32	3	z_2	42
	128		4	z_3	44
96	**96**	48	2	z_2	52
	144		3	z_3	54
128	**128**	64	2	z_2	68
	192		3	z_3	69
	256		4	z_4	72

図 5 SIMON のビットパラメータ

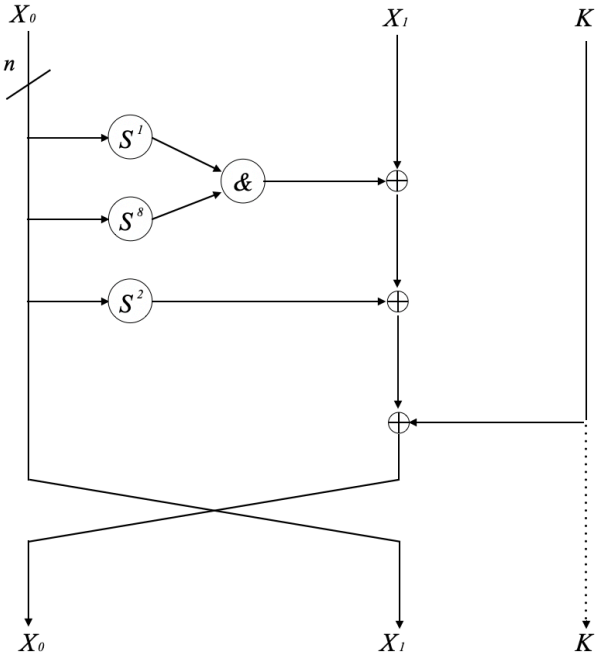

図 6 単射化した SIMON のラウンド関数

- SIMON の暗号化アルゴリズム
  文献 [6] に記載されている SIMON の暗号化アルゴリズムは以下の通りである．ここで，S^n は n ビット左循環シフト，&は論理積 (AND 演算) である．

$$\begin{aligned}
&X_0^{i+1} \leftarrow X_1^i \oplus ((S^1 X_0^i \& S^8 X_0^i) \oplus S^2 X_0^i) \oplus RK_i, \\
&X_1^{i+1} \leftarrow X_0^i.
\end{aligned}$$

- SIMON の変数の定義使用関係のグラフ
  変数同士の依存関係を有効グラフとして 7に示した．変数の値を計算するためには，その変数が終点となる

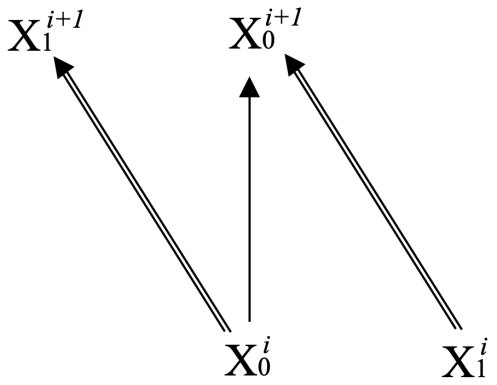

図 7　SIMON の暗号化アルゴリズムの変数の定義-使用関係

```
simon(u64 X0,u64 X1,u32 RK[68])
{
  for (i = 0; 68) {
    X1 ^= (X0 << 1 & X1 << 8) ^ (X0 << 2) ^ RK[
    i]
    X1 <-> X0;
  }
}
```

図 8　SIMON 暗号化プログラム

全ての辺の始点にある変数の値が必要である．一重線の辺については終点の値を計算した後に，始点の変数の値を別の変数の計算に使用することが出来るが，二重線の辺については終点の値を計算した後に，始点の変数の値を別の変数の計算に使用することが出来ない．

以上より, 変数名を書き換えてクリーン可逆化した SIMON の暗号化アルゴリズムは次のようになる．

$$X_1 \leftarrow X_1 \oplus ((S^1X_0 \& S^8X_0) \oplus S^2X_0) \oplus RK_i,$$
$$X_1 \leftrightarrow X_0.$$

- SIMON の Hermes プログラム
  得られた SIMON の Hermes プログラムを 8に示した．

7　おわりに

本研究では Feistel 構造をもつ軽量暗号に対して暗号アルゴリズムのクリーンな可逆化を系統的に行う手法を提案した．軽量暗号 SIMON を適用例として提案手法を適用し，適切な暗号化・復号のCプログラムを得ることができた．また，意味を変えることなく可逆プログラムに変換できることが保証された言語 A を提案した．

今後の課題としては以下が挙げられる．第 1 に提案した手法で得られたプログラムと手動で記述した Hermes プログラムを比較する必要がある．A で記述したプログラムは必ずしも可逆性を有している必要はないが，中間生成される Hermes プログラムは手動で可逆化を行ったプログラムよりも多くのリソースを必要とする可能性がある．第 2 に A と変則規則の実装を行って設計に誤りがないことを確認する必要がある．第 3 に生成されたプログラムの効率化が必要である．素朴な可逆化を行って得られる Hermes プログラムは効率が悪く，他の言語で実装したプログラムと比べてコードサイズが大きい，処理速度が低下するといった問題がある．このようにプログラムの可逆性を保つためには余分なコードが必要であるが．それに伴ってプログラムの効率が低下してしまう問題がある．可逆性を保つ余分なコードを取り除くことが可能であり，尚且つ入力と出力の対応が正しく，秘密値の操作に対する所要時間が不変であるような離可逆化による効率化を行う必要がある．

参考文献

[1] Lightweight Cryptography Working Group, et al. Cryptrec cryptographic technology guideline (lightweight cryptography). *March. Japan*, 2017.

[2] Torben Ægidius Mogensen. Hermes: a reversible language for lightweight encryption. *Science of Computer Programming*, Vol. 215, p. 102746, 2022.

[3] Robert Glück and Tetsuo Yokoyama. Reversible computing from a programming language perspective. *Theoretical Computer Science*, Vol. 953, p. 113429, 2023.

[4] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, and Dong-Geon Lee. Lea: A 128-bit block cipher for fast encryption on common processors. In *Information Security Applications: 14th International Workshop, WISA 2013, Jeju Island, Korea, August 19-21, 2013, Revised Selected Papers 14*, pp. 3–27. Springer, 2014.

[5] Ronald L Rivest. The rc5 encryption algorithm. In *International Workshop on Fast Software Encryption*, pp. 86–96. Springer, 1994.

[6] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The simon and speck families of lightweight block ciphers. *cryptology eprint archive*, 2013.