軽量暗号アルゴリズムの可逆化と可逆プログラミングによる実現
―SIMON と LEA を対象として―

2019SE040 長瀬由直　2019SE075 嶋田龍貴
指導教員：横山哲郎

1　はじめに

IoT 機器などのデバイスは，省資源でありながら，安全性の高い暗号の実装が求められる．従来の暗号アルゴリズムを IoT 機器に実装することは適切ではなく，消費電力量やエネルギー消費量を抑えられるような軽量暗号の実装が求められる．

多くの軽量暗号では，暗号化の各ステップを逆に並び替えて実行したものは，復号の処理と同じとなり，暗号文から元の平文を取得可能である．このような場合，可逆プログラミング言語で暗号化のアルゴリズムを記述すれば，その手続きを逆に実行することで，復号の手続きを実行できる．よって，可逆プログラミング言語で暗号アルゴリズムの記述を行うことによって，暗号化と復号のプログラムの保守と正当性の保証を支援し，モジュール性を高めることができる．可逆プログラミング言語 Hermes[1] は，軽量暗号アルゴリズムの記述を支援する領域特化言語である．機密度を持つ型付けにより，プログラマは情報の漏洩といった脆弱性を懸念せずコードの記述に特化できるというメリットがある．Hermes では，単射かつ可逆な文のみを記述できる．すなわち，軽量暗号アルゴリズムを可逆性制約のある Hermes プログラムで実現するために，主要な軽量暗号アルゴリズムの単射化をすることが必要である．

現段階で主要な軽量暗号アルゴリズムを Hermes の言語機能によりプログラムが記述可能か検証済みのものが少ないという背景がある．よって，Hermes の有用性評価のために，文献 [2] に載っている主要な軽量暗号が記述可能かの検証が重視される．また，Hermes が軽量暗号アルゴリズムの記述を支援できるよう，記述を行えた Hermes プログラムが，効率的な処理を行っているか，加えて，コードの理解がしやすく，読みやすいものであるかを議論することが望ましいとされる．可読性が高く，効率的な Hermes プログラムを実現できなければ，言語機能の拡張が期待される．よって，本研究の目的は，Hermes を用いて主要な軽量暗号アルゴリズムを可読性の高い，効率的で可逆なプログラムで実現することである．

単射化及び可逆化する軽量ブロック暗号を SIMON と LEA とする．それらのアルゴリズムが非単射であることを確かめる．次に，暗号アルゴリズムの出力にラウンドキーを追加することで，入出力の数を一致させ，単射性を満たすようにアルゴリズムを記述し直す．次に，アルゴリズム中の演算が可逆であることを確かめ，各ステップの可逆性を示す．次に，Hermes で単射性・可逆性を満たした可逆プログラムが存在することを示す．実現されたプログラムにおいて効率性と可読性を考察し，考察をもとに必要となる言語機能の拡張を提案する．

2　関連研究

2.1　軽量暗号

軽量暗号 [2] とは, 使用できる資源が少ない IoT 機器などでも，実装可能かつ安全性の高い暗号のことを指す．軽量暗号の実装に関する要件として，メモリサイズの制約，消費電力量の制約，処理速度の制約などが挙げられる．これまでハードウェア実装やソフトウェア実装において，軽量性に優れた様々な暗号方式が発表されてきた．代表的な軽量暗号として，SPECK，SIMON，Midori などが挙げられる．

2.2　可逆プログラミング言語

可逆プログラミング言語とは，可逆計算を応用させた分野の一つであり，可逆な文しか記述することができないという制約を持つプログラミング言語のことである．多くの可逆プログラミング言語では，代入文を記述する際，左辺と右辺に同じ変数を記述できない特徴を持っている．同じ変数が両辺に現れた場合，非可逆な文となるからである．可逆プログラミング言語には Hermes や Janus が挙げられる．文献 [3] では，Janus を使って軽量暗号アルゴリズムの記述をしている．Janus のデメリットとして，タイミング攻撃に対して強固でないと言われている．Janus は変数などの値によって処理に必要な時間が変化するため，入力値ごとの処理時間の差異が原因で秘密鍵が盗まれる恐れがある．Hermes は，Janus の弱点であるタイミング攻撃に対して強固であるプログラミング言語である．文献 [1] では，配列の添え字に秘密情報を使う暗号アルゴリズムを記述するために，配列要素に代入する変数の型が secret 型でも記述できるように，`unsafe` というキーワードが追加することによって言語機能を拡張し，軽量暗号アルゴリズムの記述可能性を広げた．

3　軽量暗号アルゴリズムの単射化・可逆化

非単射な軽量暗号アルゴリズムを，出力にラウンドキーを追加することで単射化する．単射とは，任意の出力がそれぞれ唯一の入力から得られる演算のことをいう．さらに，各ステップを可逆更新のみで表すことで単射な計算を実現する可逆プログラムが存在することを示し，Hermes プログラムを実現する．ここで，可逆更新とは，入力を 2 つの変数 x，y とし，出力を，x と $f(y)$ を第 1 引数について単射な二項演算 $\odot$ をした結果 $x \odot f(y)$ と元の入力 y

とする部分関数 g である．任意のこうした可逆更新に対して，出力である $x \odot f(y)$ と y を，入力として使用し，$x \odot f(y)$ と $f(y)$ を $\odot$ について逆である二項演算 $\oslash$ をした結果，元の入力 x，y を得ることができる．可逆更新を図 1 に示す．

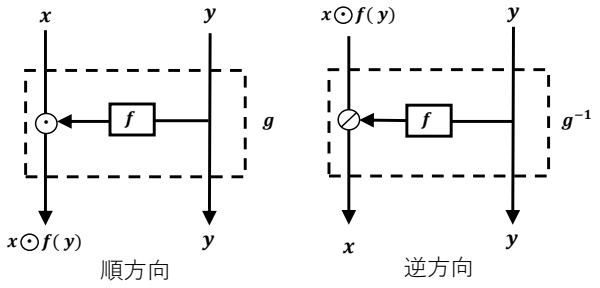

図 1 可逆更新 (文献 [4] の図を改変)

このように必ずしも単射ではない演算 $\odot$ 及び f から単射な演算 g を構成することを可逆化という．

可逆更新の例を図 2 に示す．入力を x, y_1, y_2 とし，出力を y_1 と y_2 に非単射な演算 $\otimes$ をした $y_1 \otimes y_2$ と，x との第 1 引数について単射な二項演算 $\odot$ をした結果 $x \odot (y_1 \otimes y_2)$ と，元の入力 y_1, y_2 とする．次に出力である $x \odot (y_1 \otimes y_2)$ と y_1，y_2 を入力として使用し，x と $y_1 \otimes y_2$ を $\odot$ について逆である二項演算 $\oslash$ をした結果，元の入力 x，y_1，y_2 を得ることができる．非単射な演算を導入しても，このように出力に入力の値を保持すれば，図 2 全体の演算 g は単射となる．

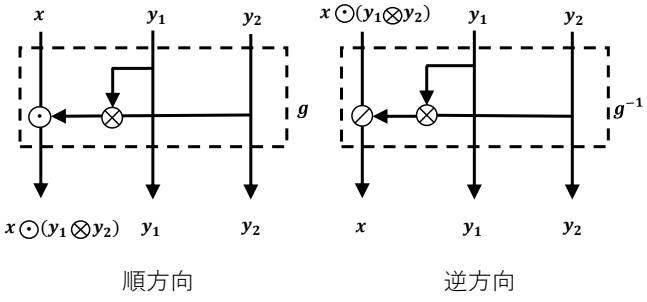

図 2 可逆更新の例

3.1 可逆なビット演算

多くの軽量暗号アルゴリズムで使われるビット演算には，NOT 演算，XOR 演算 $\oplus$，2^{32} を法とした剰余加算 $\boxplus$，j ビット左ローテート S^j がある．図 1 における $\odot$ を $\oplus$ や $\boxplus$，S^j と置き換えることで可逆更新となる．また，SIMON のアルゴリズムでは，2 つの入力を j ビット左ローテートさせたものを AND 演算している．図 1 において関数 $f(y) = S^1 y \ \& \ S^8 y$ とし $\odot$ を $\oplus$ に置き換えることで，この AND 演算を用いる演算も可逆更新となる．このように，多くの軽量暗号アルゴリズムで使われているビット演算は，可逆更新で実現できる．

3.2 アルゴリズムの単射化・可逆化

SIMON，LEA が表す計算が非単射であることを示し，単射化・可逆化を行う．

3.2.1 SIMON

図 3 における点線の矢印とその先の K を除いたものが SIMON のラウンド関数である．非単射な演算は灰色で示し，単射化した部分を点線で示す．水色の演算は，第 1 引数について単射な演算である．入力ビット数は x，y とラウンドキー K がそれぞれ 64 であるのに対して，出力ビット数は x' と y がそれぞれ 64 である．このように入力ビット数より出力ビット数の方が少ないので，このラウンド関数は単射ではない．

出力にラウンドキーを追加することで，単射化する．図 3 における破線を実線にした図を考える．入力は x, y とラウンドキー K であるのに対して，出力は x', y とラウンドキー K である．図 3 における破線を実線にしたラウンド関数は，図 1 における $\odot$ を第一引数について単射な二項演算 $\oplus$ にした 3 つの可逆更新からなる．単射関数の合成は単射であることから，単射な可逆更新を組み合わせたこのラウンド関数は単射である．また，このラウンド関数は単射であるため，規定のラウンド数分関数を繰り返しても，それらは可逆更新の組み合わせであるため単射である．

3.2.2 LEA

図 4 における点線の矢印とその先の $k_0 \sim k_5$ を除いたものが LEA のラウンド関数である．非単射な演算は灰色で示し，単射化した部分を点線で示す．水色の演算は，第 1 引数について単射な演算である．入力ビット数は a，b，c，d，ラウンドキー $k_0 \sim k_5$ がそれぞれ 32 であるのに対して，出力ビット数は a，b，c，d がそれぞれ 32 である．このように入力ビット数より出力ビット数の方が少ないので，このラウンド関数は単射ではない．

出力にラウンドキーを追加することで，単射化する．図 4 における破線を実線にした図を考える．入力は a，b，c，d，ラウンドキー $k_0 \sim k_5$ であるのに対して，出力は a'，b'，c'，d'，ラウンドキー $k_0 \sim k_5$ である．図 4 における破線を実線にしたラウンド関数は，図 1 における $\odot$ を第一引数について単射な二項演算 $\oplus$ と $\boxplus$ にした 6 つの可逆更新からなる．単射関数の合成は単射であることから，単射な可逆更新を組み合わせたこのラウンド関数は単射である．また，このラウンド関数は単射であるため，規定のラウンド数分関数を繰り返しても，それらは可逆更新の組み合わせであるため単射である．

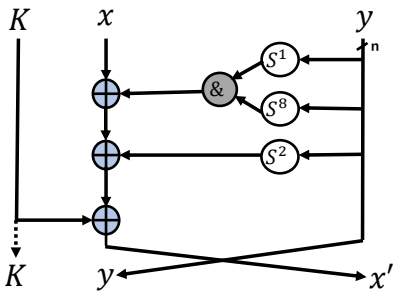

図 3 SIMON のラウンド関数

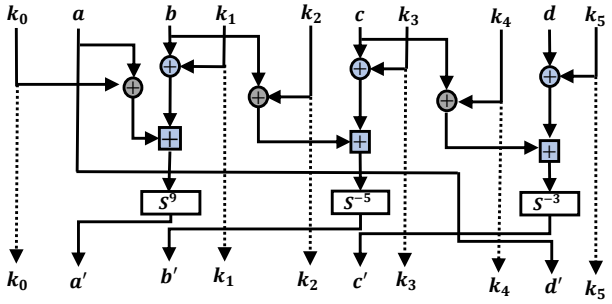

図 4 LEA のラウンド関数

プログラム 1 Hermes による SIMON の暗号化プログラム

```
1 enc_simon(u64 x, u64 y, u64 mk[68])
2 {
3   for (i = 0; 68) {
4     x ^= (y << 1 & y << 8) ^ (y << 2) ^ mk[i];
5     x <-> y;
6     i++;
7   }
8 }
```

プログラム 2 Hermes による LEA の暗号化プログラム

```
1 enc_lea(u32 a, u32 b, u32 c, u32 d, u32 mk[144])
2 {
3   for (i = 0; 24) {
4     d ^= mk[i * 6 + 5];
5     d += c ^ mk[i * 6 + 4];
6     d >>= 3;
7
8     c ^= mk[i * 6 + 3];
9     c += b ^ mk[i * 6 + 2];
10    c >>= 5;
11
12    b ^= mk[i * 6 + 1];
13    b += a ^ mk[i * 6 + 0];
14    b <<= 9;
15
16    a <-> b;
17    b <-> c;
18    c <-> d;
19    i++;
20  }
21 }
```

3.3 Hermes プログラムによる実現

実現した SIMON，LEA の Hermes プログラムを示す．プログラム中の << は左ローテート，>> は右ローテート，<-> は両辺の変数の値を交換する演算子を示している．また，Hermes プログラムでは，引数の渡し方は参照渡しである．したがって，プロシージャの実行後における計算結果は，返却値ではなく，参照が渡された変数に保持される．

3.3.1 SIMON

実現した SIMON の Hermes プログラムをプログラム 1 に示す．この手続きを呼び出す際に，適切な実引数として，符号なし 64 ビット整数型 x と y には暗号化をする平文を，mk[68] にはラウンドキーを渡す必要がある．プログラム 1 は，for 文内が暗号化のラウンド関数である．4 行目において，ローテート演算，AND 演算，XOR 演算の処理を 1 行にまとめた．x に対して，y をローテート演算し AND 演算したものと XOR 演算する．続いて，演算結果と y をローテート演算したものと XOR 演算する．続いて，演算結果とラウンドキー mk で XOR 演算する．これらは可逆更新である．図 3 の変数 x における各処理と，プログラム 1 の 4 行目の処理が対応している．これらの処理を for 文を用いて 68 回繰り返しているが，for 文内の演算はすべて可逆更新であり，プログラム 1 全体は可逆更新の組み合わせである．

3.3.2 LEA

実現した LEA の Herme プログラムをプログラム 2 に示す．この手続きを呼び出す際に，適切な実引数として，符号なし 32 ビット整数型 a から d までは暗号化をする平文を，mk[144] にはラウンドキーを渡す必要がある．プログラム 2 は，for 文内が暗号化のラウンド関数である．暗号化処理を行う際，変数 b は更新前の変数 a を使用し，変数 c は変数 b を使用し，変数 d は更新前の c を使用する必要がある．よって，可逆言語で LEA の暗号化プログラムを記述する際，変数 d，c，b，a の順に処理を行う必要がある．プログラム 4 行目で，d とラウンドキー mk との XOR 演算を行っており，プログラムの 5 行目で c と d の剰余加算を行い，プログラム 6 行目で d をビット数分ローテートさせる演算を行っている．これらの演算は可逆な演算である．図 4 の変数 d における各処理と，プログラム 2 の 4 行目から 6 行目までの処理が対応している．プログラム 2 の 8 行目から 14 行目の演算も，XOR 演算，2^{32} を法とした剰余加算，ローテート演算をしているため，それらは可逆な演算である．これらの処理を for 文を用いて 24 回繰り返しているが，for 文内の演算はすべて可逆更新であり，プログラム 2 全体は可逆更新の組み合わせである．

3.4 可読性・効率性の議論

単射化したラウンド関数の図とプログラムを照らし合わせた際に，図に表された計算がプログラム中の for 文内

の各行に対応しているため，理解がしやすいといえる．また，実引数として符号なし整数型の変数が渡され，`for` 文内の処理をそれらの変数を用いて行っているため，配列参照を行わずに済み，効率化が図られている．また，ラウンドキーも出力とみなしプログラムを実行することで，ゴミが発生することなくクリーン性がある．加えて，1 回の走査で処理が行えるため効率的である．

4 言語機能の拡張

言語機能の拡張をすることで，現状の Hermes よりもプログラミングしやすくなることを目指す．

4.1 定数配列

LEA や SIMON の鍵拡張処理において，定数配列を使っている．Hermes では定数型の配列の宣言は行えない．よって，実際に使用するために，言語機能を拡張することが求められる．定数配列の構文規則を図 5 に示す．

$$d \quad ::= \quad \cdots \mid \textbf{const}\ \ id\ \texttt{[}\ e\ \texttt{]}\ \texttt{=}\ \texttt{\{}\ c^{+}\ \texttt{\}}\ \texttt{;}\ \ d \qquad \text{(const array declation)}$$

図 5 定数配列の構文規則

4.2 ドット演算子

特定のメンバへアクセスする場合にドット演算子が使用される．ドット演算子の機能を加えるために，言語機能を拡張する．ドット演算子構文の構文規則を図 6 に示す．

$$e \quad ::= \quad \cdots \mid id\ .\ l \qquad \text{(dot expression)}$$

図 6 ドット演算の構文規則

4.3 共用体

共用体とは，複数の異なる型のメンバが 1 つのメモリ領域を共有する機能である．共用体によって 2 つのオブジェクトがメモリ領域を共有することで，扱うデータサイズを処理によって使い分けることで効率性の向上を目指す．共用体の構文規則を図 7 に示す．

$$\begin{array}{rcll} proc & ::= & \cdots \mid d\ \ proc & \text{(union procedure)} \\ d & ::= & \cdots \mid \texttt{UNION}\ \ id\ \texttt{\{} & \\ & & \quad t\quad l\ \texttt{;} & \\ & & \quad t\quad l\ \texttt{;} & \text{(union declation)} \\ & & \texttt{\}}\ \texttt{;} & \end{array}$$

図 7 共用体の構文規則

4.4 考察

定数型で宣言を行った配列は，定数として扱えるため，ゼロクリアする必要がない．そのため配列要素を代入する際，`call` 文と `uncall` 文の記述は不要であり，ソースコードの行数を削減できると考えられる．結果として，より簡潔なプログラムが作成可能になり，可読性が向上すると考察する．共用体を使えば複数のメンバがメモリ領域の共有が可能になる．SPN 構造の軽量ブロック暗号では，平文を 4×4 の行列形式で表し，行列の 1 要素ごとに処理を行っている．共用体の宣言において，8 ビット整数型の配列と，32 ビット整数型の配列を宣言し，各処理ごとに利用する配列を使い分ければ，現状の Hermes よりも効率的になると考察する．

5 おわりに

非単射な SIMON・LEA の暗号アルゴリズムを単射化し，その全ステップが可逆更新の組み合わせであることを明示した．次に，暗号アルゴリズムを自然な 1 パスのクリーンな Hermes プログラムで実現し，Hermes が一定の記述力を持つことを示した．また，Hermes プログラムの可読性と効率性を高めるために，定数配列，UNION，ドット演算に相当する言語機能の拡張を行った．

今後の課題として，非単射である SIMON・LEA の鍵拡張アルゴリズムの単射化及び全ステップの可逆化を示すことが挙げられる．また，Hermes を用いて主要な軽量暗号アルゴリズムを可読性の高い，効率的で可逆なプログラムで実現することができたが，提案した言語機能を用いたプログラムの実現ができておらず，実現したプログラムが効率性と可読性を満たしているか考察することが今後の課題とされる．

参考文献

[1] Mogensen, T.Æ.: Hermes: A reversible language for lightweight encryption, *Science of Computer Programming*, Vol.215, p.102746 (2022).

[2] CRYPTREC 軽量暗号ワーキンググループ：CRYPTREC 暗号技術ガイドライン（軽量暗号）(2017).

[3] Táborský, D., Larsen, K.F. and Thomsen, M.K.: Encryption and Reversible Computations - Work-in-progress Paper, *Reversible Computation - 10th International Conference, RC 2018, Leicester, UK, September 12-14, 2018, Proceedings* (Kari, J. and Ulidowski, I., Eds.), Lecture Notes in Computer Science, Vol.11106, Springer, pp.331–338 (2018).

[4] Axelsen, H.B., Glück, R. and Yokoyama, T.: Reversible Machine Code and Its Abstract Processor Architecture, *Computer Science - Theory and Applications, Second International Symposium on Computer Science in Russia, CSR 2007, Ekaterinburg, Russia, September 3-7, 2007, Proceedings* (Diekert, V., Volkov, M.V. and Voronkov, A., Eds.), Lecture Notes in Computer Science, Vol.4649, Springer, pp.56–69 (2007).