Vol.2010-DPS-142 No.56
Vol.2010-CSEC-48 No.56
2010/3/5

ハッシュ関数 Lesamnta-LW の実装及び評価

大和田 徹[†]　井手口 恒太[†]　吉田 博隆[†]

暗号学的なハッシュ関数は，認証やデジタル署名等様々な用途で利用されている重要な暗号プリミティブである．SHA-1 の危殆化に伴い，次世代ハッシュ関数の実用化が望まれているが，特に安心安全をターゲットにした情報システムにおいては，情報の完全性保証が極めて重要であり，次世代ハッシュ関数の重要性は更に増大していくと考えられる．これに対し，Lesamnta（レザンタ）-LW [1][1]は，十分な安全性を持ち，かつ，ハードウェアでの軽量実装性を目指して設計された新たなハッシュ関数である．本稿では，Lesamnta-LWのハードウェア，及びソフトウェア実装評価について報告する．ハードウェア実装評価については，コスト最適化実装を行ない，ソフトウェア実装評価については，Intel Core2 Duo [2]，及び，“Westmere”を対象CPUとした速度最適化実装を行ない，各々，SHA-256 の実装結果と比較した．

Implementation and Evaluation of a Hash Function Lesamnta-LW

Toru Owada[†]　Kota Ideguchi[†]　and Hirotaka Yoshida[†]

Cryptographic hash functions are important cryptographic primitives which are used in various applications such as authentication and signature schemes. Given the current situation where the security of SHA-1 is undermined, it is desired that the next-generation hash functions are in use. Especially, confidentially and authentication of information in security systems are very important hence the next-generation hash functions are increasingly important. Lesamnta-LW[1][1] is a hash function which is designed to achieve high security as well as lightweight hardware implementations. In this report, we show our results on hardware and software implementations of Lesamnta-LW and then compare them with our results on SHA-256. In hardware, we show the result on an optimized implementation of Lesamnta-LW in terms of cost. In software, we show the results on optimized implementations of Lesamnta-LW on Intel Core2 Duo [2] and “Westmere”.

1. はじめに

昨今，極普通の日常生活においても，地デジ対応テレビ，携帯電話，音楽プレイヤなどを介し，何らかのデジタル情報を利用することは当然となっており，又，通信，蓄積を問わずデジタル情報の活用範囲増大に伴い，デジタル情報の安全な利活用を担保する情報セキュリティ技術の重要性がますます増大している．

その中で，暗号学的なハッシュ関数は，認証やデジタル署名等様々な用途で利用されている重要な暗号プリミティブである．しかしながら，SHA-1 の危殆化に伴い，ハッシュ関数の信頼性基盤が揺らいでおり，安全性および効率性を兼ね備えた次世代ハッシュ関数の実用化が望まれている．これを受け，米国の国立標準技術研究所（National Institute of Standards and Technology, NIST）では，次世代の汎用ハッシュ関数 SHA-3（Secure Hash Algorithm 3）の選定中である．

又，情報セキュリティ技術のうち，今後の進展が期待される，安心安全をターゲットにした情報システムにおいては，情報の完全性保証が極めて重要である．特に，センサネットなど端末ノードから発信される情報の信頼性に依拠するユビキタス情報システムにおいて，その重要性は更に増大していく．センサネットにおいては，センサネット端末等，演算能力の限定されたネットワークノードと，上記ノードからの情報を集約するサーバなどのネットワークノードの存在が仮定されるケースがあり，そのようなネットワークでの通信において用いられるハッシュ関数には，端末ノードにおける軽量実装性と，サーバにおける実用上問題のない処理性能の両立が求められる．

上記背景に対し，Lesamnta-LW は，十分な安全性を保ちつつも，ハードウェア実装での軽量実装性と，PC 等での良好なソフトウェア処理性能の両立を目指して設計された新たなハッシュ関数である．

本稿では，各種システムへの組込みを念頭に，Lesamnta-LW のハードウェア，及びソフトウェア実装評価について報告する．ハードウェア実装評価については，先に述べた軽量実装性評価の観点から，コスト最適化を重視したハードウェア実装アーキテクチャと，その実装評価について述べる．ソフトウェア実装評価については，一般的な PC 環境として，32 ビットおよび 64 ビット CPU である Intel Core2 Duo を対象 CPU とし，アセンブリ言語による速度最適化を優先した実装評価について述べる．又，ハードウェア，ソフトウェア実装の双方において，SHA-256 を対照とした実装性能の比較を行なった．

尚，Intel の次世代 CPU である“Westmere”には，AES を高速に処理する専用命令

[†] 株式会社 日立製作所 システム開発研究所
〒244-0817 横浜市戸塚区吉田町 292
Systems Development Laboratory, Hitachi, Ltd.
292 Yoshida-cho, Totsuka-ku, Yokohama, Kanagawa 244-0817, Japan

Vol.2010-DPS-142 No.56
Vol.2010-CSEC-48 No.56
2010/3/5

である AES-NI 命令セットが追加実装されることが公開されており[2], 同 CPU 上では AES のソフトウェア処理速度が従来 CPU と比較して格段に向上することが見込まれている．ここで，AES-NI 命令は，AES のみならず，AES の非線形変換部，線形変換部を，安全性評価の確立した既存部品として採用しているハッシュ関数の処理速度向上にも寄与するとの報告がある[3][4]．Lesamnta-LW のブロック暗号部も AES の非線形変換部，線形変換部を採用していることから，本稿では，AES-NI 命令を用いた Lesamnta-LW 処理の高速化についても述べる．

先ず，2 章にて，Lesamnta-LW の仕様を記載する．その後，3 章にて，ハードウェア実装結果，4 章にて，ソフトウェア実装結果を示す．

2. Lesamnta-LW の仕様

本章では, ハッシュ関数アルゴリズム Lesamnta-LW の仕様を記載する. Lesamnta-LW は，l ビットの長さからなるメッセージから，256 ビットのメッセージダイジェストを計算するアルゴリズムである．Lesamnta-LW では，32 ビットを 1 ワードと定義する．

2.1 メッセージパディング

メッセージパディングでは, l ビットの入力メッセージに対してパディングを行い，パディング後のメッセージ長を 128 ビットの倍数にする（最小 256 ビット）．
以下にパディング処理を 3 ステップに分けて説明する．

(1) 入力メッセージの最後に 1 個の“1”を付加する．
(2) k+63 個の“0”を付加する．ここで k は $l + k \equiv 0 \pmod{128}$ を満たす非負整数とする．
(3) 最後の 64 ビットに l を 2 進表現した値を付加する．

例えば ASCII 文字列“abc”の 3 文字を入力した場合，パディング後のメッセージ M は次のようになる．
入力メッセージ =“abc”, l = 24, M = 61626380 00…00 00000018（256 ビット）．

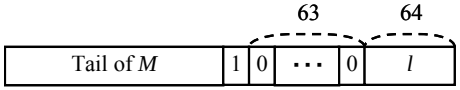

図 1 最終 2 ブロックのメッセージパディング（$l \equiv 0 \pmod{128}$）

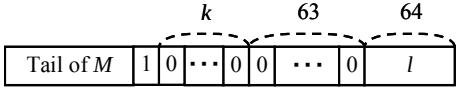

図 2 最終 2 ブロックのメッセージパディング（$l \neq 0 \pmod{128}$）

2.2 メッセージダイジェストの計算

メッセージダイジェストの計算処理は，圧縮関数 h により構成される．本処理は初期ベクトルと N ブロックからなるパディング後のメッセージ M とし, 次の手順でメッセージダイジェストを計算する．

(1) 入力メッセージをパディング処理する．
(2) パディングしたメッセージをN個の 128 ビットのメッセージブロック$M^{(1)}$, $M^{(2)}$,…, $M^{(N)}$に分割する．
(3) 変数$H^{(0)}$に初期ベクトルを設定する．

そして，次に示す疑似コードに従い$M^{(1)}, M^{(2)},\ldots, M^{(N)}$の計算を行う．

```
for i = 1 to N
        H^(i) = h(H^(i-1), M^(i))
end for
```

$M^{(N)}$の処理が完了すると, メッセージダイジェストは以下の 8 ワードで表される 256 ビットの文字列として得られる．

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}.$$

2.3 圧縮関数 h

圧縮関数hは中間値$H^{(i-1)}$とメッセージブロック$M^{(i)}$を暗号化関数Eにより処理することで$H^{(i)}$を生成する関数である．圧縮関数hは以下で定義される．

$$H^{(i-1)} = H_0^{(i-1)} \| H_1^{(i-1)},$$
$$h(H^{(i-1)}, M^{(i)}) = E(H_0^{(i-1)}, M^{(i)} \| H_1^{(i-1)}).$$

暗号化関数Eの入力値H_0, H_1, Mはそれぞれ 128 ビットのデータである．$H_0^{(i-1)}$は 128 ビットの暗号鍵として処理される．このような構成をLW1 モードと呼ぶ[5].

2.4 ブロック暗号

ブロック暗号の暗号化関数Eは圧縮関数hで使用され，128 ビットの暗号鍵と 256 ビットのメッセージを入力とし，256 ビットの暗号文を出力する関数である．暗号化関数Eは，ラウンド定数C，鍵スケジュール関数のラウンド関数f_K，および主攪拌関数のラウンド関数f_Mから構成される．ラウンド数は 64 である．
ラウンドrにおける主攪拌関数と鍵スケジュール関数の入力値を（$x_0^{(r)}, x_1^{(r)}, \cdots, x_7^{(r)}$），（$k_0^{(r)}, k_1^{(r)}, \cdots, k_3^{(r)}$）とする．$x_i^{(r)}$, $k_i^{(r)}$は 32 ビットワードとする．

Vol.2010-DPS-142 No.56
Vol.2010-CSEC-48 No.56
2010/3/5

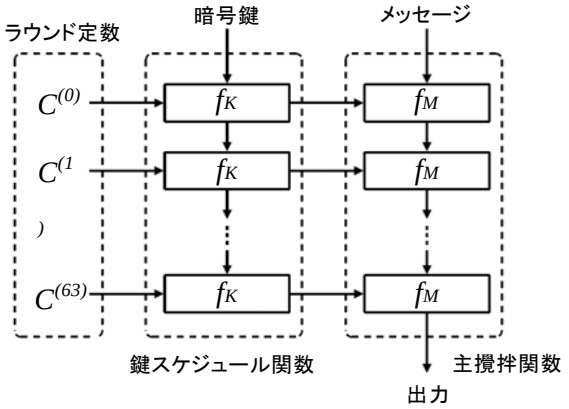

図 3　暗号化関数 E の構造

2.4.1 主撹拌関数

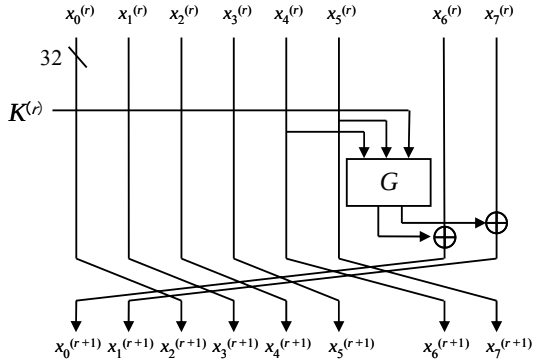

図 4　主撹拌関数のラウンド関数f_M

主撹拌関数のラウンド関数f_Mは，排他的論理和，非線形変換のG関数，およびワード単位の置換により構成される．G関数の入力はラウンド関数の入力の$x_4^{(r)}$，$x_5^{(r)}$と鍵スケジュール関数で生成される 1 ワードのラウンド鍵$K^{(r)}$であり，その出力は$x_6^{(r)}$，$x_7^{(r)}$に排他的論理和される．f_Mは以下で定義される．

$$x_0^{(r+1)} \| x_1^{(r+1)} = (x_6^{(r)} \| x_7^{(r)}) \oplus G(K^{(r)}, (x_4^{(r)} \| x_5^{(r)})),$$
$$x_2^{(r+1)} = x_0^{(r)}, \quad x_3^{(r+1)} = x_1^{(r)}, \quad x_4^{(r+1)} = x_2^{(r)}, x_5^{(r+1)} = x_3^{(r)}, \quad x_6^{(r+1)} = x_4^{(r)}, \quad x_7^{(r+1)} = x_5^{(r)}.$$

G 関数は 96 ビット入力 64 ビット出力の非線形変換を行う関数であり，AddRoundKey()，SubBytes()，MixColumns()，ByteTranspos()から構成される．

G ＝ ByteTranspos()○ MixColumns()○ SubBytes()○ AddRoundKey().

AddRoundKey()，SubBytes()，MixColumns()，ByteTranspos()の各処理において，変換対象となる 8 バイトを（s_0 , s_1, …, s_7）とする．以下に各変換処理の詳細を記載する．

AddRoundKey()では 32 ビットのラウンド鍵をs_0 , s_1, s_2, s_3に排他的論理和する．

$[s'_0, s'_1, s'_2, s'_3] = [s_0, s_1, s_2, s_3] \oplus K^{(r)}$,　$[s'_4, s'_5, s'_6, s'_7] = [s_4, s_5, s_6, s_7]$.

SubBytes()は，非線形のバイト置換であり，AES の置換テーブルである S-box によりバイト置換する．

s'_i = S-box(s_i), for $0 \le i < 8$.

MixColumns()は，有限体GF(2^8)上の行列を用いた線形変換を行う．

$$\begin{bmatrix} s'_i \\ s'_{i+1} \\ s'_{i+2} \\ s'_{i+3} \end{bmatrix} = \begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \begin{bmatrix} s_i \\ s_{i+1} \\ s_{i+2} \\ s_{i+3} \end{bmatrix}, \quad \text{for } i = 0, 4 .$$

$s'_i = (\{02\} \cdot s_i) \oplus (\{03\} \cdot s_{i+1}) \oplus s_{i+2} \oplus s_{i+3}$,
$s'_{i+1} = s_i \oplus (\{02\} \cdot s_{i+1}) \oplus (\{03\} \cdot s_{i+2}) \oplus s_{i+3}$,
$s'_{i+2} = s_i \oplus s_{i+1} \oplus (\{02\} \cdot s_{i+2}) \oplus (\{03\} \cdot s_{i+3})$,
$s'_{i+3} = (\{03\} \cdot s_i) \oplus s_{i+1} \oplus s_{i+2} \oplus (\{02\} \cdot s_{i+3})$, for $i = 0, 4$.

ByteTranspos()は，バイト単位の転置を行う．

$s'_0 = s_4$, $s'_1 = s_5$, $s'_2 = s_2$, $s'_3 = s_3$, $s'_4 = s_0$, $s'_5 = s_1$, $s'_6 = s_6$, $s'_7 = s_7$.

2.4.2 鍵スケジュール関数

鍵スケジュール関数のラウンド関数f_Kでは初めにラウンド鍵生成処理を行う．rラウンド目のラウンド鍵$K^{(r)}$を以下のように生成する．

$K^{(r)} = k_0^{(r)}$.

次に鍵スケジュール関数の内部状態を以下のように更新する．

$k_0^{(r+1)} = k_3^{(r)} \oplus P(C^{(r)}, k_2^{(r)})$, $k_1^{(r+1)} = k_0^{(r)}$, $k_2^{(r+1)} = k_1^{(r)}$, $k_3^{(r+1)} = k_2^{(r)}$.

P関数は 64 ビット入力 32 ビット出力の非線形変換を行う関数であり，ラウンド定数（2.4.4節参照）とラウンド関数f_Kの入力である$k_2^{(r)}$を引数とする．

P ＝ KeyLinear() ○ SubWords()○ AddRoundConstant().

Vol.2010-DPS-142 No.56
Vol.2010-CSEC-48 No.56
2010/3/5

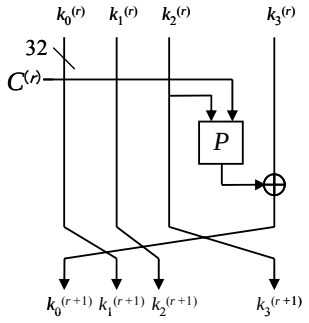

図 5　鍵スケジュール関数のラウンド関数f_K

AddRoundConstant(), SubWords(), KeyLinear() において変換対象となる 4 バイトを（a_0, a_1, a_2, a_3）とする．以下に各変換処理の詳細を記載する．

AddRoundConstant()では 32 ビットのラウンド定数をa_0 , a_1, a_2, a_3に排他的論理和する．

$[a'_0, a'_1, a'_2, a'_3] = [a_0, a_1, a_2, a_3] \oplus C^{(r)}$.

SubWords()は，非線形のバイト置換であり，AES の置換テーブルである S-box によりバイト置換する．

a'_i = S-box(a_i), for $0 \le i < 4$.

KeyLinear()は，有限体GF(2^8)上の行列を用いた線形変換を行う．

$$\begin{bmatrix} a'_0 \\ a'_1 \\ a'_2 \\ a'_3 \end{bmatrix} = \begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}.$$

$a'_0 = (\{02\}\cdot a_0) \oplus (\{03\}\cdot a_1) \oplus a_2 \oplus a_3$, $a'_1 = a_0 \oplus (\{02\}\cdot a_1) \oplus (\{03\}\cdot a_2) \oplus a_3$,
$a'_2 = a_0 \oplus a_1 \oplus (\{02\}\cdot a_2) \oplus (\{03\}\cdot a_3)$, $a'_3 = (\{03\}\cdot a_0) \oplus a_1 \oplus a_2 \oplus (\{02\}\cdot a_3)$.

2.4.3 初期値

Lesamnta-LWの初期値$H^{(0)}$は以下に示す 8 個の 32 ビットワードを使用する．尚，初期値は 16 進表現である．

$H_0^{(0)}$ = 00000256,　$H_1^{(0)}$ = 00000256,　$H_2^{(0)}$ = 00000256,　$H_3^{(0)}$ = 00000256,
$H_4^{(0)}$ = 00000256,　$H_5^{(0)}$ = 00000256,　$H_6^{(0)}$ = 00000256,　$H_7^{(0)}$ = 00000256.

2.4.4 ラウンド定数

ラウンド定数$C^{(0)}, C^{(1)}, \ldots, C^{(63)}$は以下の 64 個の 32 ビットワードとなる．尚，ラウンド定数は 16 進表現である．

a432337f, 945e1f8f, 92539a11, 24b90062, 6971c64c, d6e3f449, 2c2f0da9, 33769295,
eb506df2, 708cebfe, b83ab7bf, 97df0f17, 9223b802, 7fa29140, 0ff45228, 01fe8a45,
ed016ee8, 1da02ddd, ee8aba1b, 46c4c223, 53cd0d24, d1b46d24, c1fb4124, c3f2a4a4,
c3b39814, c3bbbf82, 759191b0, 0eb23236, b7fd6c86, a0d48750, 141a90ea, 6f65b45d,
e0d2092b, 470fd445, e5df4528, 1cbbe8a5, eea9c2b4, c618f4d6, aee8345a, 783be0cb,
5412e979, 3c712e0f, 87567c21, 2619bca4, df0efb14, c02c13e2, 75e3643c, d571a007,
9a766de0, 134ecdbc, d9a41537, 9becdb46, a556b1a8, 14aad635, efabe566, abde566c,
ceb6064d, f4e87f69, 286e7ccd, e8337039, 2bf51d27, 85a6fa44, cb7913c8, 196f2279.

ラウンド定数を生成するための擬似コードを図 6に示す. ラウンド定数はLFSRをもとに生成されており，その定義多項式$g(x)$は次式で与えられる．

$g(x) = x^{32} + x^{31} + x^{29} + x^{28} + x^{26} + x^{25} + x^{24} + x^{23} + x^{20} + x^{19} + x^{17} + x^{16} + x^{15} + x^{12} + x^{11} + x^8 + 1$.

```
ConstantGenerator(word C[64])
begin
  word c;
  c = ffffffff;   /* in hexadecimal */
  for i = 0 to (64 * 3) - 1
    /* Galois-type LFSR */
    if c ∧ 00000001 = 00000001
      c = (c >> 1) ⊕ dbcdcc80;
    else
      c = c >> 1;
    end if
    if i mod 3 = 0
      C[i/3] = c;
    end if
  end for
end
```

図 6 ラウンド定数生成擬似コード

Vol.2010-DPS-142 No.56
Vol.2010-CSEC-48 No.56
2010/3/5

3. Lesamnta-LW のハードウェア実装

本章では，Lesamnta-LW のハードウェア実装について述べる．先ず，Lesamnta-LW のアルゴリズムをハードウェア実装の観点から分析，検討する．その結果に基づき，特長を活かしたハードウェア実装アーキテクチャと，その実装結果を示す．

3.1 ハードウェア実装観点からの分析，検討

Lesamnta-LW は，鍵長 128bit，ブロック長 256bit のブロック暗号 E をベースに LW1 モードで圧縮関数 h を形成した構造となっている． LW1 モードは，MMO モード[6]のようなメッセージのフィードフォワードを行なわないことが特徴の一つであり，LW1 モードのハードウェア実装時には，MMO モードにおけるメッセージ格納用のレジスタや中間ハッシュ値生成用の bit EOR 等のモード処理用の付加回路を必要としない．ブロック暗号 E のラウンド関数のハードウェアに対し，初期値，中間値の選択用のセレクタの付加で LW1 モードを構成可能であることから，Lesamnta-LW の構造は，低いハードウェアコストで構成可能であると言える．

圧縮関数 h は 64 段のブロック暗号 E で構成されるが，論理規模と動作周波数のバランスの観点からブロック暗号 E の 64 回のループ処理にて構成するのが基本構成となる．ブロック暗号 E のラウンド関数のハードウェア化には，鍵レジスタ 128bit，ブロック暗号レジスタ 256bit の計 384bit のレジスタが必要である．鍵レジスタ長がブロック暗号レジスタの 1/2 であることはハードウェアコスト上有利であると言える．

ブロック暗号Eを構成する，鍵スケジュール関数のラウンド関数f_K，主攪拌関数のラウンド関数f_Mは，ビット幅は異なるものの，共に一般化Feistelネットワーク構造（GFN）を採っている．P関数，G関数による処理対象は入力値のうちの 1/4 だけであり，残り 3/4 はワード置換の，疎なネットワーク構造である．

これは，P関数，G関数に由来するクリティカルパス（ハードウェアの最大処理周波数の決定要因であるレジスタ間の組み合わせ論理回路の遅延が最大であるパス：最大遅延パス）が処理ビット幅の一部に限定的に現れることを示す．この特徴から，ラウンド関数f_K，f_Mは，最大 3 段まで直列接続したとしても，クリティカルパスが直列に接続せず，最大遅延の増大が発生しない．

これは，処理遅延量の増加なしに複数段のラウンド関数の並列処理が可能であることを示し，ハードウェアコストの増大を許容できるのであれば，性能向上に向けて有利なファクタとして作用する．

P 関数は，鍵加算，S-box による非線形変換，行列演算による線形変換によって構成され，G 関数は，概略，2 個の P 関数を並置し，その出力の一部をバイトスワップで接続した構成となっている．P 関数，G 関数はほぼ同一遅延量のクリティカルパスを持ち，主攪拌関数，鍵スケジュール関数を並列実装する際に，一方が動作周波数向上のボトルネックとなることのない，遅延バランスのよい構成と言える．

又，P 関数，G 関数は安全性評価と実装技術の確立した既存の部品，具体的には AES[7]の S-box，行列演算部を採用している．確立した実装技術[8][9]を用いることができるのは，実用性の観点からは大きなメリットである．

G 関数が，2 個の P 関数を並置した構成となっていることから，P 関数に相当するモジュールを 1 個だけ実装し，3 回ループで G 関数，P 関数処理を行なう構成が可能である．但し，論理規模削減のペナルティとして，必要な処理クロック数は 3 倍となり，クリティカルパスの遅延量は経路セレクタの挿入分大きくなる．

3.2 実装アーキテクチャ

Lesamnta-LWの基本的なハードウェア構成を図 7に示す．鍵スケジュール関数処理部として，鍵スケジュール関数のラウンド関数f_Kを 1 段分処理可能なモジュールと，主攪拌関数処理部として，主攪拌関数のラウンド関数f_Mを 1 段分処理可能なモジュールとを備える構成である．処理に必要なサイクル数は 64 である．

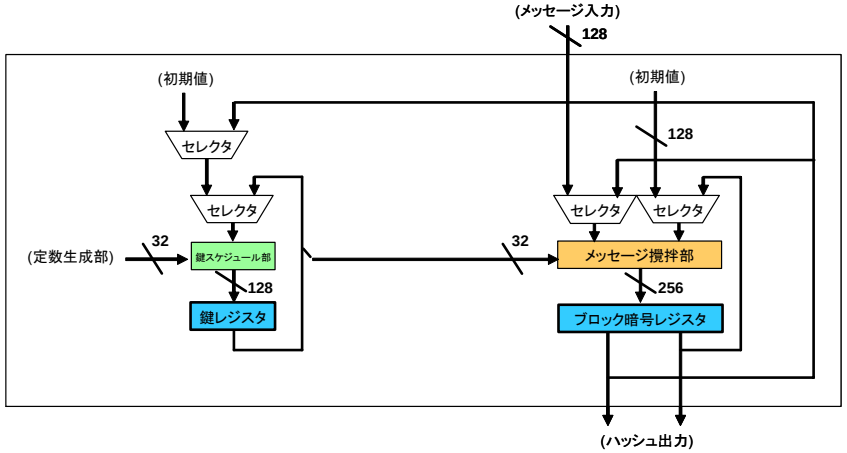

図 7　Lesamnta-LW のハードウェア構成 (基本型)

3.1 節の検討から，論理規模削減にはG関数，P関数の共有構成が有効であることが分かっている． G関数，P関数をループ処理とすることで，鍵スケジュール関数のラウンド関数f_K相当のサイズで，同関数に加え，主攪拌関数のラウンド関数f_Mの処理も可能とするもので，経路セレクタ分の論理追加はあるものの，関数処理部本体の論理

Vol.2010-DPS-142 No.56
Vol.2010-CSEC-48 No.56
2010/3/5

量を 1/3 に縮減した構成となる．処理サイクルは基本型比 3 倍の 192 となる．規模優先型のハードウェア構成を図 8 に示す．尚，詳細なセレクタ構成は省略してある．

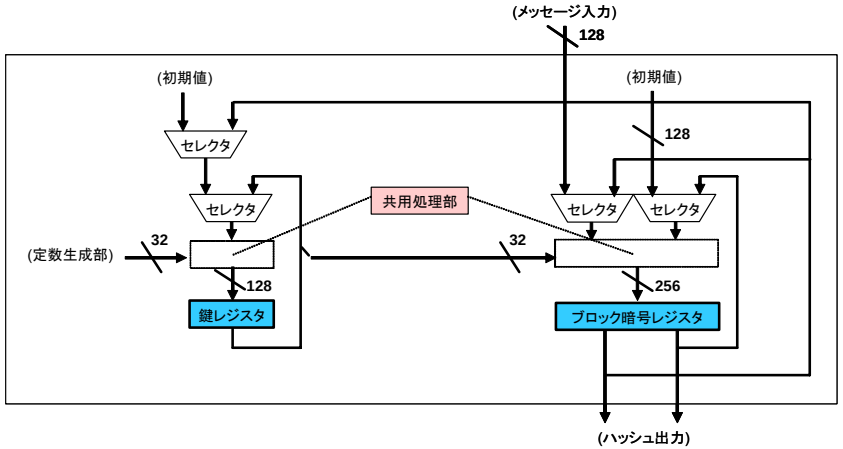

図 8　Lesamnta-LW のハードウェア構成 (規模優先型)

3.3 実装結果

ハードウェアでの軽量実装性評価を念頭に，3.2節で示した 2 種類のLesamnta-LW実装アーキテクチャの実装評価を行なった．90nmCMOSスタンダードセルライブラリを用いた際の同評価結果を表 1に示す．論理合成時の合成条件は，論理規模優先の設定とした．論理規模最小化を優先し，S-boxは合成体ベースの記述とした．又，比較対照として，SHA-256 に対する実装結果を示す．SHA-256 は論理規模優先の実装とし，実装コストの高い 32bit算術加算器はリップルキャリー演算器を用いることでサイズを縮減し，又，算術加算器の一部共有により論理量削減を図る構成を採った．

比較結果から，Lesamnta-LW は，SHA-256 比 42%減の論理規模での実装が可能であると言える．但し，性能は同 92%減となり，性能コスト上のトレードオフ関係にある．

表 1　ハードウェア実装結果

アルゴリズム	実装アーキテクチャ	論理規模 (k gates)	動作周波数 (MHz)	処理スループット (Mbps)
Lesamnta-LW	基本型	12.3	177.6	355.24
	規模優先	8.2	188.3	125.55
SHA-256	規模優先	14.2	220.8	1662.4

90nm CMOS スタンダードセル

4. Lesamnta-LW のソフトウェア実装

本章では，Lesamnta-LW のソフトウェア実装について述べる．PC 等でのソフトウェア処理性能評価を念頭に，32 ビット，64 ビットプロセッサである Intel Core2Duo E6600 を対象プロセッサとして選択した．又，Intel の次世代 CPU “Westmere”に搭載される AES 専用命令 AES-NI 命令を用いた場合のソフトウェア実装見積もりを行なった．言語は，アセンブリ言語[10]を用い，処理速度を重視した実装とした．処理性能測定には，CPU のサイクルカウンタ（RDTSC 命令）を用いた．以下，各々についての実装手法及び，実装結果を示す．

4.1 Core2Duo 上の実装

Lesamnta-LW のソフトウェア実装に当たって処理性能，及びメモリ容量に最も影響を与えるのは，P 関数，G 関数の実装手法であり，以下の手法で，処理高速化を行なった．Lesamnta-LW では，各々64 回の P 関数，G 関数処理を行なうが，コードキャッシュ容量を超えない範囲で，複数回をアンロールした実装とした．P 関数，G 関数には，AES と同一の S-box，行列演算中のガロア体上の乗算が含まれるが，これを，AES 向けの高速実装手法として知られる，S-box 層と線形変換層の合成関数を表参照で実装する手法を採用した．G 関数が，実質的には並置した P 関数であるため，両関数で用いる合成関数の参照表は共有可能であり，1 バイト入力 4 バイト出力の参照表(1 枚当たり 1 キロバイト)4 枚が必要数となる．

又，3 章で示したとおり，ラウンド関数f_K，f_Mは，最大 3 段までの並列処理が可能である．この特徴を利用し，独立な命令発行を期待したコード記述を行なうことでCPUの有する複数のALUの並列稼動率向上による性能向上を図ることができる．

4.2 Core2Duo 上の AES-NI 命令を用いた実装

AES-NI 命令セットは，AES を高速に処理するための専用命令セットで，AES の 1 ラウンド分の暗号処理を行なう AESENC や AES の最終ラウンドの暗号処理を行なう

Vol.2010-DPS-142 No.56
Vol.2010-CSEC-48 No.56
2010/3/5

AESENCLAST 等，複数の命令で構成されている．このうち，Lesamnta-LW の処理に活用可能なのは，AESENC 命令である．

AESENC 命令は，本来，各々128 ビットの処理中間値と中間鍵を入力とし，AES の 1 ラウンド分の処理を行なう命令であるが，入力値の選択と，AESENC 命令前後のバイト置換で, Lesamnta-LW の *P* 関数, *G* 関数の処理高速化に用いることが可能である．又, 上記データ置換には既存の SSE2 命令を活用することで高速な処理が可能である．

Lesamnta-LWの*G*関数では，SubBytes，MixColumnsの 2 つをAESENC命令で処理可能である．*G*関数処理のうち，AddRoundKey処理を行なった 64 ビットデータを，$x = x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7$とし，SSEレジスタ上に，下記左辺に示すフォーマットでxを配置する．

$$\begin{pmatrix} x_0 & x_4 & 0 & 0 \\ 0 & x_1 & x_5 & 0 \\ 0 & 0 & x_2 & x_6 \\ x_7 & 0 & 0 & x_3 \end{pmatrix} \rightarrow(\text{AESENC})\rightarrow \begin{pmatrix} y_0 & y_4 & * & * \\ y_1 & y_5 & * & * \\ y_2 & y_6 & * & * \\ y_3 & y_7 & * & * \end{pmatrix}$$

この入力データに対し，中間鍵入力を 0 として AESENC 命令を実行すると，その出力は上記右辺となる．y は x に対し，SubBytes と MixColumns を施した出力となる．* は無効データである．上記 y に対し，ByteTranspos 処理を行なうことで，所望の *G* 関数処理となる．同様の手法で *P* 関数の処理も可能である．

AESENC命令が 128 ビットデータを対象とした処理であるのに対し，Lesamnta-LW の*P*関数は 32 ビット幅, *G*関数は 64 ビット幅の関数であり, SSEレジスタを各々, 1/4, 1/2 だけ活用することとなる．ここで，AESのラウンド関数は，MixColumnsにおける一つの行列演算に入力される 4 バイトにのみ依存関係があり．MixColumnsの 4 つの行列演算の間には依存関係がない．この関係を利用し，複数の処理対象データを並置して入力することで，一つのAESENC命令で複数の*P*関数，*G*関数を処理することが可能である．ラウンド関数f_K，f_Mが，最大 3 段まで並列処理可能であることと合わせ，依存関係のないデータを並置することで，*P*関数，*G*関数を処理するためのAESENC命令数を削減することが可能である．

尚，同命令を用いたコードの性能見積もりには，以下の手法を用いた．

(1) シミュレータ[11]を用い，作成コードの動作確認を行なう．
(2) 作成コード上の AESENC 命令を，同命令と同一の処理スループットとレイテンシを持つ既存命令に置換したコードを作成し，同コードの，Core2Duo 実機上での速度測定値を見積もり値とする．

尚，置換に妥当な既存命令を，[12]では PMULUDQ，[3]では（MOVDQU, MULPS, MULPS, XORPS）としており，本稿では両者による見積もり値を算出している．

4.3 実装結果

表 2，表 3 に，Core2Duo 上での実装結果，及び，Core2Duo 上における AES-NI 命令適用時の性能見積もり結果と，比較対照としての SHA-256 の実装数値を示す．PC における潤沢なメモリ環境を鑑み，性能数値のみの比較としている．

Lesamnta-LW は，*P* 関数，*G* 関数における合成関数など，32 ビット幅演算に適した処理が多く，64 ビット幅の処理により高速化されるのは，GFN による 2 ワード置換など一部演算に限られる．これにより，32 ビットモード，64 ビットモード間での性能差は大きくない．又，32 ビットモードにおいて，AES-NI 命令を適用することで，約 40% の性能向上が見込まれることが分かった．

SHA-256 と比較においては，Lesamnta-LW は，ハードウェア軽量性を追及している代価として，性能優位性はないとの結果が得られた．しかしながら，現在の PC の極めて高い演算能力を考えると，Lesamnta-LW の性能数値は十分実用に堪えるものと言える．

表 2　実装結果 (32 ビット Core2Duo)

アルゴリズム	言語		Bulk Speed (Cycles/Byte)
Lesamnta -LW	アセンブリ言語		94.3
	アセンブリ言語 w/AES-NI	[12]手法	57.3
		[3]手法	58.4
SHA-256	ANSI C		29.3[13]

表 3　実装結果 (64 ビット Core2Duo)

アルゴリズム	言語	Bulk Speed (Cycles/Byte)
Lesamnta -LW	アセンブリ言語	83.0
SHA-256	ANSI C	20.1[13]

5. まとめ

本稿では，ハッシュ関数 Lesamnta-LW の，ハードウェア実装，及び，ソフトウェア実装を行ない，その結果を評価した。

ハードウェア実装においては，Lesamnta-LW のブロック暗号 *E* のラウンド関数が，比較的小さなビット長の鍵レジスタで構成でき，又，鍵スケジュール部，主攪拌部で用いる主たる変換処理が処理回路共有に適していること，更に，LW-1 モードが，レジ

Vol.2010-DPS-142 No.56
Vol.2010-CSEC-48 No.56
2010/3/5

スタや排他的論理輪などの演算回路の付加なしに構成可能であることを明らかにした．又，90nm スタンダードセルライブラリを用いた際の実装評価値を明らかにした．

ソフトウェア実装においては，Lesamnta-LW の鍵スケジュール部，主攪拌部の主たる変換処理が，AES と共通であることから，32 ビット・64 ビットプロセッサ上での表参照手法など，既存の高速化手法の適用が容易であること，又，AES 処理専用命令の適用による高速化が可能であることを明らかにした. 又, Intel Core2 Duo, “westmere” 上での実装評価値，見積もり値を明らかにした．

これらの結果から，Lesamnta-LW は，ハードウェアでの軽量実装性を備え，PC 等においても実用的なソフトウェア処理性能を有するアルゴリズムであると考える。

謝辞 本研究の一部は，高度通信・放送研究開発に係る委託研究制度の一環として独立行政法人情報通信研究機構から委託を受け実施している「次世代ハッシュ関数の研究開発」の成果の一部である．

参考文献

1) 廣瀬勝一, 井手口恒太, 桑門秀典, 大和田徹, 吉田博隆, “A Lightweight Hash Function: Lesamnta-LW,” 2010 年暗号と情報セキュリティシンポジウム, SCIS2010 講演予稿集, 3D1-1.
2) Intel Corp, "Intel's Advanced Encryption Standard (AES) Instructions Set," http://software.intel.com/en-us/articles/advanced-encryption-standard-aes-instructions-set/
3) Ryad Benadjila, Olivier Billet, Shay Gueron and Matt Robshaw, “The Intel AES Instructions Set and the SHA-3 Candidates,” ASIACRYPT 2009, Nov 2009.
4) 大和田徹, 井手口恒太, 吉田博隆, “Lesamnta v2 のソフトウェア実装及び評価,” 2010 年暗号と情報セキュリティシンポジウム, SCIS2010 講演予稿集, 3D1-4.
5) S. Hirose, K. Ideguchi, H. Kuwakado, T. Owada and H. Yoshida, “Construction of Lightweight Hash Function Based on Dedicated Block Cipher,” SCIS2010, 4D1-4.
6) Matyas, S., Meyer, C., and Oseas, J. Generating, “Strong one-way functions with cryptographic algorithms,” IBM Technical Disclosure Bulletin 27, 10a (1985), 5658-5659.
7) NIST, Federal Information Processing, “FIPS Pub 197, Advanced Encryption Standard (AES)”
8) Akashi Satoh et al., “A Compact Rijndael Hardware Architecture with S-Box Optimization,” ASIACRYPT2001, Dec 2001.
9) 森岡澄夫, 佐藤証, “共通鍵暗号 AES の低消費電力論理回路構成法,” 情報処理学会論文誌, 第 44 巻, 第 5 号, 2003.
10) Intel Corp, “Intel 64 and IA-32 Architectures software Developer's Manual.”
11) http://software.intel.com/en-us/blogs/2008/08/11/emulation-of-new-instructions/
12) M. Kounavis and S. Gueron, “Vortex: A New Family of One Way Hash Functions based on Rijndael Rounds and Carry-less Multiplication,”, http://eprint.iacr.org/2008/464.pdf
13) “Classification of the SHA-3 Candidates,” http://www.uni-weimar.de/cms/fileadmin/medien/medsicherheit/Research/SHA3/Classification_of_the_SHA-3_Candidates.pdf

[1] Lesamntaは株式会社日立製作所の登録商標です。
[2] IntelはIntel Corporationの登録商標、Core2はIntel Corporationの商品名称です。