暗号アルゴリズム「RC6」
詳細評価（HW実装評価）レポート

２００１年1月１９日

目次

１．アルゴリズム概要

RC6 は、Ron Rivest, Matt Robshaw, Ray Sidney および Yiqun Yin により 1998 年に設計されたブロック暗号であり、AES の最終審査対象 5 候補のひとつに選定されている。RC6 はブロック長、ラウンド数、鍵サイズがともに可変なスケーラブルブロック暗号である点がひとつの特徴である。このうちＡＥＳに投稿されたものは、ブロック長 128 ビット、段数 20 段、鍵サイズ 128、192、256 ビットのものであるので、本報告では、このパラメータをもつ RC6 の中で、鍵サイズが 128 ビットのものを評価対象とする。

RC6 は、これに先立つ 1995 年に設計されたスケーラブルブロック暗号 RC5 をもとに、AES コンテスト向けに再設計されたものと考えてよい。RC5 の特長である記述の簡易性とスケーラビリティを受け継ぎ、さらにソフトウエアでの高速性を追求したものである。RC6 はテーブル参照を一切おこなわず、32 ビット単位の算術加算・乗算と、データ依存回転シフトによってデータの攪拌を行っている。これは 32 ビットプロセッサではメモリアクセスの頻度のきわめて少ない高速な実装が可能なことを示している。

RC6 は、32 ビットプロセッサでは AES の最終審査対象 5 候補のうち最も高速であると言っておおむねよいが、一方 32 ビット乗算やデータ依存回転シフトの多用は、プラットフォームによっては速度の低下をまねくことがある。例えばローエンド IC カード向けの 8 ビットマイコンでは、RC6 は低速である。またハードウェアでは、RC6 のスループット（平文入力から暗号文出力までの時間）は非常に低速である。このように RC6 は結果的に AES の標準プラットフォームであった 32 ビットプロセッサにターゲットを絞った暗号であるということができる。

２．アルゴリズムRC6の説明

RC6は完全にパラメータ制御可能な暗号アルゴリズムの一種である。RC6の各バージョンは、RC6-*w/r/b* と書くことでさらに正確に表現できる。ここで、ワード長は*w*ビット、暗号化には負でない値のラウンド数ｒ が使われ、またb は暗号鍵の長さをあらわすバイト数である。ブロック長が128ビットの場合は、*w*=32および*r*=20が推奨値で、単にRC6といった場合はこのバージョンを指す。文章中でこれ以外の特別な*w* あるいは*r* の値を使う場合、RC6-*w/r* の様にパラメータを指定する。64ビットのブロック長に対しては*w*=16ビットと指定することになる。鍵サイズは0から256バイトの範囲の値をとることができるが、16バイト、24バイト、そして32バイトの鍵長のRC6のバージョンが最もよく使われている。

RC6-*w/r/b* では、すべての類型においても同様に、4個の*w*ビット長ワードの単位に対する、以下の6個の基本演算がおこなわれる。lg *w* は、2を底とする*w* の対数をあらわす。

$a + b$　　整数加算（モジュロ2^w）

$a - b$　　整数減算（モジュロ2^w）

$a \oplus b$　　wビット長のワードのビット毎の排他的論理和

$a \times b$　　整数乗算（モジュロ2^w）

$a<<<b$　　wビット長のワードaの左回転。回転量はb の最下位 lg w ビットで与えられる。

$a>>>b$　　wビット長のワードaの右回転、回転量はb の最下位 lg wビットで与えられる。

RC6の記述中に使われる“ラウンド”という言葉は、通常のDES式ラウンド数の発想にほぼ近い。つまり、データの一方の半分を他方の半分で更新し、次にそれらを入れ替える操作をいう。

２．１　データランダマイズ部

RC6 では、w ビットのレジスタ A,B,C,D が使われ、最初これらには入力された平文が置かれ、暗号化の最後の段階では出力暗号文が置かれる。平文あるいは暗号文の第 1 バイトは、A の最下位のバイトに置かれる。平文あるいは暗号文の最後のバイトは、D の最上位バイトに置かれる。ここで、$(A,B,C,D)=(B,C,D,A)$という式は、式の右側のレジスタの値を左側のレジスタに同時にまとめて代入することを意味する。図 1 に、RC6 による暗号化および復号化の擬似コードを示し、RC6 による暗号化を図式で表したものを図 2 に示す。

一般的なFeistel構造の場合は、暗号化と復号化において、共通のデータランダマイズ部を利用することが可能であるが、拡張Feistel構造を持つRC6では、暗号化と復号化で別々の構成を持たなければならない。

RC6-*w*/*r*/*b*による暗号化

入力：　4個の*w*ビット長の入力レジスタ*A,B, C, D*に配置された平文。
ラウンド数*r*。
*w*ビット長ラウンド鍵の配列　*S*[0,...,2*r*+3]。

出力：　*A,B,C,D*に配置された暗号文。

手順：

$B = B + S[0]$
$D = D + S[1]$
for $i = 1$ **to** r **do**
　{
　　$t = (B \times (2B + 1)) \lll \lg w$
　　$u = (D \times (2D + 1)) \lll \lg w$
　　$A = ((A \oplus t) \lll u) + S[2i]$
　　$C = ((C \oplus u) \lll t) + S[2i + 1]$
　　$(A, B, C, D) = (B, C, D, A)$
　}
$A = A + S[2r + 2]$
$C = C + S[2r + 3]$

RC6-*w*/*r*/*b*による復号化

入力：　4個の*w*ビット長の入力レジスタ*A,B,C,D*に配置された暗号文。
ラウンド数*r*。
*w*ビット長ラウンド鍵の
配列　*S*[0,... ,2*r*+3]。

出力：　*A,B,C,D*に配置された平文

手順：

$C = C - S[2r + 3]$
$A = A - S[2r + 2]$
for $i = r$ **downto** 1 **do**
　{
　　$(A, B, C, D) = (D, A, B, C)$
　　$u = (D \times (2D + 1)) \lll \lg w$
　　$t = (B \times (2B + 1)) \lll \lg w$
　　$C = ((C - S[2i + 1]) \ggg t) \oplus u$
　　$A = ((A - S[2i]) \ggg u) \oplus t$
　}
$D = D - S[1]$
$B = B - S[0]$

図 1.　RC6 の暗号化および復号化の擬似コード

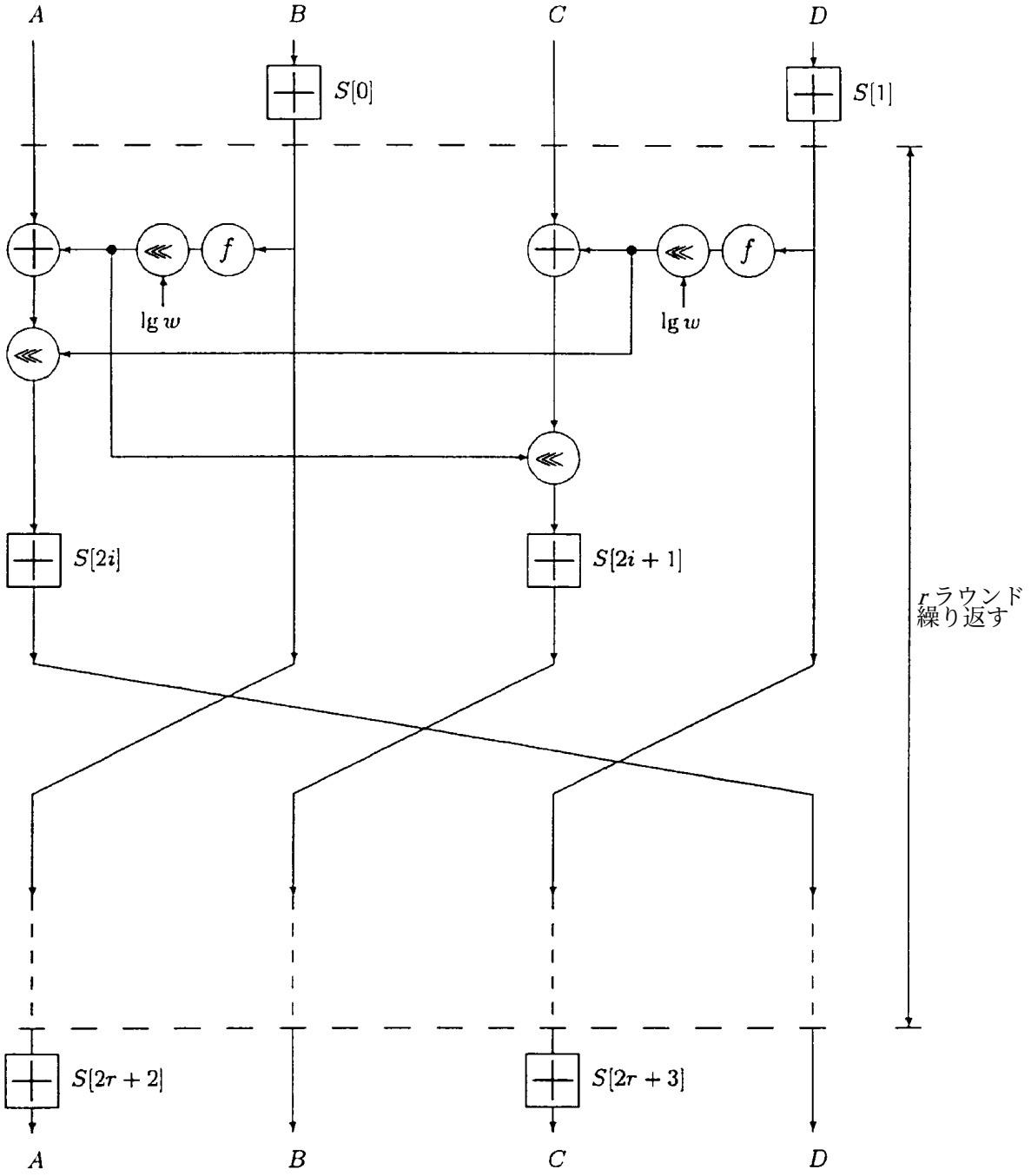

図2. データランダマイズ部

２．２　鍵スケジュール部

RC6-*w/r/b* の鍵スケジュールは、RC5-*w/r/b* の鍵スケジュールと実質的に等価である。事実上唯一の違いは、RC6-*w/r/b* ではより多くのワードが生成され、それらが暗号復号双方で使われる配列S[0,...,2*r*+3]に配置されることである。RC5鍵スケジュールと同様のものを採用した理由は、これには既に6年にわたる広範囲の研究実績が存在しているからである。

図3に、RC6の鍵スケジュールの擬似コードを記す。ユーザは、0≦*b*≦256である*b*バイトの鍵を与える。最終的にはこの鍵を使って2*r*+4ワード（各々*w*ビット）の拡張鍵を生成する。

まず、鍵の長さがゼロではない整数で表されるワード数と等しくなるまでゼロのバイトを追加する。これらの鍵の各バイトはリトル・エンディアン方式によって*c*個の*w*ビット長の配列*L*[0],...,*L*[*c*-1]に前もって配置される。このようにして、鍵の第1バイトは*L*[0]の最下位バイトに置かれ、以下同様に進み、最後に*L*[*c*-1]には必要ならば上位のバイトにゼロが埋められる（ただし*b*=0の場合は、*c*=1および*L*[0]=0となる。）。生成される1ワード*w*ビットの追加ラウンド鍵のワード数は2*r*+4であり、これらは配列*S*[0,...,2*r*+3]に配置される。（鍵の長さが異なっていても、ゼロを埋めることで同じ配列*L*[0],...,*L*[*c*-1]ができる場合は、鍵スケジュールは同じものになる。といっても一般的には鍵が異れば、鍵スケジュールも異なる。）定数 P_{32}=B7E15163 と Q_{32}=9E3779B9（16 進数）は、RC5 の場合と同じく“ 魔法の定数 ”である。P_{32} の値は、*e*-2 の二項展開から導かれる。*e* は自然対数の底である。Q_{32} の値は、*Φ*-1 の二項展開から導かれる。*Φ*は黄金比である。他のワード長の RC6 に対しても同様に、P64 およびその他に対する RC5 の定義を適用することが可能である。これらの値は幾分任意性を持つものであり、他の値を使って、“ あつらえ ”のつまり独自バージョンの RC6 をつくることも可能である。

RC6-*w/r/b*の鍵スケジュール

入力： ワード数cの配列 $L[0,\cdots,c\text{-}1]$に前もって配置されたbバイトのユーザ提供鍵。

ラウンド数r。

出力： wビット長ラウンド鍵の配列 $S[0,\ldots,2r+3]$。

手順：

$S[0] = P_w$

for $i = 1$ **to** $2r + 3$ **do**
$S[i] = S[i-1] + Q_w$

$A = B = i = j = 0$

$v = 3 \times \max\{c, 2r + 4\}$
for $s = 1$ **to** v **do**
{
$A = S[i] = (S[i] + A + B) \lll 3$
$B = L[j] = (L[j] + A + B) \lll (A + B)$
$i = (i + 1) \bmod (2r + 4)$
$j = (j + 1) \bmod c$
}

図 3. 鍵スケジュール部

３．評価方針

今回の性能評価は、SBOX、乗算器、加算器、ラウンド関数等といった基本的な機能を実現している個々のパーツ(以下、primitive)の評価(回路規模、処理速度等)を実施することによりアルゴリズム全体の性能を見積る方法で評価を行い、また、個々の primitive においては、実際の LSI 作成に則した条件を付加して評価を行うことを前提に行った。なお、本報告で用いる評価環境は、我々が H/W 評価経験のある三菱 0.35μm CMOS ASIC ライブラリを用い、回路記述には Verilog-HDL、Synthesis には Design Compiler を用い、回路規模(ゲート数)およびクリティカルパス長、処理速度等の性能見積を行った。

３.１　回路規模・性能の見積方法

理想的と考えられる評価は、個々のアルゴリズムに対し、公平な評価指標のもとに、最大限の optimize（SBOX の小型化および最速化、および SBOX やラウンド関数のような複数回使用される回路の実装個数と繰り返し回数のトレードオフ等の検討）を試みて回路規模、性能を見積ることである。しかし、この評価方法では、開発時間、開発資源（論理合成ツール等のソフトウェア、コンピュータ等のハードウェアおよび開発者等）が大量に必要となり、本詳細評価期間では、全てのアルゴリズムを公平な指標のもとに optimize することは困難である。そこで特定のアルゴリズムだけを最適化することはせずに、アルゴリズムを全て実装（回路規模は大きくても構わない）し、クリティカルパス長の短縮（処理速度向上）を重視して評価を行った。

そこで今回の評価方法は、以下の通りに行った。

① 原則として、評価対象アルゴリズム全体(鍵スケジュールおよびすべての内部鍵レジスタを含む)をすべて H/W 実装することを前提とする。
② 入力鍵サイズは 128bit とし、性能評価時に、暗号化、復号部分と拡大鍵生成部分の分離を容易にするため、拡大鍵は全てレジスタに格納する。
③ SBOX、乗算器、加算器、ラウンド関数等といった基本的な機能を実現している個々のパーツ(以下、primitive)の評価(回路規模、処理速度等)を実施する。
④ primitive の使用個数の総和を求め回路規模とする。
⑤ クリティカルパス上に存在する primitive の遅延値すべての総和を求めクリティカルパス長とする。

⑥ 「処理速度」を以下のように定義する。

$$\mathrm{Throughput\ [Mbps]} = \frac{\text{ブロックサイズ（128[bit]）}}{\{\text{暗号化（または復号）回路のクリティカルパス [ns]}\}}$$

⑦ テーブル参照型 SBOX の実装は、論理合成ツール（Design Compiler）のみを用い optimize を行う。

⑧ 我々がハードウェア評価経験のあるツールを用いる。

・開発言語：Verilog-HDL

・シミュレータ：Verilog-XL

・デザインライブラリ：三菱 0.35μm CMOS ASIC ライブラリ

・ 論理合成および性能評価：Design Compiler（version 1998.08）

⑨ 評価するときの環境条件は、Worst ケースとする。

※注…Worst ケースとは、ハードウェアが動作する環境が劣悪、かつ、LSI 製造時のバラツキ度合いが悪い方の状態にあることを意味する。一般的に、ハードウェアの性能を議論するときには、議論の対象となっているハードウェアの動作を保証するという意味で、Worst ケースで性能を論じることがほとんどである。よって、これ以降は、何も断りが無い場合は全て Worst ケースで議論を行う。

3.2 設計上の留意点

primitive の回路設計は、Verilog-HDL で記述し、設計する際の細かなブロック分けは、可能な限りアルゴリズム開発者のブロック分けに準じるように留意し行った。

3.3 Synthesis（論理合成）条件

Synthesis(論理合成)ツールは Synopsys 社の Design Compiler(version 1998-08)を用い、すべてのアルゴリズムおよび primitive に同一の条件を付加した。この条件は、速度を最大、かつ、実際の LSI 作成に則するようなものである。以下に条件を示す。

① 加算・減算・乗算回路は、Synopsys の Design Ware Basic Library 内にある最速なライブラリを用いる。

・ 32bit 加算器は Synopsys の Design Ware Basic Library 内にある DW01_add(cla)[Carry look-ahead synthesis model]を用いる。

・ 32bit 減算器は Synopsys の Design Ware Basic Library 内にある

DW01_sub(cla)[Carry look-ahead synthesis model]を用いる。

・ 32bit 乗算器は Synopsys の Design Ware Basic Library 内にある DW02_mult(csa)[Carry-save array synthesis model]を用いる。

② その他の条件は、基本的に default 値を用いる。ただし、fanout に関する条件は以下の理由で付加する。

○fanout 条件を全く付加しないことによる問題点

ある primitive の入力端子に fanout 条件が付けられていないとその入力端子は駆動能力が「∞」のドライバより駆動されていることになり、『primitive レベルでの評価からアルゴリズム全体を試算し評価する結果』と『アルゴリズム全体のレベルでの評価する結果』でかなりの違いが出てきてしまう。

実際の LSI 設計では、各 primitive の負荷状況(fanout 等)やドライバの駆動能力は、キャラクタライズ（characterize）という手法を用い、各入出力ピンの負荷状況を自動的に設定する方法、もしくは、各入出力端子に fanout 条件を付加する方法等を用いて LSI を設計、作成している。そこで、今回は、アルゴリズム全てを実装せずに、primitive レベルから性能を見積る方法を行うため、characterize 手法は使用せず、以下のように入出力端子に fanout 設定を行う。

① primitive の入力ピンの fanin が 1 になるように条件をつける。

② primitive の出力から最大 40 個の primitive が並列に接続されても耐えうるように、primitive の出力ピンの fanout は 40 になるように条件をつける。

４．各 primitive の見積り

評価方法に従って評価を実行するために、RC6 の構造を詳細にチェックし、各 primitive の見積りを行った。ここでは､回路規模や速度に関しては考慮せずに、論理的にどのような構成のものがいくつあるかと言うことについて見積もりを行っている。

４．１　データランダマイズ部

表１にあるように、暗号化部に関しては､Round 関数が 20 個の 20 段構成となっている。その他には、最初と最後の部分の拡大鍵との 32 ビット加算が 4 個存在している。復号化部しても、暗号化部同様に、$Round^{-1}$ 関数が 20 個の 20 段構成となっている。その他には、最初と最後の部分の拡大鍵との 32 ビット減算が 4 個存在している。

Round 関数は、32 ビット排他的論理和が 2 個、32 ビット加算が 2 個、F 関数が 2 個、32 ビット可変ローテーションシフトが 4 個で構成されている。Round 関数で使用されている F 関数は、32 ビット乗算が 1 個と 2×a+1 の演算によって構成されている。ここで、2×a+1 の演算は、ハードウェアとしては、結線のみの構成で実現されるため primitive 構成の見積りとしては含んでいない。

復号化部に使用される $Round^{-1}$ 関数では、32 ビットの加算部分が 32 ビットの減算に替わるだけで、ほぼ同等の primitive 構成となっている。

表 1．データランダマイズ部の各 primitive の見積り

暗号化部	Round 関数	20 個
	32 ビット加算	4 個

復号化部	$Round^{-1}$ 関数	20 個
	32 ビット減算	4 個

Round 関数	32 ビット EXOR	2 個
	32 ビット加算	2 個
	F 関数	2 個
	32 ビット可変 LOT	4 個

$Round^{-1}$ 関数	32 ビット EXOR	2 個
	32 ビット減算	2 個
	F 関数	2 個
	32 ビット可変 LOT	4 個

F 関数	32 ビット乗算	1 個

F 関数	32 ビット乗算	1 個

４．２　鍵スケジュール部

表 2 にあるように、鍵スケジュール部に関しては、初期生成部が 1 個、拡大鍵生成ブロック A と拡大鍵生成ブロック B がそれぞれ 132 個で構成されている。

初期生成部に関しては、32 ビット加算が 43 個、拡大鍵生成ブロック A では 32 ビット加算が 2 個と 32 ビットローテーションシフトが 1 個、拡大鍵生成ブロック B では 32 ビット加算が 2 個と 32 ビットの可変ローテーションシフトが 1 個で構成されている。

鍵スケジュール部の場合は、データランダマイズ部と異なり、暗号化部と復号化部において、共通の鍵スケジュール部を利用することが可能である。

表 2. 鍵スケジュール部の各 primitive の見積り

鍵スケジュール部	**初期生成部**	**1 個**
	拡大鍵生成ブロック A	**132 個**
	拡大鍵生成ブロック B	**132 個**

※暗号化･復号とも同じ構成

初期生成部	**32 ビット加算**	**43 個**
拡大鍵生成ブロック A	**32 ビット加算**	**2 個**
	32 ビット ROT	**1 個**
拡大鍵生成ブロック B	**32 ビット加算**	**2 個**
	32 ビット可変 ROT	**1 個**

５．ハードウェア評価結果

各 primitive の見積もり結果をもとにして、評価方針通りに評価を行った。

表 3 は、データランダマイズ部の各 primitive の実装結果である。ここで、Round 関数や $Round^{-1}$ 関数の実装に関しては、下位の primitive 毎に設計して積み上げる方式ではなく、実際に Verilog-HDL で記述し、論理合成を行った結果である。

表 3. データランダマイズ部の primitive 実装結果

	回路規模[Gate]	クリティカルパス[ns]
32 ビット乗算	5,446	23.50
32 ビット加算	859	3.45
32 ビット減算	1,087	3.69
32 ビット EXOR	267	1.08
32 ビット可変ローテート(Left)	3,336	3.33
32 ビット可変ローテート(Right)	3,428	3.30
2to1selector*128bit	581	1.98
128 ビットレジスタ	1,029	0.80

表 4 は、鍵スケジュール部の各 primitive の実装結果である。鍵スケジュール部に関しては、暗号化部と復号化部において、共通に利用することが可能である。

表 4. 鍵スケジュール部の primitive 実装結果

	回路規模[Gate]	クリティカルパス[ns]
初期生成部	36,924	148.35
拡大鍵生成ブロック A	2,080	6.89
拡大鍵生成ブロック B	4,888	8.49
2to1selector*128bit*11	12,344	0.80
128 ビットレジスタ*11	6,391	1.98

表 3、表 4 の primitive 実装結果をもとに、データランダマイズ部、鍵スケジュール部をアルゴリズムの構成を再現すると、表 5 のような結果となった。

表 5. データランダマイズ部、鍵スケジュール部の評価結果

	回路規模[Gate]	クリティカルパス[ns]
Round 関数	18888	34.69
32 ビット加算	859	3.45
暗号化部	381182	697.25
$Round^{-1}$ 関数	19528	31.21
32 ビット減算	1087	3.69
復号化部	394894	627.89
2to1selector*128bit	581	1.98
32 ビット EXOR	267	1.08
128 ビットレジスタ	1029	0.80
データランダマイズ部	777685	698.05
初期値生成	36924	148.35
拡大鍵生成ブロック A	2080	6.89
拡大鍵生成ブロック B	4888	8.49
2to1selector*128bit *11	12344	0.80
128 ビットレジスタ*11	6391	1.98
鍵スケジュール部	975391	2181.29
合計	1753076	698.05

以上の実装結果および見積もりにより、RC6 の H/W 詳細評価は、表 6 のような結果となった。なお、アルゴリズムの処理速度を見積もるため、本報告ではクリティカルパスに鍵スケジュールは含まれていないことに注意する。

表 6. RC6 の H/W 詳細評価結果

回路規模[Gate]	クリティカルパス[ns]	処理速度[Mbps]
1,753,076	698.05	183.36

６．まとめ

RC6については、5章の評価結果のように、約183Mbpsの処置速度であることから、128ビット暗号としてAESに採用されたRijndael等と比較した場合、相当低速なアルゴリズムであると言える。また、回路規模に関しては、拡張Feistel構造を採用しており、暗号化と復号化で共通の構成を採用できないこと、また、乗算回路を多数使用していることから、約1.7Mgateと非常に大きいアルゴリズムである。実際の回路実装を考慮した場合、Round関数等を全て実装することは無く、1個実装し、ループして使用することが考えられるが、その場合においても、約100～200Kgate前後の規模となることが予想される。従って、処理速度から、高速デジタル回線等の適用も困難であり、かつ、回路規模から、ICカードや携帯端末などの小型なデバイスを使用するアプリケーションは実装困難なアルゴリズムであると考えられる。

－　以上　－