The RC6 Block Cipher: A simple fast secure AES proposal

Ronald L. Rivest	MIT
Matt Robshaw	RSA Labs
Ray Sidney	RSA Labs
Yiqun Lisa Yin	RSA Labs

(August 21, 1998)

Outline

- Design Philosophy
- Description of RC6
- Implementation Results
- Security
- Conclusion

Design Philosophy

- Leverage our experience with RC5: use *data-dependent rotations* to achieve a high level of security.
- Adapt RC5 to meet AES requirements
- Take advantage of a new primitive for increased security and efficiency: *32x32 multiplication*, which executes quickly on modern processors, to compute rotation amounts.

Description of RC6

Description of RC6

- RC6-w/r/b parameters:
  - *Word size* in bits: w (32)(lg(w) = 5)
  - Number of *rounds*: r (20)
  - Number of *key bytes*: b (16, 24, or 32)
- Key Expansion:
  - Produces array S[0 … 2r + 3] of w-bit *round keys.*
- Encryption and Decryption:
  - Input/Output in 32-bit registers A,B,C,D

RC6 Primitive Operations

RC5

$A + B$ Addition modulo 2^w

$A - B$ Subtraction modulo 2^w

$A \oplus B$ Exclusive-Or

$A \lll B$ Rotate A left by amount in low-order $\lg(w)$ bits of B

$A \ggg B$ Rotate A right, similarly

$(A,B,C,D) = (B,C,D,A)$ Parallel assignment

$A \times B$ Multiplication modulo 2^w

RC6 Encryption (Generic)

```
B = B + S[ 0 ]
D = D + S[ 1 ]
for i = 1 to r do
   {
      t = ( B x ( 2B + 1 ) ) <<< lg( w )
      u = ( D x ( 2D + 1 ) ) <<< lg( w )
      A = ( ( A ⊕ t ) <<< u ) + S[ 2i ]
      C = ( ( C ⊕ u ) <<< t ) + S[ 2i + 1 ]
      (A, B, C, D) = (B, C, D, A)
   }
A = A + S[ 2r + 2 ]
C = C + S[ 2r + 3 ]
```

RC6 Encryption (for AES)

```
B = B + S[ 0 ]
D = D + S[ 1 ]
for i = 1 to 20 do
  {
    t = ( B x ( 2B + 1 ) ) <<< 5
    u = ( D x ( 2D + 1 ) ) <<< 5
    A = ( ( A ⊕ t ) <<< u ) + S[ 2i ]
    C = ( ( C ⊕ u ) <<< t ) + S[ 2i + 1 ]
    (A, B, C, D) = (B, C, D, A)
  }
A = A + S[ 42 ]
C = C + S[ 43 ]
```

RC6 Decryption (for AES)

$C = C - S[43]$
$A = A - S[42]$
for $i = 20$ **downto** 1 **do**
{
$(A, B, C, D) = (D, A, B, C)$
$u = (D \times (2D + 1)) \lll 5$
$t = (B \times (2B + 1)) \lll 5$
$C = ((C - S[2i + 1]) \ggg t) \oplus u$
$A = ((A - S[2i]) \ggg u) \oplus t$
}
$D = D - S[1]$
$B = B - S[0]$

Key Expansion (Same as RC5's)

- Input: array L[0 … c-1] of input key words
- Output: array S[0 … 43] of round key words
- Procedure:

```
S[ 0 ] = 0xB7E15163
for  i = 1  to  43  do S[i] = S[i-1] + 0x9E3779B9
A = B = i = j = 0
for  s = 1  to  132  do
   {  A = S[ i ] = ( S[ i ] + A + B ) <<< 3
      B = L[ j ] = ( L[ j ] + A + B ) <<< ( A + B )
      i = ( i + 1 )   mod 44
      j = ( j + 1 )  mod c          }
```

From RC5 to RC6
in seven easy steps

(1) Start with RC5

RC5 encryption inner loop:

```
for i = 1 to r do
    {
        A = ( ( A ⊕ B ) <<< B ) + S[ i ]
        ( A, B ) = ( B, A )
    }
```

Can RC5 be strengthened by having rotation amounts depend on *all* the bits of B?

Better rotation amounts?

- Modulo function?
  Use low-order bits of (B mod d)
  Too slow!
- Linear function?
  Use high-order bits of (c x B)
  Hard to pick c well!
- Quadratic function?
  Use high-order bits of (B x (2B+1))
  Just right!

$B \times (2B+1)$ is *one-to-one* mod 2^w

Proof: By contradiction. If $B \neq C$ but

$B \times (2B + 1) = C \times (2C + 1) \pmod{2^w}$

then

$$(B - C) \times (2B+2C+1) = 0 \quad \pmod{2^w}$$

But $(B-C)$ is nonzero and $(2B+2C+1)$ is odd; their product can't be zero! ☐

Corollary:

B uniform → $B \times (2B+1)$ uniform

(and high-order bits are uniform too!)

High-order bits of B x (2B+1)

- The high-order bits of

  $$f(B) = B \times (2B + 1) = 2B^2 + B$$

  depend on all the bits of B .
- Let $B = B_{31}B_{30}B_{29} \ldots B_1B_0$ in binary.
- Flipping bit i of input B
  - Leaves bits 0 … i-1 of f(B) unchanged,
  - Flips bit i of f(B) with probability one,
  - Flips bit j of f(B) , for $j > i$, with probability approximately 1/2 (1/4…1),
  - is likely to change some high-order bit.

(2) Quadratic Rotation Amounts

```
for i = 1 to r do
    {
      t = ( B x ( 2B + 1 ) ) <<< 5
      A = ( ( A ⊕ B ) <<< t ) + S[ i ]
      ( A, B ) = ( B, A )
    }
```

But now much of the output of this nice multiplication is being wasted...

(3) Use t, not B, as xor input

```
for i = 1 to r do
  {
    t = ( B x ( 2B + 1 ) ) <<< 5
    A = ( ( A ⊕ t ) <<< t ) + S[ i ]
    ( A, B ) = ( B, A )
  }
```

Now AES requires 128-bit blocks.
We could use two 64-bit registers, but
64-bit operations are poorly supported with typical
C compilers...

(4) Do two RC5's in parallel

Use four 32-bit regs (A,B,C,D), and do RC5 on (C,D) in parallel with RC5 on (A,B):

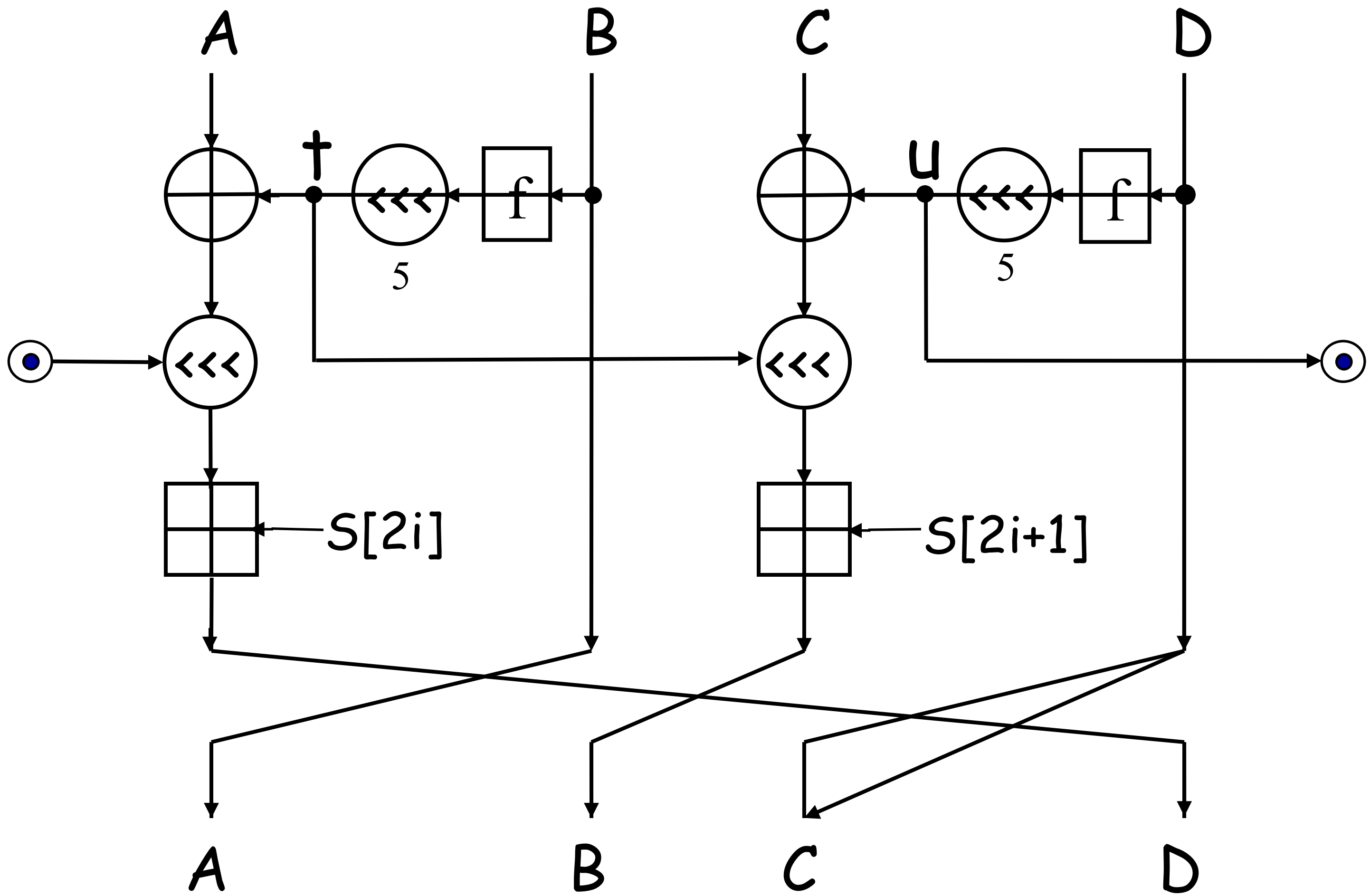
```
for i = 1 to r do
   {
      t = ( B x ( 2B + 1 ) ) <<< 5
      A = ( ( A ⊕ t ) <<< t ) + S[ 2i ]
      ( A, B ) = ( B, A )
      u = ( D x ( 2D + 1 ) ) <<< 5
      C = ( ( C ⊕ u ) <<< u ) + S[ 2i + 1 ]
      ( C, D ) = ( D, C )
   }
```

(5) Mix up data between copies

Switch rotation amounts between copies, and cyclically permute registers instead of swapping:

```
for i = 1 to r do
  {
    t = ( B x ( 2B + 1 ) ) <<< 5
    u = ( D x ( 2D + 1 ) ) <<< 5
    A = ( ( A ⊕ t ) <<< u ) + S[ 2i ]
    C = ( ( C ⊕ u ) <<< t ) + S[ 2i + 1 ]
    (A, B, C, D) = (B, C, D, A)
  }
```

One Round of RC6



(6) Add Pre- and Post-Whitening

```
B = B + S[ 0 ]
D = D + S[ 1 ]
for i = 1 to r do
   {
      t = ( B x ( 2B + 1 ) ) <<< 5
      u = ( D x ( 2D + 1 ) ) <<< 5
      A = ( ( A ⊕ t ) <<< u ) + S[ 2i ]
      C = ( ( C ⊕ u ) <<< t ) + S[ 2i + 1 ]
      (A, B, C, D) = (B, C, D, A)
   }
A = A + S[ 2r + 2 ]
C = C + S[ 2r + 3 ]
```

(7) Set r = 20 for high security

(based on analysis)

```
B = B + S[ 0 ]
D = D + S[ 1 ]
for i = 1 to 20 do
   {
      t = ( B x ( 2B + 1 ) ) <<< 5
      u = ( D x ( 2D + 1 ) ) <<< 5
      A = ( ( A ⊕ t ) <<< u ) + S[ 2i ]
      C = ( ( C ⊕ u ) <<< t ) + S[ 2i + 1 ]
      (A, B, C, D) = (B, C, D, A)
   }
A = A + S[ 42 ]
C = C + S[ 43 ]
```

Final RC6

RC6 Implementation Results

CPU Cycles / Operation

	Java	Borland C	Assembly
Setup	110000	2300	1108
Encrypt	16200	616	254
Decrypt	16500	566	254

Less than two clocks per bit of plaintext !

Operations/Second (200MHz)

	Java	Borland C	Assembly
Setup	1820	86956	180500
Encrypt	12300	325000	787000
Decrypt	12100	353000	788000

Encryption Rate (200MHz)

MegaBytes / second
MegaBits / second

	Java	Borland C	Assembly
Encrypt	0.197	5.19	12.6
	1.57	*41.5*	*100.8*
Decrypt	0.194	5.65	12.6
	1.55	*45.2*	*100.8*

Over 100 Megabits / second !

On an 8-bit processor

- On an Intel MCS51 (1 Mhz clock)
- Encrypt/decrypt at 9.2 Kbits/second (13535 cycles/block; from actual implementation)
- Key setup in 27 milliseconds
- Only 176 bytes needed for table of round keys.
- Fits on smart card (< 256 bytes RAM).

Custom RC6 IC

- 0.25 micron CMOS process
- One round/clock at 200 MHz
- Conventional multiplier designs
- 0.05 mm^2 of silicon
- 21 milliwatts of power
- Encrypt/decrypt at 1.3 Gbits/second
- With pipelining, can go faster, at cost of more area and power

RC6 Security Analysis

Analysis procedures

- Intensive analysis, based on most effective known attacks (e.g. linear and differential cryptanalysis)
- Analyze not only RC6, but also several "simplified" forms (e.g. with no quadratic function, no fixed rotation by 5 bits, etc…)

Linear analysis

- Find approximations for r-2 rounds.
- Two ways to approximate $A = B \lll C$
  - with one bit each of A, B, C (type I)
  - with one bit each of A, B only (type II)
  - each have bias 1/64; type I more useful
- Non-zero bias across f(B) only when input bit = output bit. (Best for lsb.)
- Also include effects of multiple linear approximations and linear hulls.

Security against linear attacks

Estimate of number of plaintext/ciphertext pairs required to mount a linear attack.

(Only 2^{128} such pairs are available.)

Rounds		Pairs	
8		2^{47}	
12		2^{83}	
16		2^{119}	
20 ←	RC6 →	2^{155}	Infeasible
24		2^{191}	

Differential analysis

- Considers use of (iterative and non-iterative) (r-2)-round *differentials* as well as (r-2)-round *characteristics*.
- Considers two notions of "difference":
  - exclusive-or
  - subtraction (better!)
- Combination of quadratic function and fixed rotation by 5 bits very good at thwarting differential attacks.

An iterative RC6 differential

A	B	C	D
1<<16	1<<11	0	0
1<<11	0	0	0
0	0	0	1<<s
0	1<<26	1<<s	0
1<<26	1<<21	0	1<<v
1<<21	1<<16	1<<v	0
1<<16	1<<11	0	0

- Probability = 2^{-91}

Security against differential attacks

Estimate of number of plaintext pairs required to mount a differential attack.

(Only 2^{128} such pairs are available.)

Rounds		Pairs	
8		2^{56}	
12		2^{117}	
16		2^{190}	Infeasible
20	← RC6 →	2^{238}	
24		2^{299}	

Security of Key Expansion

- Key expansion is identical to that of RC5; no known weaknesses.
- No known weak keys.
- No known related-key attacks.
- Round keys appear to be a "random" function of the supplied key.
- Bonus: key expansion is quite "one-way"---difficult to infer supplied key from round keys.

Conclusion

- RC6 more than meets the requirements for the AES; it is
  - simple,
  - fast, and
  - secure.
- For more information, including copy of these slides, copy of RC6 description, and security analysis, see
  www.rsa.com/rsalabs/aes

(The End)