Cryptanalysis

Block Ciphers-3

John Manferdelli
JohnManferdelli@hotmail.com

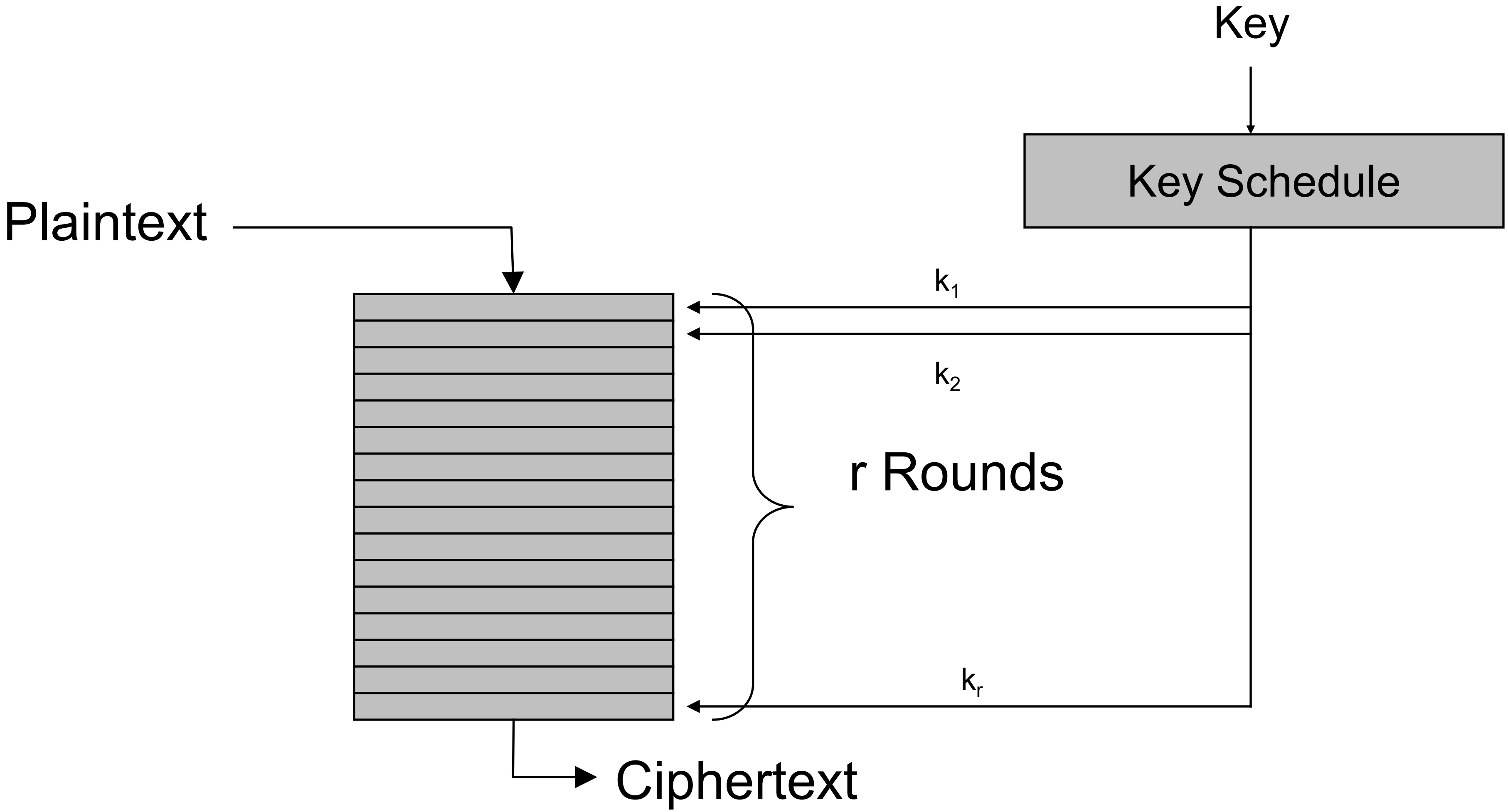
This material is provided without warranty of any kind including, without limitation, warranty of non-infringement or suitability for any purpose. This material is not guaranteed to be error free and is intended for instructional use only

AES History

- Call for DES successor 1/97
- Nine Submissions
  - CAST-256, CRYPTON, DEAL, DFC (cipher), E2, FROG, HPC, LOKI97, MAGENTA, MARS, RC6, Rijndael, SAFER+, Serpent, and Twofish.
- Finalists
  - MARS, RC6, Rijndael, Serpent, and Twofish
- And the winner is Rijndael: FIPS 197 published 11/2001

- Good References:
  - Daemen and Rijimen, *The Design of Rijndael*. Springer.
  - Ferguson et. al., *The Twofish Encryption Algorithm*. Wiley.
  - Tons of contemporaneous material, thesis, etc. Almost all on WWW.

AES



AES Requirements

- 128, 192, 256 bit keys
- Algorithms will be judged on the following factors:
  - Actual security of the algorithm compared to other submitted algorithms (at the same key and block size).
  - The extent to which the algorithm output is indistinguishable from a random permutation on the input block.
  - Soundness of the mathematical basis for the algorithm’s security.
  - Other security factors raised by the public during the evaluation process, including any attacks which demonstrate that the actual security of the algorithm is less than the strength claimed by the submitter.
  - Claimed attacks will be evaluated for practicality.
- Key agility (NSA): “Two blocks encrypted with two different keys should not take much more time than two blocks encrypted with the same key.

DESX and whitening

- Attacks like differential and linear cryptanalysis are easier since we can direct observe the input to the first round and output of the last round directly.

- Rivest and Killian:
  - $\mathrm{DESX}(k_1,k_2,k_3,x)= k_3 \oplus \mathrm{DES}(k_1, k_2 \oplus x)$

- Strategy adopted by almost all the AES participants.

Mars (Multiplication, Addition, Rotation and Substitution)

Basic Structure

1. Whiten
2. 8 rounds of key independent mixing
3. 16 rounds of keyed Feistel transforms (2 S-boxes)
4. 8 rounds of key independent mixing
5. Whiten

RC6 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.

Slide by Ron Rivest (Second AES Conference)

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

Slide by Ron Rivest (Second AES Conference)

RC6 Primitive Operations

$A + B$	Addition modulo 2^w
$A - B$	Subtraction modulo 2^w
$A \oplus B$	Exclusive-Or
$A <<< B$	Rotate A left by amount in low-order lg(w) bits of B
$A >>> B$	Rotate A right, similarly
$(A,B,C,D) = (B,C,D,A)$	Parallel assignment
$A \times B$	Multiplication modulo 2^w

Slide by Ron Rivest (Second AES Conference)

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 ]
```

Slide by Ron Rivest (Second AES Conference)

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 ]
```

Slide by Ron Rivest (Second AES Conference)

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 x ( 2D + 1 ) ) <<< 5
    t = ( B x ( 2B + 1 ) ) <<< 5
    C = ( ( C - S[ 2i + 1 ] ) >>> t ) ⊕ u
    A = ( ( A - S[ 2i ] ) >>> u ) ⊕ t
    }
D = D - S[ 1 ]
B = B - S[ 0 ]
```

Slide by Ron Rivest (Second AES Conference)

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  {
     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
     }
```

Slide by Ron Rivest (Second AES Conference)

Encryption Rate (200MHz)

MegaBytes/second
MegaBits /second

Slide by Ron Rivest (Second AES Conference)

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

Over 100 Megabits / second !

Security against linear attacks

- Estimate of number of plaintext/cipher-text 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}	
24	← →	2^{191}

Infeasible

Slide by Ron Rivest (Second AES Conference)

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.

Slide by Ron Rivest (Second AES Conference)

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

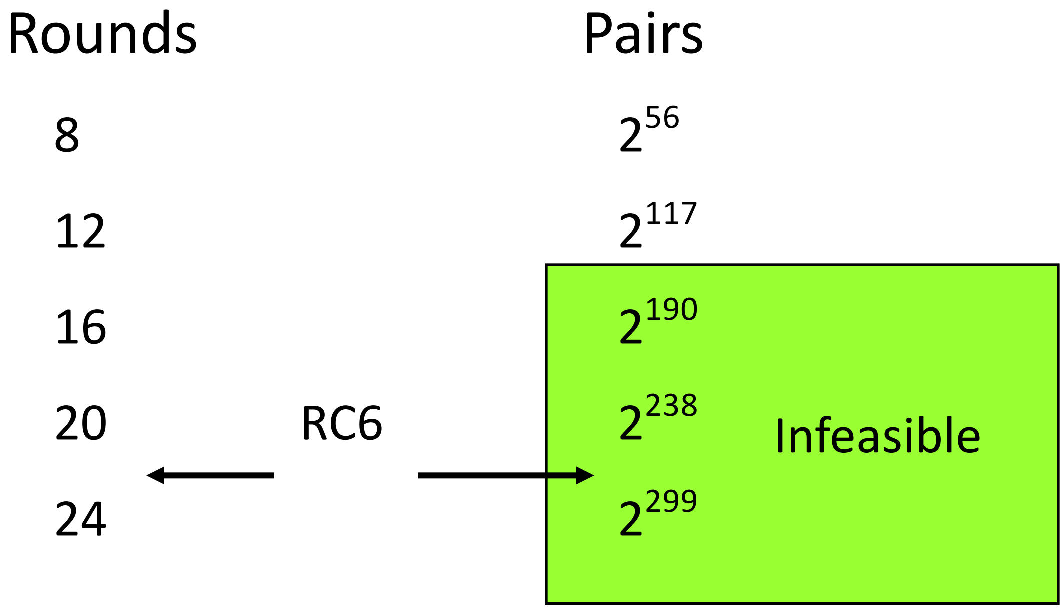
- Probability = 2^{-91}

Slide by Ron Rivest (Second AES Conference)

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}
20	2^{238}
24	2^{299}

Slide by Ron Rivest (Second AES Conference)

Twofish Observations

- Didn’t use multiplication unlike other candidates
- Uses same primitives for key schedule generation as basic round functions
- Key dependent S-box built from two 256 S-Boxes.
- Two non-independent S-Boxes built from 8 fixed 16 element permutations picked for statistical properties.

Twofish

- Basic Structure for 128-bit operation.
  - Construct 40 32-bit round keys K_0 ,…, K_{39}
  - Input Whiten
  - 16 Keyed rounds
  - Output Whiten (after switching left and right blocks)
- Input bytes p_0, p_1, …, p_{15}. Little endian as 32-bit words.
  - $P_0=p_0+p_12^8+p_22^{16}+p_32^{24,}$ $P_1=p_4+p_52^8+p_62^{16}+p_72^{24}$
  - $P_2=p_8+p_92^8+p_{10}2^{16}+p_{11}2^{24,}$ $P_3=p_{12}+p_{13}2^8+p_{14}2^{16}+p_{15}2^{24}$
- Same for Output c_0, …, c_{15} = C_0, C_1, C_2, C_3
- Output of round r designated R_0^r, R_1^r, R_2^r, R_3^r
- $R_0^r=P_0$, $R_1^r=P_1$, $R_2^r=P_2$, $R_3^r=P_3$

Twofish

PHT:

$a' = a+b \pmod{2^{32}}$

$b' = a+2b \pmod{2^{32}}$

$$\text{MDS}= \begin{matrix} \texttt{0x01} & \texttt{0xef} & \texttt{0x5b} & \texttt{0x5b} \\ \texttt{0x5b} & \texttt{0xef} & \texttt{0x5b} & \texttt{0x01} \\ \texttt{0xef} & \texttt{0x5b} & \texttt{0x01} & \texttt{0xef} \\ \texttt{0xef} & \texttt{0x01} & \texttt{0xef} & \texttt{0x5b} \end{matrix}$$

GF(256) calculations (MDS) use modulus $x^8+x^6+x^5+x^3+1$ over GF(2).

Twofish

- Input Whiten
  - $R_0^0=P_0 \oplus K_0$, $R_1^0=P_1 \oplus K_1$,
  - $R_2^0=P_2 \oplus K_2$, $R_3^0=P_3 \oplus K_3$
- 16 Keyed Rounds
  - $F_1(X,Y,r)$, $F_2(X,Y,r)$ defined later
  - $R_0^{r+1}= \mathrm{ror}(R_1^r \oplus F_1(R_0^r, R_1^r,r+1), 1)$
  - $R_1^{r+1}= \mathrm{rol}(R_1^r \oplus F_2(R_0^r, R_1^r,r+1), 1)$
  - $R_2^{r+1}=R_0^r$, $R_3^{r+1}=R_1^r$
- Output Whiten (after switching left and right blocks)
  - $C_0 =R_3^{16} \oplus K_{36}$, $C_1 =R_4^{16} \oplus K_{37}$,
  - $C_2 =R_0^{16} \oplus K_{38}$, $C_3 =R_1^{16} \oplus K_{39}$
- $F_1(X,Y,r) = g(X)+g(\mathrm{ror}(Y,8))+K_{2r+4} \pmod{2^{32}}$
- $F_2(X,Y,r) = g(X)+2g(\mathrm{ror}(Y,8))+K_{2r+5} \pmod{2^{32}}$
- g(x)= h(x,S), where h and S are defined below

Twofish Key Schedule

$$\mathrm{RS}= \begin{matrix} 01 & a4 & 55 & 87 & 5a & 58 & db & 9e \\ a4 & 56 & 82 & f3 & 1e & c6 & 68 & e5 \\ 02 & a1 & fc & c1 & 47 & ae & 3d & 19 \\ a4 & 55 & 87 & 5a & 58 & db & 9e & 03 \end{matrix}$$

k= 2, Key M consists of 16 bytes m_0, m_1, …, m_{15} or 4 32-bit words (little endian) M_0 , M_1, M_2, M_3.

$M_e = M_0 \;,\; M_2$

$M_o = M_1 \;,\; M_3$

$(s_{i,0}, s_{i,1}, s_{i,2}, s_{i,3})^T =$ RS $(m_{8i}, m_{8i+1}, \ldots, m_{8i+7})^T$, k= 0,1

Twofish key schedule and S-Boxes

$r = 2^{24}+2^{16}+2^{6}+1$

$A_i = h(2i\ r, M_e)$

$B_i = rol(h((2i+1)\ r, M_o),8)$

$K_{2i} = (A_i + B_i) \pmod{2^8}$

$K_{2i+1} = rol((A_i + 2B_i) \pmod{2^8}),9)$

$S_i = s_{i,0} + s_{i,1}2^8+s_{i,2}2^{16}+s_{i,3}2^{24}$

$S=(S_1, S_0)$

The Function h

$h(X, L_0, L_1)$

$l_{i,j} = \mathrm{int}(L_i/2^{8j}) \pmod{2^8}$

$x_j = \mathrm{int}(X/2^{8j}) \pmod{2^8}$

$y_{i,j} = x_j$

$y_0 = q_1[q_0[q_0[y_{2,0}] \oplus l_{1,0}] \oplus l_{0,0}]$

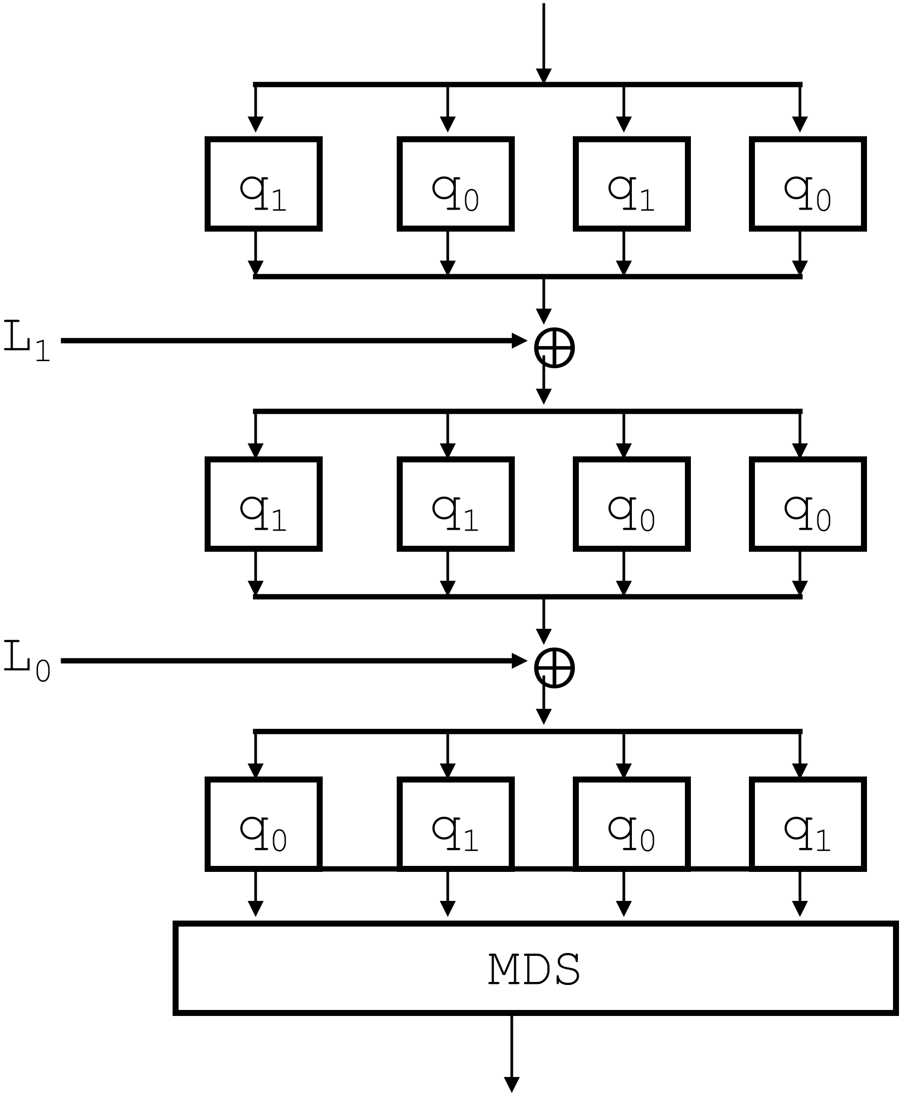
$y_1 = q_0[q_0[q_1[y_{2,1}] \oplus l_{1,1}] \oplus l_{0,1}]$

$y_2 = q_1[q_1[q_0[y_{2,2}] \oplus l_{1,2}] \oplus l_{0,2}]$

$y_3 = q_0[q_1[q_1[y_{2,3}] \oplus l_{1,3}] \oplus l_{0,3}]$

$(z_0, z_1, z_2, z_3)^T = \mathrm{MDS}(y_0, y_1, y_2, y_3)^T$

The Function h



q_0, q_1

For q_0

t_0= [8 1 7 d 6 f 3 2 0 b 5 9 e c a 4]

t_1= [e c b 8 1 2 3 5 f 4 a 6 7 0 9 d]

t_2= [b a 5 e 6 d 9 0 c 8 f 3 2 4 7 1]

t_3= [d 7 f 4 1 2 6 e 9 b 3 0 8 5 c a]

For q_1

t_0= [2 8 b d f 7 6 e 3 1 9 4 0 a c 5]

t_1= [1 e 2 b 4 c 3 7 6 d a 5 f 9 0 8]

t_2= [4 c 7 5 1 6 9 a 0 e d 8 2 b 3 f]

t_3= [b 9 5 1 c 3 d e 6 4 7 f 2 0 8 a]

q_0, q_1

a_0= int(x/16), b_0= x (mod 16)

a_1= $a_0 \oplus b_0$, b_1= $a_0 \oplus \mathrm{ror}_4(b_0, 1) \oplus 8a_0$

a_2= $t_0[a_1]$ b_2=$t_1[b_1]$

a_3= $a_2 \oplus b_2$, b_3= $a_2 \oplus \mathrm{ror}_4(b_2 ,1) \oplus 8a_2$

a_4= $t_2[a_3]$, b_4=$t_3[b_3]$

y= $16b_4+a_4$

Review: Arithmetic of GF(2^n)

- Suppose m(x) is an irreducible polynomial of degree n over GF(2): $m(x)= x^n + m_{n-1} x^{n-1} + \ldots + m_{0.}$
- Let a(x) and b(x) be polynomials of degree <n. They form a vector space of dimension n over GF(2). Coefficients of like exponent "add": $(a_{n-1} x^{n-1} + \ldots + a_0)+ (b_{n-1} x^{n-1} + \ldots + b_0)= (a_{n-1}+ b_{n-1})x^{n-1} + \ldots + a_0 + b_0)$
- Euclidean algorithm: for a(x), b(x) polynomials of degrees mcn, there are polynomials q(x), r(x), deg r(x) <n such that a(x)=q(x)b(x)+r(x)
- Polynomials over GF(2) modulo m(x) form a field (with 2^n elements). Multiplication is multiplication of polynomials mod m(x).
- Inverses exist : If a(x) and b(x) are polynomials their greatest common denominator d(x) can be written as

  d(x)= a(x)u(x)+b(x)v(x) for some u(x), v(x).

In particular, if a(x) and b(x) are co-prime: 1= a(x)u(x)+b(x)v(x) for some u(x), v(x).

Example of multiplication and inverse

- m(x)= x^2 +x +1. m(x) is irreducible (otherwise it would have a root in GF(2)
- x+(x+1) =1, 1+(x+1)= x
- (x+1)(x+1)= x^2+2x+1=x^2+1= (x) + (x^2 +x +1)= x (mod m(x))
- (x+1) and m(x) are co-prime in fact,

  1= (x+1)(x) + (x^2+x+1)(1)
- So "x" is the multiplicative inverse of "x+1" in GF(4).
- Usually elements of GF(2^n) are written in place notation so x^5+x^3+x^2+1= 101101.

Rijndael Overview

- Input
  - p consisting of N_b words
  - k with N_k words (N_k= 4,6,8)
- State
  - 4 rows, N_b columns
- Key
  - 4 rows, columns
- Output
  - c consisting of N_b words
- All tables filled first column first $s_{0,0}$, $s_{1,0}$, $s_{2,0}$, $s_{3,0}$, $s_{0,1}$, …
-
- Design Philosophy
  - Wide Trails
- 32-bit word operations
- Non-linear substitution uses arithmetic over GF(2)
- Mixing uses polynomial arithmetic mod (x^4+1)

Rijndael Round Structure

N_r= max(N_k, N_b)+6

N_r	**N_b=4**	**N_b= 6**	**N_b=8**
N_k=4	10	12	14
N_k=6	12	12	14
N_k=8	14	14	14

Rijndael State Layout

State: $s_{i,j}$, i= Nb (mod 4), j= [Nb/4], Nb=4j+i

For Nb= 4

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

Rijndael Key Layout

- Keys: $k_{i,j}$, i= Nk (mod 4), j= [Nk/4], for Nk= 4

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Rijndael Algorithm

```
Rijndael (p, k, Nb, Nk)  {
    ComputeRoundKeys(K, W[0…Nr])
    state= p
    AddRoundKey(0, state)
    for (i=1, i<=Nr, i++) {
        for each byte, b in state
            ByteSub(b)
        ShiftRow(state)
        if(i<Nr)
            MixCol(state)
        AddRoundKey(i, state)
        }
    c= state
    }
```

Inverse Rijndael Algorithm

```
InvRijndael (c, k, Nb, Nk)  {
    ComputeRoundKeys(K, W[0…Nr])
    state= c
    for (i=0, i<Nr, i++) {
        AddRoundKey(Nr-i, state)
        if(i>0)
            InvMixCol(state)
        InvShiftRow(state)
        for each byte, b in state
            InvByteSub(b)
        }
    AddRoundKey(0, state)
    p= state
    }
```

ByteSub Primitive

```
ByteSub(b)
   if b==0
      t= 0
   else
      t= $b^{-1}$
return(Mt + a)
```

M= circ(1,0,0,0,1,1,1,1)

a= $(1,1,0,0,0,1,1,0)^T$

Arithmetic over GF(2) with m(x)= $x^8+x^4+x^3+x+1$.

ByteSub Data

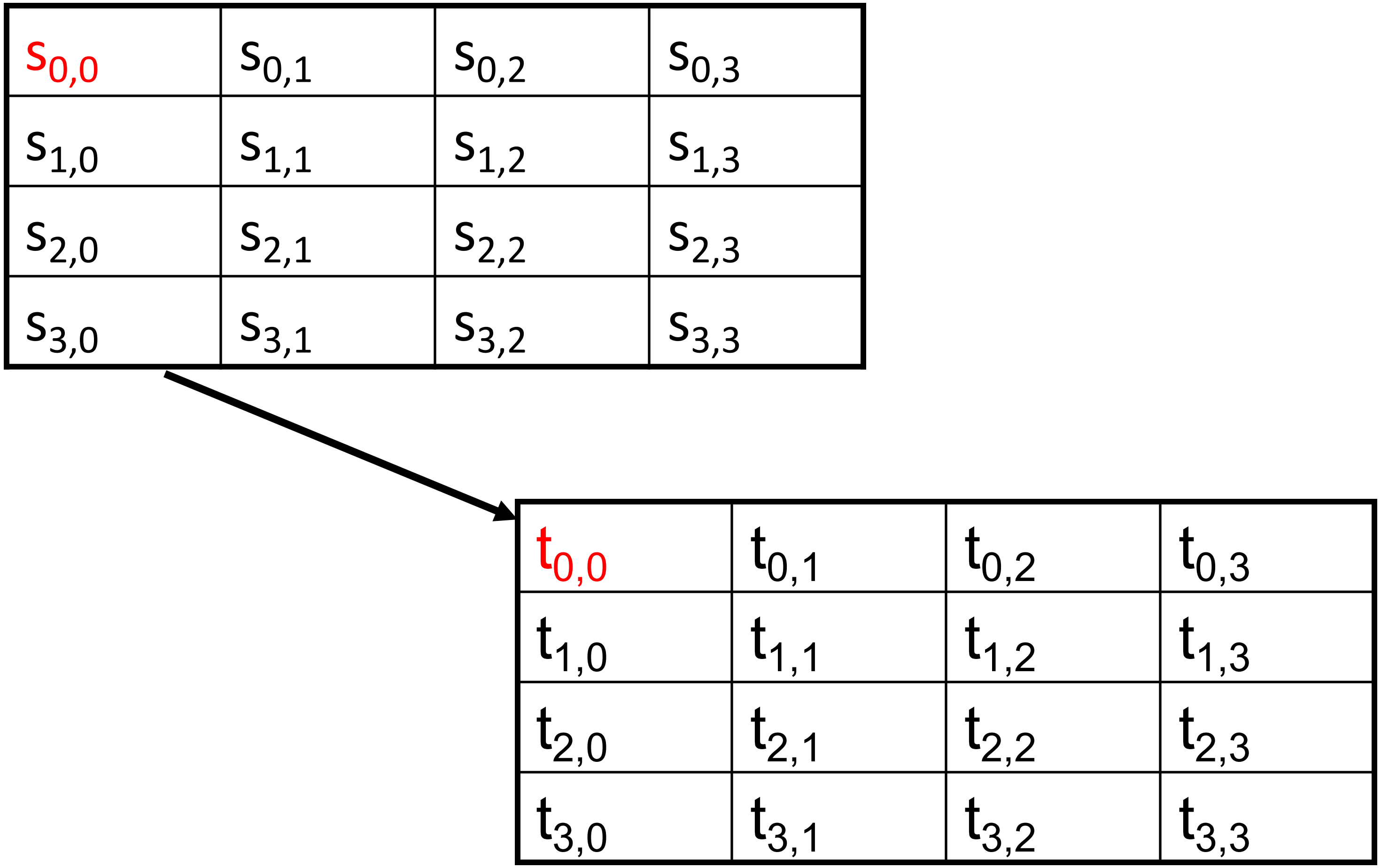
M:

1	0	0	0	1	1	1	1
1	1	0	0	0	1	1	1
1	1	1	0	0	0	1	1
1	1	1	1	0	0	0	1
1	1	1	1	1	0	0	0
0	1	1	1	1	1	0	0
0	0	1	1	1	1	1	0
0	0	0	1	1	1	1	1

a:

1
1
0
0
0
1
1
0

Bytesub



$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

$t_{0,0}$	$t_{0,1}$	$t_{0,2}$	$t_{0,3}$
$t_{1,0}$	$t_{1,1}$	$t_{1,2}$	$t_{1,3}$
$t_{2,0}$	$t_{2,1}$	$t_{2,2}$	$t_{2,3}$
$t_{3,0}$	$t_{3,1}$	$t_{3,2}$	$t_{3,3}$

Rijndael Primitives

```
ShiftRow(state)
    shift row 1 by 0.
    shift row 2 by 1.
    shift row 3 by 2 if Nb<8, 3 otherwise.
    shift row 3 by 3 if Nb<8, 4 otherwise.
MixCol(state)
    multiply each column of state by c(x) (mod $x^4+1$)
    $c(x)= 0x03\ x^3 + 0x01\ x^2 + 0x01\ x + 0x02$
InvMixCol(state)
    multiply each column of state by d(x) (mod $x^4+1$)
    $d(x)= 0x0b\ x^3 + 0x0d\ x^2 + 0x09\ x + 0x0e$
AddRoundKey(i, state)
    state= state + W[i]
```

ShiftRow

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

➔

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,3}$	$s_{1,0}$	$s_{1,1}$	$s_{1,2}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

Before ShiftRow

```
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f
```

After ShiftRow

```
0x00 0x05 0x0a 0x0f 0x04 0x09 0x0e 0x03
0x08 0x0d 0x02 0x07 0x0c 0x01 0x06 0x0b
```

MixCol

$s_{0,0}$	$s_{0,1}$	$s_{0,3}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,3}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,3}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,3}$	$s_{3,3}$

$t_{0,0}x^3+t_{1,0}x^2+t_{2,0}x+t_{3,0}=$
$(0x03x^3+0x01x^2+0x01x+0x02)$ x $(s_{0,0}x^3+s_{1,0}x^2+s_{2,0}x+s_{3,0})(\text{mod } x^4+1)$

$t_{0,0}$	$s_{0,1}$	$s_{0,3}$	$s_{0,3}$
$t_{1,0}$	$s_{1,1}$	$s_{1,3}$	$s_{1,3}$
$t_{2,0}$	$s_{2,1}$	$s_{2,3}$	$s_{2,3}$
$t_{3,0}$	$s_{3,1}$	$s_{3,3}$	$s_{3,3}$

RoundKeys

```
ComputeRoundKeys(K[4*Nk], W[Nb*(Nr+1)]) {
    for(i=0; i<Nk; i++)
        W[i]= (K[4i], K[4i+1], K[4i+2], K[4i+3])
    for(i=Nk; i<Nb*Nr+1); i++) {
        t= W[i-1]
        if((i mod Nk)==0)
            t= SubByte(RotByte(t)) + RCon(i/Nk)
        else if( (i mod Nk)==0)    // only if Nk>6
            t=SubByte(t)           // only ifNk>6
        }
    W[i]= W[i-Nk] + t
    }
```

Roundkeys Primitives

```
SubByte(w)
    w= ByteSub(w)

RotByte(w= (a, b, c, d))
    w= (b, c, d, a)

RCon[i]= (RC[i], 0x00, 0x00, 0x00);
RC[1]= 0x01
RC[i+1]=  RC[i]**(i) [multiply by “x” in polynomial
representation]
```

Cryptographic Effect

- Linear Mixing (diffusion)
  - MixCol
  - ShiftRow
- Non-Linear Mixing (confusion)
  - ByteSub
- Avalanche
  - MixCol
  - ShiftRow
  - RoundKeys

Design Criteria for ByteSub

- Invertibility
- Minimize largest non-trivial correlation between input and output (Linear resistance)
- Minimize max xor table (Differential resistance)
- Complexity of Algebraic expression in GF(2^8)
- Simplicity of description

Design Criteria for Shiftrow

- Four different offsets
- Resistance against truncated differentials
- Resistance against square attack
- Simplicity

Design Criteria for KeySched

- Invertibility
- Speed
- Eliminate symmetry with round constants (weak key resistance, related key resistance)
- Diffusion of key differences
- Partial knowledge of cipher key doesn’t reveal others
- Round differences don’t reveal cipher key differences
- Don’t need to precompute entire schedule
- Simplicity

Branch Number

- Let W(a)= number of non-zero (active) bytes
- Branch Number of F= $\min_{a \neq 0}$ W(a)+W(F(a))
- Prop ratio of differential trail $\sim$ prop ratio of active S-boxes
- Correlation of linear trail $\sim$ product of correlations of active S-boxes
- Wide Trail Strategy

Differential Trail

- If β= $\rho^{(r)}\ \rho^{(r-1)} \dots \rho^{(1)}$, $\mathbf{\Omega}$= (ω(0), ω(1), …, ω(r)) is a differential trail whose probability is the number of a(0) for which the differential tail follows the difference pattern divided by the number of possible a(0).
- The weight of a differential trail is the sum of the weights of its differential steps.: $w_r(\mathbf{\Omega}) = \sum_i w^{r(i)} (\omega(i\text{-}1), \omega(i))$.
- The differential trail imposes restrictions on the intermediate states a(i).
- **Theorem**: $Pr(a', b') = \sum_{\omega(0)=a', \omega(r)=b'} Pr(\mathbf{\Omega})$, $Pr(\mathbf{\Omega}) \sim \exp_2(-w_r(\mathbf{\Omega}))$ where $w_r(\mathbf{\Omega}) = \sum_i w^{r(i)}(\omega(i\text{-}1), \omega(i))$..

Weight Bundle

Define $w_b(a)$ as the bundle weight of a.

$B_d(\theta)$= $\min_{(a,\ b\neq a)}$ ($w_b(a\oplus b)+w_b(\theta(a) \oplus \theta(b))$).

$B_l(\theta, a)$= $\min_{(a,\ b,\ C(a\cdot x,\ b\ \cdot\theta(x))\ \neq 0)}$ ($w_b(a)+w_b\,(b)$).

- **Theorem**: In an alternating key block cipher with $\gamma\lambda$ round functions, the number of active bundles in a two round trail is ≧ the bundle branch number of λ. If φ= $\gamma\theta\gamma\lambda$ is a four round function, $B(\varphi)$ ≧ $B(\lambda)$x$B^{c(\varphi)}$ where B can be either the linear or differential branch number. The linear and differential branch numbers for an AES round is 5.

- Inverse provides linear/differential immunity, linear diffusion provides algebraic complexity.

Design strategy for Rijndael

- Choose number of rounds so that there is no correlation over all but a few rounds with amplitude significantly larger than $2^{nb/2}$ by insuring there are no linear trails with correlation contribution above $nk^{-1}2^{nb/2}$ and no differential trails with weight below nb.
- Examine round transformations ρ=$\lambda\gamma$, where l is the mixing function and γ is a bricklayer function that acts on bundles of nt bits. Block size is nb=m nt. The correlation over γ is the product of correlations over different S-box positions for given input and output patterns. Define weight of correlation as -lg(Amplitude).
- If output selection pattern is ≠0, the S-box is active. Looking for maximum amplitude of correlations and maximum difference propagation probability.
- The weight of a trail is the sum of the active S-box positions, so it is greater than or equal to number of active S-boxes times the minimum correlation weight per S-box.
- Wide trail: design round transformations so there are no trails with low bundle weight.

Rijndael Performance on 200MHz PII

(KeyLen, BlockLen)	Seed (Mb/sec)	Cycles/Blk
(128,128)	70.5	363
(192, 128)	59.3	432
(256, 128)	51.2	500

AES Finalist Bakeoff

	MARS	RC6	Rijndael (AES)	Serpent	Twofish
General Security	3	2	2	3	3
Implementation	1	1	3	3	2
SW Perf	2	2	3	1	1
Smart Card Perf	1	1	3	3	2
HW Perf	1	2	3	3	2
Design features	2	1	2	1	3

Score: 1 (low) to 3 (high). From NIST report 2 Oct 2000.

Modes of operation, block ciphers as primitives for integrity operations

Padding

- Suppose $E_K(P)$ has a block size of n bytes
  - For AES-128, n=16.

- If the message, M, to be encrypted is m bytes, we do
- the following:
  - Break M into k=int((m+15)/n) byte blocks, P_1, P_2, …,P_k
  - blocks. The last block may be only partially filled.
  - If P_k is partially full, append the byte 0x80 and as many 0x00
  - bytes as needed to fill this block. Encrypt the now full blocks P_1,
  - P_2, …,P_k using whatever block cipher mode of operation is
  - employed.
  - If P_k is full, append an additional block, P_{k+1} , of n bytes, the first
  - byte of the new block begins with the byte 0x80 followed by n-1
  - zero bytes, 0x00 bytes. Encrypt the blocks P_1, P_2, …,P_{k+1} using whatever block cipher mode of operation is employed

Padding

- With this padding, the message is “uniquely readable.”

- Decrypt each block of the transmitted cipher-text,
- C_1, C_2, …,C_k, according to the block cipher mode of
- operation employed, to obtain, P_1, P_2, …,P_k.
  - Check that the final (up to n) bytes have the prescribed
  - padding of the form 0x80 0x00*. If not, this is an error.
  - If the pad is correct, remove the padding to reconstruct the
  - original plain text.

ECB

- Given padded input blocks to obtain, P_1, P_2, …,P_k.
  - C_i= $E_K(P_i)$
  - P_i= $E_K^{-1}(C_i)$

- If a block of plaintext is repeated, the corresponding
- cipher text blocks are identical.
- What happens if we use this to encode only two
- messages?
  - “Attack”
  - “Sleep”
- Don’t use ECB

CBC

- To encrypt in CBC mode, given padded input blocks to obtain,
- P_1, P_2, …,P_k.
  - Generate a random n-byte block IV. Really, random.
  - P_0=IV
  - C_0= P_0
  - $C_{i+1}= E_K(P_i \oplus C_i)$
  - Transmit C_0, C_1, …,C_k.
- To decrypt C_0, C_1, …,C_k:
  - $P_i= E_K^{-1}(C_i) \oplus C_{i-1}$, i= 1,2,…,k

- Same plaintext block has different encryption depending on
- position in message stream.

Joux's padding attack on CBC

- We want to decrypt a block c= $c_1||c_2\ldots||c_{16}$ corresponding to plaintext p= $p_1||p_2||\ldots||p_{16}$ which was padded and CBC encrypted.
- Suppose we have a padding oracle which returns "padding error" or "success" when we submit a message to decrypt.
- Choose a randomly selected block $r=r_1||r_2\ldots||r_{16}$.
- Send the oracle the message r||c
  - If the oracle returns "success," $r\oplus E_K^{-1}(c)$ must be a valid pad.
    - Most likely, ???????????????0x80.
    - Padding could have been 0x80 (0x00*) but that's easy to incorporate.
    - So $E_K^{-1}(c)=r\oplus p=$???????????????0x80. Thus $p_{16}=r_{16}\oplus 0x80$.
  - If oracle returns "padding error", bump r_{16}. Eventually, we find
  - $p_{16}=r_{16}\oplus 0x80$ for some r_{16}. Thus we find p_{16} in at most 2^8 steps.
- Now set $r_{16}=r_{16}\oplus 0x80$, last plaintext byte will be 0x00.
  - Repeat with r_{15} to get p_{15}, etc.
- Decrypting all of c takes at most 16 x 2^8 oracle calls.

CTR

- To encrypt in CTR, given padded input blocks to obtain, P_1, P_2, …,P_k:
  - Generate a random n, 32-byte block IV.
  - CTR_0=IV||0^{32}
  - CTR_{i+1}=CTR_i+1
  - C_i= $E_K(CTR_i)\bigoplus P_i$
  - Transmit CTR_0, C_1, …,C_k.
- To decrypt IV, C_1, …,C_k:
  - P_i= $E_K\,(CTR_i)\bigoplus C_i$
- Same plaintext block has different encryption
- depending on position in message stream.

Cryptographic hashes

- Pre-image resistance:
  - Given y: y=h(x), x, unknown, it is computationally infeasible to compute x
- Second pre-image resistance:
  - Given y and x with y=h(x), it is computationally infeasible to compute x’≠x: h(x)=h(x’).
- Collision resistance:
  - It is computationally infeasible to compute x’ ≠x: h(x)=h(x’).

What cryptographic hashes good for

- Unforgeable fixed length identifier
  - If hash(MyProgram.exe)= x and you find a program Unknown.exe with hash(Unknown.exe)= x then Unknown.exe is MyProgram.exe.
  - If I make a promise written in ASCII text, MyPromise.txt, and $\text{Promise}_{\text{Hash}}$= hash(MyPromise.txt), you and I can give $\text{Promise}_{\text{Hash}}$ to an escrow agent without telling them the promise. If we get into a dispute and I claim I never made the promise, you can go to the agent with IClaimJohnPromised.txt and ask them to hash it, if hash(IClaimJohnPromised.txt)= $\text{Promise}_{\text{Hash}}$then you win!
  - Message authentication codes
  - Suppose I send you an encrypted message, C and you decrypt it as P.
  - How do you know the cipher text wasn't tampered with?
  - I also send Hash($K_{\text{integrity}}$||C). $K_{\text{integrity}}$ is another secret key you and I share.
  - If you compute Hash($K_{\text{integrity}}$||C) after receiving the message and it's the
  - same as the hash I sent, the message was correctly transmitted (and the
  - corresponding plaintext is correct).

Block cipher based hash functions

- Padded message to hash is M= m[0], m[1],…m[k]
  - H[0] = IV
  - $H[i+1]= E_{m[i]}(H[i]) \oplus H[i]$
  - Hash value is H[k].
- Coppersmith (85) showed that $H[i+1]= E_{m[i]}(H[i])$ was
- susceptible to collision attacks
- This hash construction takes an arbitrarily long set of bits
- and maps it to an 8n-bit binary string.

Hash Padding

- Extension attack:
  Without padding, hash(a)=hash(b) → hash(a||c)=hash(b||c).
- Hash pad scheme:
  Append 0x80 (0x00*) like block cipher except for last 64-bits then append message length size (64 bits)

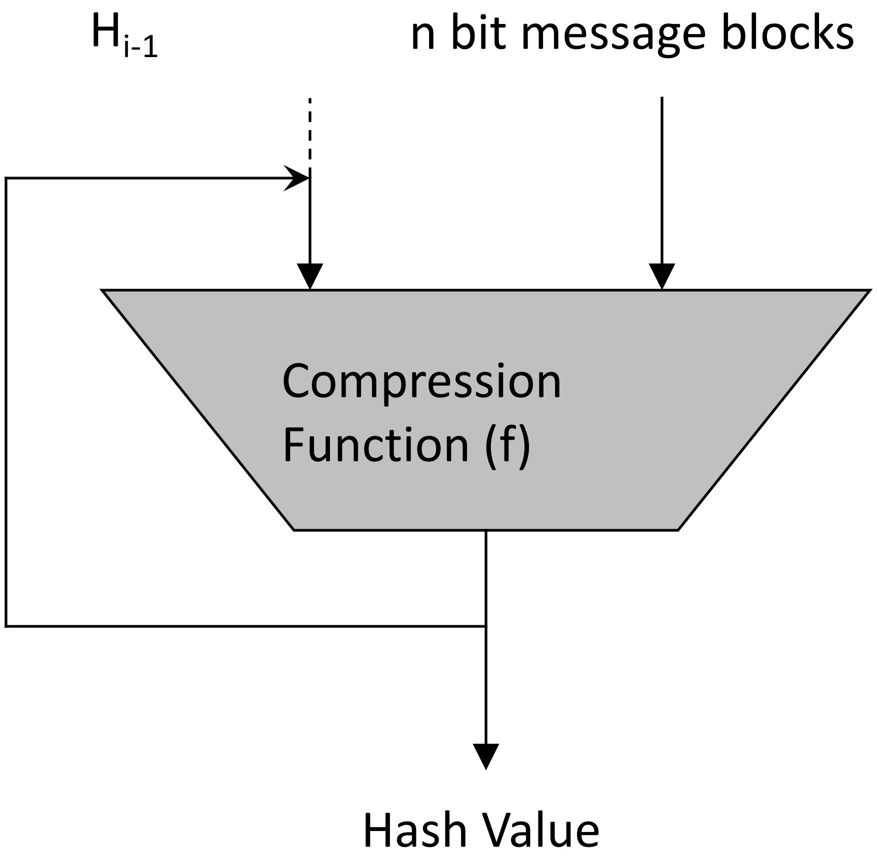
Block cipher based hash functions

- E(k,m) acts as a “compression function.”
- Ideal cipher: For random k, x →E(k, x) acts like a random
- permutation. Consequence: each of the possible 2^n images is equally likely
- For Davis Meyer:
  - $\Pr[E(m,h) \oplus h = h'] = \Pr[E(m,h) = h' \oplus h] = \Pr[E(m,h) = h''] = 2^{-n}$.
- $\Pr[E(m,h) \oplus h = E(m,h') \oplus h'] = 2^{-n/2}$ (Rogaway-Shrimpton).
- Collision resistance can be no better than $2^{-n/2}$:
  - If there are N distinct outputs of a hash and the has function is “perfectly random,” after 2 output, the probability that the second pick is not the same as the first is (1-1/N). After k picks, the probability of “non-collision” is $(1-1/N)(1-2/N)\ldots(1-(k-1)/N) \approx 1-\exp(-k^2/(2n))$. This is about ½ when $k = N^{1/2}$.

Authenticated Encryption

- Example with CTR mode
  - Added input blocks to obtain, P_1, P_2, …,P_k, shared encryption key K, shared integrity key $K_{integrity}$.
  - Generate a random n-4 byte block IV.
  - CTR_0=IV||0^{32}
  - CTR_{i+1}=CTR_i+1
  - C_i= $E_K(CTR_i) \bigoplus P_i$
  - T=MAC($K_{integrity,}$C)= hash($K_{integrity}$||C)
  - Transmit IV, C_1, …,C_k, T.
  - Actually transmit
  - T=MAC($K_{integrity,}$C)= hash(hash($K_{integrity}$||C)||$K_{integrity}$)
  - to avoid "extension" attacks.

General Merkle/Damgard Construction



Input: $x=x_1||\ldots||x_t$
Input is usually padded

H_0= IV
H_i= $f(H_{i-1}, x_i)$
$h(x)$= $g(h_t)$

Graphic by Josh Benaloh

Proofs about compression function

- **Theorem:** If g: $\{0,1\}^m \rightarrow \{0,1\}^n$, for a sequence of n bit blocks, **x**= $x_1, x_2, \ldots, x_t$, we can define a hash function h: $\{0,1\}^* \rightarrow \{0,1\}^n$ by H_0= c, H_{i+1}= g($H_i||x_i$) with h(x)=H_t. h is collision resistant if g is.

  - Proof: Let **x**= $x_1, x_2, \ldots, x_t$ and **x'**= $x_1', x_2', \ldots, x_{t'}'$ be two strings with h(**x**)=h(**x'**) and let H_i, $H_{i'}$ be the intermediate values. Suppose there is an i<t: $H_{t-i}=H_{t'-i}'$ and $H_{t-i-1}{}^{1}H_{t'-i-1}'$. Then g($H_{t-i-1}||x_i$)= g($H_{t-i-1}'||x_i'$) so g is not collision resistant. Otherwise, $H_i=H_i'$ and either $x_i=x_i'$, it, in which case there is nothing to prove or some $x_i \neq x_i'$ (but then g($H_i||x_i$) = g($H_i'||x_i'$) and again g is not collision resistant) or g(H_{t-1})= g($H_j'||x_j'$), j>t and again g is not collision resistant.

GCM

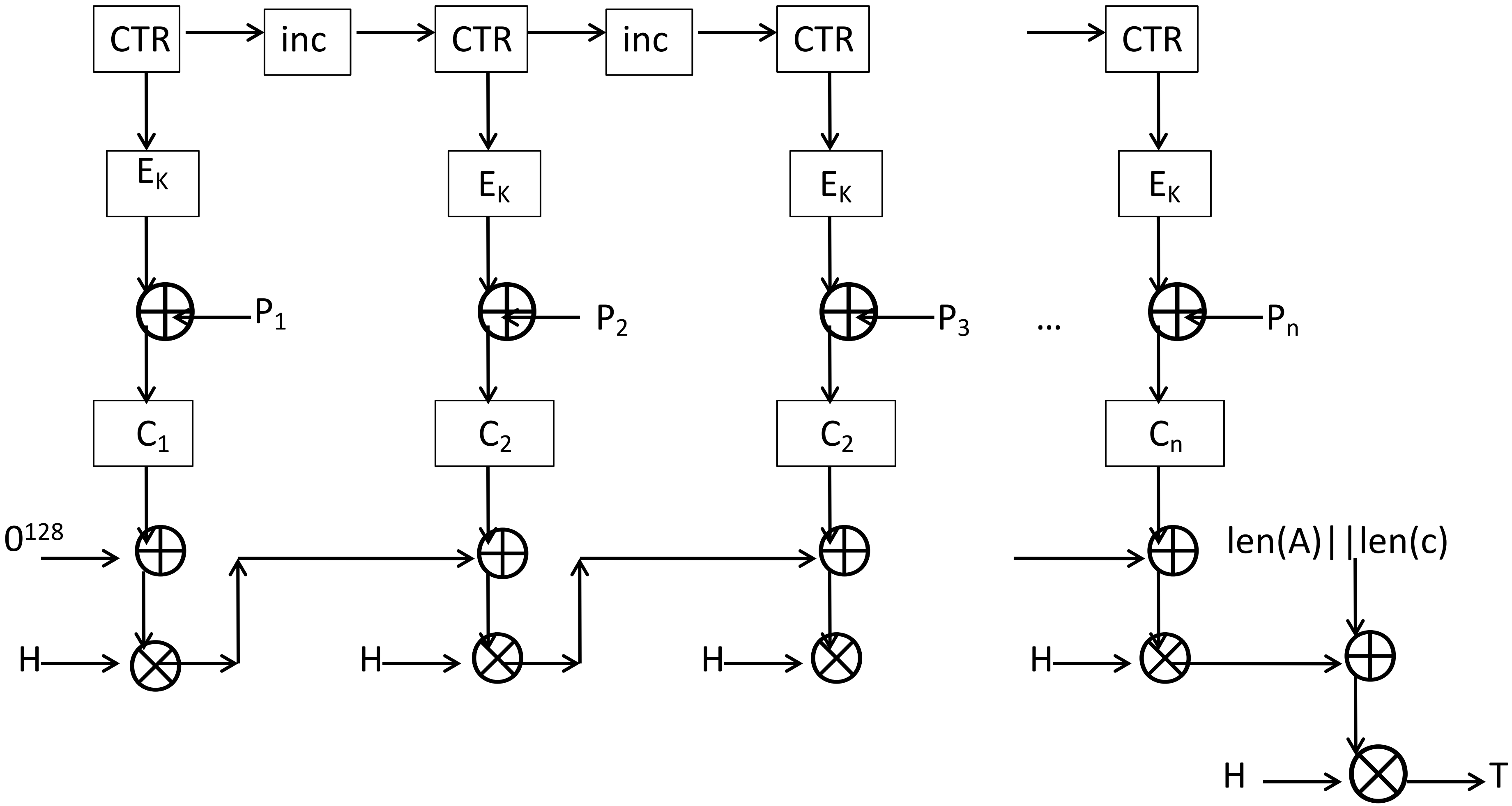
- Given m blocks of authentication data, $A=A_1, A_2,\ldots,A_m$ and n blocks, $P=P_1, \ldots, P_n$ of message, authentication length parameter, t.

```
GCM(K,IV,A,P)
    $H= E_K(0^{128})$, HeGF($2^{128}$) with minimal polynomial $p(x)= x^{128}+x^7+x^2+x+1$
    $J_0= IV \,||\, 0^{31} \,||\, 1$
    $C= GCTR(K,J_0+1,P)$
    $u= 128\lceil len(C)/128 \rceil - len(C)$, $v= 128\lceil len(A)/128 \rceil - len(A)$
    $S= GHASH(H,A||0^v||C||0^u||len(A)_{64}||len(C)_{64})$,
    $T= MSB_t(J_0,S)$
    return (C,T)
```

```
GHASH(H,X), $X= X_1, \ldots, X_m$
    $Y_0= 0^{128}$
    for(i=1;i<=m;i++)
        $Y_i= (Y_{i-1} \oplus X_i) \cdot H$
    return $Y_m$
```

```
GCTR(K,CTR,P), $P=P_1, \ldots, P_n$
    for(i-1; i<n,i++) {
        $C_i= (P_i \oplus E_K(X_i)$
        CTR= CTR+1
    $C_n{}^*= X_n{}^* \oplus MSB_r(E_K(CTR))$, $r=len(X_n{}^*)$
    return C
```

GCM


CTR
inc
CTR
inc
CTR
CTR
E_K
E_K
E_K
E_K
P_1
P_2
P_3
…
P_n
C_1
C_2
C_2
C_n
0^{128}
len(A)||len(c)
H
H
H
H
H
T

Padding attacks in the news

- “Lucky Thirteen: Breaking the TLS and DTLS Record
- Protocols,” Nadhem J. AlFardan and Kenneth G. Paterson

  We present a family of attacks that apply to CBC-mode in all TLS and DTLS implementations that are compliant with TLS 1.1 or 1.2, or with DTLS 1.0 or 1.2.

- http://www.isg.rhul.ac.uk/tls/TLStiming.pdf

End