AES and Attacks on Cryptographic Hashes

John Manferdelli
jlm@cs.washington.edu
jmanfer@microsoft.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
• Square begets Rijndael (1998)
  – Rijndael Designers: Vincent Rijmen and Joan Daemen
• 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
• FIPS 197 published 11/2001
AES

Plaintext → Key Schedule → Ciphertext

Key

r Rounds

k₁ k₂kr

AES

Plaintext → Ciphertext
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.”
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.
- Leverage our experience with RC5: use data-dependent rotations to achieve a high level of security.

RC6 Design Philosophy
Security against differential attacks

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

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

<table>
<thead>
<tr>
<th>Rounds</th>
<th>Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>$2^{56}$</td>
</tr>
<tr>
<td>12</td>
<td>$2^{117}$</td>
</tr>
<tr>
<td>16</td>
<td>$2^{190}$</td>
</tr>
<tr>
<td>20</td>
<td>$2^{238}$</td>
</tr>
<tr>
<td>24</td>
<td>$2^{299}$</td>
</tr>
</tbody>
</table>

RC6

Infeasible

Slide by Ron Rivest (Second AES Conference)
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, \( N_k \) columns

• Output
  • \( c \) consisting of \( N_b \) words

All tables filled first col first \( s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0}, s_{0,1}, \ldots \)
Rijndael Overview

- 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

\[ N_r = \max(N_k, N_b) + 6 \]

<table>
<thead>
<tr>
<th>( N_b )</th>
<th>( N = 8 )</th>
<th>( N = 6 )</th>
<th>( N = 4 )</th>
<th>( N_k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>14</td>
<td>12</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>( N_k )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( N_k = 8 )</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>( N_k = 6 )</td>
<td></td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( N_k = 4 )</td>
<td></td>
<td>14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rijndael State Layout

State: $s_{i,j}$, $i = \text{Nb} \pmod{4}$, $j = \lfloor \text{Nb}/4 \rfloor$, $\text{Nb}=4j+i$

For $\text{Nb}=4$

<table>
<thead>
<tr>
<th>$s_{0,0}$</th>
<th>$s_{0,1}$</th>
<th>$s_{0,2}$</th>
<th>$s_{0,3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_{1,0}$</td>
<td>$s_{1,1}$</td>
<td>$s_{1,2}$</td>
<td>$s_{1,3}$</td>
</tr>
<tr>
<td>$s_{2,0}$</td>
<td>$s_{2,1}$</td>
<td>$s_{2,2}$</td>
<td>$s_{2,3}$</td>
</tr>
<tr>
<td>$s_{3,0}$</td>
<td>$s_{3,1}$</td>
<td>$s_{3,2}$</td>
<td>$s_{3,3}$</td>
</tr>
</tbody>
</table>
Rijndael Key Layout

Keys: $k_{i,j}$, $i = Nk \mod 4$, $j = \lfloor Nk/4 \rfloor$

For $Nk = 4$

<table>
<thead>
<tr>
<th>$k_{0,0}$</th>
<th>$k_{0,1}$</th>
<th>$k_{0,2}$</th>
<th>$k_{0,3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_{1,0}$</td>
<td>$k_{1,1}$</td>
<td>$k_{1,2}$</td>
<td>$k_{1,3}$</td>
</tr>
<tr>
<td>$k_{2,0}$</td>
<td>$k_{2,1}$</td>
<td>$k_{2,2}$</td>
<td>$k_{2,3}$</td>
</tr>
<tr>
<td>$k_{3,0}$</td>
<td>$k_{3,1}$</td>
<td>$k_{3,2}$</td>
<td>$k_{3,3}$</td>
</tr>
</tbody>
</table>
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
}
```
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 $m \leq n$, 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 by following theorem: 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 $u(x)$, $v(x)$. 
Example of multiplication and inverse

- 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
  - \( m(x) = x^2 + x + 1 \). \( m(x) \) is irreducible (otherwise it would have a root in GF(2))
  - \( x + (x+1) = 1, 1 + (x+1) = x \)
  - \((x+1)(x+1) = x^2 + 2x + 1 = x^2 + 1 = (x) + (x^2 + x + 1) = x \pmod{m(x)}\)
  - \((x+1)\) 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 \( x^5 + x^3 + x^2 + 1 = 101101 \).
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)= x⁸+x⁴+x³+x+1.
### ByteSub Data

<table>
<thead>
<tr>
<th>a:</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>M:</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **ByteSub Data** represents a pattern of bytes that can be seen as a binary representation of various values.
Bytesub

<table>
<thead>
<tr>
<th></th>
<th>S₀,₀</th>
<th>S₀,₁</th>
<th>S₀,₂</th>
<th>S₀,₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₀,₀</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S₁,₀</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S₂,₀</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S₃,₀</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>t₀,₀</th>
<th>t₀,₁</th>
<th>t₀,₂</th>
<th>t₀,₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₀,₀</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t₁,₀</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t₂,₀</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t₃,₀</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>(s_{0,0})</th>
<th>(s_{0,1})</th>
<th>(s_{0,2})</th>
<th>(s_{0,3})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s_{1,0})</td>
<td>(s_{1,1})</td>
<td>(s_{1,2})</td>
<td>(s_{1,3})</td>
<td></td>
</tr>
<tr>
<td>(s_{2,0})</td>
<td>(s_{2,1})</td>
<td>(s_{2,2})</td>
<td>(s_{2,3})</td>
<td></td>
</tr>
<tr>
<td>(s_{3,0})</td>
<td>(s_{3,1})</td>
<td>(s_{3,2})</td>
<td>(s_{3,3})</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(s_{0,0})</th>
<th>(s_{0,1})</th>
<th>(s_{0,2})</th>
<th>(s_{0,3})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s_{1,3})</td>
<td>(s_{1,0})</td>
<td>(s_{1,1})</td>
<td>(s_{1,2})</td>
<td></td>
</tr>
<tr>
<td>(s_{2,2})</td>
<td>(s_{2,3})</td>
<td>(s_{2,0})</td>
<td>(s_{2,1})</td>
<td></td>
</tr>
<tr>
<td>(s_{3,3})</td>
<td>(s_{3,0})</td>
<td>(s_{3,1})</td>
<td>(s_{3,2})</td>
<td></td>
</tr>
</tbody>
</table>
MixCol

\[
t_{0,0}x^3 + t_{1,0}x^2 + t_{2,0}x + t_{3,0} = \left(0x03x^3 + 0x01x^2 + 0x01x + 0x02\right) x \left(s_{0,0}x^3 + s_{1,0}x^2 + s_{2,0}x + s_{3,0}\right) \pmod{x^4+1}
\]
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(Nk>6 && (i mod Nk)==0)
            t=SubByte(t)
        W[i]= W[i-Nk] + t
    }
}
RoundKeys Primitives

SubByte\( (w= (a,b,c,d))\))
\[\begin{align*}
w &= (\text{ByteSub}(a), \text{ByteSub}(b), \text{ByteSub}(c), \text{ByteSub}(d)) \\
\text{return}(w)
\end{align*}\]

RotByte\( (w= (a,b,c,d))\))
\[\begin{align*}
w &= (b,c,d,a) \\
\text{return}(w)
\end{align*}\]

\(\text{RCon}[i]= (\text{RC}[i], 0x00, 0x00, 0x00)\);
\(\text{RC}[1]= 0x01\)
\(\text{RC}[i+1]= \text{RC}[i]^{"0x2"} \quad [\text{multiply by "x"}]\)
# AES Finalist Bakeoff

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

<table>
<thead>
<tr>
<th></th>
<th>MARS</th>
<th>RC6</th>
<th>Rijndael (AES)</th>
<th>Serpent</th>
<th>Twofish</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Security</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Implementation</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>SW Perf</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Smart Card Perf</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>HW Perf</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Design features</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Algebraic Attacks - Preview

- XSL, Courtois, Pieprzyk, Murphy, Robshaw
  1. Generate equations of higher degree than the original equations by multiplying equation of an active S-box by passive S-box equations
  2. Solve the equations in the formal terms of the equations
- Estimate of linearly independent equations is necessary
- Claim that solving the equations for AES was possible because the estimated number of linearly independent equations was adequate generated excitement.
- Coppersmith cast doubt on the number of linearly independent equations.
Stream Ciphers

• Synchronous stream ciphers
  – The keystream is generated independently of the plaintext and the ciphertext
  – Using Keyed PRNG

• Asynchronous stream ciphers
  – The keystream is generated as a function of the key, $K$, and at most $t$ previous ciphertext symbols.
Stream Cipher Encryption and PRNGs

Plaintext: ■■■■■■■■■■■■■■■■■■■■
PRNG(seed): □□□□□□□□□□□□□□□□
Ciphertext: ■■■■■■■■■■■■■■■■■■■■

Encryption Equation: $c_j = p_j \oplus k_j$
Synchronous Stream using Linear Feedback Shift Register (LFSR)

Recurrence: \( s_{j+L} = \sum_{j=0, 1 \ldots L-1} a_j \cdot s_{j+L-1} \)

Polynomial: \( f(x) = \sum_{j=0, 1 \ldots L-1} a_j \cdot x^j - x^L \)

\( a_i, s_j \in F_q \)
LFSR-based keystream generator

- Nonlinear combination generators
- Nonlinear filter generators
**RC4**

**Initialization**

\[ S[0..255] = 0, 1, \ldots, 255 \]

\[ K[0..255] = \text{Key}, \text{Key}, \text{Key}, \ldots \]

for \( i = 0 \) to 255

\[ j = (j + S[i] + K[i]) \mod 256 \]

swap \( S[i] \) and \( S[j] \)

i=j= 0

**Iteration**

\[ i = (i + 1) \mod 256 \]

\[ j = (j + S[i]) \mod 256 \]

swap \( S[i] \) and \( S[j] \)

\[ t = (S[i] + S[j]) \mod 256 \]

Output \( S[t] \)
RC-4 Facts

• RC4 implements a permutation of the $2^N$, $N=2^n$, where $n=8$ is word size.

• RC4 cannot enter states $s$: $i=a$, $j=a+1$, $S[a+1]=1$. There are $N^2$ of these.

• Notation
  • $S_r, i_r, j_r$ and $t_r$ denote the RC4 state during initialization after using key words $[0, 1, \ldots, r]$.
  • $I(S)$, $J(S)$, $T(S)$, $Z(S)$ are the state indicies, output index and first output word of RC4 (i.e.- just after initialization is complete).
  • $T(S)=S[1]+S[S[1]]$, $Z(S)=S[T(S)]$.
  • Key is $l$-words long

• References
  • Fluhrer, Mantin and Shamir. Attacks on RC4 and WEP.
  • Mantin, Master’s Thesis.
Attacks on RC4 and FSRs

• Simple xor attack on stream without MAC or how to swindle your bank.

• Reproduce internal state
• Solve for “taps”
• Look for short cycles

• Alleged RC4 resists these.
  – RC4 is a good stream cipher if you throw away first bunch of bytes
Bias in Second Byte of RC4

Let $S_i$ be the state at time $i$ and let $\langle z_i \rangle$ be the output sequence.

Theorem: $P(\text{z}_2=0)= \frac{2}{N}$. (roughly twice what we expect from a random cipher)


Round 1:
- $i=1$, $X=S_0[1]+0$. Exchange $S_0[1]$ and $S_0[Y]$.

Round 2:

So $P(\text{z}_j = 0) \sim \frac{1}{N} + \frac{1}{N} (1-1/N) \sim \frac{2}{n}$.

By Bayes, if $z_2 = 0$, we can extract byte of state with probability $1/2$. 

Cryptographic Hashes

A cryptographic hash ("CH") is a "one way function," \( h \), from all binary strings (of arbitrary length) into a fixed block of size \( n \) (called the size of the hash) with the following properties:

1. Given \( y = h(x) \) it is infeasible to calculate a \( x' \neq x \) such that \( y = h(x') \). ("One way," "non-invertibility" or "pre-image" resistance). Functions satisfying this condition are called One Way Hash Functions (OWHF).

2. Given \( u \), it is infeasible to find \( w \) such that \( h(u) = h(w) \). (weak collision resistance, 2\textsuperscript{nd} pre-image resistance).

3. It is infeasible to find \( u, w \) such that \( h(u) = h(w) \). (strong collision resistance). Note 3\( \rightarrow \)2. Functions satisfying this condition are called Collision Resistant Functions (CRFs).

- Just like Symmetric ciphers ratio of work factor for computation of hash vs work factor to break hash should be very high.
- Adversary has complete information on computing hash and (obviously) can compute as many hashes from the target as she wants.
Observations on Cryptographic Hashes

- Hashes are a strong “checksum”
- OWHF and CRF conditions make CHs satisfy many of the properties of “random functions”
  - Small changes should create large changes (otherwise the pre-image of near neighbors are near neighbors making collisions easy to find)
  - Small input changes should be statistically unrelated (uncorrelated) to changes in a subset of the hash bits
  - Analysis of CHs very similar to Symmetric Cipher techniques

Popular practical cryptographic hashes
- MD4, MD5 (now “broken”)
- SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 (last 4 are “SHA-2”)
- RIPEMD
Observations

• Collision Resistance $\rightarrow$ 2nd pre-image resistance
• Let $f(x) = x^2 - 1 \pmod{p}$.
  – $f(x)$ acts like a random function but is not a OWHF since square roots are easy to calculate mod $p$.
• Let $f(x) = x^2 \pmod{pq}$.
  – $f(x)$ is a OWHF but is neither collision nor 2nd pre-image resistant
• If either $h_1(x)$ or $h_2(x)$ is a CRHF so is $h(x) = h_1(x) || h_2(x)$
• MDC+signature & MAC+unknown Key require all three properties
What are Hash Functions Good for?

• Modification Detection Codes (MDCs): This is a strong checksum (integrity check). Sometimes called “unkeyed” hashes.

• Message Authentication Code (MACs): If shared secret is part of the hash, two parties can determine authenticated integrity with CHs. Called “keyed hashes”.

• Message Digests (MDs): Encrypting (with private key) the CH of a message (its MD) acts as a certification that the message was “approved” by possessor of private key. This is called a Digital Signature. [Note: you could “sign’ the whole message rather than the hash but this would take oodles of time by comparison.]
What are Hash Functions Good for?

• Identity: Uniquely and securely identifies bit streams like programs. Hash is strong name for program.

• Entropy mixing: Since CHs are random functions into fixed size blocks with the properties of random functions, they are often used to “mix” biased input to produce a “seed” for a pseudo-random number generator.

• Password Protection: Store salted hash of password instead of password (Needham).

• Bit Commitment
One-Way Functions

Hashes come from two basic classes of one-way functions

– Mathematical
  • Multiplication: \( Z = X \cdot Y \)
  • Modular Exponentiation: \( Z = Y^X \pmod{n} \) (Chaum vP Hash)

– Ad-hoc (Symmetric cipher-like constructions)
  • Custom Hash functions (MD4, SHA, MD5, RIPEMD)
Chaum-vanHeijst-Pfitzmann Compression Function

• Suppose \( p \) is prime, \( q=(p-1)/2 \) is prime, \( a \) is a primitive root in \( F_p \), \( b \) is random.

• \( g: \{1,2,\ldots,q-1\}^2 \to \{1,2,\ldots,p-1\}, q=(p-1)/2 \) by:
  – \( g(s, t) = a^s b^t \pmod{p} \)

• Not used in practice: too slow.

• Reduction to discrete log:
  Suppose \( g(s, t) = g(u, v) \) can be found. Then \( a^s b^t \pmod{p} = a^u b^v \pmod{p} \).
  So \( a^{s-u} \pmod{p} = b^{v-t} \pmod{p} \). Let \( b = a^x \pmod{p} \). Then \( (s-u) = x(y-t) \pmod{p-1} \).
  But \( p-1 = 2q \) so we can solve for \( x \), thus determining the discrete log of \( b \).
Merkle/Damgards 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
Padding

• Standard technique
  – Let last message block have k bits. If k=n, make a new block and set k=0.
  – Append a 1 to last block leaving r=n-k-1 remaining bits in block.
  – If r>=64, append r-64 0s then append bit length of input expressed as 64 bit unsigned integer
  – If r<64, append n-r 0’s (to fill out block), append n-64 0’s at beginning of next block then append bit length of input expressed as 64 bit unsigned integer
Technique for CHs from Block Ciphers

Let input be $x = x_1 \parallel x_2 \parallel \ldots \parallel x_t$ where each $x_i$ is $n$ bits long. Let $g$ be a function taking an $n$ bit input to an $m$ bit input. Let $E(k, x)$ be a block cipher with $m$ bit keyspace and $n$ bit block. Let $H_0 = IV$.

Construction 1
$H_i = E(g(H_{i-1}), x_i) \oplus H_{i-1}$

Construction 2
$H_i = E(x_i, H_{i-1}) \oplus H_{i-1}$

Construction 3
$H_i = E(g(H_{i-1}), x_i) \oplus x_i \oplus H_{i-1}$

Note: Because of collisions $n$ should be $> 64$. Ideally, $m=n$ and $g = id$. DES with $n=64$ is too small. AES with $n=m=128$ is better.
Attacks on Cryptographic Hashes

- Birthday (Yuval) attacks
  - Probability of collision determined by “Birthday Paradox” calculation:
    - \((1 - 1/n) (1 - 2/n) \ldots (1-(k-1)/n) = (n!/(k!))/n^k\)
    - Probability of collision is >.5 when \(k^2 > n\).
    - Need \(2^{80}\) blocks for SHA.
    - \(1+x \leq e^x, \prod_{i=1}^{i=k} (1-i/n) \leq e^{-k(k-1)/(2n)}\)

- Dobbertin Attacks on MD4
  - Collision attack based on compression function weakness

- Biham, Chen, Chabaud, Joux, Wang et al, Differential attacks on RIPEMD-128, HAVAL, MD4, MD5, SHA-0, SHA-1
Attacks on Cryptographic Hashes

- Berson (1992) using differential cryptanalysis on 1 round MD-5.
- Boer and Bosselaers (1993), Pseudo collision in MD5.
- Biham and Chen (2004), Collisions in SHA-0.
- Chabaud and Joux (2004), Collisions in SHA-0.
- Wang, Feng, Lai, Yu, (2004), MD4, MD5, RIPEMD
- Wang et al, (2004, 2005), SHA-1

- SHA-1 has stood up best: best known theoretical attack (11/05) requires $2^{64}$ operations.
Prefix and suffix attacks

Hash(m₁ || m₂) = Hash(m₂), if internal state collides

To fix: h_{DBL}(h(m) || m)

HMAC: keyed-hash message authentication code

Two popular constructions

HMAC₀_K(x) = Hash(k | p | m | k), p is a pad

HMAC₁_K(x) = SHA-1(K ⊕ opad || SHA-1(K ⊕ ipad) || x)
A Cryptographic Hash: SHA-1

Slide by Josh Benaloh
SHA-0/1

Do until no more input blocks {
  If last input block
    Pad to 512 bits by adding 1 then 0s then 64 bits of length.
  $M_i =$ input block (32 bits)
    $i = 0, ..., 15$
  $W_t =$ $M_t$, $t = 0, ..., 15$
  $W_t =$ ( $W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}$ ) $\ll 1$
    $t = 16, ..., 79$
  $a =$ $A$; $b =$ $B$; $c =$ $C$; $d =$ $D$; $e =$ $E$
  for ($t = 0$ to $79$) {
    $x =$ ( $a \ll 5$ ) + $f_t$( $b$,$c$,$d$) + $e$+$W_t$+$K_t$
    $e =$ $d$; $d =$ $c$; $c =$ $b$ $\ll 30$
    $b =$ $a$; $a =$ $x$
  }
  $A += a$; $B += b$; $C += c$; $D += d$; $E += e$
}

Absence of this term is only difference between SHA-0 and SHA-1

$A =$ 0x67452301, $B =$ 0xefcdab89,
$C =$ 0x98badcfe, $D =$ 0x10325476
$E =$ 0xc3d2e1f0

$F_t(X,Y,Z) =$ $X \land Y \lor (\neg X)\land Z$,
    $t =$ 0, ..., 19
$F_t(X,Y,Z) =$ $X \oplus Y \oplus Z$,
    $t =$ 20, ..., 39
$F_t(X,Y,Z) =$ $X \land Y \lor (X \land Z) \lor (Y \land Z)$,
    $t =$ 40, ..., 59
$F_t(X,Y,Z) =$ $X \oplus Y \oplus Z$, $t =$ 60, ..., 79

$K_t =$ 0x5a827999, $t =$ 0, ..., 19
$K_t =$ 0x6ed9eba1, $t =$ 20, ..., 39
$K_t =$ 0x8f1bbcdc, $t =$ 40, ..., 59
$K_t =$ 0xca62c1d6, $t =$ 60, ..., 79
A Cryptographic Hash: SHA-1
A Cryptographic Hash: SHA-1

Depending on the round, the “non-linear” function $f$ is one of the following.

\[
f(X,Y,Z) = (X \land Y) \lor ((\neg X) \land Z)
\]
\[
f(X,Y,Z) = (X \land Y) \lor (X \land Z) \lor (Y \land Z)
\]
\[
f(X,Y,Z) = X \oplus Y \oplus Z
\]
A Cryptographic Hash: SHA-1

What’s in the final 32-bit transform?

• Take the rightmost word.
• Add in the leftmost word rotated 5 bits.
• Add in a round-dependent function $f$ of the middle three words.
• Add in a round-dependent constant.
• Add in a portion of the 512-bit message.
Breaking news on “Chinese” Attacks on Hashes

- Don’t use MD4 or you’ll look really really silly.
- Don’t use MD5.
- Don’t use RIPEMD-128
- SHA-1 appears to have collision attacks of the order $2^{64}$
- Use SHA-2 functions
  - Truncate to provide legacy compatibility if you have to (i.e. – gun to head)
  - Required by “Suite B” Standards
SHA-2

• FIPS 180-2, 8/02.
  – SHA-224 (truncated) added 2/04

• Great increase in mixing between bits of the words compared to SHA-1.

• US Patent 6,829,355
• Inventor: Glenn Lilly
• Assignee: NSA
• Can obtain source from
Other Cryptographic Hashes and Performance

<table>
<thead>
<tr>
<th>Hash Name</th>
<th>Block Size</th>
<th>Relative Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD4</td>
<td>128</td>
<td>1</td>
</tr>
<tr>
<td>MD5</td>
<td>128</td>
<td>.68</td>
</tr>
<tr>
<td>RIPEMD-128</td>
<td>128</td>
<td>.39</td>
</tr>
<tr>
<td>SHA-1</td>
<td>160</td>
<td>.28</td>
</tr>
<tr>
<td>RIPEMD-160</td>
<td>160</td>
<td>.24</td>
</tr>
</tbody>
</table>
What to take home

• Symmetric ciphers and hashes provide key ingredients for “distributed security”
  – Fast data transformation to provide confidentiality
  – Integrity
  – Public key crypto provides critical third component (trust negotiation, key distribution)
• It’s important to know properties of cryptographic primitives and how likely possible attacks are, etc.
  – Most modern ciphers are designed so that knowing output of n-1 messages provides no useful information about n\textsuperscript{th} message.
  – This has an effect on some modes of operation.
General Modern References

Blake, Seroussi, and Smart, Elliptic Curves in Cryptography, Cambridge
Bressoud and Wagon, Computational Number Theory. Key Press.
Bach and Shallit, Algorithmic Number Theory.
Biham and Shamir, Differential Cryptanalysis of DES. Springer.
Boneh, Twenty Years of attacks on RSA. Notices AMS.
Buchmann, Introduction to Cryptography. Springer.
Cohen, A Course in Computational Algebraic Number Theory. Springer.
Damgard, Lectures on Data Security. Springer.
Koblitz, Algebraic Aspects of Cryptography. Springer.
Konheim, Cryptography: A Primer. Wiley.
General Modern References

Landau, DES, AES, Survey article. Notices AMS. 
MacWilliams et. al., Theory of Error Correcting Codes. North Holland. 
Menezes, van Oorshot, Vanstone, Handbook of Applied Cryptography. 
(Online: http://www.cacr.math.uwaterloo.ca/hac/). CRC Press. 
Rhee, Cryptography and Secure Communications. 
Rivest, Class notes on Security and Crypto online. (web.mit.edu). 
Schneier, Applied Cryptography. Wiley. 
Stinson, Cryptography: Theory and Practice. CRC Press. 

Homework 7

1. We saw that a typical round of AES consisted of the following operations:
   
   for each byte, b in state
      ByteSub(b)
   ShiftRow(state)
   if(i<Nr)
      MixCol(state)
   AddRoundKey(i, state)

   For the 128 bit key, 128 bit block size version of Rijndael, using lookup tables to reduce the computations required and assuming basic operations (32 bit lookup, 32 bit xor, etc) all take about .001 microseconds and your code/data budget is under 16 MB, design a implementation of the round operations that is faster than implementing each of the primitive operations (ByteSub, ShiftRow, MixCol).

   How long does each round take (about)?

   Counter mode use of AES is used by selecting a nonce (n) and constructing cipher blocks AES_k(n||ctr), AES_k(n||ctr+1), AES_k(n||ctr+2),….

   The resulting bits are xored into the plaintext (as with the stream cipher).
   What properties of AES make this safe? Can the keystream be generated in parallel and stored for later use? What performance properties does this mode have over ECB?
Homework 7

2. Show that \( f(x) = x^2 \pmod{pq} \) is a One-Way Function but is not Collision Resistant, where \( p \) and \( q \) are prime.

3. Linear Feedback Shift Registers Cryptosystem:
Suppose \( X \) is a cryptosystem implemented by a 5 element linear feedback shift register which generates a psuedo random stream \( s_0, s_1, s_2, \ldots \) so

\[
\begin{align*}
    s_{n+5} &= a_4 s_{n+4} \oplus a_3 s_{n+3} \oplus a_2 s_{n+2} \oplus a_1 s_{n+1} \oplus a_0 s_n
\end{align*}
\]

If the first 10 output bits of the pseudo random generator are 1110100010, what are the next 3 bits? Assume \( n \) is the register length. About how many consecutive bits do you need to break a LFSR? How does this compare to a stream generator on an \( n \) bit state that is not linear?
Homework 7

4. Given i = 64, j = 245 and S is as stated below, what are the next 4 bytes of output of RC4? Estimate the speed of encrypting the next 4 bytes of output of an RC4 cipher on a computer in which assignment addition and logical AND requires .001 microseconds.

5. Suppose two parties share a secret key k and wish to communicate a series of “yes/no” answers over a public channel without disclosing the answers. Design a protocol to do this using a MAC. Be careful to make sure the adversary cannot figure out all the answers if they know whether the “code” for a few of the yes/no answers.
Backup
Differential Cryptanalysis: Overview

Let \( P=(P_L, P_R) \), \( P^*=(P_L^*, P_R^*) \) and \( C=(C_L, C_R) \), \( C^*=(C_L^*, C_R^*) \) be pairs of inputs and outputs with prescribed xors

\[
P'=(P_L', P_R') = (P_L, P_R) \oplus (P_L^*, P_R^*)
\]

\[
C'=(C_L', C_R') = (C_L, C_R) \oplus (C_L^*, C_R^*)
\]

Output xor depends non uniformly on key bits.
Let non uniform distribution “vote” on set containing keys.

Uses chosen plaintext/ciphertext pairs to get enough compliant pairs by following the xor of two plaintexts through rounds of DES.
Examine last round to discover key
Differential Profile of single S-box

• For prescribed input and output differences \( x', y' \) set
  \[ D_j(x', y') = \{ u : S_j(u \oplus x') \oplus S_j(u) = y' \} \], then
  – Note that \( u, u \oplus x', u \oplus k, u \oplus x' \oplus k \) will all appear in this set
  – \( k \in x \oplus D_j(x', y') \), if \( x \) is an input (pre-key) to \( S_i \).

• \(|D_j(x', y')| \) has non uniform distribution.
• For given input difference about 80% of the output differences are possible.
• \( p = |D_j(x', y')| / 2^m \), \( m \) is the dimension of the space of \( a' \).
• Shamir and Biham denote this as \( x' \rightarrow y', p \).
# S1 Differential Distribution

**S box 1**

<table>
<thead>
<tr>
<th>In</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>64</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>12</td>
<td>4</td>
<td>10</td>
<td>6</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>12</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>6</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>12</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>4</td>
<td>10</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>8</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>8</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>d</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>f</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>14</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>12</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
S1 Differential Distribution: another view

S box 1

D1(00, 0d): 0 found
D1(01, 0d): (0a, 0b) (0b, 0a) (22, 23) (23, 22) (3e, 3f) (3f, 3e) 6 found
D1(02, 0d): (08, 0a) (0a, 08) (29, 2b) (2b, 29) (35, 37) (37, 35) 6 found
D1(03, 0d): (14, 17) (17, 14) 2 found
D1(04, 0d): (13, 17) (17, 13) (1b, 1f) (1f, 1b) (2a, 2e) (2e, 2a) (3b, 3f) (3f, 3b) 8 found
D1(05, 0d): (01, 04) (04, 01) 2 found
D1(06, 0d): (21, 27) (27, 21) 2 found
... ... ...
D1(33, 0d): (07, 34) (0d, 3e) (1a, 29) (29, 1a) (34, 07) (3e, 0d) 6 found
D1(34, 0d): (06, 32) (10, 24) (16, 22) (1c, 28) (22, 16) (24, 10) (28, 1c) (32, 06) 8 found
D1(35, 0d): (00, 35) (35, 00) 2 found
D1(36, 0d): (02, 34) (0d, 3b) (34, 02) (3b, 0d) 4 found
Example: Differential Cryptanalysis of S1 through a single round

Consider input texts and output xors from S1

P₁ = 0x01, P₁* = 0x35 which produce output xor C₁' = 0x0d. (So C₁' = 0x34).
P₂ = 0x22, P₂* = 0x15 which produce output xor C₂' = 0x03. (So C₂' = 0x34).

Then

D₁(0x34, 0xd) = {0x06, 0x10, 0x16, 0x1c, 0x22, 0x24, 0x28, 0x32}.
D₁(0x34, 0x3) = {0x01, 0x02, 0x15, 0x21, 0x35, 0x36}.

And

(1) k ∈ P₁ ⊕ D₁(0x34, 0xd)
(2) k ∈ P₂ ⊕ D₁(0x34, 0x3)

(1) reduces the possible key set to {0x07, 0x33, 0x11, 0x25, 0x17, 0x23, 0x1d, 0x29}
(2) reduces the possible key set to {0x20, 0x14, 0x23, 0x17, 0x34, 0x00}.
The intersection (and actual possibilities) are { 0x17, 0x23}
One Round Differential used to analyze 4 round DES

Method

Use 1 round characteristic to right. Undo effect of permutation matrix and solve each S box separately. This allows us to solve for 48 key bits. This 1 round characteristic will be used to estimate input xor in subsequent rounds.
Differential Cryptanalysis of 4 rounds

- \( D' = a' \oplus B' \oplus L_4' \)
- \( d' = R_4' \)
- Because \( b' = L_0' \), the output xor of \( S_2, S_3, \ldots, S_8 \) in round 2 is 0. This gives 28 bits of \( B' \) and hence 28 bits of \( D' \) is known.
- Since \( B' \) is known, we can calculate \( D' = B' \oplus L_4' \) using 4 encrypted pairs for each of the 7 relevant \( S \) boxes. All key candidates are in this set which gives \( 7 \times 6 = 42 \) bits of key with high probability.
Computing a single characteristic

- The first and most important differential is \((L',0) \rightarrow (L',0)\), \(p=1\).
- Another is \((L',0x60000000) \rightarrow (L' \oplus 0x00808200,0x60000000)\), \(p=1/4\).
- Construction:
  - \(E(0x60000000)= E(0110 0000 \ldots 0000)= 001100 000000 \ldots 000000\)
  - \(S_1(001100)' \rightarrow 0xe\) with \(p=1/4\), \(S_j(0)' \rightarrow 0\) with \(p=1\), \(j>1\) and \(P(0xe0000000)=0x00808200\).
Multi-round Characteristics

- Sequence of Differentials with identified input and output xors. Each round differential occurs with probability $p_i$.

- Overall probability: $p = \prod p_i$

- Characteristic to the right is a three round characteristic with probability $(14/64)^2$

- Used to approximate differentials through multiple rounds.

- Each pair following the characteristic at each round is called a “right pair”. Other pairs are “wrong pairs.”

- Wrong pairs get distributed uniformly; right pairs follow overall characteristic probability.
Three Round Characteristic

- This characteristic occurs with probability \( p = 1/16 \) and forms an estimate for the differential input of the 4\(^{th} \) round of the 6 rounds.

- \((00\ 20\ 00\ 08\ 00\ 00\ 00\ 04) \rightarrow (00\ 00\ 04\ 00\ 00\ 20\ 00\ 08)\) with \( p = 1/16 \) is another such characteristic.
Differential Cryptanalysis of 6 rounds

- Suppose \((L_{i-1}, R_{i-1}), k_i\) are the inputs to round \(i\). \(P_L = L_0, P_R = R_0\).
- \(L_6 = R_4 \oplus f(k_6, R_6) = L_3 \oplus f(k_6, R_6) \oplus f(k_4, R_3)\)
- \(L_6' = L_3' \oplus f(k_6, R_6) \oplus f(k_6, R_6*) \oplus f(k_4, R_3) \oplus f(k_4, R_3*)\)
- \(L_6' = C_L\) and \(R_6 = C_R\) are known.
- Estimate \(L_3' = 40000000\), \(R_3' = 40080000\), using the differential.
- Set \(S = P^{-1}(C_L \oplus 40000000) = f(k_6, C_R) \oplus f(k_6, C_R*) \oplus f(k_4, R_3) \oplus f(k_3, R_3*) = S_1(E_1) || S_2(E_2) || ... || S_8(E_8)\) where \(E_1|| E_2|| ... || E_8\) are the bits obtained by applying \(E\) to 40080000.
- \(E_1|| E_2|| ... || E_8 = 001000000000000001010000...0 = 08||00||01||10||00||00||00||00\).
- Since the input Xors to \(S_2, S_5, S_6, S_7, S_8\) are 0, \(f(k_4, R_3) \oplus f(k_4, R_3*)\) is 0 in the corresponding output bit positions and we are left with the simple differential: \(P^{-1}(C_L \oplus 40000000) = f(k_6, C_R) \oplus f(k_6, C_R*)\) for \(S_2, S_5, S_6, S_7, S_8\).
Differential Cryptanalysis of 6 rounds

• First characteristic yields 30 bits of key. Second one adds another 12 bits of key.
• Recall $P^{-1}(C_L \oplus 40000000) = f(k_6, C_R) \oplus f(k_6, C_R^*)$ for $S2, S5, S6, S7, S8$
• This occurs with $p = 1/16$.
• Straightforward implementation yielding 30 keybits:
  – Set up $2^{30}$ counters
  – Bump counter for suggested key for each pair of $n$ chosen texts
  – Correct key be will “voted” at least $1/16 n$ time (“right pairs”)
  – Incorrect keys will be voted randomly each with probability $1/2^{30}$
Differential Cryptanalysis of 6 rounds

• Improving the “signal to noise” ratio by “filtering” pairs
  – For each of S2, S5, S6, S7, S8 with input xor x’ and output xor y’, look at $x \oplus D_j(x’,y’)$. 
  – If this is empty, this must be wrong pair.
  – For any given S box the, this happens with probability .2.
  – The probability that all 5 S boxes have non-empty candidate key sets is $(.8)^5=.33$. Call this set of pairs RP and the complement WP.
  – RP contains 1/3 of the pairs, WP contains 2/3
  – In RP, the probability of a “correct vote” is 3/16
Algebraic Attacks

• As we’ve seen, ciphertext can be expressed as algebraic function of keys and plaintext (Lagrange Interpolation Theorem).
• Sometimes key bits are expressible as functions of plain and cipher texts.
• These are easy to solve if the equations are linear even for very large key spaces.
• These are very hard to solve if the equations are even quadratic (NP-hard in fact, see “General System of Quadratic Equations” slide).
• General problem is “Find one solution of a system of $m$ equations in $n$ variables of bounded degree, $D$, over $K$ (usually finite):
  \[ \sum_b a_b x^b + c_i = 0, \quad x^b = x_1^{b_1} x_2^{b_2} \ldots X_n^{b_n}, \quad \sum_i b_i \leq D \]
• We refer to this problem as SolveAlgebraic($K,D,m,n$) and often abbreviate equations as $l_j(x)=0$. 
Solving SolveAlgebraic(K,D,m,n)

- Classic Technique is Grobner Basis, see
  - Lauritzen, Concrete Abstract Algebra. Cambridge.
  - Cox, Little, O’Shea, Using Algebraic Geometry. Springer.
- Grobner uses Buchberger’s Algorithm which is doubly exponential time in the worst case since the monomial grow very rapidly and singly exponential time on average.
- This is not practical for \( n > 15 \).
- However, we can do better with an overdefined set of equations \( m > n \).
- Note first that if we pick \( m \) random equations \( m > n \) they will likely be inconsistent.
- Let’s see how we might solve overdetermined systems by solving them as we do linear equations after we prove that solving even quadratic systems of equations is NP hard.
SHA-0 Strategy (Chabaud and Joux)

- Basic idea is to look for small differences that can be tracked through rounds like differential cryptanalysis.
- Consider three approximations to the SHA-0 compression function.
  - SHI-1
    - Use Xor instead of Add
    - Make $f^{(i)}$ linear
  - SHI-2
    - Use Xor instead of Add
    - Restore $f^{(i)}$ to original values
  - SHI-3
    - Restore Add
    - Make $f^{(i)}$ linear
SHI-1 Finding Collisions

- Assume the $W^{(i)}$ are unrelated and follow progress of a change to $W^{(1)}$.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$W^1$+ROL$_5$(A)+f(B,C,D)+E+K</td>
<td>A</td>
<td>ROL$_{30}$(B)</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>2</td>
<td>$W^2$+ …</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>3</td>
<td>ROL$_{30}$(-)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>ROL$_{30}$(W$^1$+ROL5(A)+f(B,C,D)+E+K)</td>
</tr>
<tr>
<td>6</td>
<td>$W^6$+ … - fixes $W^1$ perturbation</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**SHI-1 Error Propagation in Hash**

<table>
<thead>
<tr>
<th>State</th>
<th>$\Delta$</th>
<th>$W_1^{(i)}$</th>
<th>$W_6^{(i+1)}$</th>
<th>$W_1^{(i+2)}$</th>
<th>$W_{31}^{(i+3)}$</th>
<th>$W_{31}^{(i+4)}$</th>
<th>$W_{31}^{(i+5)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A^{(i)}$</td>
<td>$A_1^{(i)}$</td>
<td>$A^{(i+1)}$</td>
<td>$A^{(i+2)}$</td>
<td>$A^{(i+3)}$</td>
<td>$A^{(i+4)}$</td>
<td>$A^{(i+5)}$</td>
<td></td>
</tr>
<tr>
<td>$B^{(i)}$</td>
<td>$B_{31}^{(i+1)}$</td>
<td>$C_{31}^{(i+2)}$</td>
<td>$D_{31}^{(i+3)}$</td>
<td>$E_{31}^{(i+4)}$</td>
<td>$E^{(i+5)}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C^{(i)}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D^{(i)}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$E^{(i)}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Perturbation**
On bit 1

** Corrections defining masks**
Message Expansion

• Process of expanding from 16 32 bit words to 80 32 bit words in the compression function is called message expansion
  – MD5
    • Permutations
  – SHA-0
    • Linear code (LFSR)
  – SHA-1
    • Linear code with rotation
• Has profound effect on possible disturbance vectors in Differential attacks
• Being studied to provide greater protection
• Replace xor with modular addition to prevent codeword difference propagation
• Conditions on chaining variables for local collision (Prob between $2^{-39}$ and $2^{-42}$)