Lecture 6: Two Fish on the Rijndael

The algorithm might look haphazard, but we did everything for a reason. Nothing is in Twofish by chance. Anything in the algorithm that we couldn’t justify, we removed. The result is a lean, mean algorithm that is strong and conceptually simple.

Bruce Schneier

Breaking Grades File

- Not in my office or any UVA computer
  - Do not try to break into any UVA computer
- Home PC: C:\cs588\grades.txt (encrypted)
  - If you obtain that file, it tells you what to do next
- Adelphia Cable Modem
- My browser is set to disallow ActiveX, allow Java and JavaScript

Clipper

- 1993 – AT&T markets secure telephony device
- Law enforcement: US courts can authorize wire taps, must be able to decrypt
- NSA proposes Clipper Chip
  - Secret algorithm (Skipjack), only implemented in hardware

Key Escrow

- NSA has copy of special key, can get with a court order
- Sender transmits $E(M,k) \parallel \text{LEAF}$ (“law enforcement agents’ field”)
- Holder of special key can decrypt $\text{LEAF}$ to find message key and decrypt message

LEAF

$\text{LEAF} = E(E(k,u) \parallel n \parallel a, f)$

- $k$ = message key
- $u$ = 80-bit special key (unique to chip)
- $n$ = 30-bit identifier (unique to chip)
- $a$ = escrow authenticator
- $f$ = 80-bit key (same on all chips)
Wire Tap

- FBI investigating Alice, intercepts Clipper communication
- Uses $f$ to decrypt LEAF:
  $$D(E((E(k, u) \parallel n \parallel a), f)) = E(k, u) \parallel n \parallel a$$
- Delivers $n$ and court order to 2 escrow agencies, obtains $u$
- Decrypts $E(k, u)$ to obtain message key and decrypt message

Two Escrow Agencies

- Proposal didn’t specify who (one probably NSA)
- Divide $u$ so neither one can decrypt messages on their own (even if they obtain $f$)
  
  One gets $u \oplus X$, other gets $X$

Clipper Security

- How do you prevent criminals from transmitting wrong LEAF?
  - NSA solution: put it in hardware, inspect all Clipper devices
    - Still vulnerable to out-of-the-box device

Clipper Politics

- Not widely adopted, administration backed down
  - Secret algorithm
  - Public relations disaster
    - Didn’t involve academic cryptographers early
    - Proposal was rushed, in particular hadn’t figured out who would be escrow agencies
  - See [http://www.eff.org/pub/Privacy/Key_escrow/Clipper/](http://www.eff.org/pub/Privacy/Key_escrow/Clipper/)
  - Future?: Senators have called for new Clipper-like restrictions on cryptography
  - Lessons learned well for AES process

AES

- 1996: NIST initiates program to choose Advanced Encryption Standard to replace DES
- Requests algorithm submissions: 15
- Requirements:
  - Secure for next 50-100 years
  - Performance: faster than 3DES
  - Support 128, 192 and 256 bit keys
    - Brute force search of $2^{128}$ keys at 1 Trillion keys/second would take $10^{19}$ years ($10^9 \times$ age of universe)
    - Must be a block cipher

AES Process

- Open Design
  - DES: design criteria for S-boxes kept secret
- Many good choices
  - DES: only one acceptable algorithm
- Public cryptanalysis efforts before choice
  - Heavy involvements of academic community, leading public cryptographers
- Conservative (but quick): 4 year+ process
AES Round 1

- 15 submissions accepted
- Weak ciphers quickly eliminated
  - Magenta broken at conference!
- 5 finalists selected: MARS (IBM), RC6 (Rivest, et. al.), Rijndael (top Belgium cryptographers), Serpent (Anderson, Biham, Knudsen), Twofish (Schneier, et. al.)
  - Security v. Performance is main tradeoff
    - How do you measure security?
    - Simplicity v. Complexity
    - Need simplicity to be able to analyze and implement efficiently

Breaking a Cipher

- Real World Standard
  - Attacker can decrypt secret messages
  - Reasonable amount of work, actual amount of ciphertext
- “Academic” Standard
  - Attacker can determine something about the message
  - Given unlimited number of chosen plaintext ciphertext pairs
  - Can perform a very large number of computations, up to, but not including, $2^n$, where $n$ is the key size in bits (i.e., assume that the attacker can’t mount a brute force attack, but can get close)

AES Evaluation Criteria

1. Security
   - Most important, but hardest to measure
   - Resistance to cryptanalysis, randomness of output
2. Cost and Implementation Characteristics
   - Licensing, Computational, Memory
   - Flexibility (different key/block sizes), hardware implementation

From RC5 to RC6 in seven easy steps

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 \( S[0, \ldots, 2r + 3] \) of \( w \)-bit round keys.
- Encryption and Decryption:
  - Input/Output in 32-bit registers A,B,C,D

Design Philosophy

- Leverage 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.
Data-Dependent Rotations

\[
\begin{array}{cccccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} & \text{g} & \text{h} \\
\text{d} & \text{e} & \text{f} & \text{g} & \text{h} & \text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[X \oplus X' = \Delta X\]

\[X_1 = X \ll f(X, k)\]

\[X_1' = X' \ll f(X', k)\]

Can we say anything about \(\Delta X_1 = X_1 \oplus X_1'\)?

- Same number of bits are still different, but can't tell which ones.
- \(\ll n\) means rotate left by amount in low order \(\log_2 w\) bits of \(n\) (word size \(w = 32, 5\) bits)

Better rotation amounts?

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

Properties \(B \times (2B + 1)\) should have:

1. One-to-one (can invert for decryption)
2. Good distribution – if \(B\) is well distributed, so is \(B \times (2B + 1)\)
3. High order bits depend on all bits of \(B\) (diffusion)
4. Easy to calculate efficiently (if your hardware has 32-bit multiplies)

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

**Proof:** By contradiction: Assume \(B \neq C\)

\[B \times (2B + 1) - C \times (2C + 1) = 0 \mod 2^w\]

Then:

\[2B^2 + B - (2B^2 + C) = 0 \mod 2^w\]

\[(B - C) \times (2B + 2C + 1) = 0 \mod 2^w\]

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!)

3. High-order bits of \(B \times (2B + 1)\) depend on all bits of \(B\) (diffusion)

\[
\begin{array}{cccccccc}
B_31 & B_{30} & B_{29} & \ldots & B_0 \\
B_31 & B_{30} & B_{29} & \ldots & B_0 \\
B_0 & B_3 & B_{26} & \ldots & B_{l-1} \\
B_1 & B_3 & B_{26} & \ldots & B_{l-1} \\
\end{array}
\]

\[f(B) = F_3 \oplus F_0 = F_1 \oplus F_0 \]

\[F_j = (1 \times B_j) + \sum (B_i \times B_{j-i}) + C_{i-1} \mod 2\]

(1) Start with RC5

RC5 encryption inner loop:

\[
\text{for } i = 1 \text{ to } r \text{ do}
A = ((A \oplus B) \ll S[i]) + S[i]
\]

\[(A, B) = (B, A)\]

\(\ll\) only depends on 5 bits of \(B\)

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

\[\text{• Modulo function?} \]

\[\text{• Linear function?} \]

\[\text{• Quadratic function?} \]

\[\text{Better rotation amounts?} \]

\[\text{Properties } B \times (2B + 1) \text{ should have:} \]

\[\text{3. High-order bits of } B \times (2B + 1) \text{ depend on all bits of } B \text{ (diffusion)} \]

\[f(B) = F_3 \oplus F_0 = F_1 \oplus F_0 \]

\[F_j = (1 \times B_j) + \sum (B_i \times B_{j-i}) + C_{i-1} \mod 2\]
Diffusion, cont.

\[ F_i = B_i + \sum_{j=0}^{i-1} (B_j \times B_{i-j}) + C_{i-1} \mod 2 \]

\[ C_i = B_i + \sum_{j=0}^{i-1} (B_j \times B_{i-j}) + C_{i-1} \div 2 \]

- Flipping bit \( B_i \):
  - Leaves bits \( F_0 \) to \( F_{i-1} \) of \( f(B) \) unchanged,
  - Flips bit \( F_i \) always
  - Flips bit \( F_j \) for \( j > i \), with probability approximately \( \frac{1}{2} \)
    - Different for different \( j \)'s, but \( F_i \) depends on \( B_j \) for all \( i > j \).
  - Is likely to change some high-order bits

---

(2) Quadratic Rotation Amounts

\[
\begin{aligned}
\text{for } i = 1 \text{ to } r \text{ do} \\
\quad t = (B \times (2B + 1)) <<< 5 \\
\quad A = ((A \oplus t) <<< t) + S[2i] \\
\quad (A, B) = (B, A)
\end{aligned}
\]

But now much of the output of multiplication is being wasted (only 5 top bits used)...

---

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

\[
\begin{aligned}
\text{for } i = 1 \text{ to } r \text{ do} \\
\quad t = (B \times (2B + 1)) <<< 5 \\
\quad A = ((A \oplus t) <<< t) + S[2i] \\
\quad (A, B) = (B, A)
\end{aligned}
\]

RC5 used 64-bit blocks
AES requires 128-bit blocks
Double size of \( A \) and \( B \)?
64-bit registers and operations are poorly supported by typical compilers and hardware

---

(4) Do two RC5’s in parallel

\[
\begin{aligned}
M_0 &= A_0B_0A_1B_1 \ldots \\
M_r &= A_rB_rA_{r+1}B_{r+1} \ldots \\
\text{for } i = 1 \text{ to } r \text{ do} \\
\quad t = (B \times (2B + 1)) <<< 5 \\
\quad A = ((A \oplus t) <<< t) + S[2i] \\
\quad (A, B) = (B, A)
\end{aligned}
\]

\[
\begin{aligned}
\quad u = (D \times (2D + 1)) <<< 5 \\
\quad C = ((C \oplus u) <<< u) + S[2i+1] \\
\quad (C, D) = (D, C)
\end{aligned}
\]

Same thing for next 64 bits

---

(5) Mix up data between copies

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

\[
\begin{aligned}
\text{for } i = 1 \text{ to } r \text{ do} \\
\quad t = (B \times (2B + 1)) <<< 5 \\
\quad u = (D \times (2D + 1)) <<< 5 \\
\quad A = ((A \oplus t) <<< u) + S[2i] \\
\quad C = ((C \oplus u) <<< t) + S[2i+1] \\
\quad (A, B, C, D) = (B, C, D, A)
\end{aligned}
\]
Key Expansion (Same as RC5’s)

- **Input:** array L[0 \ldots c -1] of input key words
- **Output:** array S[0 \ldots 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
  ```

What do $/pi/e/\Phi$ have to do with cryptography?

- Used by RC5, RC6, Blowfish, etc. in magic constants
- Mathematical constants have good pseudorandom distribution
- Since they are public and well-known, no fear that choice is a trap door

(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 \oplus t) <<< u) + S[2i]
  C = ((C \oplus 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

```
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 \oplus t) <<< u) + S[2i]
  C = ((C \oplus 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 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]
```

Blowfish

- [Schneier93]
- 64-bit block cipher
- Much faster than DES
- Variable key length: 32-448 bits
- Many attempted crytanalyses, none successful yet
- Widely used: ssh, OpenBSD, PGPFone
Key-Dependent S-Boxes

- Differential Cryptanalysis depends on analyzing S-box input/output different probabilities
- Change the S-boxes so you can't do analysis

Blowfish → Twofish

- Blowfish: runs encryption 521 times to produce S-boxes
  - Too slow for AES, requires too much memory for smart cards
- Twofish
  - Provides options for how many key-dependent S-boxes (tradeoff security/time-space)
  - Also: increase block size (128 required by AES), change key schedule, etc.

Two Fish


Choosing AES

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Speed (16)</th>
<th>Speed (8)</th>
<th>Safety Factor</th>
<th>Simplicity (code size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serpent</td>
<td>62</td>
<td>69</td>
<td>3.56</td>
<td>241 KB</td>
</tr>
<tr>
<td>MARS</td>
<td>23</td>
<td>34</td>
<td>1.90</td>
<td>85 KB</td>
</tr>
<tr>
<td>RC6</td>
<td>15</td>
<td>43</td>
<td>1.18</td>
<td>48 KB</td>
</tr>
<tr>
<td>Rijndael</td>
<td>18</td>
<td>20</td>
<td>1.11</td>
<td>98 KB</td>
</tr>
<tr>
<td>Twofish</td>
<td>16</td>
<td>18</td>
<td>2.67</td>
<td>104 KB</td>
</tr>
</tbody>
</table>

AES Winner: Rijndael

Invented by Joan Daemen and Vincent Rijmen

Rijndael. A variant of Square, the chief drawback to this cipher is the difficulty Americans have pronouncing it.

Bruce Schneier

Selected as AES, October 2000

Rijndael Overview

- Key sizes: 128, 192, 256 bits
- Block sizes: 128, 192, 256 bits
- 10 rounds (including initial AddKey)
  - Academic break on 9 rounds, 256-bit key gives safety factor of $10/9 = 1.11$
  - Requires $2^{230}$ work and $2^{85}$ chosen related-key plaintexts (why is this considered a break for 256-bit key but not 128-bit key?)
  - “Our results have no practical significance for anyone using the full Rijndael.”
Rijndael Round

1. Byte substitution using non-linear S-Box (independently on each byte)
2. Shift rows (square)
3. Mix columns – matrix multiplication by polynomial
4. XOR with round key

Rijndael Design

- Resistant to linear and differential cryptanalysis
- Differential trail
  - Probability that a given difference $a'$ pattern at input produces an output difference of $b'$
  - Choose S-box and multiplication polynomial to minimize maximum difference probability

Charge

- Designing and picking a Cipher that will last 50 years is hard
  - Advances in computing power
  - Advances in cryptanalysis
  - Performance/security tradeoff keeps changing – need something that works today and in 2050
- This week: talk or email me about your project ideas
- Next time:
  - Key Distribution