Implementation and Performance analysis of Skipjack & Rijndael Algorithms

By
Viswanadham Sanku

Topics

• Skipjack cipher operations
• Design principles & cryptanalysis
• Implementation & optimization
• Results
• Rijndael cipher operations
• Design principles & cryptanalysis
• Implementation & optimization
• Results
• Problems encountered
• Conclusions
Skipjack cipher operations

Background

- Skipjack is a block cipher designed in 1993 by NSA
- Unclassified on June 24th, 1998
- 64-bit block size and 80-bit key
- 32 rounds, composed of round functions
- Two round functions RuleA, RuleB and their inverses (decryption)

Key expansion

- Each of the 32 rounds uses 4 byte subkey
- 10 byte key is repeated itself to make it 128 byte
Design principles & cryptanalysis

Design Principles

• Symmetry in encryption & decryption protects against chosen plaintext and chosen ciphertext attacks.
• Not too much Symmetry
  Symmetry is broken with round counters. protects against symmetry based attacks
• 80-bit Key
  Effective size against differential & linear attacks

Cryptanalysis

• Less diffusion in B rounds and A⁻¹ rounds
• Bad interactions between the round-types
• A one bit difference in input to the F Table may cause a difference of only one bit in its output
• Eli Biham had shown an attack on 31 round skipjack using impossible differentials
• Less security margin

Implementation & optimization

Implementation

• Implemented with MS VC++ (Windows) and gnu C (Linux)
• Implemented in ECB, CBC, CFB, OFB and CTR modes in 64-bit mode
• Used PKCS#5 padding
  Pad the input by appending 8 - (n mod 8) bytes to the end of the message, each having the value 8 - (n mod 8), the number of bytes being added
• Implemented KAT, MCT tests to verify the correctness of the implementation

Optimization techniques

• Operations on integers, except Gfunction. Int operations are faster than other types.
• Unroll the rounds
• #define macro substitutions. Reduces run-time overtime of function call
• With prior key knowledge, subkey table representing the Gpermutaion function
  ftable[inbyte ^ keybyte]
• Data movement can be minimized by rotating the names of variables w1,w2,w3, w4 instead of the contents.

(Note: last two techniques not used in my implementation)
Results

350 MHz Pentium II, Visual C++

500 MHz Pentium III, GNU C
Rijndael cipher operations

Background

- Rijndael is a block cipher designed by Joan Daemen and Vincent Rijmen.
- Selected as the proposed AES by NIST. Will be official sometime spring 2001.
- Keys lengths of 128, 192, or 256 bits and blocks lengths of 128, 192 or 256 bits are supported.
- Block length and key length can be extended to multiples of 32 bits.
- Number of rounds depend on block and key lengths.
- Each round transformation is composed of four different transformations: ByteSub, ShiftRow, MixColumn, and AddRoundKey, except the final round, which does not have MixColumn.
- Intermediate cipher result is called the State.

Number of rounds = \( \text{Max}(NB, NK) + 6 \)

\( NB \) = # of 32-bit blocks in input block

\( NK \) = # of 32-bit blocks in key
Rijndael cipher operations

Encryption
128/192/256 bit block
\[ \text{Key Addition} \]
\[ \text{MAX(NB,NK) + 4 - 1 Round Transformations} \]
\[ \text{Final Transformation} \]
128/192/256 bit cipher

Round Transformations
ByteSub
ShiftRow
MixColumn
AddRoundKey

Decryption
128/192/256 bit block
\[ \text{Key Addition} \]
\[ \text{MAX(NB,NK) + 4 - 1 Inv Round Transformations} \]
\[ \text{Inv Final Transformation} \]
128/192/256 bit cipher

Inverse Round Transformation
ByteSub
ShiftRow
MixColumn
AddRoundKey
Rijndael cipher operations

- **ByteSub**
- **ShiftRow**
- **MixColumn**

**Key Expansion**

- Expanded key size = \( NB \times (NR+1) \)
- Each round key consists of next \( NB \) 32-bit blocks taken from expanded key
- Inverse round key is obtained by applying InverseMixColumn to all Round Keys except the first and the last one.

**Examples**

- For MixColumn:
  \[ c(x) = 03 \times x^3 + 01 \times x^2 + 01 \times x + 02 \]
- For InverseMixColumn:
  \[ d(x) = 0B \times x^3 + 0d \times x^2 + 09 \times x + 0E \]
Design principles & cryptanalysis

Design Principles

- Symmetry in encryption & decryption protects against chosen plaintext and chosen ciphertext attacks.
- Linear-mixing layer (ShiftRow & MixColumn) which guarantees high diffusion and non-linear S-boxes. Protects against linear and differential cryptanalysis.
- Non-Feistel round transformation.

Cryptanalysis

- Rijndael has adequate security margin, against linear and differential cryptanalysis, as well as other types of attacks.
- Square attack can be used to break 7 rounds of the cipher.
- Niels Ferguson showed attacks on 8 rounds of cipher for key sizes 192 & 256 bits.

Implementation & optimization

Implementation

- Implemented with MS VC++ (Windows) and gnu C (Linux).
- Implemented in ECB, CBC, CFB, OFB and CTR modes for 128-bit input block and 128/192/256 bit-key sizes.
- Used PKCS#5 padding:
  - Pad the input by appending $8 \cdot (n \mod 8)$ bytes to the end of the message, each having the value $8 \cdot (n \mod 8)$, the number of bytes being added.
- Implemented KAT, MCT tests to verify the correctness of the implementation.

Optimization techniques

- #define macro substitutions. Reduces run-time overhead of function call.
- Use pre-calculated tables in place of (ByteSub+ShiftRow+MixColumn) transforms.
Implementation & optimization

Optimization technique
Prepare (ByteSub + ShiftRow + MixColumn) transform Tables

Table 1 = S(2X) | S(X) | S(X) | S(3X)

350 MHz Pentium II, Visual C++

500 MHz Pentium III, GNU C

Results

(Note: Brian Gladman implementation for GNU C compiler is not available)
Results

350 MHz Pentium II, Visual C++

Key setup (clock cycle count)

Note: Key setup cycles for implementations by Viswanadh & Rijmen is total cycle count for encryption key setup and decryption key setup.

500 MHz Pentium III, GNU C

Key setup (Mbits/sec)

Note: Brian Gladman implementation for GNU C compiler is not available.

Results

350 MHz Pentium II, Visual C++

(Filesize: 51 Mbytes; keysize: 128bits; blocksize: 128bits)

Speed (Mbits/sec)
Skipjack Vs Rijndael

- The Best possible speeds I could able to achieve for
  Skipjack = 9 Mhz and Rijndael = 27 Mhz (128 bit block&key)
  Major contribution for the difference is the number of rounds
  (skipjack = 32, Rijndael = 10) and then the round transformation
  tables used in Rijndael.
- Skipjack G permutation is applied to one 16-bit word in each round.
  While Rindael round transformation (ByteSub+Shift+Mix) is applied
  to every byte of the block.
- Rijndael has pretty good security margin compared to Skipjack.
Problems encountered

• Clock cycle counter code would not work with MSVC++, because the code used constants CPUID, RDTSC, which the compiler can not understand. Instead they have to be force fed to the compiler using '_emit' instruction along with the opcode.

```c
double cycles(void) {   unsigned long hi,lo;
    __asm { _emit 0x60         //PUSHAD: Save all registers
             _emit   0x0f  _emit   0xa2 //CPUID: Serialize instruction execution
             _emit   0x0f  _emit   0x31 //RDTSC: Read clock cycle count into A,D
             mov  lo,ax   mov  hi,edx //Get values from A,D registers
             _emit  0x61 }         //POPAD: Restore the registers
    return 4294967296.0 * hi + lo;     // 2^32 * hi + lo
}
```

• Though the constants CPUID, RDTSC works with GNU C compiler, the count is not returning proper cycle count values. So I had to depend on the absolute time functions. 

Could not able to figure it out Yet.

Conclusions

• Rijndael is more flexible cipher than skipjack and more secure
• Rijndael yields greater speeds than skipjack with proper implementations
• To implement a cipher it is very important to analyze the cipher. So that the implementation can be optimized to yield far better performances than the straight forward implementation.
• The speed of the cipher depends on complier as well. Same implementation may yield widely varied performances on different platforms.
• Diffusion and security margin may not say anything about how secure the cipher is, but they are important in the analysis of the cipher.