Computer Science 418

The Advanced Encryption Standard (AES)

Mike Jacobson

Department of Computer Science
University of Calgary

Week 5

Outline

Skipjack and the Clipper Chip

After DES became obsolete, the United States *National Security Agency* (NSA) wanted to take control of the cipher standard selection process

- Proposed the *Skipjack Algorithm* implemented on the *Clipper Chip*
- Standardized by NIST as Escrowed Encryption Standard (EES) in Feb. 1994 (see FIPS 185).

In blunt violation of Kerckhoff's principle, the details of Clipper and Skipjack were initially classified and kept secret.

Due to wide distrust of the NSA in the US and abroad, this never really caught on in the public sector.

AES Competition

In 1997, NIST put out a call soliciting candidates to replace DES using a process that was completely transparent and public. Requirements:

- possible key sizes of 128, 192, and 256 bits
- plaintexts and ciphertexts of 128 bits
- should work on a wide variety of hardware (from Smart Cards to PCs)
- fast
- secure
- world-wide royalty-free availability (!)

Selection Criteria

Candidates were selected according to:

- security – resistance against all known attacks
- cost — speed and code compactness on a wide variety of platforms
- simplicity of design

Most important: *public* evaluation process

- series of three conferences: algorithms, attacks, evaluations presented and discussed
- final selection done by NIST

Finalists

NIST initiated a public (world-wide) process of candidate submission and evaluation for the *Advanced Encryption Standard*.

21 algorithms were submitted on June 15, 1998, of which 15 were announced as candidates on August 20, 1998.

Five finalists were selected in August 1999:

- MARS (from IBM)
- RC6 (from RSA Labs)
- Rijndael (by two Belgians: Daemen and Rijmen)
- Serpent (Anderson, Biham, Knudson, a multi-national team)
- Twofish (Schneier et al)

The Winner: Rijndael

All five were very good algorithms. Rijndael (pronounced "Reign Dahl" or "Rhine Dahl", but NOT "Region Deal") was chosen as the AES.

- Inventors: Vincent Rijmen and Joan Daemen.

The Rijndael algorithm uses two different types of arithmetic:

- Arithmetic on bytes (8 bit vectors—actually, elements of the finite field $GF(2^8)$ of 256 elements)
- 4-byte vectors (actually polynomial operations over $GF(2^8)$).

Arithmetic on Bytes

Consider a byte $b = (b_7, b_6, \ldots, b_1, b_0)$ (an 8-bit vector) as a polynomial with coefficients in $\{0, 1\}$:

$$b \mapsto b(x) = b_7 x^7 + b_6 x^6 + \cdots + b_1 x + b_0 \ .$$

Rijndael makes use of the following operations on bytes, interpreting them as polynomials:

1. Addition
2. Modular multiplication
3. Inversion

Under these operations, polynomials of degree ≤ 7 with coefficients in $\{0, 1\}$ form the *field* $GF(2^8)$.

By associating bytes with these polynomials, we obtain these operations on bytes.

Addition of Bytes in Rijndael

Polynomial addition by taking X-OR of coefficients.

$$\begin{array}{cccccccccc} & b_7x^7 & + & b_6x^6 & +\cdots+ & b_1x & + & b_0 \\ + & c_7x^7 & + & c_6x^6 & +\cdots+ & c_1x & + & c_0 \\ \hline & (b_7 \oplus c_7)x^7 & + & (b_6 \oplus c_6)x^6 & +\cdots+ & (b_1 \oplus c_1)x & + & (b_0 \oplus c_0) \end{array}$$

The sum of two polynomials taken in this manner yields another polynomial of degree 7.

In other words, component-wise X-OR of bytes is identified with this addition operation on polynomials.

Modular Multiplication in Rijndael

Polynomial multiplication (coefficients are in $\{0,1\}$) modulo

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

(remainder when dividing by $m(x)$, analogous to modulo arithmetic with integers).

The remainder when dividing by a degree 8 polynomial will have degree ≤ 7. Thus, the "product" of two bytes is associated with the product of their polynomial equivalents modulo $m(x)$.

Note 1

$m(x)$ is the lexicographically first polynomial that is *irreducible* over $GF(2)$, *i.e.* does not split into two polynomials of smaller positive degree with coefficients in $\{0,1\}$.

Inversion of Bytes in Rijndael

$b(x)^{-1}$, the inverse of $b(x) = b_7x^7 + b_6x^6 + \cdots + b_1x + b_0$, is the degree 7 polynomial with coefficients in $\{0,1\}$ such that

$$b(x)b(x)^{-1} \equiv 1 \pmod{m(x)} \ .$$

Note that this is completely analogous to the case of integer arithmetic modulo n.

The "inverse" of the byte $b = (b_7, b_6, \ldots, b_1, b_0)$ is the byte associated with the inverse of $b(x) = b_7x^7 + b_6x^6 + \cdots + b_1x + b_0$.

Rijndael uses inverse as above in its ByteSub operation.

Arithmetic on 4-byte Vectors

In Rijndael's MixColumn operation, 4-byte vectors are considered as degree 3 polynomials with coefficients in $GF(2^8)$. That is, the 4-byte vector (a_3, a_2, a_1, a_0) is associated with the polynomial

$$a_3x^3 + a_2x^2 + a_1x + a_0,$$

where each coefficient is a byte viewed as an element of $GF(2^8)$ (addition, multiplication, and inversion of the coefficients is performed as described above).

Operations on 4-byte Vectors

We have the following operations on these polynomials:

1. addition: component-wise "addition" of coefficients (addition as described above)
2. multiplication: polynomial multiplication (addition and multiplication of coefficients as described above) modulo $M(x) = x^4 + 1$. Result is a degree 3 polynomial with coefficients in $GF(2^8)$.

Note 2

Using $M(X) = x^4 + 1$ makes for very efficient arithmetic (simple circular shifts)

4-byte Vectors in Rijndael

In MixColumn, the 4-byte vector (a_3, a_2, a_1, a_0) is replaced by the result of multiplying $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ by the fixed polynomial

$$c(x) = 03x^3 + 01x^2 + 01x + 02$$

and reducing modulo $x^4 + 1$.

The coefficients of $c(x)$ are given as bytes in hex notation.

Literature

Website: `http://csrc.nist.gov/CryptoToolkit/aes/rijndael/`

AES Description:
`http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`

See also `http://www.iaik.tu-graz.ac.at/research/krypto/AES/` for other information (ECRYPT network).

Rijndael Properties

Designed for block sizes and key lengths to be any multiple of 32, including those specified in the AES.

Iterated cipher: number of rounds N_r depends on the key length. 10 rounds for 128 bit keys, 12 rounds for 192 bit keys, and 14 rounds for 256 bit keys.

Algorithm operates on a 4×4 array of bytes (8 bit vectors) called the *state*:

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

Properties, cont.

The algorithm uses addition, multiplication, and inversion on bytes as well as addition and multiplication of 4 byte vectors.

Rijndael is a product cipher, but NOT a Feistel cipher like DES. Instead, it has three *layers* per round:

- a linear mixing layer (ShiftRows, transposition, and MixColumns, a linear transformation; for diffusion over multiple rounds)
- a non-linear layer (SubBytes, substitution, done with an S-box)
- a key addition layer (AddRoundKey, X-OR with key)

Overview

The Rijndael algorithm (given plaintext M) proceeds as follows:

1. Initialize `State` with M :

$$\begin{array}{|c|c|c|c|} \hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \leftarrow \begin{array}{|c|c|c|c|} \hline m_0 & m_4 & m_8 & m_{12} \\ \hline m_1 & m_5 & m_9 & m_{13} \\ \hline m_2 & m_6 & m_{10} & m_{14} \\ \hline m_3 & m_7 & m_{11} & m_{15} \\ \hline \end{array}$$

where M consists of the 16 bytes $m_0, m_1, \ldots, m_{15}$.

Overview, cont.

2. Perform AddRoundKey, which X-OR's the first `RoundKey` with `State`.
3. For each of the first $N_r - 1$ rounds:
   - Perform SubBytes on `State` (using an S-box on each byte of `State`),
   - Perform ShiftRows (a permutation) on `State`,
   - Perform MixColumns (a linear transformation) on `State`,
   - Perform AddRoundKey.
4. For the last round:
   - Perform SubBytes,
   - Perform ShiftRows,
   - Perform AddRoundKey.
5. Define the ciphertext C to be `State` (using the same byte ordering).

The SubBytes Operation

Each byte of `State` is substituted independently, using an invertible S-box (see p. 16 of FIPS 197 for the exact S-Box).

Algebraically, SubBytes performs on each byte:

- an inversion as described above (the inverse of the zero byte is defined to be zero here), followed by
- an affine transformation, *i.e.* a linear transformation (like in linear algebra), and the addition of a fixed vector. More exactly, the i=th bit of the output byte is

$$b'_i = b_i \oplus b_{i+4 \bmod 8} \oplus b_{i+5 \bmod 8} \oplus b_{i+6 \bmod 8} \oplus b_{i+7 \bmod 8} \oplus c_i$$

where b_i is the i-th input bit and c_i is the i-th bit of $c = (11000110)$.

Inverse of SubBytes

The inverse of SubBytes (called InvSubBytes) applies the inverse S-box to each byte State (see p. 22 of FIPS 197 for the inverse of the S-Box).

Algebraically, you first apply the inverse affine transformation to each bit and then byte inversion.

The ShiftRows Operation

Shifts the first, second, third, and last rows of State by 0, 1, 2, or 3 cells to the left, respectively:

$$\begin{array}{|c|c|c|c|} \hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \longleftarrow \begin{array}{|c|c|c|c|} \hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ \hline s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ \hline s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \\ \hline \end{array}$$

The inverse operation InvShiftRows applies right shifts instead of left shifts.

The MixColumns Operation

Each column of State is a 4-byte vector which can be interpreted as a four-term polynomial with coefficients in $GF(2^8)$ as described above. For example:

$$(s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0}) \mapsto s_{3,0}x^3 + s_{2,0}x^2 + s_{1,0}x + s_{0,0} = col_0(x) \ .$$

Let $a(x) = 3x^3 + x^2 + x + 2$ be fixed.

Then MixColumns multiplies $col_i(x)$ by $a(x)$ as described above (multiplication of two 4-byte vectors), resulting in a new 4-byte column.

MixColumns: Algebraic Description

MixColumns can also be described as a linear transformation applied to each column of State, *i.e.* multiplying each 4-element column vector by the 4×4 matrix.

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 1 \end{pmatrix}$$

Note that rows 0, 1, 2, 3 of this matrix are circular shifts of row 0 by 0, 1, 2, 3 cells to the right.

InvMixColumns: Algebraic Description

The inverse (called InvMixColumns) multiplies each column of State by the inverse of $a(x) \pmod{x^4+1}$ which is

$$a^{-1}(x) = Bx^3 + Dx^2 + 9x + E$$

in hex notation.

It can also be described as multiplication by the following matrix (in hex):

$$\begin{pmatrix} E & B & D & 9 \\ 9 & E & B & D \\ D & 9 & E & B \\ B & D & 9 & E \end{pmatrix}$$

The AddRoundKey Operation

In AddRoundKey, each column of State is X-ORed with one word of the round key:

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \leftarrow \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \oplus \begin{pmatrix} w_{0,i+0} & w_{0,i+1} & w_{0,i+2} & w_{0,i+3} \\ w_{1,i+0} & w_{1,i+1} & w_{1,i+2} & w_{1,i+3} \\ w_{2,i+0} & w_{2,i+1} & w_{2,i+2} & w_{2,i+3} \\ w_{3,i+0} & w_{3,i+1} & w_{3,i+2} & w_{3,i+3} \end{pmatrix}$$

Here $w_{i+0} = (w_{0,i+0}, w_{1,i+0}, w_{2,i+0}, w_{3,i+0})$ is the first round key for round i, made up of four bytes.

AddRoundKey is clearly its own inverse.

Key Schedule

The key schedule uses:

- the S-box from SubBytes
- cyclic left shifts by one byte on 4-byte vectors
- multiplication by powers of x (each such power is interpreted as a 4-byte vector)

Consider 128-bit Rijndael. There are 10 rounds plus one preliminary application of AddRoundKey, so the key schedule must produce 11 round keys, each consisting of four 4-byte words, from the 128-bit key (16 bytes).

KeyExpansion

Produces an expanded key consisting of the required 44 words (assuming 128-bit key).

In the following, the key $K = (k_0, k_1, k_2, k_3)$, where the k_i are 4-byte words, and the expanded key is denoted by the word-vector $(w_0, w_1, w_2, \ldots, w_{44})$.

1. for $i \in \{0, 1, 2, 3\}$, $w_i = k_i$
2. for $i \in \{4, \ldots, 44\}$:

$$w_i = w_{i-4} \oplus \begin{cases} \text{SubWord}(\text{RotWord}(w_{i-1})) \oplus \text{Rcon}_{i/4} & \text{if } 4 \mid i \\ w_{i-1} & \text{otherwise} \end{cases}$$

KeyExpansion, cont.

The components of KeyExpansion are:

- RotWord is a one-byte circular left shift on a word.
- SubWord performs a byte substitution (using the S-box SubBytes on each byte of its input word).
- Rcon is a table of round constants (Rcon_j is used in round j). Each is a word with the three rightmost bytes equal to 0 and the leftmost byte a power of x

KeyExpansion is similar for 192 and 256-bit keys.

Decryption

To decrypt, perform cipher in reverse order, using inverses of components and the reverse of the key schedule:

1. AddRoundKey with round key N_r
2. For rounds $N_r - 1$ to 1 :
   - InvShiftRows
   - InvSubBytes
   - AddRoundKey
   - InvMixColumns
3. For round 1 :
   - InvShiftRows
   - InvSubBytes
   - AddRoundKey using round key 1

Note 3

Straightforward inverse cipher has a different sequence of transformations in the rounds. It is possible to reorgainize this so that the sequence is the same as that of encryption (see A2 of FIPS-197).

Strengths of Rijndael

Secure against all known attacks at the time; some newer attacks seem to pose no real threat

Non-linearity resides in S-boxes (SubBytes):

- linear approximation and difference tables are close to uniform (thwarting linear and differential cryptanalysis)
- no fixed points ($S(a) = a$) or opposite fixed points ($S(a) = \overline{a}$)
- not an involution ($S(S(a)) \neq a$, or equivalently, $S(a) = S^{-1}(a)$)

ShiftRows and MixColumns ensure that after a few rounds, all output bits depend on all input bits (great diffusion).

Strengths, cont.

Secure key schedule (great confusion):

- knowledge of part of the cipher key or round key does not enable calculation of many other round key bits
- each key bit affects many round key bits

Very low memory requirements

Very fast (hardware and software)

Weaknesses of Rijndael

Decryption is slower than encryption.

Decryption algorithm is different from encryption (requires separate circuits and/or tables).

- Depending on the mode of operation, however, this may not be an issue (*i.e.* OFB, CTR, CFB).