Advanced Encryption Standard

Rijndael is the new Advanced Encryption Standard.

It was invented by Joan Daemen and Vincent Rijmen.

It is a block cipher. The block length and key length can be chosen independently to be 128, 192 or 256 bits.

It has 10, 12 or 14 rounds, depending on the block and key lengths. The rounds do not have a Feistel structure.

It was designed to be simple, to be resistant against all known attacks and to have fast and compact code on many platforms.

Mathematical preliminaries for Rijndael.

Byte operations are done with arithmetic in the field $GF(2^8)$.

A byte $b_7 b_6 \ldots b_1 b_0$ is considered a polynomial with coefficients in $\{0, 1\}$:

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

Example: The byte `0xB7` = `1011 0111` is the polynomial

$$x^7 + x^5 + x^4 + x^2 + x + 1.$$

Bytes are added with XOR ($\oplus$). Addition is associative and commutative. The identity element is `0x00`. Every byte is its own additive inverse.

Bytes are multiplied modulo $m(x) = x^8 + x^4 + x^3 + x + 1$ (= `0x11B`). Multiplication is associative and commutative. The identity element is `0x01`. Every non-zero polynomial (byte) has a unique inverse with respect to this multiplication. The inverse may be computed by the extended Euclidean algorithm for GCD of the polynomial with $m(x)$. This multiplication is denoted $\bullet$.

Multiplication of $b(x)$ by $x =$ `0x02` is a left shift of one bit position followed by a conditional XOR with $m(x)$: XOR with $m(x)$ iff the bit shifted out was `1`. Therefore, multiplication of two polynomials may be performed by up to 8 repeated left shifts and conditional XORs.

The byte inverse is used in `ByteSub` and in the key schedule. Byte multiplication is used in the 32-bit operations of `MixColumn`.

Code to multiply two bytes

To multiply bytes $c = a \bullet b$:

$c = 0$

for (i=0; i<8; i++) $\{$

if (b & $2^i \neq 0$) $c = c \oplus a$

$a = a + a$ // $+$, not $\oplus$

if ($a \geq 256$) $a = a \oplus$0x11B

$\}$

Example. Multiply 0xB7•0xA5.

The first step is to multiply 0xB7 by x^i for $0 \leq i \leq 7$:

$$0xB7 \bullet 0x01 = 0xB7$$

$$0xB7 \bullet 0x02 = 0x75$$

$$0xB7 \bullet 0x04 = 0xEA$$

$$0xB7 \bullet 0x08 = 0xCF$$

$$0xB7 \bullet 0x10 = 0x85$$

$$0xB7 \bullet 0x20 = 0x11$$

$$0xB7 \bullet 0x40 = 0x22$$

$$0xB7 \bullet 0x80 = 0x44$$

Since $0xA5 = 10100101 = 80 \oplus 20 \oplus 04 \oplus 01$, we have

$$0xB7 \bullet 0xA5 = 0xB7 \bullet (80 \oplus 20 \oplus 04 \oplus 01)$$

$$= 0x44 \oplus 0x11 \oplus 0xEA \oplus 0xB7 = 0x08.$$

Thirty-two bit word operations.

Thirty-two bit words are regarded as four bytes, which are the coefficients of a polynomial of degree three with coefficients in $GF(2^8)$.

Addition of two 32-bit words is simple: Just add (XOR) the coefficients. This is the same as XORing the two 32-bit words.

Multiplication of two 32-bit words is done by multiplying the polynomials modulo $M(x) = x^4 + 1$. This multiplication is denoted $\otimes$. If

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

and

$$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0,$$

then

$$d(x) = a(x) \otimes b(x) = d_3x^3 + d_2x^2 + d_1x + d_0$$

may be computed by

$$d_0 = a_0 \bullet b_0 \oplus a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$$

$$d_1 = a_1 \bullet b_0 \oplus a_0 \bullet b_1 \oplus a_3 \bullet b_2 \oplus a_2 \bullet b_3$$

$$d_2 = a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \oplus a_3 \bullet b_3$$

$$d_3 = a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$$

Multiplication of a cubic polynomial by x consists of a circular left shift of the bytes in the word representing the polynomial.

Example.

Multiply $\texttt{0xB7A5662F} \otimes \texttt{0x03010102}$ modulo $M(x) = x^4 + 1$.

We use the formulas above with $a_0 = \texttt{0x2F}$, $a_1 = \texttt{0x66}$, $a_2 = \texttt{0xA5}$, $a_3 = \texttt{0xB7}$, $b_0 = \texttt{0x02}$, $b_1 = \texttt{0x01}$, $b_2 = \texttt{0x01}$ and $b_3 = \texttt{0x03}$. In the formula for d_0 we have

$$a_0 \bullet b_0 = \texttt{0x2F} \bullet \texttt{0x02} = \texttt{0x5E}$$

$$a_3 \bullet b_1 = \texttt{0xB7} \bullet \texttt{0x01} = \texttt{0xB7}$$

$$a_2 \bullet b_2 = \texttt{0xA5} \bullet \texttt{0x01} = \texttt{0xA5}$$

$$a_1 \bullet b_3 = \texttt{0x66} \bullet \texttt{0x03} = \texttt{0xAA}$$

and so

$$d_0 = a_0 \bullet b_0 \oplus a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$$

$$= \texttt{0x5E} \oplus \texttt{0xB7} \oplus \texttt{0xA5} \oplus \texttt{0xAA} = \texttt{0xE6}.$$

Similarly,

$$d_1 = \texttt{0xCC} \oplus \texttt{0x2F} \oplus \texttt{0xB7} \oplus \texttt{0xF4} = \texttt{0xA0}$$

$$d_2 = \texttt{0x51} \oplus \texttt{0x66} \oplus \texttt{0x2F} \oplus \texttt{0xC2} = \texttt{0xDA}$$

$$d_3 = \texttt{0x75} \oplus \texttt{0xA5} \oplus \texttt{0x66} \oplus \texttt{0x71} = \texttt{0xC7}.$$

Finally, $\texttt{0xB7A5662F} \otimes \texttt{0x03010102} = \texttt{0xC7DAA0E6}$.

Rijndael has 10, 12 or 14 rounds, depending on the block and key lengths. The block length and key length can be chosen independently to be 128, 192 or 256 bits. Let Nb be the length of the block in 32-bit words (Nb $=$ 4, 6 or 8). Let Nk be the length of the key in 32-bit words (Nk $=$ 4, 6 or 8). Let Nr be the number of rounds. Then Nr $=$ 14 if either Nb or Nk $=$ 8. Otherwise, Nr $=$ 12 if either Nb or Nk $=$ 6. Finally, Nr $=$ 10 if both Nb and Nk $=$ 4.

Different parts of the Rijndael algorithm operate on the intermediate result, called the *State*. The State is a rectangular array of bytes with four rows and `Nb` columns.

The Key is expanded and placed in an array `W[Nb*(Nr+1)]`. The first `Nk` words of `W` are the Key. Each subsequent word is the XOR of the previous word and the word `Nk` words back in the array, except that words whose subscript is a multiple of `Nk` have the previous word transformed before the XOR.

`ByteSub(State)` transforms each byte in the State by replacing it with its multiplicative inverse in $GF(2^8)$ (except that `0x00` is unchanged) and then applying an affine transformation to the inverse.

`ShiftRow(State)` is a circular left shift of the rows in the State by various byte offsets which depend on `Nb`.

In `MixColumn(State)` the columns of the State are considered to be cubic polynomials with coefficients in $GF(2^8)$ and each is multiplied ($\otimes$) modulo $x^4 + 1$ with the fixed polynomial

$$c(x) = \texttt{0x03}x^3 + \texttt{0x01}x^2 + \texttt{0x01}x + \texttt{0x02}.$$

This polynomial $c(x)$ is relatively prime to $x^4 + 1$ and so is invertible. The inverse of `MixColumn(State)` is multiplication by

$$d(x) = \texttt{0x0B}x^3 + \texttt{0x0D}x^2 + \texttt{0x09}x + \texttt{0x0E}.$$

`AddRoundKey(State, RoundKey)` is simply an XOR of State with RoundKey.

The Square attack is a chosen-plaintext attack that exploits the byte structure of Rijndael. It is faster than exhaustive search for Rijndael versions with up to six rounds, but does not work for seven or more rounds.

Consider a set of 256 AES states (4×4 arrays of bytes). There are 16 byte positions in a state. A byte position is called *active* if all 256 possible bytes occur in that position in the 256 states. A byte position is called *passive* if that byte is the same (constant) in all 256 states.

A Λ-set is a set of 256 AES states in which every byte position is either active or passive.

A Λ-set is a set of 256 AES states in which every byte position is either active or passive.

Applying `ByteSub` or `AddRoundKey` to a Λ-set yields a Λ-set with active bytes in the same positions.

Applying `ShiftRow` to a Λ-set yields a Λ-set with active bytes shifted.

Applying `MixColumn` to a column with one active and three passive bytes gives a column with four active bytes because every output byte of `MixColumn` is a linear combination with invertible coefficients of the four input bytes in that column.

A set of 256 Rijndael states is *balanced* if their xor is the 0 state.

Every Λ-set is balanced.

The Square attack uses a Λ-set of plaintexts with one active and 15 passive byte positions.

Trace the Λ-set through the encryption. It remains a Λ-set until the input to the `MixColumn` of the third round.

One can show that even the input to the fourth round is balanced because of the constant coefficients of the polynomial used for multiplication in `MixColumn`, but the balance is usually destroyed by the `ByteSub` of the fourth round.

Assume we have a four-round version of AES and that the fourth round has no `MixColumn`. The output of the fourth round is known because it is the ciphertext.

We determine the fourth-round key one byte at a time. Xor each putative key byte value with the corresponding ciphertext byte and pull it back through `ByteSub`. If the resulting byte is not balanced, the key byte guess is wrong. Usually, there will be just one key byte value that gives a balanced input byte to `ByteSub`.

Once the fourth round key is known, one can determine the AES key by working backwards through the key expansion algorithm.

One can extend this attack by adding a fifth round at the end and a round at the beginning. Thus one can break Rijndael reduced to five or six rounds.