The Advanced Encryption Standard

Successor of DES
DES considered insecure; 3DES considered too slow.
NIST competition in 1997
15 submissions 1998; 5 finalists 1999
Rijndael was winner, adopted 2000, now called AES.

Still considered to have very good security. The main known attack is a "related key" attack: if the attacker knows a key, and knows that you are using a "related" key, then some information leakage may occur. If AES is used correctly, keys are always chosen randomly, and therefore are never "related". So in that case, this has no practical significance.

AES parametrisable:

- Block size 128
- key sizes of 128, 192 and 256 bits
- 10, 12 or 14 rounds of encryption for each of those key sizes

Similarly to DES, AES works in rounds, with round keys.
Here, we look at AES-128.

AES is a substitution-permutation network (not a Feistel network).
Start by arranging the message in 4×4 matrix of 8-bit elements, filling it downwards and then right
Each round has following operations:

- Substitution: Operating on every single byte independently. This gives the *non-linearity* in AES.
- Byte permutation ShiftRows
- Column manipulation MixColumns. ShiftRows and MixColumns give us *diffusion* in AES.
- Xor with round key This provides the *key addition* in AES.

The 10 rounds are preceded by a key addition (thus, there are 11 key additions in total). The final round is slightly simpler: there's no MixColumns.

Byte operations in AES

AES is a byte-oriented cipher. The 128 bit "state" which is mapulated by the rounds is considered as 16 bytes, arranged in a matrix:

$$\begin{bmatrix} A_0 & A_4 & A_8 & A_{12} \\ A_1 & A_5 & A_9 & A_{13} \\ A_2 & A_6 & A_{10} & A_{14} \\ A_3 & A_7 & A_{11} & A_{15} \end{bmatrix}$$

To define the operations used in AES, we need two operations on bytes: $\oplus$ and $\otimes$. Each of those operations takes two bytes, and returns another byte. For example,

$$11000010 \oplus 00101111 = 11101101 \text{ and } 11000010 \otimes 00101111 = 00000001$$

In fact, the operation $\oplus$ is just bitwise-xor. The operation $\otimes$ on 8-bit numbers is called multiplication in $\mathbb{F}_{2^8}$, and we will define it later. Implementation: $\otimes$ is done as a lookup table in code.

Substitution

Each byte in the current 4x4 state matrix is used as an index to the S-box, obtaining a new byte for that position.

The S-box is shown on the following slide.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Substitution

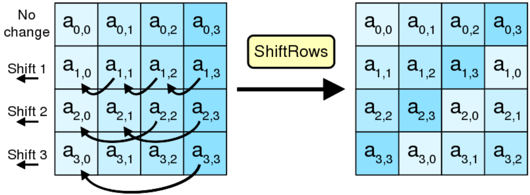
Unlike in the case of DES, the S-box isn't just an arbitrary look-up table.

We will see later that it is defined using a calculation in the field $\mathbb{F}_{2^8}$ (details later).

Implementation: done as a lookup table in code.

Shift Rows

ShiftRows performs cyclic shift on the state matrix



Source: Wikipedia

MixColumns

Mixing each column separately

Achieved by multiplying with matrix

$$\begin{bmatrix} b_{0,i} \\ b_{1,i} \\ b_{2,i} \\ b_{3,i} \end{bmatrix} = \begin{bmatrix} 0x02 & 0x03 & 0x01 & 0x01 \\ 0x01 & 0x02 & 0x03 & 0x01 \\ 0x01 & 0x01 & 0x02 & 0x03 \\ 0x03 & 0x01 & 0x01 & 0x02 \end{bmatrix} \cdot \begin{bmatrix} a_{0,i} \\ a_{1,i} \\ a_{2,i} \\ a_{3,i} \end{bmatrix}$$

In this matrix multiplication, we use $\oplus$ (xor) for addition, and the previously-mentioned "special" operation $\otimes$ for multiplication.

Adding Round Key

Key is 128 bits
Key schedule to compute 10x 128-bit round keys
They can also be represented as 4×4 matrix.
Simply xor'ed to state matrix.

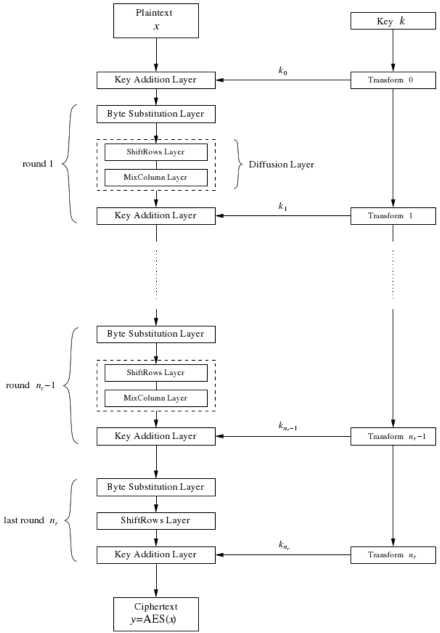
Key schedule

Derive round keys K_i as follows:
Split K into four words W_0, W_1, W_2 and W_3 of 32 bits each.

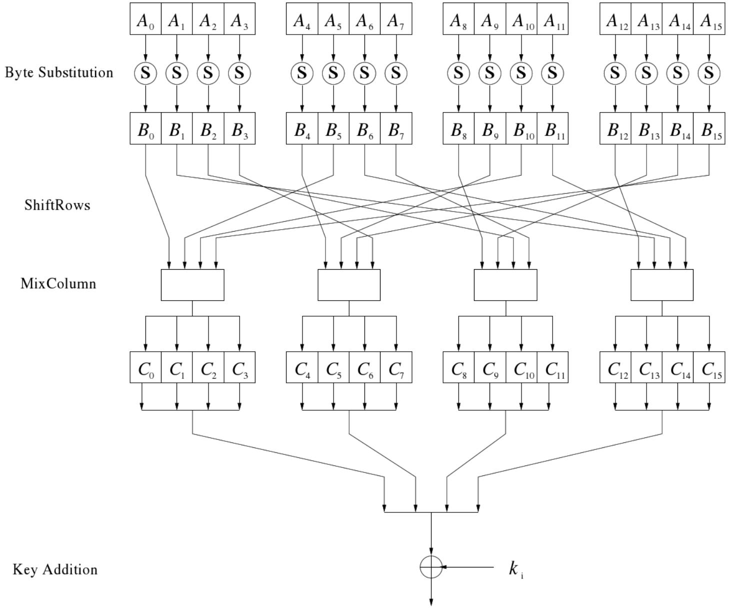
$$
\begin{aligned}
&\textbf{for } i := 1 \textbf{ to } 10 \textbf{ do}\\
&\quad T := W_{4i-1} \lll 8\\
&\quad T := \mathsf{SubBytes}(T)\\
&\quad T := T \oplus RC_i\\
&\quad W_{4i} := W_{4i-4} \oplus T\\
&\quad W_{4i+1} := W_{4i-3} \oplus W_{4i}\\
&\quad W_{4i+2} := W_{4i-2} \oplus W_{4i+1}\\
&\quad W_{4i+3} := W_{4i-1} \oplus W_{4i+2}\\
&\textbf{end}
\end{aligned}
$$

Here, RC_i are byte constants defined in AES (we will see their exact definition later). The round keys K_i are obtained as follows:

$$K_i = W_{4i}, W_{4i+1}, W_{4i+2}, W_{4i+3}.$$



Plaintext
x
Key k
Key Addition Layer
k_0
Transform 0
Byte Substitution Layer
round 1
ShiftRows Layer
MixColumn Layer
Diffusion Layer
Key Addition Layer
k_1
Transform 1
Byte Substitution Layer
round n_r-1
ShiftRows Layer
MixColumn Layer
Key Addition Layer
k_{n_r-1}
Transform n_r-1
Byte Substitution Layer
last round n_r
ShiftRows Layer
Key Addition Layer
k_{n_r}
Transform n_r
Ciphertext
y=AES(x)



A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15
Byte Substitution
S
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15
ShiftRows
MixColumn
C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15
Key Addition
k i

AES and finite fields of polynomials.

Definition

- A *polynomial* is an expression of the form

$$a_n x^n + \ldots + a_2 x^2 + a_1 x + a_0,$$

  where x is a variable symbol, and $a_0, \ldots, a_n$ are values chosen from some set.

  We say this is a "polynomial in x".
  Often $a_i \in \mathbb{Z}$ (each i), and we say it's a "polynomial over $\mathbb{Z}$".

- The a_i's are called **coefficients**, and n is called the **degree** of the polynomial.

- We denote the set of all polynomials in x over $\mathbb{Z}$ as $\mathbb{Z}[x]$.

Polynomials with bit coefficients

Instead of having the coefficients in $\mathbb{Z}$, we can use *bits*.
So the coefficients will be 0 or 1.
Because the set of bits $\{0, 1\}$ is written $\mathbb{F}_2$, we write the set of polynomials over bits as $\mathbb{F}_2[x]$.

Operations on polynomials in $\mathbb{F}_2[x]$

1. You can add them. Just add the respective coefficients, remembering that the coefficients are *bits* (thus, $1 \oplus 1 = 0$).
2. You can multiply them. You might remember how to multiply polynomials from school. Polynomials in $\mathbb{F}_2[x]$ work the same way, but again, you need to remember that the coefficients are bits.
3. You can divide one polynomial by another, yielding a quotient and remainder. See the example in the notes!
4. Combining these ideas, you can multiply two polynomials *modulo a third one*!

Irreducible polynomials

Definition

An integer n is called *prime* if its only divisors are 1 and n

The same notion for polynomials is called irreducible:

Definition

A polynomial $p(x) \in \mathbb{F}_2[x]$ is called *irreducible* if its only divisors are $p(x)$ and the constant polynomial $1 \in \mathbb{F}_2[x]$.

If $p(x)$ is an irreducible polyomial in $\mathbb{F}_2[x]$, then we write $\mathbb{F}_2[x]/p(x)$ for the set of polynomials in $\mathbb{F}_2[x]$ considered modulo $p(x)$.

Using polynomials to define a new operation on bitstrings

We identify polynomial of degree 7 with a bitstring length 8:

$$\begin{array}{cccccccc} x^7 & +x^6 & & +x^4 & +x^3 & & & +1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}$$

Multiplication of polys as a new bitstring op

Consider multiplication in $\mathbb{F}_{2^3} = \mathbb{F}_2[x]/p(x)$, with $p(x) = x^3 + x + 1$ as irreducible polynomial.
This is an operation on 3-bit strings. Example:

$$\begin{array}{ccccccc} (x^2+x+1) & \cdot & (x^2+1) & \equiv & x^2+x & \pmod{x^3+x+1} \\ 111 & \otimes & 101 & = & 110 & \end{array}$$

Observe that the choice of irreducible polynomial really matters in the definition of $\otimes$. For instance, if our choice of irreducible polynomial were $x^3 + x^2 + 1$ then the example would look like this:

$$\begin{array}{ccccccc} (x^2+x+1) & \cdot & (x^2+1) & \equiv & 1 & \pmod{x^3+x^2+1} \\ 111 & \otimes & 101 & = & 001 & \end{array}$$

Two operations over bitstrings length 3

The previous example defined $\otimes$. So now we have two operations over bitstrings of length 3:

$\oplus$	000	001	010	011	100	101	110	111
000	000	001	010	011	100	101	110	111
001	001	000	011	010	101	100	111	110
010	010	011	000	001	110	111	100	101
011	011	010	001	000	111	110	101	100
100	100	101	110	111	000	001	010	011
101	101	100	111	110	001	000	011	010
110	110	111	100	101	010	011	000	001
111	111	110	101	100	011	010	001	000

$\otimes$	000	001	010	011	100	101	110	111
000	000	000	000	000	000	000	000	000
001	000	001	010	011	100	101	110	111
010	000	010	100	110	011	001	111	101
011	000	011	110	101	111	100	001	010
100	000	100	011	111	110	010	101	001
101	000	101	001	100	010	111	011	110
110	000	110	111	001	101	011	010	100
111	000	111	101	010	001	110	100	011

Note that 000 is a special element. It is the identity element for $\oplus$, and it is a "destructor" for $\otimes$, i.e. $000 \otimes b_2 b_1 b_0 = 000$. Each element in the table for $\oplus$ has an inverse w.r.t. 000. Also, 001 is the identity element for $\otimes$, and each element in the table for $\otimes$ has an inverse w.r.t. 001. All this means that we have defined a mathematical structure called a *field*.

Bitstring operation, continued

The $\otimes$ operation is computed as follows:

- Each bitstring $a_2 a_1 a_0$ is interpreted as a polynomial $a_2 x^2 + a_1 x + a_0$.
- The two polymomials are multiplied together, and reduced modulo our chosen polynomial, which is this one: $x^3 + x + 1$.
- The result is converted back into a 3-bit string.

You can check that the field properties are satisfied.

Connection with AES

In AES, we use the field

$$\mathbb{F}_2[x] \;/\; (x^8 + x^4 + x^3 + x + 1)$$

This gives us two operations $\oplus$ and $\otimes$ on bytes. For example:

$$0x53 \otimes 0xCA = 0x01$$

These two operations is used to define the MixColumns operation and the S-boxes of AES.

Substitution in AES

The substitution operation for a byte B is defined as follows.

1. First compute the multiplicative inverse of B in the AES field, to obtain $B' = [x_7, \ldots, x_0]$. In this step, the zero element is mapped to $[0, \ldots, 0]$.
2. Then compute a new bit vector $B'' = [y_7, \ldots, y_0]$ with the following transformation in $\mathbb{F}_2$ (observe that the vector addition is the same as an xor $\oplus$):

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

The result of the substitution is B''.

Key schedule in AES

Recall that the key schedule algorithm in AES used some constants, $RC_1, \ldots, RC_{10}$. We didn't define these, but we can do so now.

$$RC_i = x^{i-1} \mod x^8 + x^4 + x^3 + x + 1$$

Thus, RC_1 is the byte 00000001, RC_3 is the byte 00000100, and (a bit harder to calculate!) RC_{10} is the byte 00110110.

AES security

AES has been subjected to a huge amount of analysis and attempted attacks, and has proved very resilient. So far, there are only very small "erosions" of AES:

- There is a meet-in-the-middle key recovery attack for AES-128. It requires 2^{126} operations, so it is only about four times faster than brute-force.
- There is a "related key" attack on AES-192 and AES-256. This means that if you use two keys that are related in a certain way, the security may be reduced. But this is an "invalid" attack, since correct use of AES means you will always choose random keys.

People have also studied simplified versions of AES, e.g. by considering a reduced number of rounds. A large number of small erosions exist in this situation too.
The Snowden documents revealed that the NSA has teams working on breaking AES, but there is no evidence that they have achieved much beyond what is publicly known.

AES security

Does AES satisfy the security definition for block ciphers?

More informally: suppose an adversary has access to an API in the manner of the definition on slide 37. Could the adversary distinguish whether the API is using AES with a random key, or is using a random permutation?

The answer to the informal question is: it is believed that an adversary could not distinguish this, whatever number of queries the adversary could practically make. In other words, it is believed that AES is secure.

The first question, on whether AES satisfies the definition for block ciphers, is not well-posed. The definition allows us to choose an arbitrary key length, and look at how the adversary's advantage diminishes as that key length increases. However, AES is defined only for three key lengths.