2.2.5 Advanced Encryption Standard (AES)

A collection of proposals has been studied by the (American) National Institute of Standards and Technology (NIST for short) for a new industrial standard for a block cipher.

The names of these proposals are CAST-256, CRYPTON, DFC, DEAL, E2, FROG, HPC, LOKI97, MAGENTA, MARS, RC6, RIJNDAAEL, SAFER+, SERPENT and TWOFISH (see the web page Advanced Encryption Standard).

The second round of the selection was concluded in August 1999. The following contenders remained in the race: MARS (IBM), RC6TM (RSA Laboratories), RIJNDAEL (Daemen and Rijmen), SERPENT (Ross Anderson, Eli Biham, and Lars Knudsen) and TWOFISH (Bruce Schneier e.a.).

On October 2, 2000, the final selection was made: RIJNDAEL!

2.2.6 Rijndael

Like most modern block ciphers, Rijndael is an iterated block cipher: it specifies a transformation, also called the round function, and the number of times this function is iterated on the data block to be encrypted/decrypted, also referred to as the number of rounds.

Rijndael is a substitution-linear transformation network, i.e. it is not a Feistel-type block cipher like e.g. DES. The block length and the key length can be independently specified to 128, 192, and 256 bits.

The number of rounds in Rijndael depends in the following way on the the block length and the key length.

cipher \ key	128	192	256
128	10	12	14
192	12	12	14
256	14	14	14

The round function consists of the following operations:

- ByteSub (affects individual bytes),
- ShiftRow (shifts rows),
- MixColumn (affects each column),
- RoundKey addition (overall XOR).

These are applied to the intermediate cipher result, also called the State: a 4×4, 4×6, resp. 4×8 matrix of which the entries consist of 8 bits, i.e. one byte. For example, when the block length is 192, one gets

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

where each $a_{i,j}$ consists of 8 bits, so it has the form $\{(a_{i,j})_0, (a_{i,j})_1, \ldots, (a_{i,j})_7\}$. For example, $a_{0,0} = \{1, 0, 1, 1, 0, 0, 0, 1\}$.

Sometimes, we use the one-dimensional ordering (columnwise) i.e.
$a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}, a_{0,1}, \ldots, a_{3,5}$.

We define N_b as the number of columns in the array above. So, the the block cipher length is $32\,N_b$ bits, or $4\,N_b$ bytes (each byte consists of 8 bits), or N_b 4-byte words.

Similarly, the Cipher Key length consists of $32\,N_k$ bits, or $4\,N_k$ bytes, or N_k 4-byte words.

□ One Round

ByteSub

This is the only non-linear part in each round.

Apply to each byte $a_{i,j}$ two operations:

1) Interpret $a_{i,j}$ as element in GF(2^8) and replace it by its multiplicative inverse,

   if it is not 0, otherwise leave it the same.

2) Replace the resulting 8-tuple, say $(x_0, x_1, \ldots, x_7)$ by

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}.$$

The finite field GF(2^8) is made by means of the irreducible polynomial
$m(\alpha) = 1 + \alpha + \alpha^3 + \alpha^4 + \alpha^8$. This polynomial is not primitive!

Note that both operations are invertible.

```
<<Algebra`FiniteFields`
```

```
f256 = GF[2, {1, 1, 0, 1, 1, 0, 0, 0, 1}];
one = f256[{1, 0, 0, 0, 0, 0, 0, 0}]
α = f256[{0, 1, 0, 0, 0, 0, 0, 0}]
```

```
{1, 0, 0, 0, 0, 0, 0, 0}₂
```

```
{0, 1, 0, 0, 0, 0, 0, 0}₂
```

```
in = {0, 1, 0, 0, 0, 0, 0, 0};
```

$$\text{pol} = \sum_{i=1}^{8} \text{in}[[i]]\,\alpha^{i-1}$$

```
inver = 1 / pol
```

```
{0, 1, 0, 0, 0, 0, 0, 0}₂
```

```
{1, 0, 1, 1, 0, 0, 0, 1}₂
```

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix};$$

```
b = {1, 1, 0, 0, 0, 1, 1, 0};
Mod[A.inver[[1]] + b, 2]
```

```
{1, 1, 1, 0, 1, 1, 1, 0}
```

Instead of performing these calculations, one can also replace them by one substitution table: the ByteSub S-box.

ShiftRow

The rows of the State are shifted cyclically to the left using different offsets: do not shift row 0, shift row 1 over c_1 bytes, row 2 over c_2 bytes, and row 3 over c_3 bytes, where

	c_1	c_2	c_3
128	1	2	3
192	1	2	3
256	1	3	4

.

So

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

becomes

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,0}$
$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,0}$	$a_{2,1}$
$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$

MixColumn

Interpret each column as a polynomial of degree 3 over GF(2^8) and multiply it with

$$(1+\alpha)\,x^3 + x^2 + x + \alpha$$

modulo $x^4 + 1$.

Note that the above polynomial is invertible modulo $x^4 + 1$.

```
g[x_] = (1 + α) x^3 + one x^2 + one x + α
```

```
{0, 1, 0, 0, 0, 0, 0, 0}_2 + x {1, 0, 0, 0, 0, 0, 0, 0}_2 +
 x^2 {1, 0, 0, 0, 0, 0, 0, 0}_2 + x^3 {1, 1, 0, 0, 0, 0, 0, 0}_2
```

Suppose that the first column looks like

```
col = {1 + α + α^3 + α^6 + α^7, one, α^2 + α^4 + α^5 + α^6, α};
col // TableForm
```

```
{1, 1, 0, 1, 0, 0, 1, 1}_2
{1, 0, 0, 0, 0, 0, 0, 0}_2
{0, 0, 1, 0, 1, 1, 1, 0}_2
{0, 1, 0, 0, 0, 0, 0, 0}_2
```

```
colpol[x_] = col[[1]] + col[[2]] x + col[[3]] x^2 + col[[4]] x^3
```

```
x^2 {0, 0, 1, 0, 1, 1, 1, 0}_2 + x^3 {0, 1, 0, 0, 0, 0, 0, 0}_2 +
 x {1, 0, 0, 0, 0, 0, 0, 0}_2 + {1, 1, 0, 1, 0, 0, 1, 1}_2
```

```
ownexpand[expr_] := Collect[expr /. {GF → GF$}, x] /. {GF$ → GF}
```

```
pr[x_] = ownexpand[colpol[x] * g[x]]
prod[x_] = PolynomialMod[pr[x], x^4 - 1]
```

```
x^2 {0, 1, 0, 0, 0, 1, 0, 0}_2 +
 x^6 {0, 1, 1, 0, 0, 0, 0, 0}_2 + x^5 {0, 1, 1, 1, 1, 0, 0, 1}_2 +
 x {1, 0, 0, 1, 0, 0, 1, 1}_2 + x^4 {1, 0, 1, 0, 1, 1, 1, 0}_2 +
 {1, 0, 1, 1, 0, 0, 0, 1}_2 + x^3 {1, 1, 1, 0, 1, 1, 0, 0}_2
```

```
{0, 0, 0, 1, 1, 1, 1, 1}_2 + x^2 {0, 0, 1, 0, 0, 1, 0, 0}_2 +
 x {1, 1, 1, 0, 1, 0, 1, 0}_2 + x^3 {1, 1, 1, 0, 1, 1, 0, 0}_2
```

The inverse operation is a multiplication by

```
h[x_] = (1 + α + α^3) x^3 + (1 + α^2 + α^3) x^2 + (1 + α^3) x + (α + α^2 + α^3) ;
ownexpand[PolynomialMod[g[x] * h[x], x^4 - 1]]
```

```
{1, 0, 0, 0, 0, 0, 0, 0}_2
```

```
ownexpand[PolynomialMod[prod[x] * h[x], x^4 - 1]]
```

```
x^2 {0, 0, 1, 0, 1, 1, 1, 0}_2 + x^3 {0, 1, 0, 0, 0, 0, 0, 0}_2 +
 x {1, 0, 0, 0, 0, 0, 0, 0}_2 + {1, 1, 0, 1, 0, 0, 1, 1}_2
```

Round Key Addition

XOR the whole matrix with a similar sized matrix (i.e. the Round Key) obtained from the cipher key in a way that depends on the round index.

Note that the XOR applied to a byte, really is an XOR applied to the 8 bits in the byte.

For example, if

$$\begin{array}{|c|c|c|c|c|c|}\hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & a_{0,4} & a_{0,5} \\ \hline a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ \hline a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ \hline a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|c|c|}\hline k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} & k_{0,4} & k_{0,5} \\ \hline k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} & k_{1,4} & k_{1,5} \\ \hline k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} & k_{2,4} & k_{2,5} \\ \hline k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} & k_{3,4} & k_{3,5} \\ \hline \end{array}$$

$$= \begin{array}{|c|c|c|c|c|c|}\hline u_{0,0} & u_{0,1} & u_{0,2} & u_{0,3} & u_{0,4} & u_{0,5} \\ \hline u_{1,0} & u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} & u_{1,5} \\ \hline u_{2,0} & u_{2,1} & u_{2,2} & u_{2,3} & u_{2,4} & u_{2,5} \\ \hline u_{3,0} & u_{3,1} & u_{3,2} & u_{3,3} & u_{3,4} & u_{3,5} \\ \hline \end{array} .$$

with $u_{0,0} = a_{0,0} \oplus k_{0,0}$, the coordinate-wise exclusive or.

```
a0,0 = {1, 1, 1, 1, 0, 0, 0, 0}; k0,0 = {1, 1, 0, 0, 1, 0, 1, 0};
Mod[a0,0 + k0,0, 2]
```

```
{0, 0, 1, 1, 1, 0, 1, 0}
```

There is also an initial Round Key addition and one final round that differs slightly from the others (the MixColumn is omitted) .

□ Key Schedule

The round keys are derived from the Cipher Key by the key schedule, consisting of two components. First the Cipher Key is expanded to the so-called Expanded Key of length equal to

32 $N_b(N_r + 1)$ bits, which equals 4 $N_b(N_r + 1)$ bytes,

where N_r denotes the number of rounds and where the +1 comes from the initial round key addition.

For example, when the block length is equal to 128 and the number of rounds is 10, one needs $128 \times (10 + 1) = 1408$ bits of expanded key.

Then the Round Keys are selected from this Expanded key: the initial Round Key addition uses the first 32 N_b bits, i.e. the first 4 N_b bytes, of the Expanded Key, the first round the next 32 N_b bits of the Expanded Key, etcetera.

The key expansion depends on N_k the number of 4-byte words in the key. Below, we only present the case that $N_k = 4$ or 6.

Let

$$(W_0, W_1, \ldots, W_{N_b(N_r+1)-1})$$

denote the Expanded Key in 4-byte-word notation.

Step 1: Fill $(W_0, W_1, \ldots, W_{N_k-1})$ with the Cipher Key.

```
Step 2: For $i \geq N_k$ Do  If $i$ is not divisible by $N_k$ Do $W_i = W_{i-1} \oplus W_{i-N_k}$
                              Else   apply cyclic shift to the left to the 4 bytes in $W_{i-1}$;
                                     apply ByteSub to each of the 4 bytes;
                                     XOR a round constant to the outcome;
                                     XOR with $W_{i-N_k}$.
```

The round constant in round i, $i \geq 1$, has the form $(x^{i-1}, 0, 0, 0)$. Here x^{i-1} and 0 need to be interpreted as elements in GF(2^8), so they are a byte (of 8 bits) determined by α^{i-1} resp 0 modulo $1 + \alpha + \alpha^3 + \alpha^4 + \alpha^8$.

```
i = 8;
PolynomialMod[α^i, {α^8 + α^4 + α^3 + α + 1, 2}]
```

```
{1, 1, 0, 1, 1, 0, 0, 0}_2
```

Note that each new key word depends on the last N_k key words. So, with limited memory resources one can compute the round keys on the fly.

□ Implementation Aspects

Because all elementary operations in Rijndael are based on bytes (i.e. 8 bits), it can be very efficiently implemented by means of 8-bit processors (typical for current smart cards). To prevent timing attacks certain precautions need to be taken.

On the other had, Rijndael works with columns that exist of four bytes, i.e. 32 bits. This means that PC's with 32-bit processors are very well suited for efficient implementions of Rijndael. For encryption, the steps of the round function can be combined and implemented using four Table look-ups and 5 XOR's per column. The tables then require 4 Kbytes of RAM.

Note also that there is a considerable amount of parallelism in the round function. All steps

of this transformation operate in a parallel way on bytes, rows or columns of the State. Most of the XOR's and the table look-up implementation (for ByteSub) can be done in parallel.

In applications where more encipherments take place under the same key, it is often advantageous to make one key expansion and store the outcome. If not enough RAM is available for storing the round keys or if the key changes for each encryption, e.g. in MDC constructions, one can compute the round keys 'on-the-fly', i.e. in parallel to the execution of the succesive rounds.

A disadvantage of substitution-linear transformation networks compared to Feistel structures is that different implementations for the cipher and its inverse are needed, i.e. the inverse of the round function is needed.

In Rijndael the inverse functions of ByteSub, ShiftRow, MixColumn (the RoundKey Addition is symmetric) are needed for decryption. The order of these transformations is reversed, implying that the non-linear ByteSub operation is the last step in the inverse of the round. This can not be combined using a Table look-up implementation like the one for encryption and therefore gives rise to a slight performance degradation. The decryption round keys can also be computed 'on-the-fly', however, a one-time execution of the key scheduling is needed for the generation of the first decryption round key (unlike e.g. in DES, where the first decryption key can be obtained from the Cipher Key by a single computation). Note that in some applications, like the calculation of MAC's and MDC's, or applying the cipher in CFB or OFB-mode, the inverse function is never needed.

Software implementations of Rijndael achieve encryption/decryption speeds of approximately 50 Mbits/second on a 32-bit Pentium Pro architecture (rated 200 Mhz). In dedicated hardware (ASIC), a pipelined implementation exists that reaches encryption/decryption speeds of 6 Gbits/second.