Section 4 Lecture 28 - AES (Advanced Encryption Standard)

With DES being cracked in the late 1990s, a new standard of cipher was needed. After a "competition" involving a variety of different ciphers, a version of the *Rijndael* cipher developed by Belgian cryptographs Joan Daeman and Vincent Rijmen was selected to be the new standard for security of secret documents, and given the name *Advanced Encryption Standard* (*AES*). It was officially approved as an international worldwide standard in around 2001.

Here we will discuss briefly the main principles of its operation to supplement the slide material.

AES

AES operates on a fixed message block size of 128 bits. There are variants of the keyword length (e.g. AES-192 and AES-256) but we will look only at a keyword length of 128 bits as well. This is sometimes referred to as AES-128 but generally when we refer to AES we mean a 128-bit keyword.

Like DES, AES uses a series of *rounds*, though only ten rounds are used here. For each of the rounds, we generate a *round key* from the main keyword. This is done using a *key schedule* as with DES, but is more complicated, and involves cyclic permutations, finite field polynomials and S-boxes – we will omit the details here.

State

At any time the 128-bit block we are currently working with (i.e. the encryption of the message block) is referred to as the *state*. This can be expressed as a 4 x 4 matrix consisting of the *bytes* (blocks of 8 bits) in each element – note that as expected, there are 16 bytes contained inside the block, since 16 lots of 8 bits = 128 bits.

The ordering is called *column-major ordering* which means that if our bytes are $b_1, b_2, \ldots, b_{16}$, we write them in the matrix as

b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}
b_4	b_8	b_{12}	b_{16}

They are normally written in *hexadecimal* – hopefully you know how to use hexadecimal but I will run through in the lecture slides. As a quick guide, hexadecimal is base 16 arithmetic, so we work in powers of 16. This means we need 16 symbols – we use 0-9 as standard and then A corresponds to 10, B to 11, C to 12, D to 13, E to 14 and F to 15.

To convert an 8-bit byte into hexadecimal, split it into two halves of four bits, and convert each binary half to its appropriate hexadecimal symbol. For example, the byte 11010110 splits as 1101 0110, which correspond to D and 6 in hexadecimal (just

convert them to binary and then convert to the appropriate symbol) and so this byte in hexadecimal is D6.

SubBytes

After initially XORing the current block with the appropriate *round key*, the main step in each round is to replace each byte (element of the matrix) by another byte obtained from a lookup table (also called an S-box). I have provided the lookup table separately – it is based on a construction involving finite fields and transformations, I will discuss a little further on the slides.

This step is called the *SubBytes* step. It is the crucial part of AES, as it is the part that gives us the non-linearity to stop the cipher being easily broken by linear techniques such as plaintext attacks.

ShiftRows

The next step in each round is to cyclically shift each of the rows of the state matrix, in an operation called *ShiftRows*.

- The first row remains unchanged
- The second row is shifted by one place to the left
- The third row is shifted by two places to the left
- The fourth row is shifted by three places to the left

You could say that the fourth row is shifted one place to the right, but it's easier to write them all as left shifts, as the concept then easily generalises to higher levels of the similar Rijndael collection of ciphers.

This procedure means that each column now contains a byte from each of the four given columns.

MixColumns

Next, we "mix up" the columns in an operation called, naturally enough, *MixColumns*.

To do this, we are going to effectively multiply each column by a matrix – the advantage of matrix multiplication is that each new value picks up on values from all the values in the column. Together with *ShiftRows,* this gives a powerful mixing of the bytes in our state.

The matrix for the operation is defined as $\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$. However, the multiplication by these numbers (and their addition) does not quite represent standard multiplication and addition, and is instead defined as follows:

Multiplying a byte by 1

This leaves the byte unchanged

Multiplying a byte by 2

If the first bit is 0, then just remove it and add 0 to the end – for example 01110001 would go to 11100010
If the first bit is 1, then remove it and add 0 to the end, and then XOR the answer by 00011011 (this is always fixed, it is 1B in hexadecimal) – for example 11110001 goes to 11100010, which XOR'd with 00011011 gives 11111001

Multiplying a byte by 3

Multiply by 2 as above, and then XOR the result with the original byte

Addition

Just use XOR

Note that this is very similar to standard multiplication and addition but ensures we get a byte as our answer (and not anything bigger) – can you think why?

Let's see a brief example (referenced from http://crypto.stackexchange.com)

Consider the column $\begin{pmatrix} D4 \\ BF \\ 5D \\ 30 \end{pmatrix}$. Hence we want to do the "matrix multiplication"

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} D4 \\ BF \\ 5D \\ 30 \end{pmatrix}.$$

I'll do the first entry here – you can then follow the principles to get the other entries

- We need to do 2 x D4. In binary, D4 is 11010100. Multiplying it by 2 according to the definition above gives firstly 10101000 (removing the first bit and adding 0 to the end) and then 10110011 (taking the XOR with 00011011 as the first bit was 1).
- We need to do 3 x BF. In binary, BF is 10111111. Multiplying it by 2 according to the definition gives firstly 01111110 (removing the first bit and adding 0 to the end) and then 01100101 (taking the XOR with 00011011 as the first bit was 1). Then taking the XOR of this with the original byte BF = 10111111, we get 11011010
- 1 x 5D leaves it unchanged, which is 01011101 in binary
- 1 x 30 leaves it unchanged, which is 00110000 in binary

Finally taking the XOR of our four multiples gives the XOR of

10110011
11011010
01011101
00110000

which gives 00000100, which is 04 in hexadecimal

Hence our first new column element is 04. I will leave you (we'll discuss a little further in the lecture) to show that the final new column is $\begin{pmatrix} 04 \\ 66 \\ 00 \\ E5 \end{pmatrix}$, and you can try some more in the tutorial.

AddRoundKey

Finally in each round, we XOR the final 128-bit block with the appropriate round key, which is known as *AddRoundKey*

Final steps

Finally, after all the rounds, we do *SubBytes, ShiftRows* and *AddRoundKey* again, and that concludes our cipher.

AES

There are no feasible known attacks for AES better than brute force. The only times it has been "broken" has been when it was implemented in an outer insecure system (such as its use in SSL). Attacks have been proposed but not found possible. Because of its mathematical properties, it may be that as our understanding of mathematics grows we may be able to find better ways to try and break it, but for now AES is considered secure, and is used worldwide for classified data.