A description of Rijndael

1. What to know

- Rijndael shows how advanced algebra has practical importance.
- You are not responsible for learning the details of the Rijndael algorithm.
- However, we'll be continuing to study the principles of finite fields that are involved, so there may be future homework on that aspect referring to this handout. If so, you will be responsible for learning that material.
- If you would like to learn more, you can take Math 116 (Mathematical Cryptology).

2. Background

Rijndael is the name of an encryption/decryption algorithm that won a competition to be the Advanced Encryption Standard (**AES**) of the US National Institute of Science and Technology (**NIST**). That means that it is approved for US Government use but is also supposed to be good for use by anyone.

The name Rijndael was made up by its inventors, Joan Daemon and Vincent Rijmen of Belgium, out of their own names. It is pronounced approximately "rain-del".

Rijndael is an efficient algorithm that uses a shared secret key (as opposed to a public key). It has versions in several strengths; here we'll discuss the strongest version, with a 256-bit key. Full information can be found at `http://csrc.nist.gov/CryptoToolkit/aes/rijndael/` .

3. Input and Output

The message to be sent is called "plain text"; the encrypted version is called "cipher text". Let's assume that Alice is to send a message to Bob.

First Alice and Bob need to agree on a secret key of 256 bits. This might be sent via a Public-Key system, for example, or produced via Diffie-Hellman or a variant.

Then Alice represents her message in bits (perhaps using the ASCII code) and breaks it into 128-bit blocks. If the message length is not an exact multiple of 128, the last block can be padded. In the use described here,

Alice runs each block separately through the algorithm to encrypt it and then sends the encrypted block as ciphertext.

Bob decrypts the blocks using the same secret key and a procedure that amounts to running Rijndael backwards.

4. Bytes and algebra

As you know, a *byte* means 8 bits (binary digits), e.g., `10110110`, or in hexadecimal for short, `B6`. In running the algorithm, bytes are interpreted in two different ways at different times:

1. A byte is a row vector of length 8 with entries in the field $\mathbb{F}_2$. From this point of view, if we add two bytes they are added bitwise mod 2, which in a computer is the "exclusive or" operation, `a^b` in `C++`.

2. A byte represents an element of the field $\mathbb{F}_{2^8}$ with $2^8 = 256$ elements. In this interpretation, there is a particular element α in that field and powers of α form a basis of $\mathbb{F}_{2^8}$ as an 8-dimensional vector space over the field $\mathbb{F}_2$ as coefficients. Each coefficient is one bit. Examples:

   - `10110110` means $1\alpha^7 + 0\alpha^6 + 1\alpha^5 + 1\alpha^4 + 0\alpha^3 + 1\alpha^2 + 1\alpha + 0$, or simply $\alpha^7 + \alpha^5 + \alpha^4 + \alpha^2 + \alpha$.
   - 1 of the field is `00000001` (why?).
   - α itself is `00000010` (why?).

   The α chosen is a generator of the nonzero elements of the field with the property that $\alpha^8 = \alpha^4 + \alpha^3 + \alpha + 1$.

   Just as you can multiply any two complex numbers once you know that $i^2 = -1$ in $\mathbb{C}$, you can multiply any two expressions involving powers of α once you know the equation for α^8.

   Example: What is `00010000` times itself? This is α^4, so times itself we have α^8, which can then be rewritten in terms of lower powers using the equation, yielding `00011011` as the answer.

5. The rounds of the algorithm for a block

The algorithm consists starting with the block of 128 bits, adding some bits of the key to it (mod 2), and then doing 14 rounds to "mix" it more and more. In each round the bits are changed in a nonlinear way and some "round key" is added in. The last round has a slight modification.

The round key is derived by mixing up the 256-bit key using some of the same methods (to be described) that occur in the rounds themselves. Each key bit is used in several rounds to make the rounds more interdependent.

6. One round

The steps of a round, to be applied repeatedly to each 128-bit block, are as follows.

Step 1: "byte substitution": The 128 bits are broken into 16 bytes, which in this step are treated individually.

1(a) Each byte is treated as an element of $\mathbb{F}_{2^8}$ and is mapped to its inverse in that field (with 0 going to 0).

1(b) Each byte is treated as a vector of length 8 over $\mathbb{F}_2$ and is multiplied by a matrix:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix} \mapsto \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix}$$

1(c) Each byte is treated as a vector of length 8 over $\mathbb{F}_2$ and `11000110` is added to it (mod 2). This flips four of the bits.

Step 2: "row and column mixing": The the sixteen bytes (not bits!) are put into a 4×4 table. (Here the original sixteen bytes will be labeled as $0, 1, \ldots, 15$.)

2(a) Each row is rotated by a different amount:

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

$\mapsto$

0	4	8	12
5	9	13	1
10	14	2	6
15	3	7	11

2(b) The table resulting from 2(a) is treated as a 4×4 matrix with entries in $\mathbb{F}_{2^8}$ and is multiplied on the left by the following matrix to get a new 4×4 table:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix},$$

where the entries are shown in hexadecimal for short. This procedure affects each column independently.

Step 3: "round key": The round key is added in. (As already mentioned, the round key is derived from the original 256-bit key.)

7. Comments

A multiple-round algorithm such as Rijndael is somewhat like stirring paint in a bucket. With each round, the space of possible blocks is stirred more and more, so that it becomes impossible to follow any individual particle (the input block). Every bit of the output (the ciphertext) depends in an extremely complicated way on every bit of the input (the plaintext).

Of course, such an encryption algorithm is **un**like a paint bucket in that it is possible to un-stir the output to decrypt the ciphertext. The un-stirring depends on the fact that each component of each round is one-to-one and so can be reversed.

If you look in detail at a round of Rijndael, you will notice that first the individual bytes are stirred, then the bytes are put in a grid, each row is rotated by a different amount, the columns are transformed linearly. It's sort of like dicing cheese and mooshing the pieces around. Finally, the round key is added in as a kind of control to make the final result depend crucially on the key.

The first piece of the process, 1(a), is the most important: Each byte is replaced by its inverse in a field. This is to introduce some nonlinearity. Otherwise, if you were to compose steps that are all linear transformations (plus constants) with scalars in $\mathbb{F}_2$, the end result would be trivial to decrypt.