Jonathan Chan
Ms. Huang
IB Mathematics HL
2016-01-11

Mathematics Exploration:
An investigation of Symmetric Key Cryptography

For my exploration, I read about symmetric key cryptography, how it is used, how it works, and attempted to present my own interpretation in a more clearer way for my peers — this exploration's target audience — to understand.

Contents

Introduction to Cryptographic Concepts

To most people, "encryption" means adding a password to some text or other data and getting a code back, and the only way to get the original text back is to decrypt it with the original password. This is known as symmetric key cryptography, and is the focus of this exploration.

Terminology for symmetric key cryptography (Bellare, Desai, Jokipii, & Rogaway, 2000):

Encrypt. To turn a piece of information into code that can only be understood by someone we want, and that no one else can understand.

Plain Text. The original text that we want to encrypt.

Cipher Text. The jumble of code we get from encrypting some plain text.

Private Key. The "password" that is used to encrypt and decrypt plain text and cipher text. This should be kept secret if we want to keep our data secret too.

It is called symmetric because the same private key is used for both encryption and decryption. There is such a thing as asymmetric cryptography (public key cryptography) with protocols such as RSA and Diffie-Hellman Key Exchange (Nechvatal, 1991), but this exploration will focus on symmetric key cryptography.

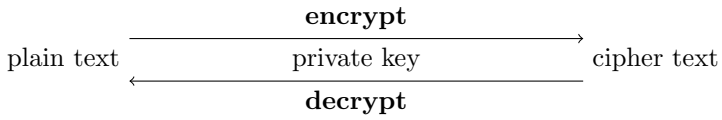

Figure 1: A diagram of symmetric encryption and decryption.

Shannon (1949), considered one of the most prominent founders of cryptography, wrote about the concepts "**confusion**" and "**diffusion**", which are used to describe components of a secure cryptographic system.

Confusion. Each bit of the plain text is somehow transformed to something in the cipher text. The cipher text is significantly different from the plain text. The relationship between the plain text, cipher text, and private key is hard to figure out, so someone with a lot of cipher texts could not figure out the private key or the plain texts.

Diffusion. The bits are somehow mixed up during the process. A small change in the plain text or the private key will lead to a completely different cipher text, and a small change in a cipher text will decrypt to a completely different plain text.

Finally, the cipher text must only be able to be decrypted if one knows the private key. The encryption/decryption process must not be a method of concealing information; only the private key is.

Real-World Applications of Symmetric Key Cryptography

Symmetric key cryptography is widely used to encrypt data on computer networks. For instance, one may use symmetric key cryptography to

- Password-protect information on a hard drive so others may not access it; or

- Password-protect a digital message so it may not be read by anyone else, such as sending a secure email that may not be read by one's email provider.

N.B. Actual implementations use symmetric key cryptography in conjunction with public key cryptography, random number generators, etc. to create better security.

The Advanced Encryption Standard (AES), designed by Daemen and Rijmen (2001), is a widely used symmetric key algorithm. For the purposes of teaching, Holden, Holden, Musa, Schaefer, and Wedig (2010) created a simplified version of AES — Simplified AES (S-AES). This exploration will investigate the mathematics behind AES and S-AES.

Background Mathematics of Symmetric Key Cryptography

Binary Arithmetic

A "bit" is a piece of binary information, either a 1 or a 0. In computer science, a "byte" is a chunk of 8 bits. S-AES uses a similar concept of the "nibble", a chunk of 4 bits (Holden et al., 2010).*

We must also remember that although bits, nibbles, and bytes are representations of numbers as well as of information in general, and so mathematical operations can be done on them.

It is conventional to add a subscript 2 after a binary number to indicate that it is binary.

Binary numbers can be added and subtracted: $1_2 + 1_2 = 10_2$. Other binary number operations (bitwise operations) include negation (NOT "$\neg$"; $\neg 1 = 0$, etc.), conjunction (AND "$\wedge$"; $1 \wedge 1 = 1$, etc.), disjunction (OR "$\vee$"; $0 \vee 1 = 1$, etc.), and exclusive disjunction (XOR "$\oplus$"; $1 \oplus 1 = 0$, etc.).

$\wedge$	0	1
0	0	0
1	0	1

$\vee$	0	1
0	0	1
1	1	1

$\oplus$	0	1
0	0	1
1	1	0

Table 1: Bitwise operations.

Abstract Algebraic Structures

N.B. What is in this section isn't the most thorough; I only included what I thought is important.

This section contains mathematical abstractions extensively documented in mathematics lecture notes such as Ikenaga (2012).

A "**set**" is generally a list of numbers, as we have learned. However, the formal definition of a set does not include arithmetic operations.

Formally, a set is a list of objects that could be numbers, geometric structures, equations, or any other type of structure. Although we can distinguish between different objects in a set, we cannot necessarily add or multiply them. (For instance, we can't "add" a triangle and a square in the set of regular polygons; it makes no sense.)

Nonetheless, if we had a set of numbers and addition and multiplication is defined (under the normal arithmetic rules we are familiar with), we get a "**commutative ring**" R, where

- Addition and multiplication are commutative and associative, and multiplication is also distributive;
- There is an element in R for every x that, when added to x, equals x (the additive identity, generally 0, such that $x + 0 = x, x \in R$);
- There is an element in R for every x that, when multiplied by x, equals x (the multiplicative identity, generally 1, such that $x \cdot 1 = x, x \in R$);
- There is an element in R for every x that, when added to x, equals 0 (the additive inverse, generally $-x$, such that $x + (-x) = 0, x \in R$); and

*This name was presumably chosen because it is smaller than the 8-bit long byte ("bite").

- For any addition or multiplication operation of elements in R, the result is also in R.

The "set" of integers is also a commutative ring because it satisfies these characteristics.

If division is also defined, then we get a "**field**" F. In addition to the rules of commutative rings,

- There is an element in F for every *nonzero* x that, when multiplied by x, equals 1 (the multiplicative inverse, generally x^{-1}, such that $x \cdot x^{-1} = 1, x \in F$).

The set of integers is not a field because division can result in fractions, which are not in the original set of integers. However, the set of rational numbers is a field because it satisfies all these conditions.

Finally, a "**finite field**" (also known as a Galois field) is a field that has a finite number of elements.

Finite Fields

There is comprehensive literature on the topic of finite fields, such as Chapter 2 of Hankerson, Menezes, and Vanstone (2004)'s textbook, or Benvenuto (2012)'s paper on finite fields in actual AES. However, these texts are quite technical; this section aims to explain the concept as simply as possible.

Finite fields are denoted with the notation $\mathbf{GF}(p^n)$, where p is a prime number and n is a positive integer, and there are exactly p^n integers in $\mathbf{GF}(p^n)$. Essentially, $\mathbf{GF}(p^n)$ is the set of numbers from 0 to $p^n - 1$, and the result of any arithmetic operations "done in" this finite field are reduced through modular reduction to yield another number in the finite field.

An irreducible polynomial of degree n is also included in the definition of a finite field. This will be explained further on.

In this section, we will explore the properties of the finite field $\mathbf{GF}(2^4)$.

Modular Arithmetic

Modular reduction (the modulo "mod" operator) is defined by

$$\text{reduced value} = \text{original value} \pmod{x} \iff \text{original value} = \text{reduced value} + kx \tag{1}$$

where $k \in \mathbb{Z}$. Essentially, $a \pmod{b}$ is equal to the remainder when a is divided by b.

Addition & Subtraction

It is conventional to enclose numbers within braces ("{}") to denote that they are a part of the finite field. In finite fields, binary digits are mapped to coefficients in a polynomial. A nibble with the digits/bits $\{(b_0 b_1 b_2 b_3)_2\}$ in $\mathbf{GF}(2^4)$ can be expressed as the polynomial

$$b_0 x^3 + b_1 x^2 + b_2 x^1 + b_3 x^0 \tag{2}$$

where x is a placeholder to help us better understand the relevant algebraic operations.

E.g., the nibble $\{1001_2\}$ in $\mathbf{GF}(2^4)$ can be expressed as

$$1x^3 + 0x^2 + 0x^1 + 1x^0 \tag{3}$$

Addition and subtraction in $\mathbf{GF}(2^4)$ works by converting the nibbles into polynomials and adding or subtracting them. Since this is all in binary (base 2), addition or subtraction of each term in the polynomials must be done modulo 2, where $1 + 1 = 2 \pmod 2 = 0$.

For instance, in $\mathbf{GF}(2^4)$

$$\begin{aligned}\{14\} + \{13\} &= \{1110_2\} + \{1101_2\} \\ &= [(1x^3 + 1x^2 + 1x^1 + 0x^0) + (1x^3 + 1x^2 + 0x^1 + 1x^0)] \pmod 2 \\ &= (2x^3 + 2x^2 + 1x^1 + 1x^0) \pmod 2 \\ &= (0x^3 + 0x^2 + 1x^1 + 1x^0) \pmod 2 \\ &= \{0011_2\} \\ &= \{3\}\end{aligned}$$

Note that addition is similar to directly adding the numbers together in binary form, but digits are not carried upwards if the operation is $1 + 1$, in which case the result is not 10_2 but 0.

Subtraction is similar: just as multiples of 2 can be subtracted from coefficients of the resultant sum, multiples of 2 can be added to a resultant difference.

In $\mathbf{GF}(2^4)$

$$\begin{aligned}\{5\} - \{6\} &= \{0101_2\} - \{0110_2\} \\ &= [(0x^3 + 1x^2 + 0x^1 + 1x^0) - (0x^3 + 1x^2 + 1x^1 + 0x^0)] \pmod 2 \\ &= (0x^3 + 0x^2 + -1x^1 + 1x^0) \pmod 2 \\ &= (0x^3 + 0x^2 + 1x^1 + 1x^0) \pmod 2 \\ &= \{0011_2\} \\ &= \{3\}\end{aligned}$$

Arithmetic in $\mathbf{GF}(2^n)$ has the special property that addition, subtraction, and bitwise exclusive disjunction (the XOR operator "$\oplus$") are equivalent, in which the result is 0 if the operands are the same, and 1 if they are different; see Table 1.

Multiplication

Multiplication is similar as well. The numbers are converted to polynomials, multiplied, reduced through modular arithmetic, and converted back.

However, the resultant polynomial product is not only reduced modulo 2, but also modulo an irreducible prime polynomial that is included in the definition of the finite field. In our case, we shall use $19_{10} = 13_{16} = 1x^4 + 1x^1 + 1x^0$, since this is used in S-AES (more on this later)[†], but any polynomial of degree n that cannot be factored within the field may be used, depending on the definition of the field.

[†] AES uses $283_{10} = 11\text{B}_{16} = x^8 + x^4 + x^3 + x^1 + 1$.

In $\mathbf{GF}(2^4)$

$$\begin{aligned}\{6\} \cdot \{11\} &= \{0110_2\} \cdot \{1011_2\} \\ &= [(0x^3 + 1x^2 + 1x^1 + 0x^0) \times (1x^3 + 0x^2 + 1x^1 + 1x^0)] \pmod{1x^4 + x + 1x^0} \pmod 2 \\ &= (1x^5 + 1x^4 + 1x^3 + 2x^2 + 1x^1 + 0x^0) \pmod{1x^4 + 1x^1 + 1x^0} \pmod 2\end{aligned}$$

Since $a \pmod b$ is the remainder of $a \div b$ (Equation 1),
$(1x^5 + 1x^4 + 1x^3 + 1x^1 + 0x^0) \pmod{1x^4 + 1x^1 + 1x^0}$ is the remainder after polynomial division.

$$\begin{array}{r|l} & x + 1 \\ \hline x^4 + x + 1 & x^5 + x^4 + x^3 + 2x^2 + x \\ & \underline{- x^5 \qquad\qquad - x^2 - x} \\ & \quad x^4 + x^3 \;\; + x^2 \\ & \underline{- x^4 \qquad\qquad - x - 1} \\ & \qquad\quad x^3 \;\; + x^2 - x - 1 \end{array}$$

Therefore

$$\begin{aligned}\{6\} \cdot \{11\} &= (1x^3 + 1x^2 + -1x^1 + -1x^0) \pmod 2 \\ &= (1x^3 + 1x^2 + 1x^1 + 1x^0) \pmod 2 \\ &= \{1111_2\} \\ &= \{15\}\end{aligned}$$

Division

Division is more complicated than multiplication.

In normal arithmetic,

$$a \div b = q \iff a \cdot \frac{1}{b} = q \tag{4}$$

That is, a divided by b is equivalent to multiplying a by the multiplicative inverse of b.

Similarly, in finite field arithmetic, we can find the multiplicative inverse of the divisor b and multiply it by the dividend a.

We can manipulate Fermat's Little Theorem for a general way to find the multiplicative inverse of numbers in finite fields. According to Fermat's Little Theorem,

$$
\begin{aligned}
a^b &= a \pmod b \\
\frac{a^b}{a} &= \frac{a}{a} \pmod b \\
a^{b-1} &= 1 \pmod b \\
\frac{a^{b-1}}{a} &= \frac{1}{a} \pmod b \\
a^{b-2} &= a^{-1} \pmod b
\end{aligned}
$$

In a finite field with b integers, the multiplicative inverse of a is a^{b-2}, with the multiplication operations done in the finite field and reduced each time.

In $\mathbf{GF}(2^4)$ with 16 integers, the multiplicative inverse of any number a is $a^{16-2} = a^{14}$ done in the finite field.

For example,

$$
\begin{aligned}
\{9\} \div \{7\} = q &= \{9\} \times \{7\}^{-1} \\
&= \{9\} \times \{7\}^{14} \\
&= \{9\} \times (\{7\} \times \{7\}) \times \{7\}^{12} \\
&= \{9\} \times \{6\} \times \{7\}^{12} \\
&= \{9\} \times (\{6\} \times \{7\}) \times \{7\}^{11} \\
&= \{9\} \times \{1\} \times \{7\}^{11} \\
&= \cdots \\
&= \{9\} \times \{6\} \\
&= \{3\}
\end{aligned}
$$

We can check that $\{9\} \div \{7\} = \{3\}$:

$$
\begin{aligned}
\{7\} \cdot \{3\} &= \{0111_2\} \cdot \{0011_2\} \\
&= [(0x^3 + 1x^2 + 1x^1 + 1x^0) \times (0x^3 + 0x^2 + 1x^1 + 1x^0)] \pmod{1x^4 + x + 1x^0} \pmod 2 \\
&= (1x^3 + 2x^2 + 2x^1 + 1x^0) \pmod{1x^4 + 1x^1 + 1x^0} \pmod 2 \\
&= (1x^3 + 2x^2 + 2x^1 + 1x^0) \pmod 2 \\
&= (1x^3 + 0x^2 + 0x^1 + 1x^0) \pmod 2 \\
&= \{1001_2\} \\
&= \{9\}
\end{aligned}
$$

Below is a list of multiplicative inverses:

$$
\begin{aligned}
\{1\}^{-1} &= \{1\} \\
\{2\}^{-1} &= \{9\} \\
\{3\}^{-1} &= \{14\} \\
\{4\}^{-1} &= \{13\} \\
\{5\}^{-1} &= \{11\} \\
\{6\}^{-1} &= \{7\} \\
\{7\}^{-1} &= \{6\} \\
\{8\}^{-1} &= \{15\} \\
\{9\}^{-1} &= \{2\} \\
\{10\}^{-1} &= \{12\} \\
\{11\}^{-1} &= \{5\} \\
\{12\}^{-1} &= \{10\} \\
\{13\}^{-1} &= \{4\} \\
\{14\}^{-1} &= \{3\} \\
\{15\}^{-1} &= \{8\}
\end{aligned}
$$

A Note on the Reducing Polynomial

The reducing polynomial must be itself irreducible within the finite field. This can be counterintuitive under normal arithmetic rules.

Firstly, the coefficients of the reducing polynomial must be non-negative integers less than the finite field's prime number (p in $\mathbf{GF}(p^n)$), otherwise the reducing polynomial can be reduced mod p, making it reducible. In $\mathbf{GF}(2^n)$, therefore, the coefficients of the reducing polynomial must be either 0 or 1.

Finally, seemingly irreducible polynomials may be reducible in the finite field. In $\mathbf{GF}(2^4)$, $x^4 + x + 1$ is irreducible. However, seemingly irreducible $x^4 + 1$ is actually reducible in the finite field because

$$
\begin{aligned}
(x+1)^4 &= x^4 + 4x^3 + 6x^2 + 4x + 1 \\
&= (x^4 + 1) \pmod 2
\end{aligned}
$$

$x + 1$ is a factor of $x^4 + 1$ in $\mathbf{GF}(2^4)$, and $x^4 + 1$ is factorable and therefore reducible.

Concluding Finite Fields

Recall the definition of a finite field. A finite field fulfills the characteristics of a field, and remains "finite" by reducing results outside the field through modular reduction.

A Case Study of S-AES

We will explore S-AES (Holden et al., 2010) by encrypting a piece of information.

Recall that nibbles and bytes are representations of information as much as representations of numbers. ASCII (Cerf, 1969) is a way to convert alphabetical letters to numbers and vice versa.

For the purposes of this demonstration, we will be encrypting the data "`IB`" expressed in binary (with ASCII), with the key "`hl`" expressed in binary. Therefore, we will be encrypting the data `0100 1001 0100 0010` with the key `0110 1000 0110 1100`.

Overview

The S-AES encryption process puts the data through the following steps:

- (Setup)
  - Key expansion
  - Add round key
- (Round 1)
  - Substitute nibbles
  - Shift rows
  - Mix columns
  - Add round key
- (Round 2)
  - Substitute nibbles
  - Shift rows
  - Add round key

First, the plain text and the key are converted into a 2×2 grid ("array") of nibbles. That is, a plain text or a key with the nibbles $N_0\ N_1\ N_2\ N_3$ will be converted to an array such as Figure 2.

N_0	N_2
N_1	N_3

Figure 2: A sample 2×2 array of nibbles denoting the positioning thereof.

Therefore, the plain text `0100 1001 0100 0010` is expressed as the array in Figure 3, and the key `0110 1000 0110 1100` is expressed as the array in Figure 4.

`0100`	`0100`
`1001`	`0010`

Figure 3: The 2×2 array representing our plain text.

0110	0110
1000	1100

Figure 4: The 2×2 array representing our key.

Terminology: the "**state array**" refers to the "current" array of nibbles. The results from manipulating the nibbles after each step are stored in the state array, and that state array is used as the input for our next step.

Definitions of Important Steps

These steps are used multiple times throughout the process.

Substitute Nibbles

Each nibble is manipulated to create a different nibble.

For a nibble n, we find the multiplicative inverse n^{-1} in $\mathbf{GF}(2^4)$. Then, that inverse $n^{-1} = \{(b_0 b_1 b_2 b_3)_2\}$ outputs $\{(b'_0 b'_1 b'_2 b'_3)_2\}$ by the equation

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{5}$$

0 cannot be inverted so 0 is used directly without being inverted.

To find the output of $n = \{0011_2\}$,

$$n^{-1} = \{1110_2\} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

$$\implies \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1+0+1+0 \\ 1+1+0+0 \\ 1+1+1+0 \\ 0+1+1+0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ 2 \\ 3 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

$$= \{1011_2\}$$

Therefore the substitute nibble of $\{0011_2\}$ is $\{1011_2\}$.

However, since there is a finite number of inputs and outputs, a computer calculates this beforehand and looks up the output from a table.

input	output	input	output	input	output	input	output
$\{0000_2\}$	$\{1001_2\}$	$\{0100_2\}$	$\{1101_2\}$	$\{1000_2\}$	$\{0110_2\}$	$\{1100_2\}$	$\{1100_2\}$
$\{0001_2\}$	$\{0100_2\}$	$\{0101_2\}$	$\{0001_2\}$	$\{1001_2\}$	$\{0010_2\}$	$\{1101_2\}$	$\{1110_2\}$
$\{0010_2\}$	$\{1010_2\}$	$\{0110_2\}$	$\{1000_2\}$	$\{1010_2\}$	$\{0000_2\}$	$\{1110_2\}$	$\{1111_2\}$
$\{0011_2\}$	$\{1011_2\}$	$\{0111_2\}$	$\{0101_2\}$	$\{1011_2\}$	$\{0011_2\}$	$\{1111_2\}$	$\{0111_2\}$

Table 2: A table of nibble substitutions.

Shift Rows

This simply means to flip the bottom two rows of an array.

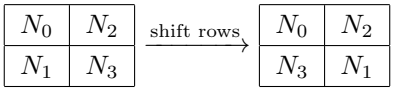

Figure 5: A diagram detailing the shift rows step.

Mix Columns

This step is the process's main source of diffusion, where the bits are reversibly "mixed" around with each other. One way to do this is to multiply each column by a constant 2×2 matrix.

In an array, each column is multiplied by the matrix $\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} = \begin{bmatrix} \{0001_2\} & \{0100_2\} \\ \{0100_2\} & \{0001_2\} \end{bmatrix} = \begin{bmatrix} 1 & x^2 \\ x^2 & 1 \end{bmatrix}$ in $\mathbf{GF}(2^4)$.

That is, a column $\begin{array}{|c|}\hline n_0 \\ \hline n_1 \\ \hline \end{array}$ has its nibbles converted to polynomial form and transformed to

$$\begin{bmatrix} N_0' \\ N_1' \end{bmatrix} = \begin{bmatrix} 1 & x^2 \\ x^2 & 1 \end{bmatrix} \begin{bmatrix} N_0 \\ N_1 \end{bmatrix} \tag{6}$$

Finally, the result is reduced modulo $x^4 + x + 1$ to yield a result in $\mathbf{GF}(2^4)$.

A column $\begin{array}{|c|}\hline \underline{1010} \\ \hline \underline{1111} \\ \hline \end{array}$ is equivalent to $\begin{bmatrix} x^3 + x \\ x^3 + x^2 + x + 1 \end{bmatrix}$.

That is transformed to

$$\begin{aligned} \begin{bmatrix} N_0' \\ N_1' \end{bmatrix} &= \begin{bmatrix} 1 & x^2 \\ x^2 & 1 \end{bmatrix} \begin{bmatrix} x^3 + x \\ x^3 + x^2 + x + 1 \end{bmatrix} \\ &= \begin{bmatrix} 1(x^3 + x) + x^2(x^3 + x^2 + x + 1) \\ x^2(x^3 + x) + 1(x^3 + x^2 + x + 1) \end{bmatrix} \\ &= \begin{bmatrix} x^3 + x + x^5 + x^4 + x^3 + x^2 \\ x^5 + x^3 + x^3 + x^2 + x + 1 \end{bmatrix} \\ &= \begin{bmatrix} x^5 + x^4 + 2x^3 + x^2 + x \\ x^5 + 2x^3 + x^2 + x + 1 \end{bmatrix} \\ &= \begin{bmatrix} x^5 + x^4 + x^2 + x \pmod 2 \\ x^5 + x^2 + x + 1 \pmod 2 \end{bmatrix} \\ &= \begin{bmatrix} x + 1 \pmod{x^4 + x + 1} \\ 1 \pmod{x^4 + x + 1} \end{bmatrix} \end{aligned}$$

Which is equal to $\begin{array}{|c|}\hline \underline{0011} \\ \hline \underline{0001} \\ \hline \end{array}$.

Setup

Key Expansion

The key expansion step is used to generate different keys for each round. Since it would be naive to use the same initial key for every round (due to the principles of confusion and diffusion), AES (Daemen & Rijmen, 2001) generates extra "round keys", or unique keys for each round, based on the initial private key. In total, there will be 3 round keys: the initial private key before round 1 (we'll call this round 0's key), one for round 1, and one for round 2.

The key expansions involve different constants for each round ("round constants"). These same constants are used every single time for any private key. It is a column of two nibbles, with the first nibble defined as the polynomial x^{n+2}, reduced through modular reduction, where n is the round number. The second nibble is always defined as `0000`. Therefore, the round constant is `1000 0000` for round 1 and `0011 0000` for round 2. These round constant columns will be the same for every private key, and manipulated along with the initial private key to generate each round's key.

To generate the first column of round 1's key:

1. Take the last column of round 0's key;
2. Flip it (applying diffusion);
3. Substitute its nibbles by looking it up on Table 2 (applying confusion);
4. Add the round constant `1000 0000` (applying confusion); and
5. Add the first column of the previous round's key (round 0's key) (applying confusion).

Hence, the column 1 of round 1's key is

$$\xrightarrow{\text{last column of round 0's key}} \begin{bmatrix}0110\\1100\end{bmatrix} \xrightarrow{\text{flip}} \begin{bmatrix}1100\\0110\end{bmatrix} \xrightarrow{\text{substitute}} \begin{bmatrix}1100\\1000\end{bmatrix}$$

$$\xrightarrow{\text{add round constant}} \begin{bmatrix}1100\\1000\end{bmatrix} \oplus \begin{bmatrix}1000\\0000\end{bmatrix} = \begin{bmatrix}0100\\1000\end{bmatrix}$$

$$\xrightarrow{\text{add 1st column of round 0's key}} \begin{bmatrix}0100\\1000\end{bmatrix} \oplus \begin{bmatrix}0110\\1000\end{bmatrix}$$

$$= \begin{bmatrix}0010\\0000\end{bmatrix}$$

To generate the second column, add that to round 0's 2nd column.

$$= \begin{bmatrix}0010\\0000\end{bmatrix} \oplus \begin{bmatrix}0110\\1100\end{bmatrix}$$

$$= \begin{bmatrix}0100\\1100\end{bmatrix}$$

We similarly generate round 2's key. Column 1:

$$\xrightarrow{\text{last column of round 1's key}} \begin{bmatrix}0100\\1100\end{bmatrix} \xrightarrow{\text{flip}} \begin{bmatrix}1100\\0100\end{bmatrix} \xrightarrow{\text{substitute}} \begin{bmatrix}1100\\1101\end{bmatrix}$$

$$\xrightarrow{\text{add round constant}} \begin{bmatrix}1100\\1101\end{bmatrix} \oplus \begin{bmatrix}0011\\0000\end{bmatrix} = \begin{bmatrix}1111\\1101\end{bmatrix}$$

$$\xrightarrow{\text{add 1st column of round 1's key}} \begin{array}{|c|}\hline 1111 \\ \hline 1101 \\ \hline \end{array} \oplus \begin{array}{|c|}\hline 0010 \\ \hline 0000 \\ \hline \end{array}$$

$$= \begin{array}{|c|}\hline 1101 \\ \hline 1101 \\ \hline \end{array}$$

Column 2:

$$= \begin{array}{|c|}\hline 1101 \\ \hline 1101 \\ \hline \end{array} \oplus \text{round 1's 2nd column}$$

$$= \begin{array}{|c|}\hline 1101 \\ \hline 1101 \\ \hline \end{array} \oplus \begin{array}{|c|}\hline 0100 \\ \hline 1100 \\ \hline \end{array}$$

$$= \begin{array}{|c|}\hline 1001 \\ \hline 0001 \\ \hline \end{array}$$

Therefore, the round keys are:

Round 0:
0110	0110
1000	1100

Round 1:
0010	0100
0000	1100

Round 2:
1101	1001
1101	0001

Add Round Key 0

We add round 0's key to the plain text array.

$$= \begin{array}{|c|c|}\hline 0100 & 0100 \\ \hline 1001 & 0010 \\ \hline \end{array} \oplus \begin{array}{|c|c|}\hline 0110 & 0110 \\ \hline 1000 & 1100 \\ \hline \end{array}$$

$$= \begin{array}{|c|c|}\hline 0010 & 0010 \\ \hline 0001 & 1110 \\ \hline \end{array}$$

This is our state array after round 0.

Round 1

We take the state array, substitute it (confusion), shift rows (diffusion), mix columns (a lot of diffusion), and finally add round 1's key.

$$\xrightarrow{\text{state array}} \begin{array}{|c|c|}\hline 0010 & 0010 \\ \hline 0001 & 1110 \\ \hline \end{array} \xrightarrow{\text{substitute}} \begin{array}{|c|c|}\hline 1010 & 1010 \\ \hline 0100 & 1111 \\ \hline \end{array} \xrightarrow{\text{shift rows}} \begin{array}{|c|c|}\hline 1010 & 1010 \\ \hline 1111 & 0100 \\ \hline \end{array}$$

$$\xrightarrow{\text{mix columns}} \begin{array}{|c|c|}\hline 0011 & 1001 \\ \hline 0001 & 1010 \\ \hline \end{array} \xrightarrow{\text{add round key 1}} \begin{array}{|c|c|}\hline 0011 & 1001 \\ \hline 0001 & 1010 \\ \hline \end{array} \oplus \begin{array}{|c|c|}\hline 0010 & 0100 \\ \hline 0000 & 1100 \\ \hline \end{array}$$

$$= \begin{array}{|c|c|}\hline 0001 & 1101 \\ \hline 0001 & 0110 \\ \hline \end{array}$$

This is our state array after round 1.

Round 2

In this round, the mix columns step is omitted because it does not provide additional security and slows down the process.

$$\xrightarrow{\text{state array}} \begin{array}{|c|c|}\hline 0001 & 1101 \\ \hline 0001 & 0110 \\ \hline \end{array} \xrightarrow{\text{substitute}} \begin{array}{|c|c|}\hline 0100 & 1110 \\ \hline 0100 & 1000 \\ \hline \end{array} \xrightarrow{\text{shift rows}} \begin{array}{|c|c|}\hline 0100 & 1110 \\ \hline 1000 & 0100 \\ \hline \end{array}$$

$$\xrightarrow{\text{add round key 2}} \begin{array}{|c|c|}\hline 0100 & 1110 \\ \hline 1000 & 0100 \\ \hline \end{array} \oplus \begin{array}{|c|c|}\hline 1101 & 1001 \\ \hline 1101 & 0001 \\ \hline \end{array}$$

$$= \begin{array}{|c|c|}\hline 1001 & 0111 \\ \hline 0101 & 0101 \\ \hline \end{array}$$

This is our final cipher text.

1001	0111
0101	0101

Figure 6: The final cipher text.

Decryption

We can do the process in reverse to confirm that we did it correctly. Since this is symmetric key cryptography, the decryption process uses the same round keys that we have already generated.

Remember that in $\mathbf{GF}(2^n)$, "adding" is equivalent to subtracting.

$$\xrightarrow{\text{cipher text}} \begin{array}{|c|c|}\hline 1001 & 0111 \\ \hline 0101 & 0101 \\ \hline \end{array} \xrightarrow{\text{add round key 2}} \begin{array}{|c|c|}\hline 1001 & 0111 \\ \hline 0101 & 0101 \\ \hline \end{array} \oplus \begin{array}{|c|c|}\hline 1101 & 1001 \\ \hline 1101 & 0001 \\ \hline \end{array} = \begin{array}{|c|c|}\hline 0100 & 1110 \\ \hline 1000 & 0100 \\ \hline \end{array}$$

$$\xrightarrow{\text{reverse shift rows}} \begin{array}{|c|c|}\hline 0100 & 1110 \\ \hline 0100 & 1000 \\ \hline \end{array} \xrightarrow{\text{reverse substitution}} \begin{array}{|c|c|}\hline 0001 & 1101 \\ \hline 0001 & 0110 \\ \hline \end{array}$$

This reversed round 2, and the result is equal to our state array after round 1 encryption.

$$\xrightarrow{\text{state array}} \begin{array}{|c|c|}\hline 0001 & 1101 \\ \hline 0001 & 0110 \\ \hline \end{array} \xrightarrow{\text{add round key 1}} \begin{array}{|c|c|}\hline 0001 & 1101 \\ \hline 0001 & 0110 \\ \hline \end{array} \oplus \begin{array}{|c|c|}\hline 0010 & 0100 \\ \hline 0000 & 1100 \\ \hline \end{array} = \begin{array}{|c|c|}\hline 0011 & 1001 \\ \hline 0001 & 1010 \\ \hline \end{array}$$

Then, we need to perform the inverse of mix columns, which is to multiply each column by the matrix $\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix} = \begin{bmatrix} \{1001_2\} & \{0010_2\} \\ \{0010_2\} & \{1001_2\} \end{bmatrix} = \begin{bmatrix} x^3+1 & x \\ x & x^3+1 \end{bmatrix}$, which was found through matrix row manipulations. This inverts the mix columns function from encryption.

$$\begin{bmatrix} N_0' \\ N_1' \end{bmatrix} = \begin{bmatrix} x^3+1 & x \\ x & x^3+1 \end{bmatrix} \begin{bmatrix} N_0 \\ N_1 \end{bmatrix} \tag{7}$$

The column $\begin{array}{|c|}\hline 0011 \\ \hline 0001 \\ \hline \end{array}$ is transformed by

$$\begin{aligned}
\begin{bmatrix} N_0' \\ N_1' \end{bmatrix} &= \begin{bmatrix} x^3+1 & x \\ x & x^3+1 \end{bmatrix} \begin{bmatrix} x+1 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} (x^3+1)(x+1)+(x)(1) \\ (x)(x+1)+(x^3+1)(1) \end{bmatrix} \\
&= \begin{bmatrix} x^4+x^3+x+1+x \\ x^2+x+x^3+1 \end{bmatrix} \\
&= \begin{bmatrix} x^4+x^3+2x+1 \\ x^3+x^2+x+1 \end{bmatrix} \\
&= \begin{bmatrix} x^4+x^3+1 \pmod 2 \\ x^3+x^2+x+1 \pmod 2 \end{bmatrix} \\
&= \begin{bmatrix} x^3+x \pmod{x^4+x+1} \\ x^3+x^2+x+1 \pmod{x^4+x+1} \end{bmatrix}
\end{aligned}$$

Which equals $\begin{array}{|c|}\hline 1010 \\ \hline 1111 \\ \hline \end{array}$.

$$\xrightarrow{\text{state array}} \begin{array}{|c|c|}\hline 0011 & 1001 \\ \hline 0001 & 1010 \\ \hline \end{array} \xrightarrow{\text{inverse mix columns}} \begin{array}{|c|c|}\hline 1010 & 1010 \\ \hline 1111 & 0100 \\ \hline \end{array} \xrightarrow{\text{reverse shift rows}} \begin{array}{|c|c|}\hline 1010 & 1010 \\ \hline 0100 & 1111 \\ \hline \end{array}$$

$$\xrightarrow{\text{reverse substitution}} \begin{array}{|c|c|}\hline 0010 & 0010 \\ \hline 0001 & 1110 \\ \hline \end{array}$$

Finally, we add round 0's key, our initial private key.

$$= \begin{array}{|c|c|}\hline 0010 & 0010 \\ \hline 0001 & 1110 \\ \hline \end{array} \oplus \begin{array}{|c|c|}\hline 0110 & 0110 \\ \hline 1000 & 1100 \\ \hline \end{array}$$

$$= \begin{array}{|c|c|}\hline 0100 & 0100 \\ \hline 1001 & 0010 \\ \hline \end{array}$$

This is equivalent to our plain text, so the decryption was successful.

Conclusion

This exploration demonstrated the mathematics behind Simplified AES (Holden et al., 2010). Although S-AES is not used in real-world situations, its theories closely mirror those of AES (Daemen & Rijmen, 2001), which *is* used in real life. AES differs only by longer pieces of information and more rounds.

I note again that the raw encryption I have shown in this exploration is not how real computers do it; the real-life process combines the AES process with other cryptographic components to create a truly secure cipher.

A further exploration could explore more *how* each step of this process increases security, and perhaps compare and contrast it to public key cryptography ciphers.

Thank you for reading through all of this. I hope it was as interesting to read as it was to write it, and that the 18 pages didn't seem quite as long.

Appendix: Swift Program

I wrote a S-AES program in the Swift programming language from scratch, to help me understand the concepts better (Chan, 2015). It is available on `https://github.com/NathanJang/swift-s-aes`

References

Bellare, M., Desai, A., Jokipii, E., & Rogaway, P. (2000). A concrete security treatment of symmetric encryption. *Proceedings of the 38th Symposium on Foundations of Computer Science*. Retrieved from `http://web.cs.ucdavis.edu/~rogaway/papers/sym-enc.pdf`

Benvenuto, C. J. (2012). *Galois field in cryptography.* University of Washington. Retrieved from `https://www.math.washington.edu/~morrow/336_12/papers/juan.pdf`

Cerf, V. (1969). *ASCII format for network interchange (RFC 20).* Retrieved from `http://www.rfc-editor.org/rfc/rfc20.txt`

Chan, J. (2015). *`swift-s-aes`* [Computer software]. Retrieved from `https://github.com/NathanJang/swift-s-aes`

Daemen, J., & Rijmen, V. (2001). Specification for the Advanced Encryption Standard (AES). *Federal Information Processing Standards Publication 197*. Retrieved from `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`

Hankerson, D., Menezes, A., & Vanstone, S. (2004). *Guide to elliptic curve cryptography.* Springer. Retrieved from `http://cs.ucsb.edu/~koc/ecc/docx/GuideEllipticCurveCryptography.pdf`

Holden, J., Holden, L., Musa, M., Schaefer, E., & Wedig, S. (2010). *A simplified AES algorithm.* Rose-Hulman Institute of Technology. Retrieved from `https://www.rose-hulman.edu/~holden/Preprints/s-aes.pdf`

Ikenaga, B. (2012). *Commutative rings and fields.* Millersville University of Pennsylvania. Retrieved from `http://sites.millersville.edu/bikenaga/linear-algebra/rings/rings.pdf`

Nechvatal, J. (1991). Public-key cryptography. *NIST Special Publication 800-2*. Retrieved from `http://www.dtic.mil/dtic/tr/fulltext/u2/a408338.pdf`

Shannon, C. E. (1949). Communication theory of secrecy systems. *Bell System Technical Journal*, *28*(4). Retrieved from `http://netlab.cs.ucla.edu/wiki/files/shannon1949.pdf`