The Advanced Encryption Standard (AES)

Lecture 13

Josh Brake
Harvey Mudd College

Outline

- Introduction to Finite (Galois) Fields
- AES Overview

Learning Objectives

By the end of this lecture you will…

- Have an operational understanding of the fundamental mathematics used in AES
- Understand the basic process of AES

Finite Fields

A finite field is a set F with two operators (+ and ·) satisfying the following:

- Closure: if $a, b \in F$, then $a + b \in F$ and $a \cdot b \in F$.
- Identity: 0 and 1 elements exist such that $a + 0 = a$ and $a \cdot 1 = a$ for any $a \in F$.
- Inverse: For any a, there exists x and y such that $a + x = 0$ and $a \cdot y = 1$. Except there is no multiplicative inverse of 0.
- **Associative**, **commutative**, and **distributive** properties hold.
- The two most common fields are $GF(p)$ and $GF(2^n)$.

History

Galois Fields are named after French mathematician Évariste Galois. He died in a duel at age 20 after being expelled from college for political protest after the French revolution. He spent six months in prison developing his mathematical ideas.

$\mathrm{GF}(\mathrm{p})$: Arithmetic modulo p

Example: $\mathrm{GF}(13)$: contains 0– 12, math is all done mod 13

- 0 and 1 are the identity elements
- Additive inverse of 4 is 9 because 4 + 9 = 0.
- Multiplicative inverse of 4 is 10 because $4 \cdot 10 \mod 13 = 1$.
- Used in public key encryption
    - Typically 256-1024 bit numbers mod a number of this length
    - Public and private keys a and b have special property that: $\mathrm{x}^{(\mathrm{a}\cdot\mathrm{b})} = \mathrm{x} \mod \mathrm{p} \forall \mathrm{x}$.

$GF(2^n)$

- Elements are polynomials in some dummy variable x with coefficients of 0 or 1.
- Hence, each coefficient operates mod 2.
- Addition, subtraction, and exclusive or are all the same.
- This makes the hardware very easy because addition simplifies to bitwise XOR.
- Operations are done modulo some characteristic irreducible (prime) polynomial. For AES, this polynomial is $m(x) = x^8 + x^4 + x^3 + x + 1$.

Advantages of Galois Arithmetic for Digital Hardware

Example: $\mathrm{GF}(2^8)$ with characteristic polynomial $\mathrm{m(x)} = \mathrm{x}^8 + \mathrm{x}^4 + \mathrm{x}^3 + \mathrm{x} + 1\,.$

- Each element is an 8-bit number (easy hardware)
- Addition is 8-bit XOR
- Multiplication is shifting

$GF(2^8)$ Hexadecimal Representation

- Express $a = x^5 + x^3 + x^2 + 1$ in hexadecimal.

$a =$ `0b00101101` $=$ `0x2D`

- Express $b = x^7 + x^6 + x^2 + x + 1$ in hexadecimal

$b =$ `0b11000111` $=$ `0xC7`

GF(2^8) Addition

Compute $c = a + b$ where $a = x^5 + x^3 + x^2 + 1$ and $b = x^7 + x^6 + x^2 + x + 1$.

a = `0b00101101` = `0x2D`

b = `0b11000111` = `0xC7`

$$\begin{array}{r} 0b00101101 \\ \oplus 0b11000111 \\ \hline 0b11101010 \end{array}$$

$\mathrm{GF}(2^8)$ Addition

Given $a = x^5 + x^3 + x^2 + 1$ and $b = x^7 + x^6 + x^2 + x + 1$.

What is the additive identity of a (i.e., what is the zero element)?

$a = $ `0b00101101` $=$ `0x2D`

0

What is the additive inverse of a (i.e., $-a$)?

$a = $ `0b00101101` $=$ `0x2D`

a

$\mathrm{GF}(2^8)$ Multiplication

What is $a(x) \cdot x$ where $a = x^5 + x^3 + x^2 + 1$?

$$
\begin{aligned}
a = x^5 + x^3 + x^2 + 1 &= 0b00101101 \\
a \cdot x = x^6 + x^4 + x^3 + x &\rightarrow 0b01011010
\end{aligned}
$$

$\mathrm{GF}(2^8)$ Multiplication

What is $b(x) \cdot x$ where $b = x^7 + x^6 + x^2 + x + 1$?

$$b = x^7 + x^6 + x^2 + x + 1 = 0b11000111$$
$$b \cdot x = x^8 + x^7 + x^3 + x^2 + x \rightarrow 0b110001110$$

$GF(2^8)$ Multiplication

$a = x^5 + x^3 + x^2 + 1$

What is the multiplicative **identity** of $a(x)$ (i.e., our 1 element?

$$a \cdot x = a \to x = 1$$

What is the multiplicative **inverse** of $a(x)$?

Tricky to find!

Galois Field Math Summary

- Addition is XOR.
- Multiplication by x is left shift.
  - If the coefficient of x^8 is 0, then you are done.
  - Otherwise, you need to reduce modulo an irreducible polynomial. For AES, this polynomial is $m(x) = x^8 + x^4 + x^3 + x + 1$.

Galois Fields Worksheet

The Advanced Encryption Standard

AES Overview

From the spec:

The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information. Encryption converts data to an unintelligible form called ciphertext; decrypting the ciphertext converts the data back into its original form, called plaintext.

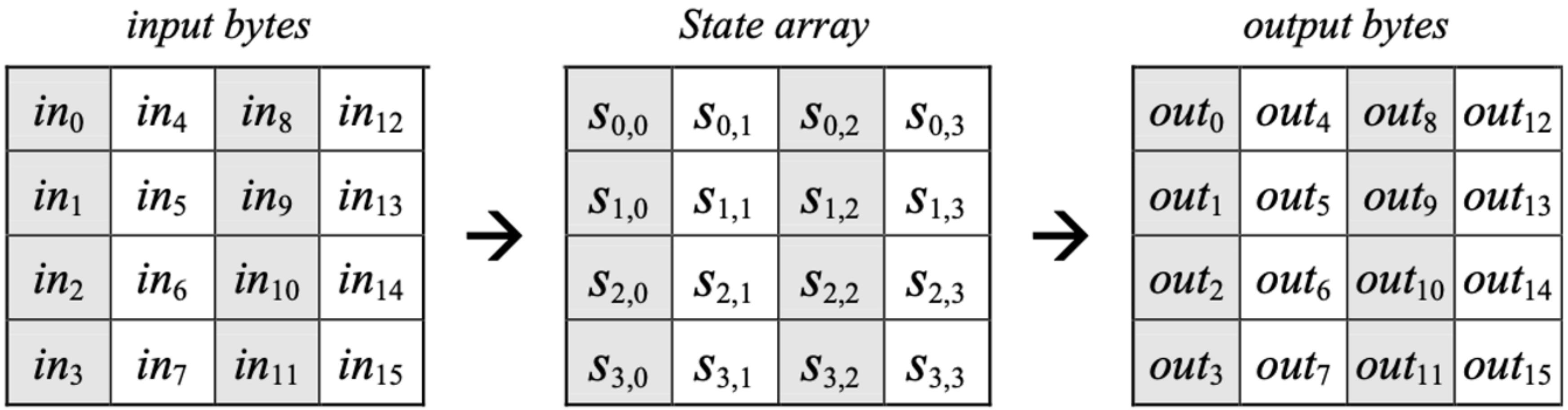
The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits.

128-bit AES

- The cipher works by taking a plain text message and scrambling it into a cyphertext message in a way that is hard to reverse.
- Scrambling is done 10 times to make it hard. The key changes from round to round.

AES Data Organization

128-bit message is organized as a matrix of 16 bytes (4x4).



input bytes

in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

→

State array

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

→

output bytes

out_0	out_4	out_8	out_{12}
out_1	out_5	out_9	out_{13}
out_2	out_6	out_{10}	out_{14}
out_3	out_7	out_{11}	out_{15}

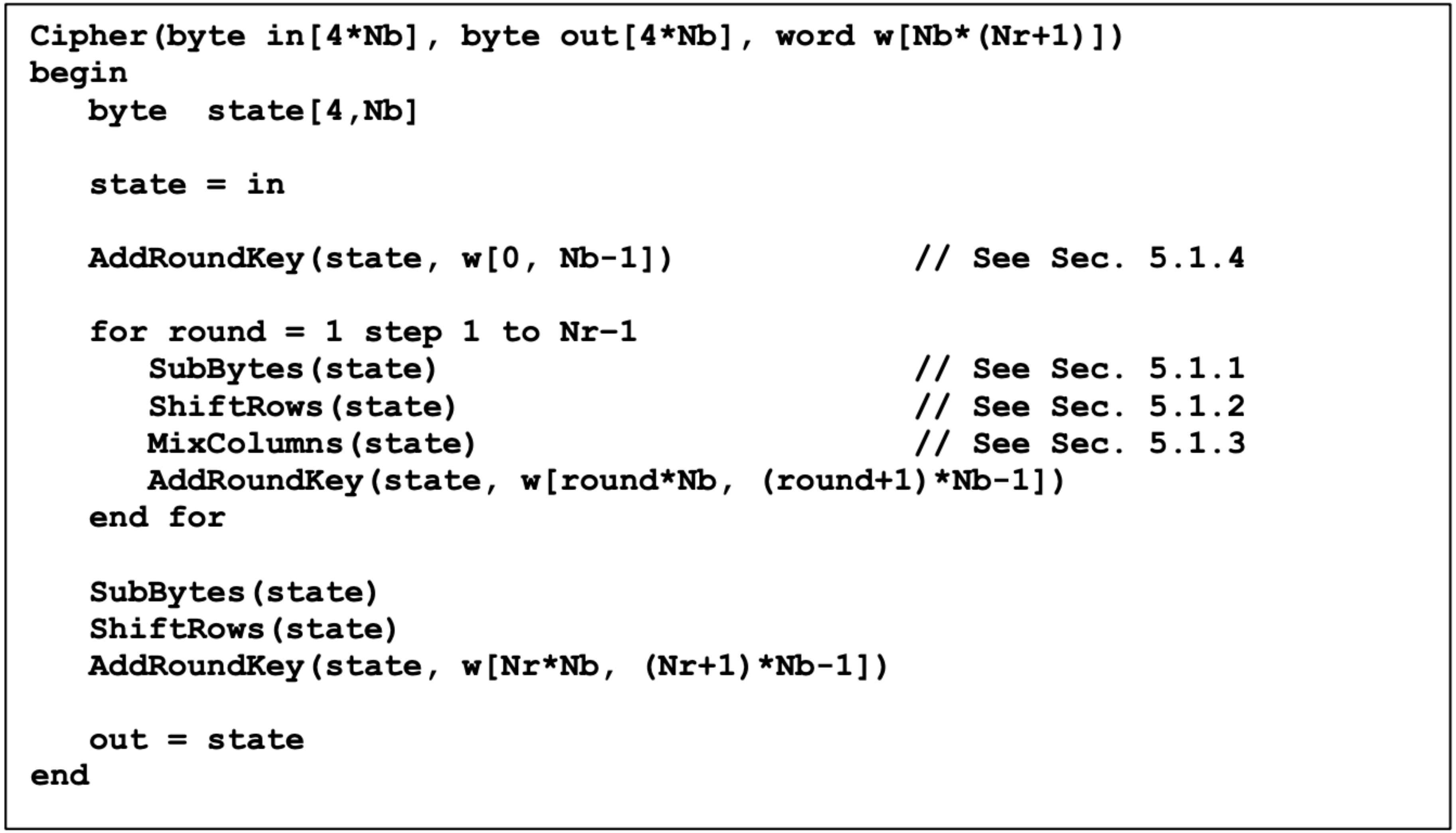
Figure 3. State array input and output.

AES Cipher Process

Each step of the cipher involves the following steps

1. `SubBytes`: take each byte and replace it with a different byte using a randomish lookup table.
2. `ShiftRows`: move bytes around in the rows
3. `MixColumns`: funky Galois multiplication on elements of the columns
4. `AddRoundKey`: XOR with the key for the current round.

AES Cipher Pseudocode



```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])            // See Sec. 5.1.4

   for round = 1 step 1 to Nr–1
      SubBytes(state)                        // See Sec. 5.1.1
      ShiftRows(state)                       // See Sec. 5.1.2
      MixColumns(state)                      // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Figure 5. Pseudo Code for the Cipher.[1]