Irving Reyes Bravo

04/14/2025

CS 4920 Spring 2025

HW 4

Problem 1.

Investigate Simplified AES (S-AES) based on GF(2^4).

a. Cite the source(s) for your S-AES investigation and learning.
Simplified AES (S-AES) is a scaled-down version of the Advanced Encryption Standard designed for "educational purpose"s. S-AES typically uses 2-3 rounds instead of AES's 10-14 rounds, making the calculations manageable for classroom demonstrations while preserving the fundamental cryptographic principles; thus, it is easier to trace by hand and therefore understand. The algorithm follows the same substitution-permutation network structure as AES, with rounds consisting of SubNibbles (analogous to SubBytes), ShiftRows, MixColumns, and AddRoundKey operations:
   1. Key Expansion: Generate round keys from the original key.
   2. Initial AddRoundKey: XOR the plaintext with the initial round key.
   3. Round 1:
      - SubNibbles: Substitute each nibble using the S-box.
      - ShiftRows: Shift the second row.
      - MixColumns: Apply matrix multiplication in GF(2^4).
      - AddRoundKey: XOR with round key 1.
   4. Round 2:
      - SubNibbles: Substitute each nibble using the S-box.
      - ShiftRows: Shift the second row.
      - AddRoundKey: XOR with round key 2. (no MixColumns in final round)
   5. Output the ciphertext

   Stallings, W. (2019). *Cryptography and Network Security* (8th ed.). Pearson Education (US). https://uccs.vitalsource.com/books/9780135764268
   Musa, M. & Schaefer, E. & Wedig, S. (2003). *A Simplified AES Algorithm and its Linear and Differential Cryptanalyses.* CRYPTOLOGIA. 27. 148-177. 10.1080/0161-110391891838.

b. What is the block length and the structure of the state array? Express them in bits.
Block Length: 16 bits
State Array Structure: A (2*2) matrix of "nibbles", for a total of 16 bits.
$$\begin{bmatrix} n_{0,0} & n_{0,1} \\ n_{1,0} & n_{1,1} \end{bmatrix}$$
Each nibble represents 4 bits, and the entire state array represents the 16-bit block being processed. This is a scaled down version of AES's (4*4) byte matrix.

c. What is the modulus/prime polynomial used for the GF(2^4) operations in S-AES?
The irreducible polynomial used for GF(2^4) operations in S-AES is: $x^4 + x + 1$
This polynomial is represented as 10011 in binary and 0x13 in hexadecimal. It serves as the modulus for all field operations in S-AES, ensuring that results of multiplication remain within the field.

d. Develop multiplication table similar to Table 5.2 for the GF(2^4) operations in S-AES.
Multiplication table for GF(2^4) using the irreducible polynomial $x^4 + x + 1$:

*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	3	1	7	5	B	9	F	D
3	0	3	6	5	C	F	A	9	B	8	D	E	7	4	1	2
4	0	4	8	C	3	7	B	F	6	2	E	A	5	1	D	9
5	0	5	A	F	7	2	D	8	E	B	4	1	9	C	3	6
6	0	6	C	A	B	D	7	1	5	3	9	F	E	8	2	4
7	0	7	E	9	F	8	1	6	D	A	3	4	2	5	C	B
8	0	8	3	B	6	E	5	D	C	4	F	7	A	2	9	1
9	0	9	1	8	2	B	3	A	4	D	5	C	6	F	7	E
A	0	A	7	D	E	4	9	3	F	5	8	2	1	B	6	C
B	0	B	5	E	A	1	F	4	7	C	2	9	D	6	8	3
C	0	C	B	7	5	9	E	2	A	6	1	D	F	3	4	8
D	0	D	9	4	1	C	8	5	2	F	B	6	3	E	A	7
E	0	E	F	1	D	3	2	C	9	7	6	8	4	A	B	5
F	0	F	D	2	9	6	4	B	1	E	C	3	8	7	5	A

e. Show that the matrix given in the following, with entries in GF(2^4), is the inverse of the matrix used in the Mix Columns step of S-AES.

$$\begin{bmatrix} x^3+1 & x \\ x & x^3+1 \end{bmatrix}$$

I need to multiply it with the original MC matrix and verify that the result is the Identity Matrix:
The MC matrix in S-AES is:

$$\begin{bmatrix} 1 & x^2 \\ x^2 & 1 \end{bmatrix}$$

I now multiply these two matrices and find the product:

1. Top-Left:
   $((x^3 + 1) * 1) + (x * x^2)$
   $= (x^3 + 1) + x^3$
   $= 1$ $\quad x^3 + x^3 = 0$ in GF(2^4)
2. Top-Right:
   $((x^3 + 1) * x^2) + (x * 1)$
   $= (x^5 + x^2) + x$ $\quad x^5 \bmod (x^4 + x + 1) = x^2$
   $= x^2 + x^2 + x$ $\quad x^2 + x^2 = 0$ in GF(2^4)
   $= x \quad = 0$
3. Bottom-Left:
   $(x * 1) + ((x^3 + 1) * x^2)$
   $= x + (x^5 + x^2)$ $\quad x^5 \bmod (x^4 + x + 1)$
   $= x + x^2 + x^2$ $\quad x^2 + x^2 = 0$ in GF(2^4)
   $= x \quad = 0$
4. Bottom-Right:
   $(x * x^2) + ((x^3 + 1) * 1)$
   $= x^3 + (x^3 + 1)$ $\quad x^3 + x^3 = 0$ in GF(2^4)
   $= 1$

The result is the Identity Matrix, confirming that the given matrix is indeed the inverse:

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

Problem 2.

Let's implement AES Mix Columns using GF(2^8) operations. Apply Mix Columns to the input to find the output. Here is the approach I used:

- **Representation:**

  Each element in GF(2^8) is represented as an integer between 0 and 255, corresponding to the binary coefficients of a polynomial of degree 7 at most.

- **Operations:**
  - Addition/Subtraction:

    In GF(2^8), addition and subtraction are the same operation – the bitwise XOR (^).
  - Multiplication:

    Implemented using a function that performs multiplication in GF(2^8) with modular reduction via the AES polynomial (0x11b).
  - Mix Columns Transformation:

    The Mix Columns operation is a matrix multiplication of the state with a fixed matrix in GF(2^8). This involves:
    - mix_single_col(column):

      Applies the Mix Columns transformation to a single column of the state.
    - mix_columns(state):

      Applies the Mix Columns transformation to the entire state matrix.

- **Input/Output Handling:**

  My program reads in 128-bit hexadecimal strings, converts them to a 4x4 matrix of bytes, applies the Mix Columns transformation, and writes the resulting hexadecimal strings to output files.

My program processes both the provided class-common inputs and my own calculations file. Below is a description of the main functions and variables within the code:

- **galois_multiply(a, b):**
  Performs multiplication in GF(2^8) using the AES polynomial for reduction.
- **mix_single_column(column):**
  Transforms a single column using the Mix Columns matrix.
- **mix_columns(state):**
  Transforms the entire state matrix.
- **hex_to_matrix(hex_string):**
  Converts a 128-bit hex string to a 4x4 matrix of bytes.
- **matrix_to_hex(matrix):**
  Converts a 4x4 matrix of bytes back to a 128-bit hex string.
- **process_file(input_filename, output_filename):**
  Processes an input file, applies Mix Columns to each line, and writes results to an output file.
- **AES_POLYNOMIAL:**
  Value "0x11b" representing ($x^8 + x^4 + x^3 + x + 1$), used for modular reduction in multiplication.

Problem 3.

The following questions are about the CBC mode.

a. Is it possible to perform encryption operations in parallel on multiple blocks of plaintext in CBC mode? How about decryption?

The formula for CBC encryption is:

$$C_i = E_K\left(P_i \bigoplus C_{i-1}\right)$$

Where C_i is the current ciphertext block, P_i is the current plaintext block, and C_{i-1} is the previous cyphertext block. Since each block needs the result of the previous block's encryption, the process must be sequential – aka encryption operations in CBC mode cannot be performed in parallel.
The formula for CBC decryption is:

$$P_i = D_K\left(C_i \bigoplus C_{i-1}\right)$$

Since all ciphertext blocks (C_i and C_{i-1}) are available before decryption begins, the decryption of each block can proceed independently and simultaneously. Thus, decryption can be performed in parallel.

b. In the CBC mode, an error in a block of transmitted ciphertext propagates. For example, an error in the transmitted C_1 (Figure 7.4) corrupts P_1 and P_2. Are any blocks beyond P_2 affected?
When an error occurs in ciphertext block C_1, P_3 and beyond will not be affected because they depend on C_2, C_3, etc., which are assumed to be error-free. Therefore, the error propagation in C_1 is limited to 2 blocks: the corresponding block (P_1) and the next block (P_2).

c. Is it possible to perform encryption operations in parallel on multiple blocks of plaintext in *s*-bit CFB mode? How about decryption?
In s-bit CFB, the encryption formula is:

$$C_i = P_i \bigoplus [\text{first } s \text{ bits of } E_K(\text{shift register})]$$

Where the shift register is updated with ach operation by shifting in the previous ciphertext bits. This sequential dependency prevents parallelization, like CBC.
The formula for decryption is:

$$P_i = C_i \bigoplus [\text{first s bits of } E_K(\text{shift register})]$$

Since the shift register contents for each block can be pre-computed from known ciphertext values, the decryption operations can be performed simultaneously.

d. In the *s*-bit CFB mode, an error in a block of transmitted ciphertext occurs. The decryption of the block therefore provides an error. Are any blocks beyond that block affected?
In s-bit CFB mode, an error is a transmitted ciphertext block affects both the decryption of the current block (containing the error) and the next b/s blocks (where b is the block size in bits). This is due to the erroneous ciphertext bits entering the shift register and remaining there for b/s iterations before being shifted out completely. Once the erroneous bits are shifted out of the register, subsequent decryptions are no longer affected.
For example, if I have 8-bit CFB mode with a 64-bit block size, an error will affect the current block plus the next blocks (64 / 8 = 8). So, the error propagation is limited and clears after b/s blocks.

Now let's solve the questions for the OFB mode.

e. Is it possible to perform encryption operations in parallel on multiple blocks of plaintext in OFB mode? How about decryption?
The encryption process in OFB mode follows this pattern:

$$O_i = E_K(O_{i-1})$$
$$C_i = P_i \bigoplus O_i$$

Where each output block depends on the previous one. Since the keystream generation is sequential, encryption operations in OFB mode cannot be performed in parallel on multiple blocks of plaintext.

f. In the OFB mode, an error in a block of transmitted ciphertext occurs. The decryption of the block therefore provides an error. Are any blocks beyond that block affected?
In OFB mode, an error in a transmitted ciphertext block only affects the decryption of that block and does not propagate to subsequent blocks. This is because the key stream generation (O_i values) is

independent of the ciphertext so each plaintext block is recovered by XORing the corresponding ciphertext block with the keystream block.
If C_j contains an error, P_j will be incorrect because $P_j = C_j \oplus O_j$, and C_j contains an error. But P_{j+1} and all subsequent blocks will be unaffected because their decryption depends only on their respective ciphertext blocks and the keystream, which is not influenced by previous ciphertext blocks.

Problem 4.

Suppose you have a true random bit generator where each bit in the generated stream has the same probability of being a 0 or 1 as any other bit in the stream and that the bits are independent, i.e., the bits are generated from identical independent distribution (IID). However, the bit stream is biased. The probability of a 1 is $0.5 + \P$ and the probability of a 0 is $0.5 - \P$, where $0 < \P < 0.5$. A simple conditioning algorithm is as follows: examine the bit stream as a sequence of nonoverlapping pairs. Discard all 00 and 11 pairs. Replace each 01 pair with 0 and each 10 pair with 1.

a. What is the probability of occurrence of each pair in the original sequence?
For non-overlapping pairs, the probability of occurrence is:

•	P(00)	= P(0) * P(0)	$= (0.5 - \P)(0.5 - \P)$	$= (0.5 - \P)^2$	$= 0.25 - \P + \P^2$
•	P(01)	= P(0) * P(1)	$= (0.5 - \P)(0.5 + \P)$	$= 0.25 - \P^2$	
•	P(10)	= P(1) * P(0)	$= (0.5 + \P)(0.5 - \P)$	$= 0.25 - \P2$	
•	P(11)	= P(1) * P(1)	$= (0.5 + \P)(0.5 + \P)$	$= (0.5 + \P)^2$	$= 0.25 + \P + \P^2$

b. What is the probability of occurrence of 0 and 1 in the modified sequence after the conditioning?
In this modified sequence: 00 and 11 pairs are discarded, 01 becomes 0, and 10 becomes 1.
The probability of getting a 0: P(output = 0)
$= P(01)/[P(01) + P(10)]$
$= (0.25 - \P^2)/(0.5 - 2\P^2)$
$= 0.5$
The probability of getting a 1: P(output = 1)
$= P(10)/[P(01) + P(10)]$
$= (0.25 - \P^2)/(0.5 - 2\P^2)$
$= 0.5$
So, the mod. seq. has an unbiased probability of 0.5 for both 0 and 1, regardless of the original bias ¶.

c. What is the expected number of input bits to produce *x* output bits?
The probability that a pair of input bits produces an output bit: P(pair produces output)
$= P(01) + P(10)$
$= (0.25 - \P^2) + (0.25 - \P^2)$
$= 0.5 – 2\P^2$
So, the expected number of pairs needed to produce one output bit is: E(pairs for one output bit)
$= 1/(0.5 - 2\P^2)$
$= 2/(1 - 4\P^2)$
Since each pair consists of 2 input bits, the expected number of input bits to produce one output bit is: E(input bits for one output bit)
$= 2 * 2/(1 - 4\P^2)$
$= 4/(1 - 4\P^2)$
To produce x output bits, the expected number of input bits is: E(input bits for x output bits)
$= x * 4/(1 - 4\P^2)$

d. Suppose that the algorithm uses overlapping successive bit pairs instead of non-overlapping successive bit pairs. That is, the first output bit is based on input bits 1 and 2, the second output bit is based on input bits 2 and 3, and so on. What is the output bit stream?

If the algorithm uses overlapping successive bit pairs instead of non-overlapping pairs, the output bit stream will have dependencies between consecutive bits, making it no longer IID. I will use the input stream [a, b, c, d, e, …]:

- First output: Based on (a.b)
- Second output: Based on (b, c)
- Third output: Based on (c, d)

This creates dependencies because consecutive output bits share an input bit. For instance, if the first output is derived from an (a,b) = (0,1) pair (producing a 0), then the second output will be derived from a (b,c) pair where b = 1. This means the second pair can only be (1,0) or (1,1), limiting the possible values for the second output bit. This dependency means the output stream would not be truly random even if the input bias is corrected.

Another approach to conditioning is to consider the bit stream as a sequence of non-overlapping groups of *n* bits each and output the parity of each group. That is, if a group contains an odd number of ones, the output is 1; otherwise the output is 0.

e. Express this operation in terms of a basic Boolean function.
For a group of n bits (b_1, b_2, …, b_n), the parity can be expressed as:

$$Parity(b_1, b_2, \ldots, b_n) = b_1 \bigoplus b_2 \bigoplus \ldots \bigoplus b_n$$

Where ⊕ represents the XOR operation. This basic Boolean function outputs 1 if there is an odd number of 1s in the input bits, and 0 if there is an even number of 1s.

f. Assume, as in before, that the probability of a 1 is 0.5 + ¶. If each group consists of 2 bits, what is the probability of an output of 1?
For a 2-bit group, the output is 1 when exactly one bit is 1; so, when the group is 01 or 10.
P(output = 1)
= P(01) + P(10) P(output = 1)
= P(0) * P(1) + P(1) * P(0) P(output = 1)
= (0.5 - ¶)(0.5 + ¶) + (0.5 + ¶)(0.5 - ¶)
P(output = 1)
= 2(0.5 - ¶) * (0.5 + ¶) P(output = 1)
= 2(0.25 - $\P^2$) P(output = 1)
= 0.5 - $2\P^2$

g. If each group consists of 4 bits, what is the probability of an output of 1?
I need to calculate the probability of having exactly k ones in a 4-bit group:
P(k ones)
= C(4,k) * $(0.5 + \P)^k$ * $(0.5 - \P)^{4-k}$
Where C(4,k) is the binomial coefficient "4 choose k".
P(output = 1)
= P(1 one) + P(3 ones) P(output = 1)
= C(4,1) * $(0.5 + \P)^1$ * $(0.5 - \P)^3$ + C(4,3) * $(0.5 + \P)^3$ * $(0.5 - \P)^1$
Simplifying:
= 4(0.5 + ¶) * $(0.5 - \P)^3$ + 4 * $(0.5 + \P)^3$ * (0.5 - ¶)
Expanding and collecting terms:
= 0.5 - $4\P^2 + 4\P^4$
For a 4-bit group, the output is 1 when there is an odd number of 1s; so 1, 3, or all 4 bits are 1.

h. Generalize the result to find the probability of an output of 1 for input groups of *n* bits.
Using the binomial theorem and properties of parity, I can derive:
P(output = 1)
= $(1 - (1 - 2\P)^n)/2$

This can be expanded as:

$$= 0.5 - C(n,2)\,¶^2 + C(n,4)\,¶^4 - C(n,6)\,¶^6 + ... + (-1)^{n-1}C(n,n)\,¶^n$$

For small values of ¶, the dominant term is $0.5 - C(n,2)\,¶^2$, which means:

- As n increases, the bias in the output decreases more rapidly
- For large n, the output approaches a probability of 0.5 (unbiased)

This shows that using parity bits from larger groups is more effective at reducing bias in the original bit stream, with the output becoming closer to a true random sequence as n increases. Thus, for an n-bit group, the output is 1 when there are an odd number of 1s.

Problem 5.

a. State and prove Fermat's Theorem.
If p is a prime number and a is an integer not divisible by p, then:

$$a^{p-1} \equiv 1\ (mod\ p)$$

Alternatively, for any integer a:

$$a^{p} \equiv a\ (mod\ p)$$

I use the set:
S = {1, 2, 3, ..., p-1}.
When I multiply each element of S by a (mod p), I get:
T = {a mod p, 2a mod p, 3a mod p, ..., (p-1)a mod p}.
Since a is not divisible by p, all elements in T are non-zero modulo p. Also, no two elements in T can be congruent modulo p. If $ja \equiv ka\ (mod\ p)$ for some j ≠ k, then $(j-k)a \equiv 0\ (mod\ p)$, which would mean p divides (j-k)a. Since p doesn't divide a, it must divide (j-k), which is impossible since both j and k are between 1 and p-1.
Therefore, T must be a permutation of S. This means:
$(1 \times 2 \times ... \times (p-1)) \times a^{(p-1)} \equiv 1 \times 2 \times ... \times (p-1)$ (mod p)
Dividing both sides by (1 × 2 × ... × (p-1)) since none of these numbers is divisible by p:
$a^{(p-1)} \equiv 1$ (mod p)

b. State and prove Euler's Theorem.

c. Prove that $f(p^i) = p^i - p^{i-1}$ if p is prime, where f(.) is the Euler's Totient Function.
Euler's totient function φ(n) counts the number of integers from 1 to n that are coprime to n. For p^i, I need to count the integers from 1 to p^i that are not divisible by p.
The numbers that are not coprime to p^i are exactly those that are divisible by p, which are:
$p, 2p, 3p, ..., p^{(i-1)} * p$
There are $p^{(i-1)}$ such numbers. Therefore: $\varphi(p^i) = p^i - p^{(i-1)}$
This can also be written as: $\varphi(p^i) = p^i(1 - \frac{1}{p})$
This formula shows that for a prime power p^i, the totient function equals the total number of integers (p^i) minus the count of integers that share a factor with p^i (which is p^{i-1}).