Universidad Nacional Autónoma de México

Facultad de Ingeniería

Ingeniería en Computación

Proyecto Final

Cifrado Simétrico AES

Profesor: Dr. Alfonso Francisco de Abiega L Eglisse

Integrantes: Cruz Calderón Jorge Luis
Guzmán Pérez Karla Isela
Montelongo Alquicira Marco Antonio
Ocaña Casillas Jonathan Leonardo
Sánchez Estrada Angel Isaac
Sánchez Zamora Jesús
Velasco Avila Cristopher

Fecha de entrega: 18 de mayo de 2025

Contenido

Constantes Globales

Nk – *#define Nk 4*

Constante entera preprocesador. Número de palabras de la clave (4 palabras de 32 bits = 128 bits).

Nb – *#define Nb 4*

Constante entera preprocesador. Número de columnas en el estado (bloque de 128 bits dividido en 4 columnas de 32 bits).

Nr – *#define Nr 10*

Constante entera preprocesador. Número de rondas de cifrado/desifrado para AES-128.

s_box – *static const uint8_t s_box[256]*

Tabla S-box (sustitución directa) utilizada en la transformación SubBytes del algoritmo AES. Cada byte de entrada se reemplaza por el valor correspondiente en esta tabla. Es una tabla constante de 256 elementos de 8 bits.

inv_s_box – *static const uint8_t inv_s_box[256]*

Tabla inversa de S-box usada en la operación InvSubBytes durante el descifrado AES. Permite revertir la sustitución de bytes hecha en el cifrado. También contiene 256 elementos constantes de 8 bits.

Rcon – *static const uint8_t Rcon[11]*

Tabla de constantes de ronda utilizadas en la expansión de la clave (Key Expansion). Contiene 11 valores (uno por cada ronda, incluyendo la inicial). Se utiliza para generar claves de ronda únicas a partir de la clave principal.

Operaciones en GF(2^8)

uint8_t xtime(uint8_t x)

Realiza la multiplicación de un byte x por x (es decir, por 2) en el campo finito GF(2^8), aplicando reducción modular con el polinomio irreducible de AES:

$$m(x) = x^8 + x^4 + x^3 + x + 1 = 0x11b,$$

lo que equivale a una operación mod 0x11b, pero optimizada usando 0x1b tras un desplazamiento.

Parámetros

- **uint8_t x:** Byte a multiplicar por 2 en GF(2^8).

Valor de retorno

- **uint8_t:** Resultado de x * 2 mod 0x11b.

Funcionamiento

- Se desplaza x un bit a la izquierda: x << 1 (multiplicación por 2).
- Si el bit más significativo de x (bit 7) era 1, se hace una reducción modular con 0x1b: *(x << 1) ^ 0x1b*.

uint8_t mul(uint8_t x, uint8_t y)

Realiza la multiplicación de dos bytes x y y en GF(2^8), utilizando el algoritmo conocido como "multiplicación rusa" (Russian Peasant Multiplication), que es eficiente y evita el uso de tablas.

Parámetros

- **uint8_t x:** Primer operando de la multiplicación (multiplicando).
- **uint8_t y:** Segundo operando (multiplicador).

Valor de retorno

- **uint8_t:** Resultado de x * y mod 0x11b, es decir, la multiplicación en el campo finito.

Funcionamiento:

1. Se inicializa result en 0.
2. Mientras y no sea cero:
    - Si el bit menos significativo de y es 1, se hace result ^= x.
    - Luego se multiplica x por 2 usando xtime(x).
    - Se divide y por 2 desplazando un bit a la derecha (y >>= 1).
3. Al finalizar, result contiene x * y en GF(2^8).

Transformaciones de Cifrado

void SubBytes(uint8_t state[4][4])

Realiza una sustitución no lineal de cada byte en la matriz de estado utilizando la tabla *S-box*.

Parámetros

- **uint8_t state[4][4]:** Matriz de 4×4 bytes que representa el estado intermedio del bloque de datos a cifrar o descifrar. La operación modifica esta matriz directamente.

Funcionamiento:

Recorre cada byte de la matriz *state* y lo reemplaza por su equivalente en la tabla *s_box*.

void ShiftRows(uint8_t state[4][4])

Desplaza cíclicamente las filas de la matriz de estado hacia la izquierda, aumentando el desplazamiento por cada fila (la primera fila no se mueve).

Parámetro:

- **uint8_t state[4][4]:** Matriz de estado de 4×4 bytes que se modificará en el lugar.

Funcionamiento:

- La fila 0 se deja sin cambios.
- La fila 1 se rota 1 byte a la izquierda.
- La fila 2 se rota 2 bytes a la izquierda.
- La fila 3 se rota 3 bytes a la izquierda (equivale a 1 byte a la derecha).

void MixColumns(uint8_t state[4][4])

Combina linealmente los bytes de cada columna del estado usando multiplicaciones en el campo finito GF(2^8).

Parámetros

- **uint8_t state[4][4]:** Matriz de estado de 4×4 bytes. El contenido de cada columna será mezclado en el lugar.

Funcionamiento:

Para cada columna, se calcula una nueva columna aplicando multiplicaciones por constantes (0x02, 0x03) en GF(2^8), seguidas de operaciones XOR entre los bytes.

void AddRoundKey(uint8_t state[4][4], uint8_t* roundKey)

Combina el estado actual con una subclave de ronda mediante la operación XOR.

Parámetros:

- **uint8_t state[4][4]:** Matriz de estado de 4×4 bytes que se modificará directamente.
- **uint8_t* roundKey:** Puntero a un arreglo de 16 bytes que representa la subclave correspondiente a una ronda del cifrado o descifrado.

Funcionamiento:

Cada byte del estado se combina mediante XOR con el byte correspondiente de la subclave. Los índices se calculan en orden columna mayor, es decir, primero se recorren las filas y luego las columnas.

Transformaciones de Descifrado

void InvSubBytes(uint8_t state[4][4])

Aplica la transformación inversa de SubBytes, sustituyendo cada byte del estado por su valor correspondiente en la tabla *inv_s_box*.

Parámetros

- **uint8_t state[4][4]:** Matriz de estado de 4×4 bytes que será modificada directamente.

Funcionamiento:

Recorre todos los elementos del estado y reemplaza cada byte usando la S-box inversa (*inv_s_box*). Esta tabla es el inverso de la S-box utilizada en la etapa de cifrado.

void InvShiftRows(uint8_t state[4][4])

Revierte la transformación ShiftRows desplazando las filas del estado hacia la derecha, en vez de hacia la izquierda, restaurando el orden original de los bytes antes del cifrado.

Parámetros

- **uint8_t state[4][4]:** Matriz de estado que será modificada en el lugar.

Funcionamiento:

- La fila 0 no se modifica.
- La fila 1 rota 1 byte a la derecha.
- La fila 2 rota 2 bytes a la derecha.
- La fila 3 rota 3 bytes a la derecha (equivalente a 1 a la izquierda).

void InvMixColumns(uint8_t state[4][4])

Revierte la operación MixColumns multiplicando cada columna del estado por la matriz inversa de MixColumns en el campo finito GF(2^8).

Parámetros

- **uint8_t state[4][4]:** Matriz de estado que será actualizada directamente.

Funcionamiento:

Para cada columna de la matriz *state*, se realiza una multiplicación por los coeficientes {0x0e, 0x0b, 0x0d, 0x09} en GF(2^8), según la matriz inversa definida por el estándar AES.

Expansión de Llave

void KeyExpansion(const uint8_t* key, uint8_t* roundKeys)

Genera todas las subclaves (también conocidas como round keys) necesarias para las 10 rondas del algoritmo AES-128 a partir de una clave original de 16 bytes. El resultado es un arreglo expandido de 176 bytes que contiene las claves de ronda utilizadas durante el proceso de cifrado y descifrado.

Parámetros:

- **const uint8_t* key:** Puntero a un arreglo de 16 bytes que representa la clave original utilizada como entrada.
- **uint8_t* roundKeys:** Puntero a un arreglo de 176 bytes donde se almacenarán las **subclaves generadas**. Cada ronda del algoritmo AES-128 utiliza una clave de 16 bytes, y se requieren 11 claves (una por ronda, incluyendo la inicial).

Funcionamiento:

1. **Inicialización**:
   Se copian los primeros 16 bytes de la clave original directamente en el arreglo roundKeys.
2. **Generación de nuevas palabras**:

   A partir de la quinta palabra (índice 4 en adelante), se generan nuevas palabras de 4 bytes usando una combinación de operaciones:

   - **RotWord**: Rota los 4 bytes de una palabra una posición hacia la izquierda.
   - **SubWord**: Sustituye cada byte por su valor correspondiente en la tabla S-box.
   - **Rcon**: Se aplica un XOR entre el primer byte de la palabra y un valor de la tabla Rcon, dependiendo del índice de la ronda.
   - **XOR**: Finalmente, la nueva palabra se obtiene mediante XOR con la palabra generada Nk posiciones atrás (4 posiciones para AES-128).
3. **Repetición**:
   Este proceso se repite hasta completar las 44 palabras (4 bytes cada una) necesarias para todas las rondas del algoritmo.

Funciones AES

void AES_encrypt(const uint8_t* input, const uint8_t* key, uint8_t* output)

Cifra un bloque de 16 bytes (*input*) utilizando una clave de 16 bytes (*key*) con el algoritmo AES-128, y coloca el resultado en *output*.

Parámetros:

- **const uint8_t* input:** Puntero a un arreglo de 16 bytes que representa el bloque de texto plano a cifrar. Cada byte corresponde a un carácter de los datos sin cifrar.
- **const uint8_t* key:** Puntero a un arreglo de 16 bytes que contiene la clave secreta AES-128. Esta clave se utiliza para generar las subclaves de ronda mediante expansión.
- **uint8_t* output:** Puntero a un arreglo de 16 bytes donde se almacenará el resultado cifrado (ciphertext). El resultado se organiza en el mismo orden que el input (columna mayor).

Funcionamiento:

1. **Inicialización**
    - Se declara la matriz *state[4][4]* que representará el bloque de entrada como una matriz de estado en orden columna mayor.
    - Se declara *roundKeys[176]* para almacenar todas las subclaves generadas por la expansión de clave (AES-128 utiliza 11 claves de ronda × 16 bytes = 176 bytes).
2. **Expansión de clave**
    - *KeyExpansion()* genera todas las subclaves necesarias a partir de la clave de 16 bytes.
3. **Copia al estado**
    - Se reorganiza el arreglo input en state, colocándolo en formato columna mayor: *state[row][col] = input[col*4 + row]*.
4. **Ronda inicial**
    - AddRoundKey aplica la primera subclave al estado antes de comenzar las rondas principales.
5. **9 Rondas principales (AES-128 = 10 rondas en total)**
    - En cada ronda:
        - SubBytes: Sustituye cada byte usando la S-box.
        - ShiftRows: Rota filas del estado.
        - MixColumns: Mezcla las columnas (difusión).
        - AddRoundKey: Aplica la subclave correspondiente.

6. **Última ronda (ronda 10)**
    - Igual que las rondas anteriores, pero sin MixColumns.
7. **Copia del resultado**
    - Se extraen los 16 bytes del estado y se colocan en *output*, manteniendo el orden columna mayor.

void AES_decrypt(const uint8_t* input, const uint8_t* key, uint8_t* output)

Descifra un bloque de 16 bytes (*input*) usando AES-128 y una clave de 16 bytes (*key*), y almacena el texto plano en *output*.

Parámetros:

- **const uint8_t* input:** Puntero a un arreglo de 16 bytes que representa el bloque cifrado (ciphertext) a descifrar.
- **const uint8_t* key:** Puntero a un arreglo de 16 bytes que contiene la clave secreta AES-128 usada para el descifrado. Debe coincidir exactamente con la clave usada en el cifrado.
- **uint8_t* output:** Puntero a un arreglo de 16 bytes donde se almacenará el texto plano resultante del proceso de descifrado.

Funcionamiento:

1. **Inicialización**
    - Igual que en el cifrado: se usa una matriz *state[4][4]* y un arreglo de subclaves *roundKeys[176].*
2. **Expansión de clave**
    - Se generan las subclaves con *KeyExpansion().*
3. **Copia al estado**
    - Se copia el input al estado en formato columna mayor.
4. **Ronda inicial**
    - Se aplica la última subclave *(roundKeys + Nr * 16),* que en AES-128 corresponde a la ronda 10.
5. **Rondas inversas**
    - Desde la ronda 9 hasta la 1:
        - InvShiftRows: Inverso de ShiftRows.
        - InvSubBytes: Inverso de SubBytes.
        - AddRoundKey: Aplica la subclave correspondiente.
        - InvMixColumns: Inverso de MixColumns.
6. **Última ronda inversa**

- Similar a las anteriores, pero sin InvMixColumns.
- Se aplica la subclave original (*roundKeys*).

7. **Copia del resultado**
    - El estado descifrado se copia a output en orden columna mayor.

Funciones para obtener y tratar los datos del usuario

void clear_input_buffer()

Se encarga de limpiar el buffer de entrada del teclado, para evitar que se queden caracteres que puedan corromper el programa.

void ascii_to_bytes(const char* input, uint8_t* output)

Toma como entrada una cadena de texto (*input*) y convierte cada uno de sus primeros 16 caracteres a su valor ASCII, almacenando esos valores en un arreglo de bytes (*output*).

Parámetros:

- **const char* input:** Puntero a una cadena de caracteres de al menos 16 elementos.
- **uint8_t* output:** Puntero a un arreglo de 16 elementos donde se almacenarán los valores ASCII.

Funcionamiento:

Dentro de un ciclo for que itera de 0 a 15, cada carácter de la cadena input es convertido explícitamente a tipo uint8_t y almacenado en la posición correspondiente del arreglo *output*. Esto se hace mediante una conversión de tipo *((uint8_t)input[i])*.

int hex_to_bytes(const char* hex_input, uint8_t* output)

Convierte una cadena de 32 caracteres que representa 16 bytes en formato hexadecimal (dos caracteres por byte) en un arreglo de 16 valores uint8_t. Cada par de caracteres hexadecimales es interpretado como un solo byte.

Parámetros:

- **const char* hex_input:** Puntero a una cadena que contiene 32 caracteres hexadecimales válidos (0-9, a-f, A-F).
- **uint8_t* output:** Puntero a un arreglo de 16 bytes donde se almacenarán los valores convertidos.

Funcionamiento:

1. Se itera de i = 0 a 15 (un total de 16 bytes).
2. En cada iteración:

- Se extraen dos caracteres consecutivos de *hex_input*, formando una subcadena *byte_str* con longitud 2 (más el carácter nulo al final).
- Se convierte la subcadena *byte_str* a un número entero base 16 usando strtol.
- Se valida que la conversión fue correcta:
    - El puntero endptr debe apuntar al final de la cadena ('\0').
    - El valor convertido debe estar entre 0 y 255.
- Si hay un error en la conversión, se imprime un mensaje y se retorna -1.
- Si la conversión es válida, se guarda el valor como uint8_t en *output[i]*.

3. Si todas las conversiones son exitosas, la función retorna 0.

int encrypt()

Solicita al usuario un texto plano y una clave de 16 caracteres, convierte ambos a arreglos de bytes (uint8_t), realiza el cifrado AES-128 sobre el texto usando la clave proporcionada y muestra el resultado en formato hexadecimal.

Funcionamiento:

1. **Declaración de variables:**
    - *input_text[17]* y *input_key[17]*: Arreglos para almacenar las entradas del usuario (16 caracteres + carácter nulo \0).
    - *plaintext[16]*, *key[16]*, *ciphertext[16]*: Arreglos de bytes para el texto plano, la clave y el texto cifrado, respectivamente.
2. **Entrada del usuario:**
    - Se solicita el texto plano. Se usa fgets para leer hasta 16 caracteres (más el nulo).
    - Se verifica que fgets no haya fallado. Si ocurre un error, se limpia el búfer de entrada y se retorna -1.
    - Se repite el mismo proceso para la clave.
3. **Validación de longitud:**
    - Se verifica que ambas cadenas (*input_text* y *input_key*) tengan exactamente 16 caracteres (excluyendo el nulo).
    - Si alguna no cumple la longitud, se imprime un mensaje de error y se retorna -1.
4. **Conversión a bytes:**
    - Se convierte cada carácter del texto y la clave a su valor ASCII usando la función ascii_to_bytes.
5. **Cifrado AES:**
    - Se llama a AES_encrypt(plaintext, key, ciphertext) para cifrar el texto usando la clave. Se espera que esta función realice el cifrado AES-128 ECB sobre un bloque de 16 bytes.
6. **Salida cifrada:**

- Se recorre el arreglo ciphertext e imprime cada byte en formato hexadecimal de dos dígitos (%02x), separados por espacios.

int decrypt()

Solicita al usuario un texto cifrado en formato hexadecimal (32 caracteres que representan 16 bytes) y una clave de 16 caracteres, los convierte en arreglos de bytes, realiza el descifrado AES-128, y muestra el resultado en formato ASCII.

Funcionamiento:

1. **Declaración de variables:**
   - *hex_input[33]*: Cadena para almacenar el texto cifrado en formato hexadecimal (32 caracteres + carácter nulo).
   - *input_key[17]*: Cadena para la clave (16 caracteres + carácter nulo).
   - *ciphertext[16], key[16], decrypted[16]*: Arreglos de bytes para el texto cifrado, la clave y el resultado descifrado, respectivamente.
2. **Entrada del usuario:**
   - Se solicita el texto cifrado en hexadecimal.
     - Se utiliza fgets para leer hasta 32 caracteres.
     - Si falla la lectura, se limpia el búfer y retorna error -1.
   - Se solicita la clave AES de 16 caracteres con la misma validación.
3. **Validación de longitud:**
   - Se verifica que *hex_input* tenga exactamente 32 caracteres y que *input_key* tenga 16.
     - Si no se cumple, se imprime un mensaje de error y se retorna -1.
4. **Conversión a bytes:**
   - *hex_input* se convierte a un arreglo de 16 bytes mediante la función *hex_to_bytes*.
     - Si hay un error (por ejemplo, caracteres no hexadecimales), se retorna -1.
   - *input_key* se convierte a bytes ASCII mediante la función *ascii_to_bytes*.
5. **Descifrado AES:**
   - Se llama a *AES_decrypt(ciphertext, key, decrypted)* para descifrar el texto cifrado.
6. **Impresión del resultado:**
   - Se imprime cada byte del arreglo *decrypted* como un carácter ASCII.

Función Principal

int main()

Se encarga de mostrar un menú al usuario para seleccionar si cifrar, descifrar o salir del programa, por medio de una variable local llamada *opcion*.

Imprime un mensaje de error en caso de que se digite algo diferente a un número. Si la variable *opción* cuenta con el valor uno lleva a cabo el cifrado, si es dos lleva a cabo el descifrado, en caso de ser tres termina con el programa, y si no es ninguno de los valores mencionados imprime un error indicando que no es un valor válido.