ECE3300 AES FINAL PROJECT

Jason Moya, Bianca Chavez, Kathleen Mach, Ramiro Ascencio

Abstract—**Advanced Encryption Standard, also known as AES, is a symmetric encryption algorithm that is used to protect sensitive information, secure communication, and hold data storage. Symmetric encryption means that there is one key for both the encrypting and decrypting processes. This algorithm converts plaintext into cipher text, ensuring that a key is needed to be able to access the original message. AES goes through various "rounds" of encryption that make the initial message harder to decode.**

I. Introduction

In a world where technology advances and evolves rapidly, The Advanced Encryption Standard (AES) serves to provide robust and secure encryption standards for the information exchanged and serves to protect the integrity of sensitive information. This academic paper delves into the history, design, procedure, and methods of AES. Starting from its response to being an upgrade from its predecessor Data Encryption Standard (DES), striving to improve on what DES falls short on, to becoming a standard in cryptography.

II. History

The Advanced Encryption Standard (AES) became a crucial development for cryptography, and became the replacement for Data Encryption Standard (DES). DES was established in the 1970's by International Business Machines (IBM), and it was adapted from their project called LUCIFER. DES was very important and influential during the time it was used, however it's key size was only 64 bits, with 8 of it's bits used for a parity test, this made it really easy to brute force due to its small key size. Due to rapidly changing technology and the development of faster computers and processors, the National Institute of Standards and Technology (NIST) sought to find a new standard, especially one that has a larger key size. Thus, in November of 2001, AES became the new encryption standard. AES offers a stronger security with key lengths of 128, 192, or 256 bits, which ensures more security and protection against most cryptographic threats.

III. Definitions

AES- Advanced Encryption Standard, a cryptographic algorithm that can be used to protect electronic data.
Cryptographic algorithm- a symmetric block cipher that can encrypt or decrypt data.
Brute Force- common trial and error method used to try and crack a security mechanism by attempting all possible combinations until the correct value is found.
Symmetric block cipher- uses the same key that works for encrypting and decrypting.
SP Network- Mathematical operations used in block cypher algorithms by using substitution and permutation of bits.
Cipher- Repeated transformations converting plaintext into ciphertext using a cipher key.
Hill Cypher- Symmetrical key encryption algorithm for encryption and decryption. Mainly focuses on matrix based approaches to perform mathematical transformations on the plaintext.
Ciphertext- encrypted text transformed from plaintext using an encryption algorithm
Round Key- unique derived encryption key that is used for each round of decryption and encryption
Encryption- converts data to an unintelligible form called ciphertext
Decryption- converts data back to it's original form, plaintext
Substitution- algorithm replaces the plaintext with the ciphertext based on a predefined cipher
Shifting- the second step in the algorithm, which allows the rows to be shifted by 1, except for the first.
Mixing- A cipher method used to mix columns to prevent a user from shifting rows back to decrypt data.
Further Encryption- a small portion of the encryption key is used to encrypt that data block
Plaintext- data before encryption
Rijndael Algorithm- another term for AES
Affine Transformation- A transformation consisting of multiplication by a matrix followed by the addition of a vector
Array- data structure consisting of a collection of elements, of same memory size, each identified by at least one array index or key **Bit**-unit of information expressed as either a 0 or 1
Block- fixed set of data that is processed in the AES algorithm
Byte - a group of binary digits or bits
Matrix-a set of numbers arranged in rows and columns so as to form a rectangular array
Cells-an electronic circuit that stores one bit of binary information
Logic Gates- a device that acts as a building block for digital circuits.
State- two dimensional array or matrix that holds a single block of decrypted or encrypted data.
FPGA- Also known as field programmable gate array, which are integrated circuits that are able to be reconfigured to meet/fit certain needs

IV. Process of AES

1) Round Key: In the first step, the round key takes the state array from the text, and performs an XOR operation (refer to Fig 1.) of the state array to pass onto the next step of substitution where the AES algorithm starts. After the Mix Columns step, the current state array is XOR'd with the previous key and then passes its result to the next round to become the next round key. The algorithm repeats until the last round where the resulting RoundKey becomes the ciphertext. Round keys play a crucial role in the effectiveness of the entire encryption process. These keys are the derivation from the original key and are implemented in each round to add more complexity to the process. Before undergoing the AES algorithm, the plaintext is converted into Hex. In each round, the round key is XORed to the current state by using the operation, addRoundKey. A new round key can be declared to different blocks of data.

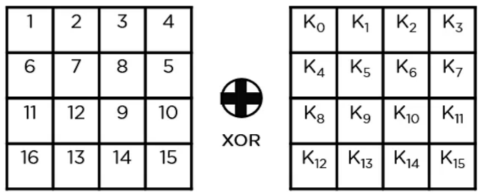


Fig. 1. Example of a round key performing XOR operation

2) Substitution: The next step in the AES process is substitution which helps with the encryption process by changing the bytes in the array which helps with the encryption portion of AES. It takes the state array from the past step and transforms every byte independently in the past state array into a hexadecimal byte for the encryption. This step is only possible by the AES lookup table that takes the bytes into it and transforms it into another state array with new values in. Then it becomes the new present array that goes into the next step. It can help with making it more difficult for people to hack into the system. Since this process uses an s-box (refer to Fig 2.) which is a lookup table that is created by an algorithm to create confusion with the hacker that is trying to get into the system. It can even pick a few certain bytes to create even more confusion in the system. In the field there should be no way that someone can track you down.

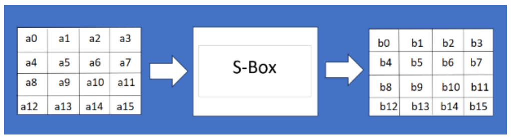


Fig. 2. Example of an S-Box being used

3) Shifting: Shifting is the next step that follows the substitution of AES. This procedure is done by shifting the rows in the array to the left (refer to Fig 3.). Each row holds bytes which are shifted a particular number of times. The first row is always unchanged, while the consecutive rows are shifted. For example, the second row is shifted once to the left, the third is shifted twice, the fourth is shifted three times, and the pattern continues depending on the number of rows in the system. When dealing with 128 bits, the normal operation is a 4x4 matrix. The purpose of the ShiftRows step is to shuffle the bytes within each row to increase the confusion and complexity between the input and output. After each shift in the row, there is a new arrangement of bytes that can prevent the attacker from recognizing patterns. Shifting is structured into AES to contribute to the overall security and complexity of the encryption process. Here is an example of when the bits are shifted in a matrix.

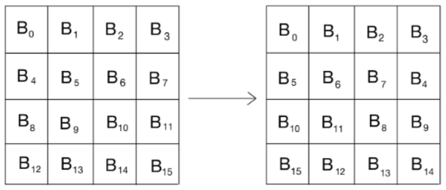


Fig. 3. Example of rows being shifted

4) Mixing: The next process of the AES is mixing which mixes up the bytes in each column to get a new column from the current state array which can help with encryption in systems. This helps with creating more confusion by multiplying the column of the array with a matrix that will create a new state array for the system. To produce these new values for the state array it is XOR to get the new bytes of each column.
The mixing step in the process helps with taking arrays into a multiplication step with a matrix which is made with a four term polynomial (refer to Fig 4.). Every column is put into through the matrix helps create another layer of security to help with securing the system and making it a lot more difficult to solve the key for the system. The process of mixing the column gives more complexity since a small input of the byte can influence a whole column in the array. By creating this new state array for the security of the system it is used for the final step of AES.

V. Serial Input Data for AES

The board being implemented for this AES process is the Nexyss A7-100T, which contains components such as:

- UART: Universal Asynchronous Receiver/Transmitter– a hardware communication protocol used for serial commu-

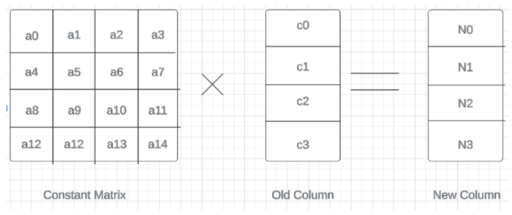


Fig. 4. Example of mixing

nication between devices, using port to port connections, and can be used for transmitting or receiving data.
- ADC: Analog to Digital Converter
- 8 seven segments
- SPI: Serial Peripheral Interface– a synchronous serial communication protocol that allows a high-speed data transfer between a master device and one or more slave devices.
- 16 switches
- 5 buttons
- PMODS: Peripheral Modules– small input output interface board standard designed by Dilgilent, and more hardware components can be possibly added to the original FPGA.
- 16 LEDs
- VGA: Video Graphics Array– analog video standard that can be used to connect computer monitors and displays.

In order to obtain data for the first step of the AES process, UART is implemented. There is a port to port connection from an external keyboard USB cable connected to the USB Host on the board, and data can then be stored.

VI. Code

A. Round Key

In this code, we will be using a 128 bit RoundKey and it will be added with the data using the XOR operation. The key we are using is a predetermined key, which is “128’h000102030405060708090a0b0c0d0e0f”.

```
module addRoundKey(data, out, key);

input [127:0] data;
input [127:0] key;
output [127:0] out;

assign out = key ^ data;

endmodule
```

B. Substitution

Within the code we are using a for loop to exchange the current data from the round key into a s-box that converts them into a different matrix

```
module subBytes(in,out);
input [127:0] in;
output [127:0] out;

genvar i;
generate
for(i=0;i<128;i=i+8) begin :sub_Bytes
  sbox s(in[i +:8],out[i +:8]);
  end
endgenerate


endmodule
```

C. Shifting

For this source file, we assign data from the last step into their respective rows. It can be seen that within every row after the first there is going to be movement depending on the position of the row to create a new complex matrix

```
module shiftRows (in, shifted);
  input [0:127] in;
  output [0:127] shifted;

  // First row (r = 0) is not shifted
  assign shifted[0+:8] = in[0+:8];
  assign shifted[32+:8] = in[32+:8];
  assign shifted[64+:8] = in[64+:8];
  assign shifted[96+:8] = in[96+:8];

  // Second row (r = 1) is cyclically left
      shifted by 1 offset
  assign shifted[8+:8] = in[40+:8];
  assign shifted[40+:8] = in[72+:8];
  assign shifted[72+:8] = in[104+:8];
  assign shifted[104+:8] = in[8+:8];

  // Third row (r = 2) is cyclically left
      shifted by 2 offsets
  assign shifted[16+:8] = in[80+:8];
  assign shifted[48+:8] = in[112+:8];
  assign shifted[80+:8] = in[16+:8];
  assign shifted[112+:8] = in[48+:8];

  // Fourth row (r = 3) is cyclically left
      shifted by 3 offsets
  assign shifted[24+:8] = in[120+:8];
  assign shifted[56+:8] = in[24+:8];
  assign shifted[88+:8] = in[56+:8];
  assign shifted[120+:8] = in[88+:8];

endmodule
```

D. Mixing

In the code below, we are mixing the matrix rows by shifting it to the left by multiplying it by 2 or 3. This code also checks for a 1 in the most significant bit, and if this is true, then it will perform the XOR operation.

```
module mixColumns(state_in,state_out);

input [127:0] state_in;
```

```
output[127:0] state_out;

function [7:0] mb2; //multiply by 2
  input [7:0] x;
  begin
      /* multiplication by 2 is shifting one
          bit to the left, and if the
          original 8 bits had a 1 @ MSB,
      xor the result with {1b}*/
      if(x[7] == 1) mb2 = ((x << 1) ^ 8'h1b);
      else mb2 = x << 1;
  end
endfunction

/*
  multiplication by 3 is done by:
    multiplication by {02} xor(the original x)
    so that 2+1=3. where xor is the addition
        of elements in finite fields
*/
function [7:0] mb3; //multiply by 3
  input [7:0] x;
  begin

      mb3 = mb2(x) ^ x;
  end
endfunction

genvar i;

generate
for(i=0;i< 4;i=i+1) begin : m_col

  assign state_out[(i*32 + 24)+:8]=
      mb2(state_in[(i*32 + 24)+:8]) ^
      mb3(state_in[(i*32 + 16)+:8]) ^
      state_in[(i*32 + 8)+:8] ^
      state_in[i*32+:8];
  assign state_out[(i*32 + 16)+:8]=
      state_in[(i*32 + 24)+:8] ^
      mb2(state_in[(i*32 + 16)+:8]) ^
      mb3(state_in[(i*32 + 8)+:8]) ^
      state_in[i*32+:8];
  assign state_out[(i*32 + 8)+:8]=
      state_in[(i*32 + 24)+:8] ^
      state_in[(i*32 + 16)+:8] ^
      mb2(state_in[(i*32 + 8)+:8]) ^
      mb3(state_in[i*32+:8]);
  assign state_out[i*32+:8]=
      mb3(state_in[(i*32 + 24)+:8]) ^
      state_in[(i*32 + 16)+:8] ^
      state_in[(i*32 + 8)+:8] ^
      mb2(state_in[i*32+:8]);

end

endgenerate

endmodule
```

E. VGA

This code controls the VGA to get our output to display onto the screen. The code uses a 100 MHz clock, and reset as it's inputs.

```
module vga_controller(
    input clk_100MHz, // from Basys 3
    input reset,    // system reset
    output video_on, // ON while pixel counts
        for x and y and within display area
    output hsync,   // horizontal sync
    output vsync,   // vertical sync
    output p_tick, // the 25MHz pixel/second
        rate signal, pixel tick
    output [9:0] x, // pixel count/position of
        pixel x, max 0-799
    output [9:0] y // pixel count/position of
        pixel y, max 0-524
    );

    parameter HD = 640;      // horizontal
        display area width in pixels
    parameter HF = 48;       // horizontal
        front porch width in pixels
    parameter HB = 16;       // horizontal back
        porch width in pixels
    parameter HR = 96;       // horizontal
        retrace width in pixels
    parameter HMAX = HD+HF+HB+HR-1; // max
        value of horizontal counter = 799
    // Total vertical length of screen = 525
        pixels, partitioned into sections
    parameter VD = 480;      // vertical
        display area length in pixels
    parameter VF = 10;       // vertical front
        porch length in pixels
    parameter VB = 33;       // vertical back
        porch length in pixels
    parameter VR = 2;        // vertical
        retrace length in pixels
    parameter VMAX = VD+VF+VB+VR-1; // max
        value of vertical counter = 524

    // *** Generate 25MHz from 100MHz
        ************************************************
   reg [1:0] r_25MHz;
   wire w_25MHz;

   always @(posedge clk_100MHz or posedge
       reset)
     if(reset)
       r_25MHz <= 0;
     else
       r_25MHz <= r_25MHz + 1;

   assign w_25MHz = (r_25MHz == 0) ? 1 : 0; //
       assert tick 1/4 of the time
    //
        ************************************************

    // Counter Registers, two each for
        buffering to avoid glitches
    reg [9:0] h_count_reg, h_count_next;
    reg [9:0] v_count_reg, v_count_next;

    // Output Buffers
    reg v_sync_reg, h_sync_reg;
    wire v_sync_next, h_sync_next;

    // Register Control
    always @(posedge clk_100MHz or posedge
    reset)
   if(reset) begin
      v_count_reg <= 0;
      h_count_reg <= 0;
      v_sync_reg <= 1'b0;
      h_sync_reg <= 1'b0;
   end
   else begin
      v_count_reg <= v_count_next;
      h_count_reg <= h_count_next;
      v_sync_reg <= v_sync_next;
      h_sync_reg <= h_sync_next;
   end

//Logic for horizontal counter
always @(posedge w_25MHz or posedge reset)
    // pixel tick
   if(reset)
      h_count_next = 0;
   else
      if(h_count_reg == HMAX)        // end
          of horizontal scan
         h_count_next = 0;
      else
         h_count_next = h_count_reg + 1;

// Logic for vertical counter
always @(posedge w_25MHz or posedge reset)
   if(reset)
      v_count_next = 0;
   else
      if(h_count_reg == HMAX)        // end
          of horizontal scan
         if((v_count_reg == VMAX))   // end
             of vertical scan
            v_count_next = 0;
         else
            v_count_next = v_count_reg + 1;

// h_sync_next asserted within the
    horizontal retrace area
assign h_sync_next = (h_count_reg >=
    (HD+HB) && h_count_reg <=
    (HD+HB+HR-1));

// v_sync_next asserted within the
    vertical retrace area
assign v_sync_next = (v_count_reg >=
    (VD+VB) && v_count_reg <=
    (VD+VB+VR-1));

// Video ON/OFF - only ON while pixel
    counts are within the display area
assign video_on = (h_count_reg < HD) &&
    (v_count_reg < VD); // 0-639 and 0-479
    respectively

// Outputs
assign hsync = h_sync_reg;
assign vsync = v_sync_reg;
assign x   = h_count_reg;
assign y   = v_count_reg;
assign p_tick = w_25MHz;

endmodule
```

F. Top File

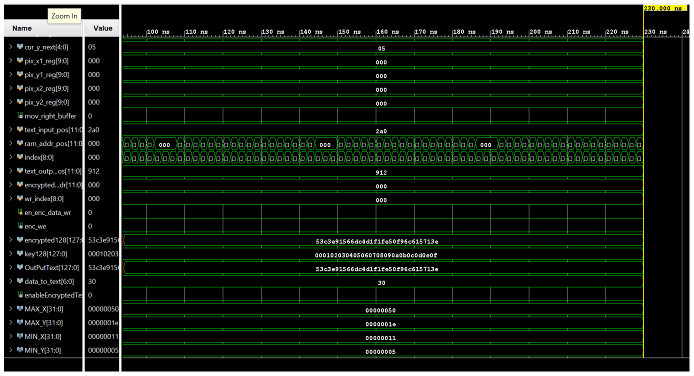
In this file, we will be connecting all our modules together to create our project.

```
module top(
      input clk,         // 100MHz
      input reset,
      input set_btn,     // btnC
      input wire psd,
      input wire psc,
      input left,        // btnL
      input right,       // btnR
      input activate_read,
      input enable_output_text,
      output hsync, vsync, // VGA connector
      output [7:0] ascii_led,
      output data_read_led,
      output [11:0] rgb, // DAC, VGA connector
      output dp_out,
      output [6:0] display_controller_cc_out,
      output [7:0] display_controller_an_out
   );

   // signals
   wire [9:0] w_x, w_y;
   wire w_vid_on, w_p_tick;
   reg [11:0] rgb_reg;
   wire [11:0] rgb_next;

   // instantiate vga controller
   vga_controller vga(.clk_100MHz(clk),
       .reset(reset), .video_on(w_vid_on),
                  .hsync(hsync),
                      .vsync(vsync),
                      .p_tick(w_p_tick),
                  .x(w_x), .y(w_y));

   wire [7:0] ascii;
   wire ascii_ready;
   keyboard_input ASCII_out(
     .clk(clk),
     .reset(reset),
     .scan_ready(ascii_ready),
     .ps2d(psd),
     .ps2c(psc),
       .ascii_code(ascii)
  );

   assign ascii_led = ascii;


   // instantiate text generation circuit
   text_screen_gen tsg(.clk(clk),
       .reset(reset), .video_on(w_vid_on),
       .set(set_btn),
                   .up(), .down(),
                       .left(left),
                       .right(right),
                       .read_txt_input(activate_read),
                   .enable_output_text(enable_output_text
                   .sw(ascii), .x(w_x),
                       .y(w_y),
                       .data_read_led(data_read_led),
                       .rgb(rgb_next));

   // rgb buffer
   always @(posedge clk)
      if(w_p_tick)
         rgb_reg <= rgb_next;

   // output
   assign rgb = rgb_reg;


   /* Clk divider for sevent segment */
   wire sev_seg_clk;
   clk_counter sevSegClk(
      .sys_clk(clk),
      .sys_rst(reset),
      .speed_sel(5'b10000),
      .block_clk(sev_seg_clk)
   );

   /* Seven segment*/
   display_controller myDisplayCntrl(
      .display_controller_clk(sev_seg_clk),
      .display_controller_dp_in(8'b11111111),
      .display_controller_an_in(8'b00000000),
      .display_controller_ssd0(ascii[3:0]),
      .display_controller_ssd1(ascii[7:4]),
      .display_controller_ssd2(8'h00),
      .display_controller_ssd3(8'h00),
      .display_controller_ssd4(8'h00),
      .display_controller_ssd5(8'h00),
      .display_controller_ssd6(8'h00),
      .display_controller_ssd7(8'h00),
      .display_controller_dp_out(dp_out),
      .display_controller_cc_out(display_controller
      _cc_out),
      .display_controller_an_out(display_controller
      _an_out)
   );

endmodule
```

G. Test Bench

Here we are testing all the modules of our project including our Top file. We are testing by going through each possible input within an initial block. It's output in the form of a timing diagram is displayed in Fig. 5.

```
module top_tb( );
   reg clk_tb;        // 100MHz Basys 3
   reg reset_tb;     // sw[15]
   reg set_btn_tb;       // btnC
   reg psd_tb; // reg wire
   reg psc_tb; // reg wire
   reg left_tb;       // btnL
   reg right_tb;      // btnR

   reg activate_read_tb;
   reg enable_output_text_tb;

   wire hsync_tb;
   wire vsync_tb; // VGA connector
   wire [7:0] ascii_led_tb;
   wire data_read_led_tb;
   wire [11:0] rgb_tb;

initial
   begin
         clk_tb = 1;
      end
         always
            begin
         #1
            clk_tb = ~clk_tb;
         end


   top myTop(
      .clk(clk_tb),        // 100MHz Basys 3
      .reset(reset_tb),     // sw[15]
      .set_btn(set_btn_tb),      // btnC
      .psd(psd_tb),
      .psc(psc_tb),
      .left(left_tb),       // btnL
      .right(right_tb),     // btnR

      .activate_read(activate_read_tb),
      .enable_output_text(enable_output_text_tb),

      .hsync(hsync_tb),
      .vsync(vsync_tb), // VGA connector
      .ascii_led(ascii_led_tb),
      .data_read_led(data_read_led_tb),
      .rgb(rgb_tb)   // DAC, VGA connector
      );


   initial
      begin

         reset_tb = 1'b1;
         set_btn_tb = 1'b1;       // btnC
         psd_tb = 1'b1; // reg wire
         psc_tb = 1'b1; // reg wire
         left_tb = 1'b0;       // btnL
         right_tb = 1'b0;      // btnR
         #10
         left_tb = 1'b1;       // btnL
         right_tb = 1'b1;

         activate_read_tb = 1'b0;
         #10
         activate_read_tb = 1'b1;
         enable_output_text_tb = 1'b0;
         #10
        // activate_read_tb = 1'b1;
         enable_output_text_tb = 1'b1;
         #100
         #100


        $finish;
      end


endmodule
```



Name	Value
cur_y_next[4:0]	05
pix_x1_reg[9:0]	000
pix_y1_reg[9:0]	000
pix_x2_reg[9:0]	000
pix_y2_reg[9:0]	000
mov_right_buffer	0
text_input_pos[11:0	2a0
ram_addr_pos[11:0	000
index[8:0]	000
text_outp...os[11:0]	912
encrypted...dr[11:0]	000
wr_index[8:0]	000
en_enc_data_wr	0
enc_we	0
encrypted128[127:(	53c3e9156
key128[127:0]	00010203
OutPutText[127:0]	53c3e9156
data_to_text[6:0]	30
enableEncryptedTe	0
MAX_X[31:0]	00000050
MAX_Y[31:0]	0000001e
MIN_X[31:0]	00000011
MIN_Y[31:0]	00000005

```
53c3e91566dc4d1f1fe50f96c615713e
000102030405060708090a0b0c0d0e0f
53c3e91566dc4d1f1fe50f96c615713e
```

Fig. 5. Testbench output

VII. Conclusion

In our project, we successfully implemented the Advanced Encryption Standard (AES) using Verilog on the Nexys A7 board, strengthening data security in today's digital world is an importnant element. Our design incorporates key components like round key generation, data substitution, shifting, and mixing columns, collectively strengthening the encryption process by creating a complex system. Utilizing the Nexys A7 board's VGA output, we effectively visualized each AES operation, providing both an insightful demonstration of encryption and an interactive educational tool. This integration not only fortifies data security but also offers a deeper understanding of cryptographic techniques.

VIII. References

[1]A. Kak, "Lecture 8: AES: The Advanced Encryption Standard Lecture Notes on " Computer and Network Security "," 2016. Available: https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf

[2]"aes/src/rtl at master · secworks/aes," GitHub. https://github.com/secworks/aes/tree/master/src/rtl (accessed Dec. 11, 2023).

[3]Giac.org, 2023. https://www.giac.org/paper/gsec/2048/s-box-modifications-effect-des-like-encryption-systems/103534 (accessed Dec. 11, 2023).

[4]"HISTORY OF DES," Umsl.edu, 2021. https://www.umsl.edu/ siegelj/information$_t heory/projects/des.netau.net/des$
[5]I. T. L. Computer Security Division, "Cryptographic Standards and Guidelines — CSRC — CSRC," CSRC — NIST, Dec. 29, 2016. https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines

[6]NIST, "Federal Information Processing Standards Publication 197 Announcing the ADVANCED ENCRYPTION STANDARD (AES)," 2001. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[7]"What are the role of S-boxes in DES?," www.tutorialspoint.com. https://www.tutorialspoint.com/what-are-the-role-of-s-boxes-in-des

[8]"What is a Cipher Key? (with pictures)," Easy Tech Junkie, Oct. 27, 2023. https://www.easytechjunkie.com/what-is-a-cipher-key.htm (accessed Dec. 11, 2023).

[9]"What is the AES algorithm?," Educative: Interactive Courses for Software Developers. https://www.educative.io/answers/what-is-the-aes-algorithm

[10]"What Is AES Encryption and How Does It Work? - Simplilearn," Simplilearn.com, Jul. 27, 2021. https://www.simplilearn.com/tutorials/cryptography-tutorial/aes-encryption: :text=The