Tim Lum + Matt Sell + Christopher Ijams
2020.01.23
CSS527 - Cryptography

ASSG 1 - Substitution Boxes

ENCIPHERMENT + AVALANCHE ANALYSIS

Prepared For:
The University of Washington Bothell

CSS527 - CRYPTOGRAPHY

Winter, Q2 - 2020

Tim Lum + Matt Sell + Christopher Ijams
2020.01.23
CSS527 - Cryptography

Index:

Tim Lum + Matt Sell + Christopher Ijams
2020.01.23
CSS527 - Cryptography

SECTION 0 - PREFACE

SECTION 0.0 - ASSIGNMENT DESCRIPTION

Given a set of substitution boxes, plaintext, and keys, perform the following operations:
- Implement a procedure to encipher the plaintext according to a given specification
- Analyze avalanche effectiveness on the ciphertext
- Modify the encipherment procedure to improve the avalanche metric

SECTION 0.1 - RUBRIC

DONE?	TASK
✓	5% - Draw a chart describing the relationship between the plaintext, key, and ciphertext
✓	30% - Implement a procedure to encipher the plaintext
✓	50% - Measure avalanche before and after modifying 1 bit in each of the 10 plaintext messages
✓	15% - Suggest and implement alterations to the encryption schema

SECTION 1 - QUESTION 1

SECTION 1.0 - CIPHER RELATIONSHIP CHART

Step 1 - Configure provided substitution boxes to a linear array

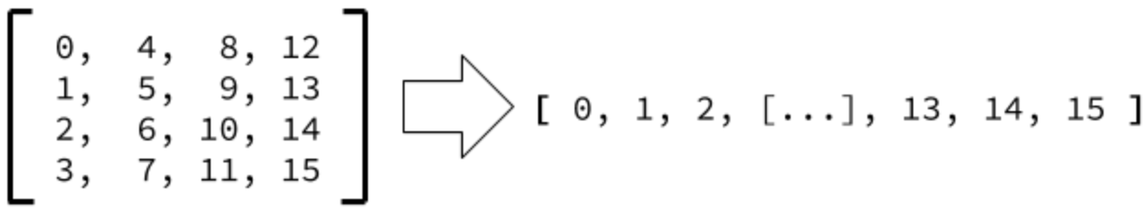

Fig. 1.0.0 - Reconfiguration of substitution boxes to a linear array with indices in range 0 (0b0000) to 15 (0b1111)

Step 2 - For every 2 byte block (a message), break into 4 bit nibbles and handle each nibble as follows to generate a pre-substitution ciphertext (psE):

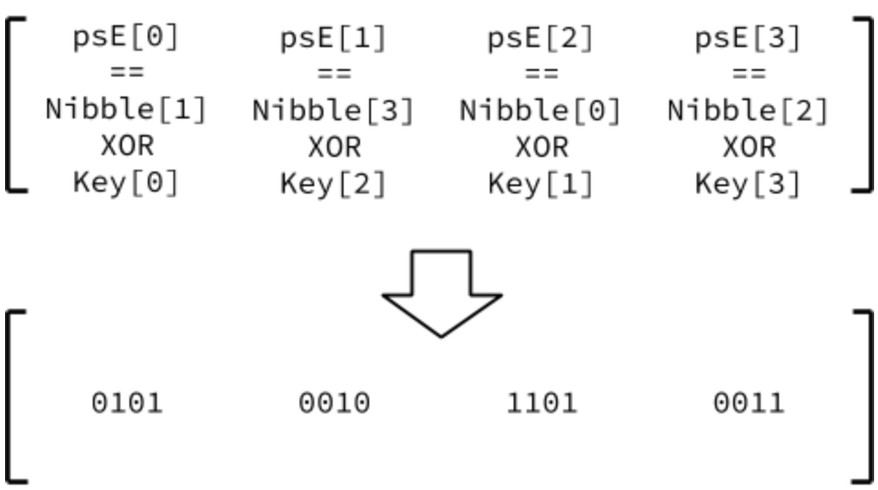

Fig. 1.0.1 - Performing the appropriate nibble XOR operations results in a pre-substitution ciphertext comprised of (4) binary sequence nibbles corresponding to integers in range 0 to 15.

Tim Lum + Matt Sell + Christopher Ijams
2020.01.23
CSS527 - Cryptography

Step 3 - Use the values represented by the pre-substitution ciphertext to perform index-based lookups from the linear substitution tables, alternating between S1 and S2.

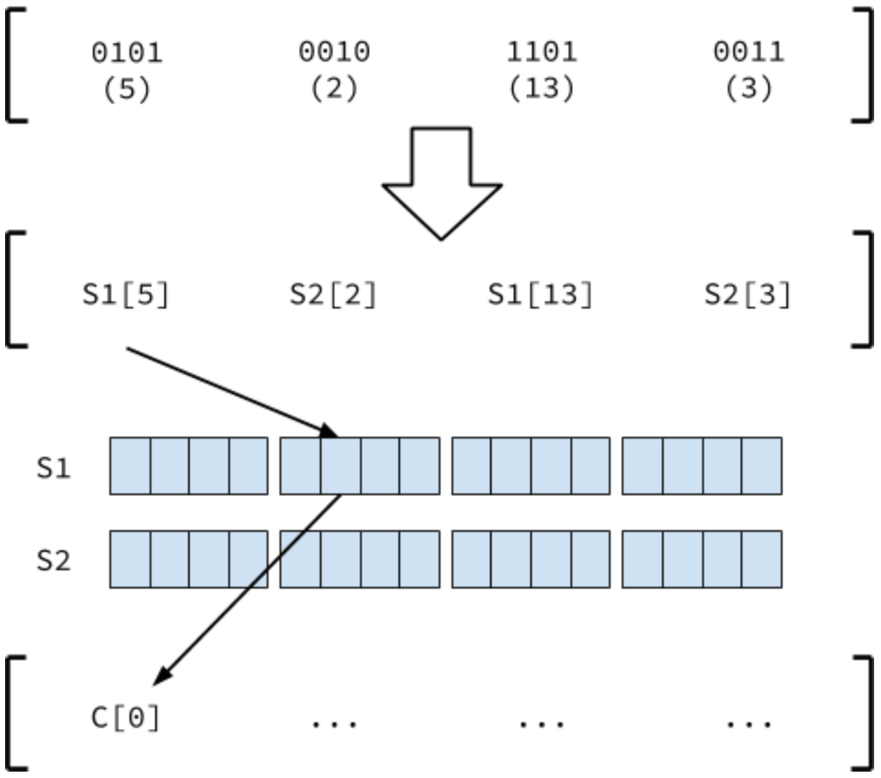

Fig. 1.0.2 - Once the pre-substitution ciphertext is acquired, the appropriate lookup in the substitution boxes S1 and S2 may be performed, resulting in the appropriate final ciphertext.

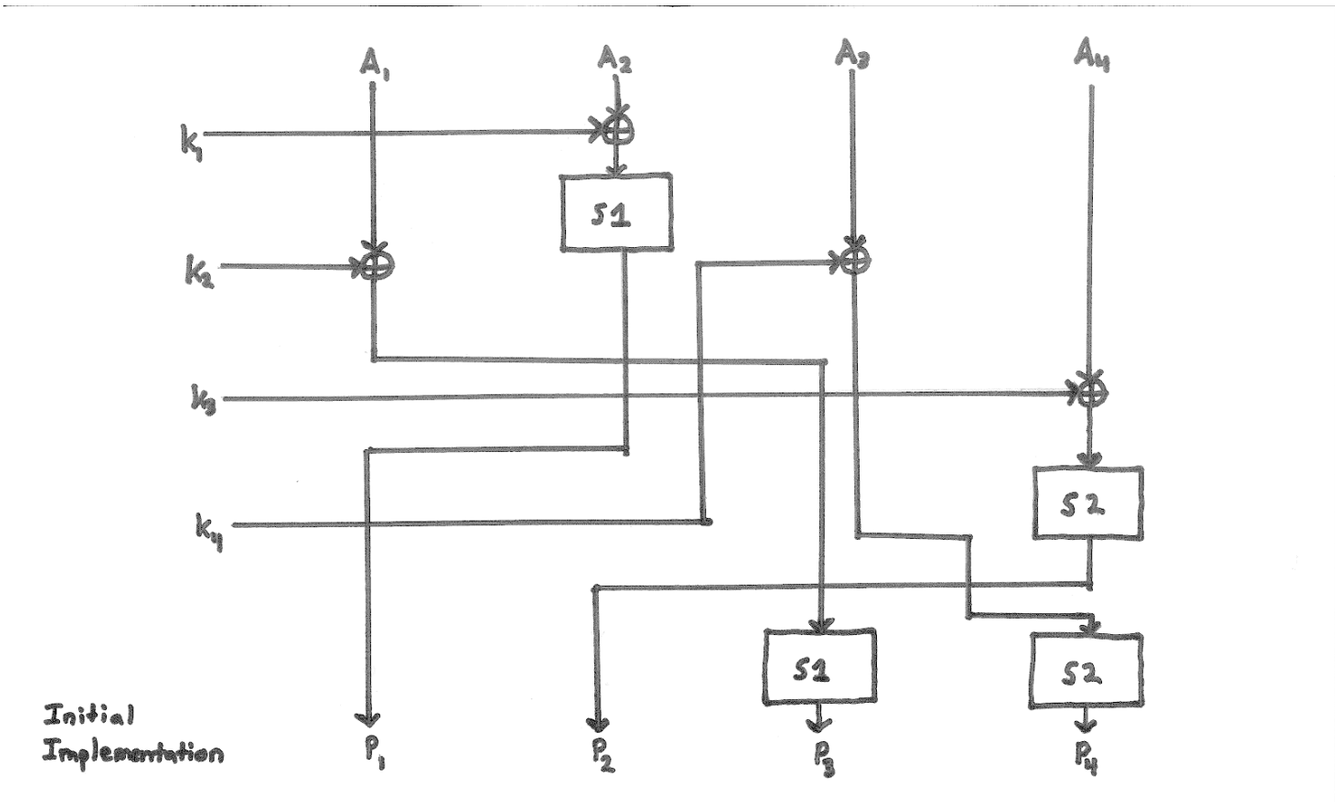

Fig. 1.0.3 - A summary view of all cryptographic operations from plaintext to ciphertext.

Tim Lum + Matt Sell + Christopher Ijams
2020.01.23
CSS527 - Cryptography

SECTION 2 - QUESTION 2

SECTION 2.0 - CIPHER IMPLEMENTATION

See [associated codebase]

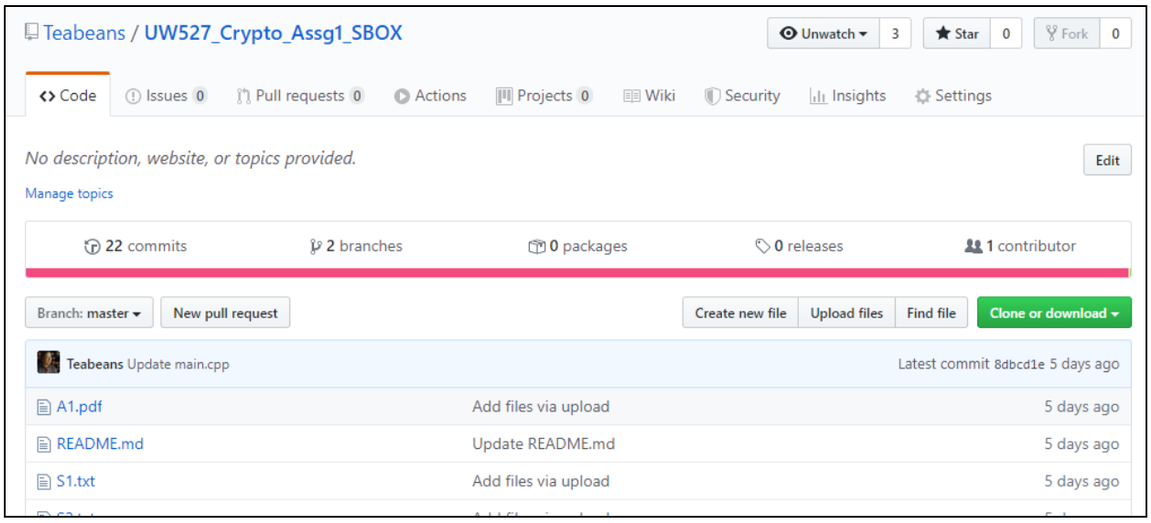

Fig 2.0.0 - Rendered ciphertext test message ("6, 0, 12, 5") from the assignment description demonstrating correctness of output.

SECTION 2.1 - CIPHERTEXT FROM KEY 1

ROW	PLAINTEXT	CIPHERTEXT
1	1001 0100 0110 0110	0111 1001 1100 1010
2	0010 1001 1100 0010	1001 1000 1111 0001
3	0101 1100 1110 0010	0000 1000 0011 0000
4	1001 1100 0010 0111	0000 0000 1100 1111
5	1011 1101 1101 1111	0011 1010 1011 0011
6	0001 1101 0001 0011	0011 0100 1001 1110
7	1110 0001 1100 0011	1100 0100 0101 0001
8	0011 1110 0000 0010	1010 1000 1000 0110
9	0101 0011 1101 1011	1011 1111 0011 0011
10	1100 0010 0111 0100	0010 0011 0111 1101

Fig 2.1.0 - Ciphertext output from key K1.

Tim Lum + Matt Sell + Christopher Ijams
2020.01.23
CSS527 - Cryptography

SECTION 2.2 - CIPHERTEXT FROM KEY 2

ROW	PLAINTEXT	CIPHERTEXT
1	1001 0100 0110 0110	1110 0011 0000 1011
2	0010 1001 1100 0010	0011 0010 0110 0000
3	0101 1100 1110 0010	0001 0010 1110 0001
4	1001 1100 0010 0111	0001 0001 0000 0110
5	1011 1101 1101 1111	1001 1011 1010 1001
6	0001 1101 0001 0011	1001 0101 0111 0111
7	1110 0001 1100 0011	1101 0101 1000 0000
8	0011 1110 0000 0010	1111 0010 0101 1111
9	0101 0011 1101 1011	0110 0110 1110 1001
10	1100 0010 0111 0100	0101 1001 1001 1100

Fig 2.2.0 - Ciphertext output from key K2.

Tim Lum + Matt Sell + Christopher Ijams
2020.01.23
CSS527 - Cryptography

SECTION 3 - QUESTION 3

SECTION 3.0 - AVALANCHE CALCULATION

See associated codebase

Alteration of 1 bit in the plaintext:

ROW	PLAINTEXT (ORIGINAL)	PLAINTEXT (ALTERED)
1	1001 0100 0110 0110	**0**001 0100 0110 0110
2	0010 1001 1100 0010	**1**010 1001 1100 0010
3	0101 1100 1110 0010	**1**101 1100 1110 0010
4	1001 1100 0010 0111	**0**001 1100 0010 0111
5	1011 1101 1101 1111	**0**011 1101 1101 1111
6	0001 1101 0001 0011	**1**001 1101 0001 0011
7	1110 0001 1100 0011	**0**110 0001 1100 0011
8	0011 1110 0000 0010	**1**011 1110 0000 0010
9	0101 0011 1101 1011	**1**101 0011 1101 1011
10	1100 0010 0111 0100	**0**100 0010 0111 0100

SECTION 3.1 - AVALANCHE RESULT

Results in alteration of 1 nibble in the output:

ROW	CIPHERTEXT (ORIGINAL)	CIPHERTEXT (ALTERED)
1	0111 1001 1100 1010	0111 1001 1**00**1 1010
2	1001 1000 1111 0001	1001 1000 **00**1**0** 0001
3	0000 1000 0011 0000	0000 1000 **110**1 0000
4	0000 0000 1100 1111	0000 0000 1**00**1 1111
5	0011 1010 1011 0011	0011 1010 10**00** 0011
6	0011 0100 1001 1110	0011 0100 1**10**0 1110
7	1100 0100 0101 0001	1100 0100 **101**0 0001
8	1010 1000 1000 0110	1010 1000 10**11** 0110
9	1011 1111 0011 0011	1011 1111 **110**1 0011
10	0010 0011 0111 1101	0010 0011 0**111** 1101

Avalanche comparison was then performed by converting the messages ([0] to [9]) to their binary string representations, then iterating along the string and counting the character differences.

```
Converting DefaultBox messages 0 to 9 to binary string...
000101111101101000111001001000011110100111000000111000001101111111001010101100111100010000

Converting ComparisonBox messages 0 to 9 to binary string...
000101110011101000111001111100011110100110010000111000000011111111001010100000111100010010
```

This procedure results in a relatively low avalanche score for the original encryption schema of about **16.25%**, with the most striking feature of the original and altered output being the 1-to-1 mapping of the nibble which received an altered bit (in the above example, the 0th index of the plaintext) and the effect in the output (above, the 2nd index of the ciphertext).

```
Comparing original ciphertext binary string to ComparisonBox binary string...
Comparing seq1 (160) against seq2 (160)...
16.25% different.
teabean@DESKTOP-E9L7QKR:/mnt/c/Users/twhlu/desktop/css527/assg1$
```

Tim Lum + Matt Sell + Christopher Ijams
2020.01.23
CSS527 - Cryptography

SECTION 4 - QUESTION 4

SECTION 4.0 - AVALANCHE IMPROVEMENT

Several approaches to improving the default avalanche were investigated including:

S-BOX TUNING:
Based on discussion regarding the engineering of the DES and AES substitution boxes, efforts were made to explore how different S-Box values could affect the avalanche calculation given the same set of keys and plaintext.

A test application was written based on the original implementation that used “Monte Carlo”-style simulation to generate random values for the substitution boxes, with each iteration checking for an improvement to the avalanche result. After several hours of running this evolutionary algorithm, an improvement of a little over 2% (final avalanche +6.93%) was achieved by using different S-Box values.

It was observed during testing that the configuration of the S-Boxes had a direct result on whether some characters in messages would collide or converge, though it bears mention that this method could only achieve a theoretical maximum of 25% avalanche where S-Box values always result in a complete inversion of the default bit state of the altered nibble. This limit is imposed by the fact that the default encryption schema imposes a 1-to-1 mapping between an altered character and an output character.

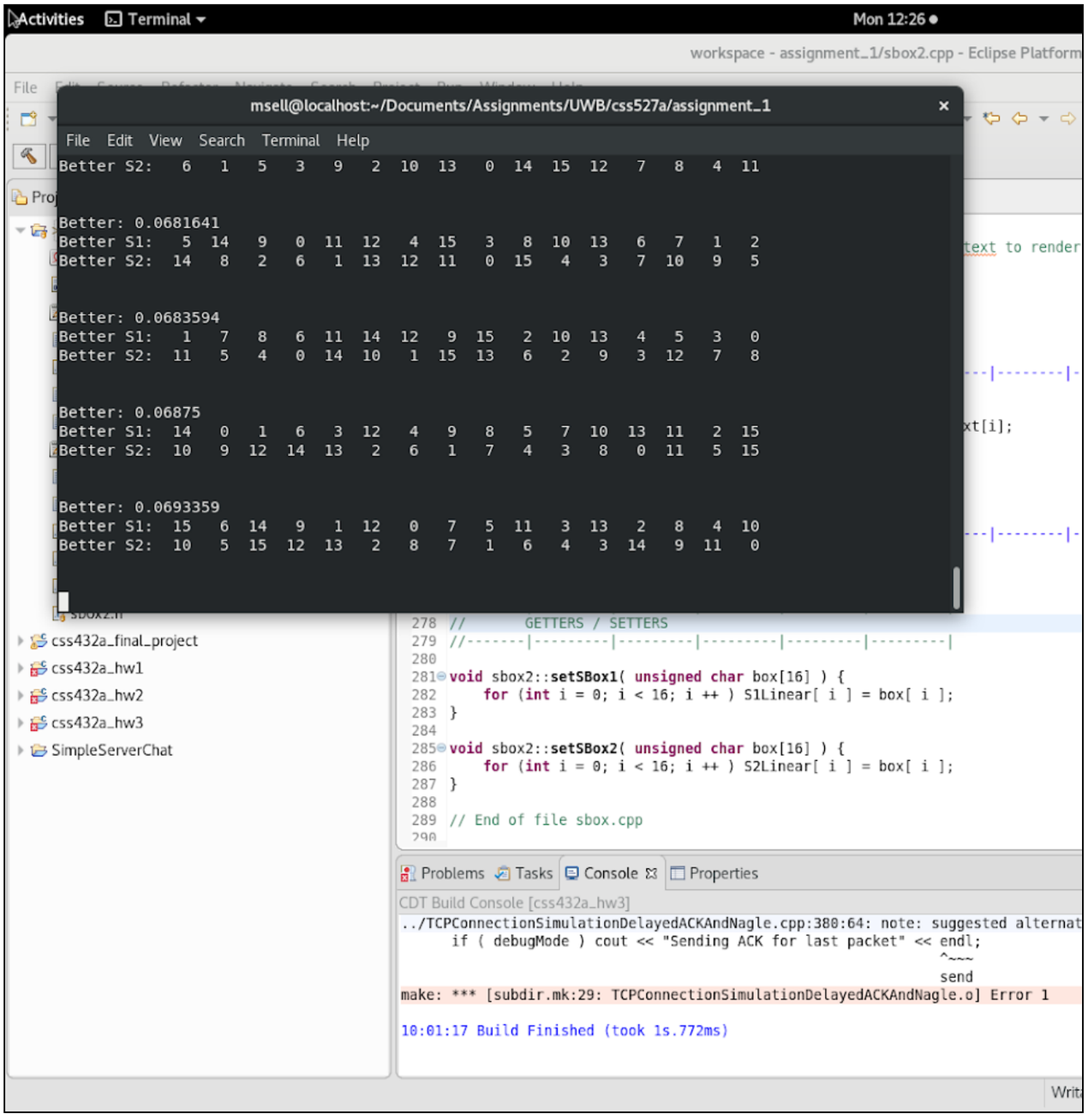

Fig. 4.0.0 - Usage of a Monte-Carlo simulation to arrive at an empirically better SBox configuration.

Tim Lum + Matt Sell + Christopher Ijams
2020.01.23
CSS527 - Cryptography

In order to escape the 12.5% limit of the original encryption schema, the altered nibble must affect more than just one character in the output. Ideally, it should affect all of them. Three strategies were pursued to achieve this effect, including:

ROUNDS - Repetitions of the encipherment schema
DIFFUSION - Wherein the effect of the changed bit is applied to adjacent characters
ROTATION - Wherein the ciphertext fed to the next round is shifted in a non-linear but deterministic manner.

After implementation, these changes result in a **+32.5%** avalanche improvement (48.75%), with alterations to the number of rounds causing fluctuations of a few percentage points.

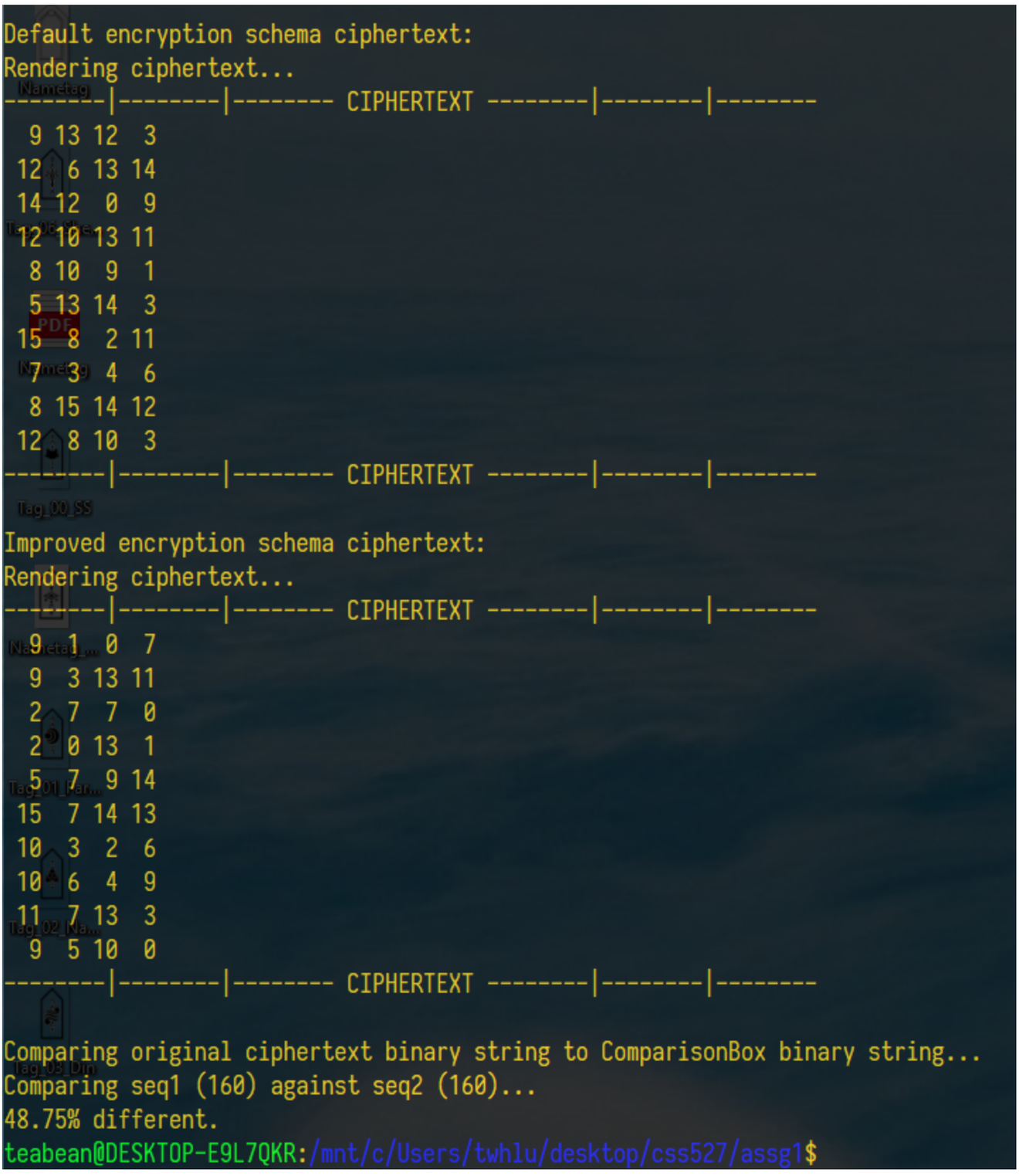

Fig. 4.0.1 - Results after the altered nibble is repeatedly diffused to adjacent characters and the ciphertext re-processed.

Tim Lum + Matt Sell + Christopher Ijams
2020.01.23
CSS527 - Cryptography

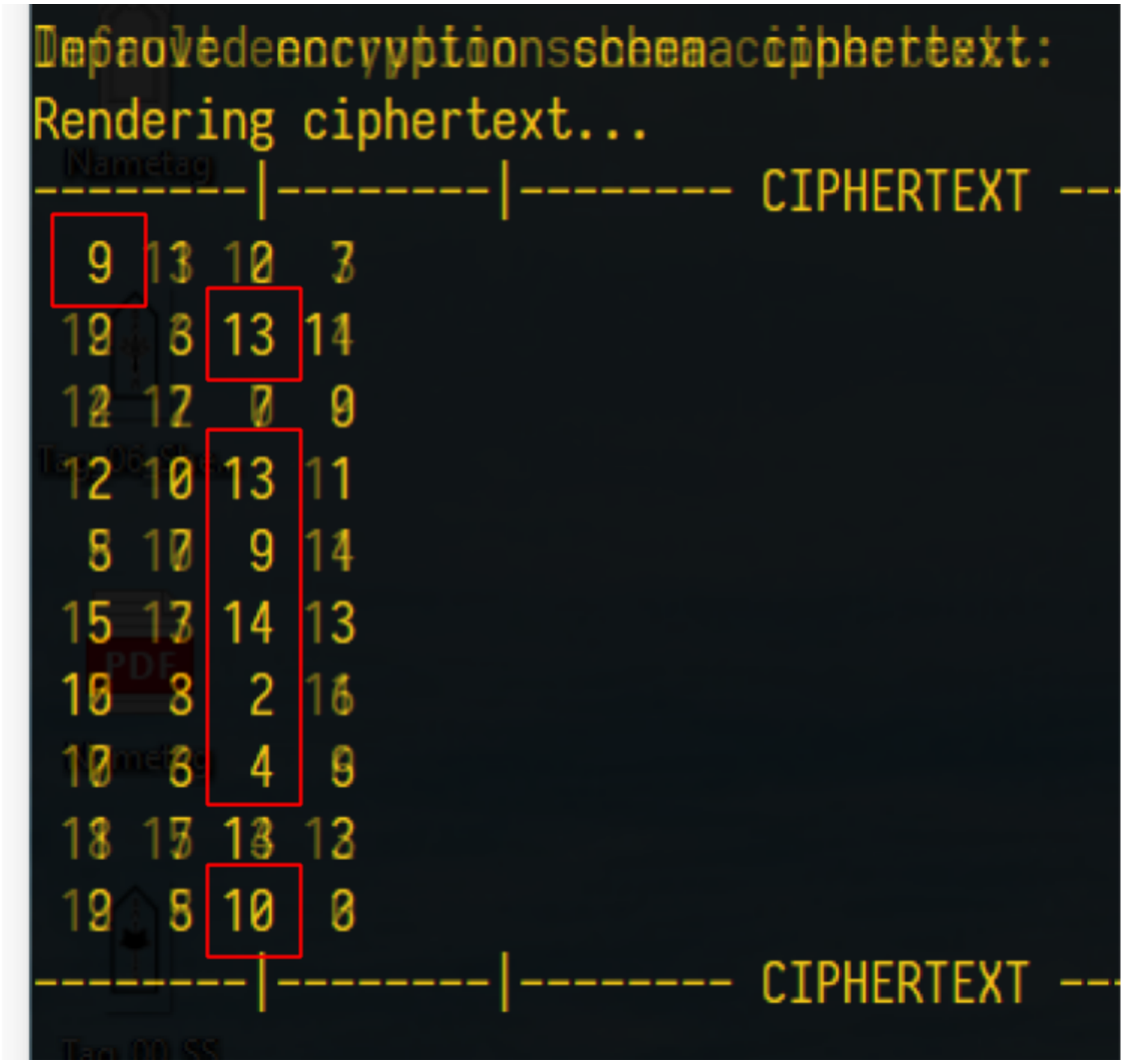

Fig 4.0.2- Closer comparison of the ciphertext outputs reveals a statistically unlikely and inconsistent collision behavior in encryption outputs. This is a bug under investigation.

ROW	CIPHERTEXT (DEFAULT)	CIPHERTEXT (IMPROVED)
1	1001 1101 1100 0011	1001 0001 0000 0111
2	1100 0110 1101 1110	1001 0011 1101 1011
3	1110 1100 0000 1001	0010 0111 0111 0000
4	1100 1010 1101 1011	0010 0000 1101 0001
5	1000 1010 1001 0001	0101 0111 1001 1110
6	0101 1101 1110 0011	1111 0111 1110 1101
7	1111 1000 0010 1011	1010 0011 0010 0110
8	0111 0011 0100 0110	1010 0110 0100 1001
9	1000 1111 1110 1100	1011 0111 1101 0011
10	1100 1000 1010 0011	1001 0101 1010 0000

Tim Lum + Matt Sell + Christopher Ijams
2020.01.23
CSS527 - Cryptography

SECTION 4.1 - CODE ALTERATION

The following tables list the original and new values for the substitution boxes:

ROW	SBOX 1 (ORIGINAL)	SBOX 1 (ALTERED)
1	1111 1010 0010 0101	1111 0001 0101 0010
2	1000 0100 1011 0110	0110 1100 1011 1000
3	0001 0000 1110 0111	1110 0000 0011 0100
4	1001 0011 1100 1101	1001 0111 1101 1010

ROW	SBOX 2 (ORIGINAL)	SBOX 2 (ALTERED)
1	0100 0000 1111 1010	1010 1101 0001 1110
2	1000 1001 0111 1101	0101 0010 0110 1001
3	0101 0001 0110 1011	1111 1000 0100 1011
4	0010 0011 1110 1100	1100 0111 0011 0000

Further, relevant code controlling alterations to the default encryption scheme are as follows:

DEFAULT ENCRYPTION

```
   // Implement the cipher to find the appropriate indices
  unsigned char cipherIndex[4];
  cipherIndex[0] = plaintextRow[1] ^ key[0];
  cipherIndex[1] = plaintextRow[3] ^ key[2];
  cipherIndex[2] = plaintextRow[0] ^ key[1];
  cipherIndex[3] = plaintextRow[2] ^ key[3];

  // Read out from S1 and S2 at the appropriate locations to find the
cipher character
  unsigned char cipherRow[4];
  cipherRow[0] = S1Linear[cipherIndex[0]];
  cipherRow[1] = S2Linear[cipherIndex[1]];
  cipherRow[2] = S1Linear[cipherIndex[2]];
  cipherRow[3] = S2Linear[cipherIndex[3]];
```

IMPROVED ENCRYPTION

```
 for( int round = 0 ; round < 15 ; round++ ) {
   // Implement the cipher to find the appropriate indices
   cipherIndex[0] = plaintextRow[1] ^ key[0];
   cipherIndex[1] = plaintextRow[3] ^ key[2];
   cipherIndex[2] = plaintextRow[0] ^ key[1];
   cipherIndex[3] = plaintextRow[2] ^ key[3];

   // Read out from S1 and S2 at the appropriate locations to find the
cipher character
   int sum = cipherRow[0] + cipherRow[1] + cipherRow[2] + cipherRow[3];
   if( sum % 2 == 0 ) {
     cipherRow[0] = this->S1Linear[cipherIndex[0]];
     cipherRow[1] = this->S2Linear[cipherIndex[1]];
     cipherRow[2] = this->S1Linear[cipherIndex[2]];
     cipherRow[3] = this->S2Linear[cipherIndex[3]];
   }
   else {
     cipherRow[0] = this->S2Linear[cipherIndex[0]];
     cipherRow[1] = this->S1Linear[cipherIndex[1]];
     cipherRow[2] = this->S2Linear[cipherIndex[2]];
     cipherRow[3] = this->S1Linear[cipherIndex[3]];
   }

   // Diffuse the effect to multiple characters of the cipher row
   unsigned char tempDiffuse1 = cipherRow[0];
   unsigned char tempDiffuse2 = cipherRow[1];
   cipherRow[0] = (cipherRow[0] + cipherRow[1] + cipherRow[2]) % 16;
   cipherRow[1] = (cipherRow[1] + cipherRow[2] + cipherRow[3]) % 16;
   cipherRow[2] = (cipherRow[2] + cipherRow[3] + tempDiffuse1) % 16;
   cipherRow[3] = (cipherRow[3] + tempDiffuse1 + tempDiffuse2) % 16;

   // Rotate L or R
   if( sum % 2 == 0 ) {
     unsigned char temp = cipherRow[0];
     cipherRow[0] = cipherRow[1];
     cipherRow[1] = cipherRow[2];
     cipherRow[2] = cipherRow[3];
     cipherRow[3] = temp;
   }
   else {
     unsigned char temp = cipherRow[0];
     cipherRow[0] = cipherRow[3];
     cipherRow[3] = cipherRow[2];
     cipherRow[2] = cipherRow[1];
     cipherRow[1] = temp;
   }
 } // End of for loop - Rounds over
```

Tim Lum + Matt Sell + Christopher Ijams
2020.01.23
CSS527 - Cryptography

APPENDIX

APPENDIX 0.0 - Assignment Description

Given the following S-Boxes:
S1= [15 10 2 5
8 4 11 6
1 0 14 7
9 3 12 13];

S2= [4 0 15 10
8 9 7 13
5 1 6 11
2 3 14 12];

Implement the following 16 bit cipher:

Plain text: P = [a1 a2 a3 a4] where a1..a4 are 4 bits each
Key: K = [k1 k2 k3 k4] where k1..k4 are 4 bits each
Cipher text: C = E(p) = [S1(a2⊕k1) S2(a4⊕k3) S1(a1⊕k2) S2(a3⊕k4)]
Example: P = [1000 1100 1101 0110], K = [0001 0011 0010 1111]
C = [S1(1101) S2(0100) S1(1011) S2(0010] = [6 0 12 5]
= [0110 0000 1100 0101]

1. Draw a chart showing the relation between P, C, and K according to this cipher [5%].

2. Implement the above cipher and calculate the cipher text for the plaintext provided in Appendix I using the two keys provided in Appendix II. [30%]

3. Measure the avalanche effect for the encryption algorithm using the provided plaintexts and keys. Change 1 bit in the input and calculate the percentage of how many bits are changed in the cipher text. Repeat this for the provided 10 plaintext inputs; this will give 10 x 16 x 2 rounds. Calculate the average avalanche effect. [50%]

4. Suggest a change to the encryption algorithm to enhance the avalanche effect. Repeat 3 and comment on your findings. [15%]

Appendix I: Test Plain Text
1001 0100 0110 0110
0010 1001 1100 0010
0101 1100 1110 0010
1001 1100 0010 0111
1011 1101 1101 1111
0001 1101 0001 0011
1110 0001 1100 0011
0011 1110 0000 0010
0101 0011 1101 1011
1100 0010 0111 0100

Appendix II: Test Keys
1010 0010 0011 1010
1110 1111 0001 1000