# Portfolio Part 1
**CSI2108**

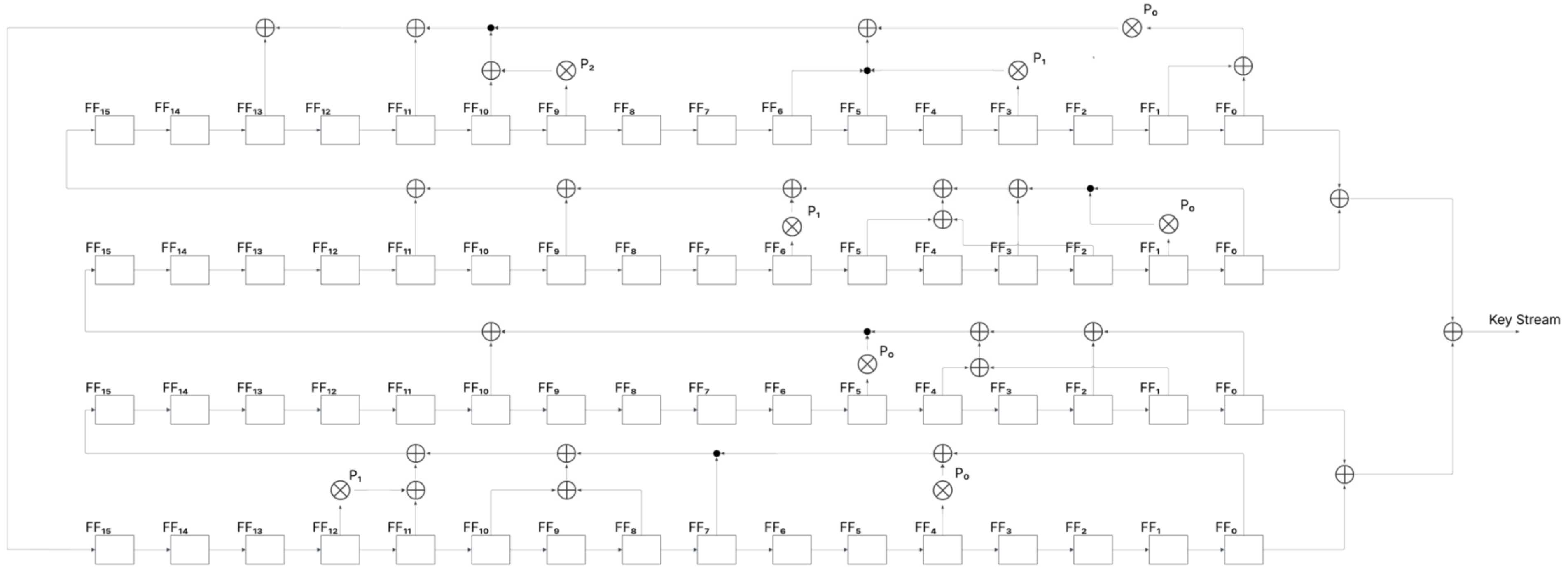
Jordan Farrow (10653054)
Edith Cowan University, Joondalup
Dr. Julia Collins

Table of Contents

## 1. Stream Cipher Design

**Figure 1**

*Pseudorandom Number Generator Structure*


P₀
P₂
P₁
FF₁₅ FF₁₄ FF₁₃ FF₁₂ FF₁₁ FF₁₀ FF₉ FF₈ FF₇ FF₆ FF₅ FF₄ FF₃ FF₂ FF₁ FF₀
P₁
P₀
FF₁₅ FF₁₄ FF₁₃ FF₁₂ FF₁₁ FF₁₀ FF₉ FF₈ FF₇ FF₆ FF₅ FF₄ FF₃ FF₂ FF₁ FF₀
Key Stream
P₀
FF₁₅ FF₁₄ FF₁₃ FF₁₂ FF₁₁ FF₁₀ FF₉ FF₈ FF₇ FF₆ FF₅ FF₄ FF₃ FF₂ FF₁ FF₀
P₁
P₀
FF₁₅ FF₁₄ FF₁₃ FF₁₂ FF₁₁ FF₁₀ FF₉ FF₈ FF₇ FF₆ FF₅ FF₄ FF₃ FF₂ FF₁ FF₀

*Note.* ⊕ represents an XOR operation being performed, ⊗ represents the multiplication between flip and switch assigned switch value, ● represents multiple bits meeting on a line which travel to the same XOR position.

## 1.1. Pseudorandom Number Generator (PRNG) Design

Seen in Figure 1, the PRNG consists of 4 separate linear feedback shift registers (LFSR), each generating the next bit in another's sequence. This is seen achieved by taking the LFSR value in flip-flop $(FF)_0$ and performing additions with the FF bits connected through XOR gates seen in the figure. The total is calculated modulo 2 resulting in a 1 or 0 value. The two states of a bit, performing an XOR operation. In addition, some FF are seen in the figure connected to switch gates, allowing closed switch values to be excluded (where p value = 0) and open switch values to be included (where p value = 1). Each LFSR has a unique set of gates and switches, using different formulas for initialising the next $FF_{15}$ bit in a another's sequence (where P = switch value, $FF_n$ = flip-flop value):

LFSR1:

$$FF_{15}\ (LFSR4) = P_0(FF_0 + FF_1) + (FF_3 + P_1) + FF_5 + FF_6 + (FF_9P_2) + FF_{10} + FF_{11} + FF_{12}$$

LFSR2:

$$FF_{15}\ (LFSR1) = FF_0 + (FF_1P_0) + FF_3 + FF_2 + FF_5 + (FF_6P_1) + FF_9 + FF_{11}$$

LFSR3:

$$FF_{15}\ (LFSR2) = FF_0 + FF_2 + FF_1 + FF_4 + (FF_5P_0) + FF_{10}$$

LFSR4:

$$FF_{15}\ (LFSR3) = (FF_4P_0) + FF_7 + FF_8 + FF_{10} + FF_{11} + (FF_{12}P_1)$$

In each LFSR there are 16 FFs, ranging from $FF_0$ to $FF_{15}$. As each clock cycle occurs and each $FF_0$ bit is taken away, all other FF bits move a placement to the right, occupying the next FF. This results in $FF_{15}$ having no bit within it, becoming occupied with the next bit generated by another LFSR. To begin the PRNG, values for each FF and switch must be chosen. Table 1 shows the initial bit values the bank can use in generating the next bit, with an example of the first 4.

**Table 1**

Bit Generation of Each LFSR Based Upon Initial Bit Choice

| Clock Cycle | LFSR | FF15 | FF14 | FF13 | FF12 | FF11 | FF10 | FF9 | FF8 | FF7 | FF6 | FF5 | FF4 | FF3 | FF2 | FF1 | FF0 | Result of each LFSR formula |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | LFSR1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| | LFSR2 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| | LFSR3 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| | LFSR4 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | LFSR1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| | LFSR2 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| | LFSR3 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| | LFSR4 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 2 | LFSR1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| | LFSR2 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| | LFSR3 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| | LFSR4 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 3 | LFSR1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| | LFSR2 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| | LFSR3 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| | LFSR4 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 4 | LFSR1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| | LFSR2 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| | LFSR3 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| | LFSR4 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

## 1.2. Generating the Keystream

Within each clock cycle, $FF_0$ bits from LFSR 1 and 2 are added modulo 2. This also occurs with LFSR 3 and 4, resulting in two bits. These bits then follow the same process, resulting in 1 final bit contributing to the keystream. The number of keystream bits and next sequence bits generated is dependent on the length of the binary plaintext to be encrypted. Table 2 shows the first 4 clock cycles and the keystream bit it produces.

**Table 2**

*Initial Bit Values of Each LFSR, Next Bit Generation and Next Keystream Bits*

| Clock Cycle | LFSR | FF15 | FF14 | FF13 | FF12 | FF11 | FF10 | FF9 | FF8 | FF7 | FF6 | FF5 | FF4 | FF3 | FF2 | FF1 | FF0 | Result of each LFSR formula | LFSR addition (before keystream) | LFSR addition 2 (keystream) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | LFSR1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
|  | LFSR2 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |  |  |
|  | LFSR3 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |  |
|  | LFSR4 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |  |  |
| 2 | LFSR1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
|  | LFSR2 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |  |  |
|  | LFSR3 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |  |
|  | LFSR4 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |  |  |
| 3 | LFSR1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
|  | LFSR2 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |  |  |
|  | LFSR3 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |  |
|  | LFSR4 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |  |  |
| 4 | LFSR1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
|  | LFSR2 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |  |  |
|  | LFSR3 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |  |
|  | LFSR4 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |  |  |
| 5 | LFSR1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
|  | LFSR2 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |  |  |
|  | LFSR3 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |  |
|  | LFSR4 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |  |  |

## 1.3. Encryption of Plaintext X

Plaintext encryption occurs through an XOR operation between each bit of the key stream and binary plaintext, producing a bit dependent of the outcome. Message X in binary is 576 bits in length (including whitespace) therefore, requires 576 clock cycles to generate a keystream where each plaintext bit has a keystream to perform the encryption. With the first keystream bits ‘01111’, the binary plaintext bits ‘01010’ produces the ciphertext ‘00101’ seen below:

1st bit:

$$0 + 0 = 0 \equiv 0 \pmod{2}$$

2nd bit:

$$1 + 1 = 2 \equiv 0 \pmod{2}$$

3rd bit:

$$1 + 0 = 1 \equiv 1 \pmod{2}$$

4th bit:

$$1 + 1 = 2 \equiv 0 \pmod 2$$

5th bit:

$$1 + 0 = 1 \equiv 1 \pmod 2$$

## 1.4. Cryptographic Properties

### *1.4.1. Periodicity*

The periodicity for each LFSR is $2^{16}$ - 1 (65,535) bit sequences before repetition occurs. As a result, no repetitions will occur for message X which only requires 576 LFSR sequences, increasing the ‘randomness’ of the PRNG. Previously 8 FF per LFSR were considered however, only provided a periodicity of $2^8 – 1$ (255) without considering closed switches. The number of bits in X is greater than the periodicity of 8 FFs, causing repetitions to occur and decreasing the ‘randomness’ of the PRNG as patterns emerge. 16 FFs allows the bank to increase the length of their message to at least 2,000 ascii characters (16,383 bits with 8 bits per ascii) without causing repetitions, whilst allowing switch gates to be used despite their disadvantage of sacrificing periodicity.

### *1.4.2. Predictability*

Although these switches decrease periodicity, it also decreases the ability for an attacker to predict and determine the key of the PRNG (initial FF and switch values). Due to the number of FFs and switches, attackers will struggle to predict and determine the correct combination of initial bits and switch values for generating the keystream and next sequence of bits. The PRNG has a key space of $(2^{64} – 1)x2^8$ ($2^{64}$ - 1 possible initial sequences and $2^8$ possible switch values) = $4.72x10^{21}$ possible initialisation bits that can be used for generating keystreams. As a result, for an attacker to generate the correct keystream and LFSR sequences, it requires 72 correctly placed bit values within a key space of 4.72x1021 possible combinations.

### *1.4.3. Non-linearity*

The inclusion of switches has caused the generation of the next bit in the sequence to become non-linear, reducing its susceptibility to cryptanalysis attacks and predictability. In addition, the circular nature where each LFSR relies on the bit of another produces nonlinearity. This is because rather than always relying on the bit created from its initial sequence, each LFSR slowly relies on the other LFSRs and their current bit sequences which then do not rely on their initial sequence as they receive bits produced from other LFSRs.

### *1.4.4. Practicality*

Due to one round of encryption occurring, messages are quickly encrypted. This will allow the bank to output a greater number of encrypted messages to the receiving bank. In addition, due to the efficiency of stream ciphers, the bank will not have to consider using high resource hardware or considering the impact its implementation will have on current performance. In addition, the bank can create the keystream once and encrypt multiple messages off that stored keystream, removing the need to run the PRNG for each message as each PRNG use would produce the same keystream. This would increase the speed and efficiency of encryption as the XOR operations between keystream and plaintext would be the only step in encryption. If the initial bit values change the PRNG would then be used, producing a keystream to replace the other.

## 1.5 Decryption of the Ciphertext

In order to decrypt the ciphertext, the receiving bank will require all initial FF and switch bit values to create the same keystream used to encrypt the ciphertext. With the keystream the bank would XOR each bit from the ciphertext and stream, resulting in the binary of the plaintext. The binary would then be converted into its decimal number and converted into its ascii equivalence. An example of the first three characters are as follows:

**1st character:**

Encrypted character bits = $00101011_2$

Aligning keystream bits = $01111111_2$

Decrypt with XOR:

$$00101011_2 \oplus 01111111_2 = 01010100_2$$

Convert:

$$01010100_2 = 84_{10} = T$$

**2nd character:**

Encrypted character bits = $01111010_2$

Aligning keystream bits = $00001000_2$

Decrypt with XOR:

$$01111010_2 \oplus 00001000_2 = 01110010_2$$

Convert:

$01110010_2 = 114_{10} = \mathrm{r}$

**3rd character:**

Encrypted character bits = $01011111_2$

Aligning keystream bits = $00111110_2$

Decrypt with XOR:

$01011111_2 \oplus 00111110_2 = 01100001_2$

Convert:

$01100001_2 = 97_{10} = \mathrm{a}$

## 1.6. Improvements

### *1.6.1 Increase the Number of Flip-Flops per LFSR*

By increasing the number of FFs within each LFSR, it will increase the stream ciphers periodicity and number of possible bit combinations, reducing predictability and increasing resistance against brute-force attacks. In comparison with the Trivium stream cipher with 288 total FFs spread across 3 interconnected LFSRs, where an LFSR has 93, another 84, and the final containing 111 FFs (Potestad-Ordóñez et al., 2020; Robshaw & Billet, 2008). This provides Trivium with a much greater periodicity of $2^{84} - 1$ bit sequences before repetitions occur, compared to this stream ciphers worst case $2^{13} - 1$ (where LFSR 1 has all switches closed). Although Trivium does not provide all 288 FFs with initial bit values, it still utilises more than the design, using 160 FFs initialised by an 80-bit key and 80-bit number used only once (nonce) (Robshaw & Billet, 2008). This is seen providing Trivium $2^{160} - 1$ possible seed combination to initialise with, compared to this ciphers $2^{64} - 1$. As a result, Trivium has a greater number of initial bit combinations, increasing the number of values an attacker would need to implement in a brute-force attack; where a ciphertext created with the designed stream cipher taking a few days to brute-force, Trivium would remain encrypted for several decades.

### *1.6.2 Use of Nonce Initial Bit Sequences*

Discussed prior, Trivium initialises its LFSRs with an 80-bit key and 80-bit nonce (Robshaw & Billet, 2008). Since the nonce sequences are used for encrypting a single message, it means same or similar messages will encrypt using a different sequence, producing different ciphertexts. This would ensure an attacker is unable to analyse repetitive ciphertexts to determine and predict patterns generated by the cipher.

### *1.6.3 Lack of Cryptanalysis Before Mainstream Implementation*

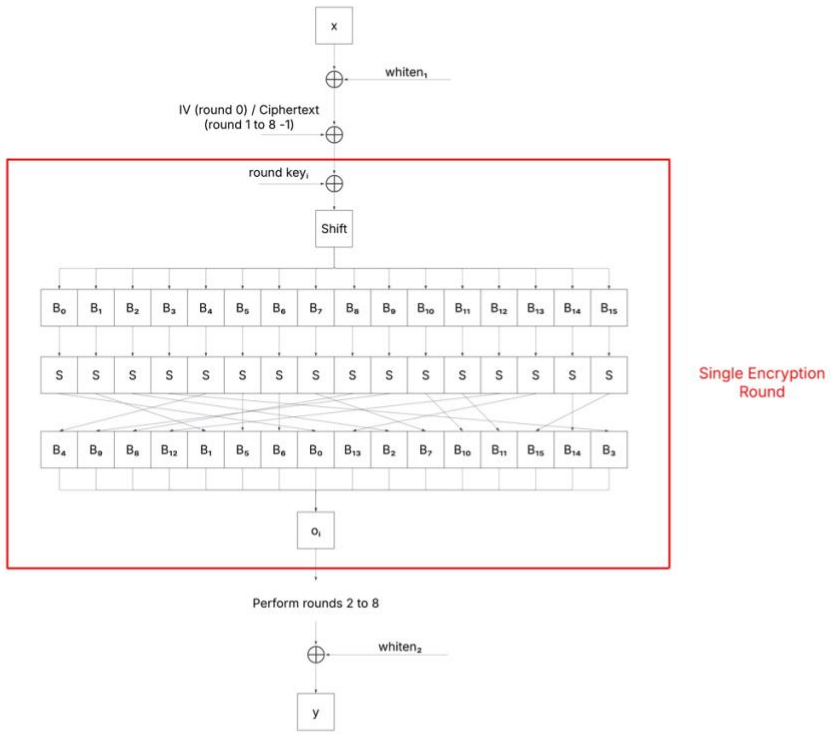
The disadvantage of implementing this cipher over another for instance, Trivium, is the lack of cryptanalysis which has been performed. Trivium is well-known and has existed for an extensive period of time therefore, has had the opportunity for weaknesses to be measured and noted. With the bank possibly implementing this cipher without a proper understanding of its resistance against cryptanalysis, the cipher could contain zero-day exploits that will negatively impact the bank and its reputation (IBM, n.d.; Australian Signals Directorate, n.d.).

# 2. Block Cipher Design

## 2.1 Elements of Block Cipher Structure

**Figure 2**

*Block Cipher Structure*


x
whiten$_1$
IV (round 0) / Ciphertext (round 1 to 8 -1)
round key$_i$
Shift
$B_0$ $B_1$ $B_2$ $B_3$ $B_4$ $B_5$ $B_6$ $B_7$ $B_8$ $B_9$ $B_{10}$ $B_{11}$ $B_{12}$ $B_{13}$ $B_{14}$ $B_{15}$
S S S S S S S S S S S S S S S S
$B_4$ $B_9$ $B_8$ $B_{12}$ $B_1$ $B_5$ $B_6$ $B_0$ $B_{13}$ $B_2$ $B_7$ $B_{10}$ $B_{11}$ $B_{15}$ $B_{14}$ $B_3$
$o_i$
Single Encryption Round
Perform rounds 2 to 8
whiten$_2$
y

### *2.1.1. Pre-Encryption Rounds*

Before a block undergoes each round of encryption, XOR operations with the plaintext are performed with the initialisation vector (IV) in round 1 or ciphertext in rounds 2 to 8 and first half of the 128-bit initial key through key whitening. These elements are discussed in section 2.3.

### *2.1.2. Round of Encryption*

Beginning in each round of encryption, an XOR operation occurs between the plaintext and 64-bit round key generated through key scheduling (discussed in 2.3.), causing confusion since the plaintext is altered with binary that is unknown to the attacker, obscuring the plaintext. This is followed with a permutation of the block by shifting all bits right a number of times depending on the current round. Table 3 notes the number of bit shifts that occur each round.

**Table 3**

*Number of Right Bit Shifts in Each Round*

| Round | Number of shifts per bit |
|---|---|
| 1,2,3,4,6 | 1 bit |
| 5 | 10 bits |
| 7 | 2 bits |
| 8 | 5 bits |

Shifting allows the bits for each ascii character to spread across the smaller blocks created in the next step, further hiding the plaintext through diffusion. Next, the block is split into 16 4-bit blocks. Seen in table 4, each block is substituted depending on its bit combination for another.

**Table 4**

*Bit Combinations and its Substitutions for S*

| Original Bit Combination | Substituted Bit Combination |
|---|---|
| 0000 | 0110 |
| 0001 | 0011 |
| 0010 | 0111 |
| 0011 | 0010 |
| 0100 | 1001 |
| 0101 | 0001 |
| 0110 | 1010 |
| 0111 | 1100 |
| 1000 | 0101 |
| 1001 | 0100 |
| 1010 | 1110 |
| 1011 | 1111 |
| 1100 | 0000 |
| 1101 | 1011 |
| 1110 | 1000 |
| 1111 | 1101 |

This causes confusion as all bits change, removing prior relations to the plaintext which remained. The substituted blocks are then permuted in the order (where || = concatenation:

$B_4 \| B_9 \| B_8 \| B_{12} \| B_1 \| B_5 \| B_6 \| B_0 \| B_{13} \| B_2 \| B_7 \| B_{10} \| B_{11} \| B_{15} \| B_{14} \| B_3$

This reforms the 64-bit block in a different order, continuing to spread the plaintext around the ciphertext causing greater diffusion. With the first round of encryption completed, the 64-bit block is used as the input for the next encryption round until all 8 rounds are completed.

#### *2.1.3. Post-Encryption Rounds*

After all rounds of encryption are performed, a final key whiten with the second half of the initial key occurs, resulting in the ciphertext of the block.

#### *2.1.4. Choice of Rounds*

8 rounds of encryption were chosen for providing strong ciphertexts and fast execution. Increasing the number of rounds did not provide a substantial increase in security over the additional time it would take, with message X requiring 80 rounds (8 rounds per block and the 8 key schedule rounds). With banks performing 2.2 million daily transactions, a compromise in the number of rounds is required to meet the speeds a bank requires; Assuming each transaction is the length of message X, the bank would be performing $1.584 \times 10^8$ encryption rounds daily (Reserve Bank of Australia, 2022).

### 2.2. Mode of Operation

The block cipher utilises cipher block chaining (CBC) for encrypting messages which exceed the 64-bit limit requiring the encryption of multiple blocks. Cipher block chaining XOR's each plaintext block (exception with block 0) with the ciphertext block before it. This creates a 'chain' effect where blocks control the encryption/decryption of another. For the first block of the message, an initialisation vector (IV) is used in place of a ciphertext. The IV is a 64-bit randomly generated nonce. Although the IV and ciphertext are known the key is not therefore, the plaintext remains secure.

#### *2.2.1 Advantages*

The IV ensures identical plaintexts produce different ciphertext since each bit is XOR'd with a unique bit combination. In addition, by requiring the ciphertext of another block for encryption, it prevents an attacker from substituting blocks with their own which were not encrypted as part of the chain.

### *2.2.2 Disadvantages*

If a block is substituted and the 'chain' is broken it causes the blocks following to decrypt incorrectly, resulting in partially correct decrypted plaintext. Furthermore, if a single bit is modified by noise/traffic, it changes the block's encrypted value causing the 'chain' to also break and have the same impact. Finally, it slows the encryption process since a block must wait for its predecessor to be encrypted before beginning its encryption, removing the ability to parallelize the encryption of a message's blocks.

## 2.3. Discussion of Additional Features

### *2.3.1. Key Schedule*

To increase the security of the block cipher, key scheduling was implemented where an initial key is used to generate a round key for each encryption round (Knudsen & Robshaw, 2011). This results in each round performing its XOR operation with a different key, producing confusion since each round has its block obscured differently, resulting in different ciphertext outputs in each round.

**Figure 3**

*Block Cipher Key Schedule Structure*

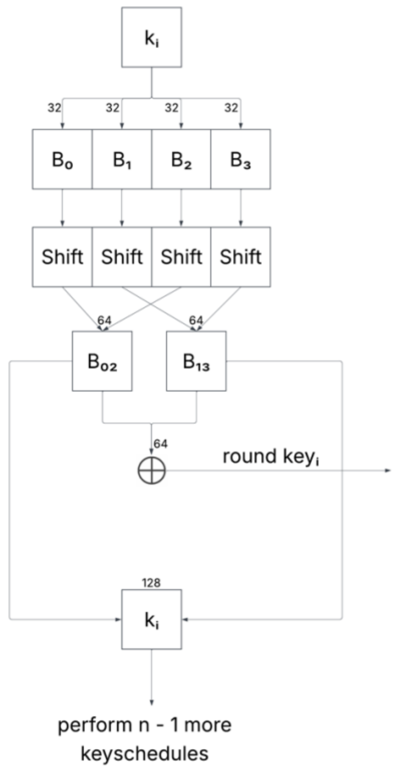


Figure 3 shows the structure of the key schedule. It begins with a 128-bit initial key being split into 4 32-bit blocks. The blocks are then shifted to the right performing the same number of shifts each round as seen in figure 2. Next, blocks 0 and 2 are concatenated, in addition to blocks 1 and 3, creating 2 64-bit blocks. An XOR operation is performed with the blocks, creating the round key to be used in a single round of encryption. To repeat the key schedule the two blocks are concatenated, forming a 128-bit key that is used as the input for the generation of the next round key.

### *2.3.2. Key Whitening*

This is implemented through applying the XOR operation between the binary plaintext and first half of the initial key before encryption rounds begin and after, with an XOR operation between the ciphertext and last half of the initial key (Knudsen & Robshaw, 2011). This improves the block cipher's resistance against attacks for instance, differential power analysis and key-search attacks (Tu et al., 2017; Knudsen & Robshaw, 2011).

### *2.3.3. Substitution-boxes (S-boxes)*

An S-box consists of a look-up table where chosen bits in a sequence form a position on the table, substituting with the position's value (Easttom, 2022). The block cipher implements S-boxes when substituting the 4-bit blocks with another seen in table 4, producing confusion as the relationship between the plaintext and the round's ciphertext reduces. Despite this, the S-box can be improved in comparison to DES and AES which use larger look-up tables, producing greater confusion over the former with only 16 possible substitutions (Easttom, 2022).

### *2.3.4. Multiple Key Lengths*

AES provides the user the ability to use 3 key lengths of 128, 192 and 256-bits. As a result, for an attacker to use a brute-force attack and retrieve the correct key, it would involve testing all keys for each length. This would improve the block cipher as currently an attacker can brute force the correct key in a few days, whilst this method would remove this possibility.

## 2.4. Evaluation of Block Cipher's Confusion

By changing a single bit in the initial key, a minor avalanche is achieved as it changes the round keys generated and therefore, changes the outcome of the XOR operations between the blocks and round keys (Easttom, 2022). In addition, by changing a single bit in the plaintext block, it alters the ascii character formed by the byte. For instance, with '01010100' (T) having a bit changed to 1, '01010101' (U) however, only changes the outcome of a single bit within the XOR operation with the round key.

The single bit change has a greater impact on the 4-bit block substitutions where a single bit changes the combination of bits, resulting in different substitutions and causing the 3 other bits in the 4-bit block to change. Furthermore, this also impacts the ascii character produced when the block is combined with another, resulting in a change of an ascii character. An example of this is the bit combination '1011' becoming '1001' due to a bit swap in the second position. Rather than the substitution '1111', '0100' is substituted, changing the nibble's outcome and ascii character of byte when converted. Finally, this then changes the ciphertext at the end of the encryption round, impacting the input for the next round and the final ciphertext produced, further impacting the result of the XOR operation with the next block's plaintext for CBC, changing the outcome of the encryption.

## 2.5 Comparisons with DES and AES

### *2.5.1. Key Space*

In comparison to DES with a key space of $2^{56}$, this block cipher has a larger key size of $2^{64}$, increasing the number of brute-force attempts before decryption occurs. Although better than DES, AES uses 128-bit round keys providing a $2^{128}$ key space which is unable to be brute forced for several decades whilst DES and the designed block cipher in a few days.

### *2.5.2. Lack of Cryptanalysis Conducted Before Mainstream Implementation*

Due to the amount of time DES and AES has existed, it has provided the opportunity for cryptanalysis to be performed to evaluate potential weaknesses. This cipher has not therefore, may contain zero-day exploits which can severely harm the bank (IBM, n.d.; Australian Signals Directorate, n.d.).

### *2.5.3. Inferior substitution*

Within DES, each of the 8 blocks receive a unique look-up table with 64 possible substitutions each in comparison to this cipher which has all blocks using the same bit combinations and only achieving 16 possible substitutions. This increases confusion as it has a greater variety of values to obscure the relationship of the plaintext with.

## 3. Recommendation

When recommending the encryption algorithm best suited for the bank, both speed and security must be considered. The bank may perform 2.2 million transactions per day (~26 transactions per second) therefore, must choose a cipher fast enough to process all transactions (Reserve Bank of Australia, 2022). In addition, the algorithm must be secure, ensuring the attacker is unable to decrypt the ciphertext and manipulate transactions. The block cipher provides more secure encryption than the stream cipher through its multiple rounds and methods of encryption whilst the stream cipher is faster, using a pseudorandom generated keystream for encryption.

### 3.1. Applying The Ciphers Within Use Cases

Both ciphers were implemented in python, each performing 2000 encryptions of the message X (performed on a MacBook Pro, M1 pro chip, 16gb). It was discovered that the stream cipher executed 2000 separate encryptions in 3.86 seconds and block cipher in 6.39 seconds, 2.53 seconds slower. Although the block cipher is slower, it outperforms the 26 executions per second required to perform the 2.2 million daily transactions, therefore, is fast enough for the bank's application.

As a result, it is recommended the bank utilises the block cipher. It provides greater security over the stream cipher, whilst also executing in a timely manner.

## 4. References

Australian Signals Directorate. (n.d.). *Zero day exploit*. Australian Cyber Security Centre. https://www.cyber.gov.au/glossary/zero-day-exploit

Easttom, C. (2022). S-box design. In W. Easttom (Eds.), *Modern Cryptography* (pp. 193-212). Springer. https://doi.org/10.1007/978-3-031-12304-7_8

IBM. (n.d.). *What is a zero-day exploit?*. https://www.ibm.com/think/topics/zero-day

Knudsen, L., & Robshaw, M. (2011). *The block cipher companion*. Springer-Verlag Berlin Heidelberg. https://doi.org/10.1007/978-3-642-17342-4

Potestad-Ordóñez, F. E., Valencia-Barrero, M., Baena-Oliva, C., Parra-Fernández, P., & Jiménez-Fernández, C. J. (2020). Breaking Trivium Stream Cipher Implemented in ASIC Using Experimental Attacks and DFA. *Sensors*, *20*(23), 6909–6927. https://doi.org/10.3390/s20236909

Reserve Bank of Australia. (2022). *Reverse Bank of Australia Annual Report – 2022*. https://www.rba.gov.au/publications/annual-reports/rba/2022/banking-and-payment-services.html

Robshaw, M., & Billet, O. (2008). *New stream cipher designs : the eSTREAM finalists*. Springer. https://doi.org/10.1007/978-3-540-68351-3

Tu, C., Zhang, L., Liu, Z., Gao, N., & Ma, Y. (2017, June). A Practical Chosen Message Power Analysis Approach Against Ciphers with the key Whitening Layers. In D. Gollmann, H. Kikuchi & A. Miyaji (Eds.), *Applied Cryptography and Network Security* (pp. 415-434). Springer. https://doi.org/10.1007/978-3-319-61204-1_21