CSE 470

CRYPTOGRAPHY AND COMPUTER SECURITY

Homework 1

GIFT-COFB Project Report

Muhammed Bedir ULUÇAY

1901042697

GIFT-COFB (GIFT with Confusion and Addition of Feedback) is a block cipher algorithm that was proposed by Indian researchers in 2017. It is a variant of the GIFT (Gifted Internet Fast Tranmission) block cipher algorithm, which was designed to be lightweight and suitable for use in resource-constrained environments such as smart cards and embedded systems.

GIFT-COFB is a 128-bit block cipher that uses a 128-bit key for encryption and decryption. It operates in a block cipher mode of operation, which means that it processes the input data in fixed-size blocks (in this case, 128 bits). The algorithm consists of four main rounds of substitution, permutation, and key addition, followed by an additional round of substitution, permutation, and key addition for decryption.

One of the key features of GIFT-COFB is its use of confusion and addition of feedback (COFB) to increase the diffusion and confusion of the data. Confusion refers to the mixing of the input data in a way that makes it difficult to determine the original values from the output data. Addition of feedback refers to the use of a feedback function to add a portion of the output data back into the input data for the next round of processing. This helps to further obscure the relationship between the input data and the output data, making it more difficult for an attacker to break the encryption.

In addition to the COFB mechanism, GIFT-COFB also uses a substitution-permutation network (SPN) structure, which is a common design used in block cipher algorithms. The SPN consists of a series of substitution boxes (S-boxes) and permutation boxes (P-boxes), which are used to transform the input data in a way that is difficult to reverse or predict. The S-boxes and P-boxes are controlled by the key, which helps to further increase the security of the algorithm.

Overall, GIFT-COFB is a relatively new block cipher algorithm that has been designed to be lightweight and suitable for use in resource-constrained environments. Its use of COFB and an SPN structure help to increase the security of the algorithm by increasing the diffusion and confusion of the data, making it more difficult for an attacker to break the encryption.

As mentioned in the previous response, GIFT-COFB is a block cipher algorithm that operates on 128-bit blocks of data using a 128-bit key. It consists of four main rounds of processing, each of which consists of a substitution step, a permutation step, and a key addition step. The key addition step involves XORing (Exclusive OR) the input data with a key value that is derived from the key.

Here is an example of the pseudocode for the encryption process in GIFT-COFB:

```
def encrypt(input_block, key):
    # Initialize the output block to the input block
    output_block = input_block

    # Perform the four main rounds of processing
    for i in range(4):
        # Substitute the output block using the S-box for round i
        output_block = substitute(output_block, S[i])

        # Permute the output block using the P-box for round i
        output_block = permute(output_block, P[i])

        # Add the key for round i to the output block using XOR
        output_block = xor(output_block, K[i])

    # Return the final output block
    return output_block
```

Here, `input_block` is the 128-bit input block to be encrypted, `key` is the 128-bit key, `S[i]` is the S-box for round `i`, `P[i]` is the P-box for round `i`, and `K[i]` is the key value for round `i`. The `substitute` and `permute` functions perform the substitution and permutation operations, respectively, using the specified S-box and P-box. The `xor` function performs an XOR operation on the two input values.

The decryption process in GIFT-COFB is similar to the encryption process, but it uses a slightly different set of S-boxes and P-boxes and a different order of operations. Here is an example of the pseudocode for the decryption process:

```
def decrypt(input_block, key):
    # Initialize the output block to the input block
    output_block = input_block

    # Perform the additional round of processing for decryption
    # Substitute the output block using the S-box for round 4
    output_block = substitute(output_block, S[4])

    # Permute the output block using the P-box for round 4
    output_block = permute(output_block, P[4])

    # Add the key for round 4 to the output block using XOR
    output_block = xor(output_block, K[4])

    # Perform the four main rounds of processing in reverse order
    for i in range(3, -1, -1):
        # Add the key for round i using XOR
        output_block = xor(output_block, K[i])

        # Substitute the output block using the S-box for round i
        output_block = substitute(output_block, S[i])

        # Permute the output block using the P-box for round i
        output_block = permute(output_block, P[i])

    # Return the output block
    return output_block
```

The GIFT-COFB algorithm uses a series of S-boxes (substitution boxes) and P-boxes (permutation boxes) to transform the input data in a way that is difficult to reverse or predict. The S-boxes and P-boxes are controlled by the key, which helps to further increase the security of the algorithm.

Here is an example of the pseudocode for the `substitute` function, which performs the substitution step in the GIFT-COFB algorithm:

```
def substitute(input_block, s_box):
    # Initialize the output block to all zeros
    output_block = [0] * 128

    # Substitute each bit of the input block using the S-box
    for i in range(128):
        output_block[i] = s_box[input_block[i]]

    # Return the output block
    return output_block
```

Here, `input_block` is the 128-bit input block to be transformed, and `s_box` is the S-box to be used for the substitution. The `s_box` is an array of 256 values, where each value is a bit (either 0 or 1). The `substitute` function loops through each bit of the `input_block` and substitutes it with the corresponding value from the `s_box`.

The permutation step in the GIFT-COFB algorithm is similar to the substitution step, but it uses a P-box (permutation box) instead of an S-box. Here is an example of the pseudocode for the `permute` function, which performs the permutation step in the GIFT-COFB algorithm:

```
def permute(input_block, p_box):
    # Initialize the output block to all zeros
    output_block = [0] * 128

    # Permute each bit of the input block using the P-box
    for i in range(128):
        output_block[i] = input_block[p_box[i]]

    # Return the output block
    return output_block
```

Input and Output Data

To encrypt a message M with associated data A and nonce N , one needs to provide the information given below.
The encryption algorithm takes as input

- An encryption key $K \in \{0, 1\}^{128}$.
- A nonce $N \in \{0, 1\}^{128}$. This can include the counter to make the nonce non-repeating.
- Associated data and message $A, M \in \{0, 1\}^*$.

It generates the following output data:

- Ciphertext $C \in \{0, 1\}^{|M|}$.
- Tag $T \in \{0, 1\}^{128}$

To decrypt (with verification) a ciphertext-tag pair (C, T) with associated data A and nonce N , one needs to provide the information given below.

- An encryption key $K \in \{0, 1\}^{128}$.
- A nonce $N \in \{0, 1\}^{128}$.
- Associated data and ciphertext $A, C \in \{0, 1\}^*$.
- Tag $T \in \{0, 1\}^{128}$

It generates the following output data:

- Message $M \in \{0, 1\}^{|C|} \cup \{\perp\}$, where $\perp$ is a special symbol denoting rejection.

DESIGN:

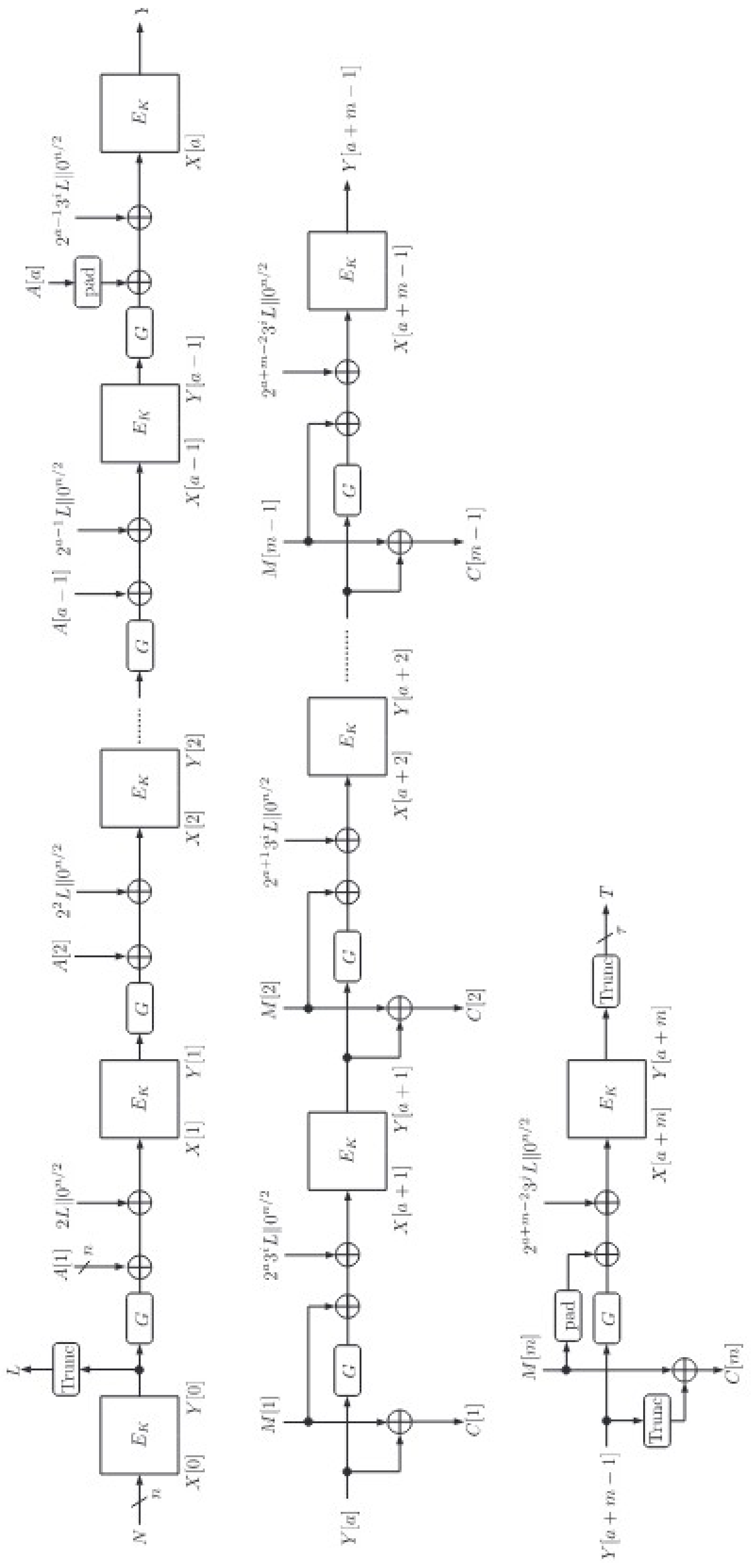

Figure 2.2: Encryption of COFB. In the rightmost figure, the case of encryption for empty M (hence a MAC for (N, A)) can be highlighted as $T = \mathsf{Trunc}_\tau(Y[a])$

ALGORITHM:

Algorithm $\mathsf{COFB}\text{-}\mathcal{E}_K(N, A, M)$

$Y[0] \leftarrow E_K(N),\ L \leftarrow \mathsf{Trunc}_{n/2}(Y[0])$
$(A[1], \ldots, A[a]) \xleftarrow{n} \mathsf{Pad}(A)$
if $M \neq \epsilon$ **then**
  $(M[1], \ldots, M[m]) \xleftarrow{n} \mathsf{Pad}(M)$
for $i = 1$ **to** $a - 1$
  $L \leftarrow 2 \cdot L$
  $X[i] \leftarrow A[i] \oplus G \cdot Y[i-1] \oplus L\|0^{n/2}$
  $Y[i] \leftarrow E_K(X[i])$
if $|A| \bmod n = 0$ **and** $A \neq \epsilon$ **then** $L \leftarrow 3 \cdot L$
else $L \leftarrow 3^2 \cdot L$
if $M = \epsilon$ **then** $L \leftarrow 3^2 \cdot L$
$X[a] \leftarrow A[a] \oplus G \cdot Y[a-1] \oplus L\|0^{n/2}$
$Y[a] \leftarrow E_K(X[a])$
for $i = 1$ **to** $m - 1$
  $L \leftarrow 2 \cdot L$
  $C[i] \leftarrow M[i] \oplus Y[i + a - 1]$
  $X[i+a] \leftarrow M[i] \oplus G \cdot Y[i + a - 1] \oplus L\|0^{n/2}$
  $Y[i+a] \leftarrow E_K(X[i+a])$
if $M \neq \epsilon$ **then**
  if $|M| \bmod n = 0$ **then** $L \leftarrow 3 \cdot L$
  else $L \leftarrow 3^2 \cdot L$
  $C[m] \leftarrow M[m] \oplus Y[a + m - 1]$
  $X[a+m] \leftarrow M[m] \oplus G \cdot Y[a+m-1] \oplus L\|0^{n/2}$
  $Y[a+m] \leftarrow E_K(X[a+m])$
  $C \leftarrow \mathsf{Trunc}_{|M|}(C[1]\|\ldots\|C[m])$
  $T \leftarrow \mathsf{Trunc}_\tau(Y[a+m])$
else $C \leftarrow \epsilon,\ T \leftarrow \mathsf{Trunc}_\tau(Y[a])$
return (C, T)

Algorithm $\mathsf{COFB}\text{-}\mathcal{D}_K(N, A, C, T)$

$Y[0] \leftarrow E_K(N),\ L \leftarrow \mathsf{Trunc}_{n/2}(Y[0])$
$(A[1], \ldots, A[a]) \xleftarrow{n} \mathsf{Pad}(A)$
if $C \neq \epsilon$ **then**
  $(C[1], \ldots, C[c]) \xleftarrow{n} \mathsf{Pad}(C)$
for $i = 1$ **to** $a - 1$
  $L \leftarrow 2 \cdot L$
  $X[i] \leftarrow A[i] \oplus G \cdot Y[i-1] \oplus L\|0^{n/2}$
  $Y[i] \leftarrow E_K(X[i])$
if $|A| \bmod n = 0$ **and** $A \neq \epsilon$ **then** $L \leftarrow 3 \cdot L$
else $L \leftarrow 3^2 \cdot L$
if $C = \epsilon$ **then** $L \leftarrow 3^2 \cdot L$
$X[a] \leftarrow A[a] \oplus G \cdot Y[a-1] \oplus L\|0^{n/2}$
$Y[a] \leftarrow E_K(X[a])$
for $i = 1$ **to** $c - 1$
  $L \leftarrow 2 \cdot L$
  $M[i] \leftarrow Y[i + a - 1] \oplus C[i]$
  $X[i+a] \leftarrow M[i] \oplus G \cdot Y[i + a - 1] \oplus L\|0^{n/2}$
  $Y[i+a] \leftarrow E_K(X[i+a])$
if $C \neq \epsilon$ **then**
  if $|C| \bmod n = 0$ **then**
    $L \leftarrow 3 \cdot L$
    $M[c] \leftarrow Y[a + c - 1] \oplus C[c]$
  else
    $L \leftarrow 3^2 \cdot L,\ c' \leftarrow |C| \bmod n$
    $M[c] \leftarrow \mathsf{Trunc}_{c'}(Y[a+c-1] \oplus C[c])\|10^{n-c'-1}$
  $X[a+c] \leftarrow M[c] \oplus G \cdot Y[a+c-1] \oplus L\|0^{n/2}$
  $Y[a+c] \leftarrow E_K(X[a+c])$
  $M \leftarrow \mathsf{Trunc}_{|C|}(M[1]\|\ldots\|M[c])$
  $T' \leftarrow \mathsf{Trunc}_\tau(Y[a+c])$
else $M \leftarrow \epsilon,\ T' \leftarrow \mathsf{Trunc}_\tau(Y[a])$
if $T' = T$ **then return** M, **else return** $\perp$

Figure 2.3: The encryption and decryption algorithms of COFB

```
mbulucay@mbulucay:~/Desktop/Crypto/1901042697/gift-cofb$ ./gift_cofb
GIFT-COFB AEAD

Key       : 37d04d0055e19837f980ddaaed01f2f9
Nonce     : 2d6ad8649c67337184d79a9fbd50e84c
Text      : f1b20766e45901a16d0e51573dd2936e757f23340cfc02d9799c72cf1bded8d3
Encrypted : ffd16adfd5f200ca5855e3753ceee9d274c3e4ae608a2feb857b215abadd40df
Tag       : 58b3d1316ea1a77b359364236cb3450d
Decrypted : f1b20766e45901a16d0e51573dd2936e757f23340cfc02d9799c72cf1bded8d3
GIFT-COFB AEAD

Key       : 37d04d0055e19837f980ddaaed01f2f9
Nonce     : 2d6ad8649c67337184d79a9fbd50e84c
Text      : f1b20766e45901a16d0e51573dd2936e757f23340cfc02d9799c72cf1bded8d3
Encrypted : a3ba8d26c4dcd4fe735930324ef93dd414f16958fff5687c1ecd51574645cdab
Tag       : 9d853d99e1ac0237306f72c2f922f849
Decrypted : f1b20766e45901a16d0e51573dd2936e757f23340cfc02d9799c72cf1bded8d3
GIFT-COFB AEAD

Key       : 37d04d0055e19837f980ddaaed01f2f9
Nonce     : 2d6ad8649c67337184d79a9fbd50e84c
Text      : f1b20766e45901a16d0e51573dd2936e757f23340cfc02d9799c72cf1bded8d3
Encrypted : 6df63ac68650781f61423ee327ee77fee7644ad5f0efe3145414c8c0bc03f4f3
Tag       : c788555a98d19ca929bbf5a5287f3b8f
Decrypted : f1b20766e45901a16d0e51573dd2936e757f23340cfc02d9799c72cf1bded8d3
GIFT-COFB AEAD

Key       : 37d04d0055e19837f980ddaaed01f2f9
Nonce     : 2d6ad8649c67337184d79a9fbd50e84c
Text      : f1b20766e45901a16d0e51573dd2936e757f23340cfc02d9799c72cf1bded8d3
Encrypted : 663f0311be01a5de24148dd21ccacd42605478c8ff53b43a8d2d37a4a931cc49
Tag       : 8807ef7f5e2a49df41c923b04b286e38
Decrypted : f1b20766e45901a16d0e51573dd2936e757f23340cfc02d9799c72cf1bded8d3
GIFT-COFB AEAD

Key       : 37d04d0055e19837f980ddaaed01f2f9
Nonce     : 2d6ad8649c67337184d79a9fbd50e84c
Text      : f1b20766e45901a16d0e51573dd2936e757f23340cfc02d9799c72cf1bded8d3
Encrypted : ba8ea12e178f08539f195759fc10f044d0c9c07d785987bb112455dffc1e0605
Tag       : 200219f09a55cf30e7e7b6f76d02013c
Decrypted : f1b20766e45901a16d0e51573dd2936e757f23340cfc02d9799c72cf1bded8d3
GIFT-COFB AEAD

Key       : 37d04d0055e19837f980ddaaed01f2f9
Nonce     : 2d6ad8649c67337184d79a9fbd50e84c
Text      : f1b20766e45901a16d0e51573dd2936e757f23340cfc02d9799c72cf1bded8d3
Encrypted : 1bb14ee7b62bd51965d0dcbd2da083a892d78b2f5f1a46b408b97f2b90baffc7
Tag       : 7a2417500a1b60ab7a3ae0e21e0ed494
Decrypted : f1b20766e45901a16d0e51573dd2936e757f23340cfc02d9799c72cf1bded8d3
GIFT-COFB AEAD

Key       : 37d04d0055e19837f980ddaaed01f2f9
Nonce     : 2d6ad8649c67337184d79a9fbd50e84c
Text      : f1b20766e45901a16d0e51573dd2936e757f23340cfc02d9799c72cf1bded8d3
Encrypted : ec9c954cfeca5d659f459dcf035823051ff097ed07a656c7ed97cfc8814ae278
Tag       : 094624215605dfd87f2e69b460355bdb
Decrypted : f1b20766e45901a16d0e51573dd2936e757f23340cfc02d9799c72cf1bded8d3
```