Merve Horuz
1801042651

BIL 470 RESEARCH PART OF THE PROJECT

Lightweight cryptography refers to cryptographic techniques and algorithms that are designed to be efficient and require minimal computational resources, such as processor power or memory. These algorithms are often used in situations where the device or system being used has limited resources, such as small embedded systems, Internet of Things (IoT) devices, and low-power devices.

Lightweight cryptography is typically used to secure communications, authenticate devices, and protect data. Examples of lightweight cryptographic algorithms include:

Symmetric encryption algorithms: These algorithms use the same secret key for both encryption and decryption. Examples of symmetric encryption algorithms include Blowfish, Twofish, RC4, and Salsa20.

Hash functions: These algorithms take an input (or "message") and produce a fixed-size output (or "hash") that is unique to the input. Hash functions are used for a variety of purposes, including data integrity checks, password storage, and data indexing.

Digital signatures: These algorithms allow a sender to sign a message with a private key, which can then be verified by anyone with the corresponding public key. Digital signatures are used to authenticate the identity of the sender and ensure the integrity of the message.

Public-key cryptography: These algorithms use a pair of keys, a public key and a private key, to perform encryption and decryption. The public key is used to encrypt messages, while the private key is used to decrypt them.

In general, lightweight cryptography is used to provide secure communication and protect data in situations where the resources available are limited. However, it is important to carefully evaluate the security needs of the application and choose cryptographic algorithms that are appropriate for the level of security required.

Lightweight Symmetric Encryption Algorithms

Lightweight symmetric encryption algorithms are designed to be fast and efficient, while still providing a reasonable level of security. They are often used in situations where computing power or bandwidth is limited, or where the volume of data to be encrypted is very large.

Some examples of lightweight symmetric encryption algorithms include:

Blowfish: Blowfish is a block cipher that uses a variable-length key and operates on 64-bit blocks. It is very fast and secure and has been widely adopted for use in a variety of applications.

Twofish: Twofish is a block cipher that uses a 128-bit key and operates on 128-bit blocks. It was designed as a successor to Blowfish and is considered to be very fast and secure.

RC4: RC4 (Rivest Cipher 4) is a stream cipher that uses a variable-length key. It is considered to be very fast and is widely used in a variety of applications, including secure sockets layer (SSL) and transport layer security (TLS) protocols.

Salsa20: Salsa20 is a stream cipher that uses a 256-bit key and operates on 64-bit blocks. It is considered to be very fast and is widely used in a variety of applications, including secure messaging and file encryption.

In general, lightweight symmetric encryption algorithms are considered to be suitable for a wide range of applications, including secure communication, data storage, and file transfer. However, they are generally not as secure as heavier, more computationally intensive algorithms, and may not be suitable for use in situations where a very high level of security is required.

There are several factors to consider when analyzing lightweight symmetric encryption algorithms:

Key size: A larger key size generally results in a stronger level of security but may also result in slower encryption and decryption times.

Speed: Lightweight algorithms should be fast and efficient, especially if they are being used on devices with limited resources.

Security: The security of an encryption algorithm is determined by how difficult it is to break the encryption without knowing the secret key. Lightweight algorithms may not be as secure as heavier, more computationally intensive algorithms, and may not be suitable for use in situations where a very high level of security is required.

Algorithm design: The design of the algorithm can also impact its security and efficiency. For example, algorithms that use a block cipher (which operates on fixed-size blocks of data) may be more secure than algorithms that use a stream cipher (which operates on a continuous stream of data).

GIFT-COFB And ISAP

GIFT-COFB

GIFT-COFB authenticated, which instantiates the COFB (COmbined FeedBack) block cipher based AEAD mode with the GIFT block cipher. COFB primarily focuses on the hardware implementation size. Here, we consider the overhead in size, thus the state memory size beyond the underlying block cipher itself (including the key schedule) is the criteria we want to minimize, which is particularly relevant for hardware implementation.

ISAP

Isap is a family of nonce-based authenticated ciphers with associated data (AEAD) designed with a focus on robustness against passive side-channel attacks. All Isap family members are permutation-based designs that combine variants of the sponge-based Isap mode with one of several published lightweight permutations. The main design goal of Isap is to provide out-of-the-box robustness against certain types of implementation attacks while allowing to add additional defense mechanisms at low cost. This is essential whenever cryptographic devices are deployed in locations that are physically accessible by potential attackers – a typical scenario in IoT (Internet of Things) applications. Secure software and firmware updates on such devices in particular are both crucial and challenging.

Notation of GIFT-COFB

For any $X \in \{0, 1\} *$, where $\{0, 1\} *$ is the set of all finite bit strings (including the empty string), we denote the number of bits of X by $|X|$. Note that $|e| = 0$.

For a string X and an integer $t \leq |X|$, Trunc t (X) is the first t bits of X.

Throughout this document, n represents the block size in bits of the underlying block cipher EK . Typically, we consider $n = 128$ and GIFT-128 is the underlying block cipher, where K is the 128-bit GIFT-128 key.

For two-bit strings X and Y, $X \| Y$ denotes the concatenation of X and Y.

A bit string X is called a complete (or incomplete) block if $|X| = n$ (or $|X| < n$ respectively). We write the set of all complete (or incomplete) blocks as B (or B < respectively). Note that, e is considered as an incomplete block and $e \in$ B <.

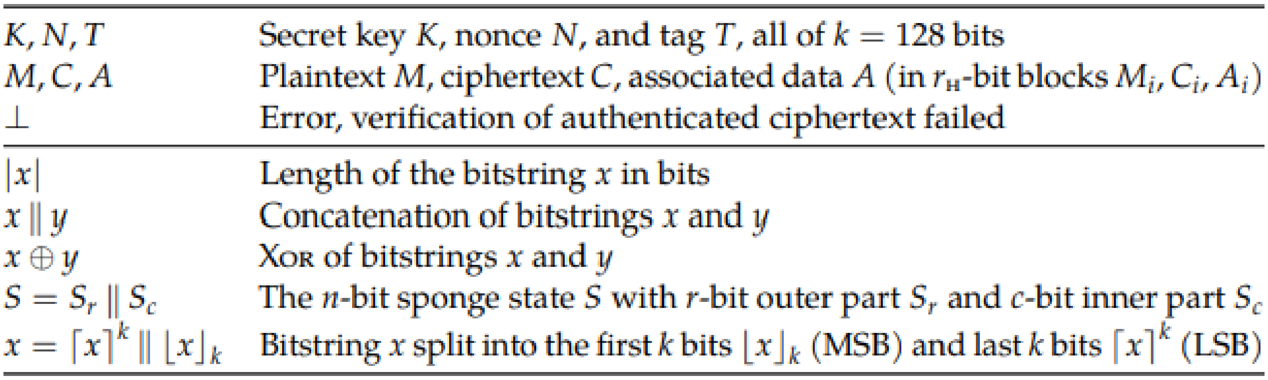
Given non-empty Z ∈ {0, 1} ∗, we define the parsing of Z into n-bit blocks as n (Z [1], Z [2], . . ., Z[z]) ←n Z, where z = ceiling(|Z|/n), |Z[i]| = n for all i < z and 1 ≤ |Z[z]| ≤ n such that

Z = (Z [1] k Z [2] k · · k Z[z]). If Z = e, we let z = 1 and Z [1] = e. We write ||Z|| = z (number of blocks present in Z).

Given any sequence Z = (Z [1], . . ., Z[s]) and 1 ≤ a ≤ b ≤ s, we represent the sub sequence (Z[a], . . ., Z[b]) by Z[a..b].

For integers a ≤ b, we write [a..b] for the set {a, a + 1, . . ., b}.

Notation of ISAP



K, N, T	Secret key K, nonce N, and tag T, all of $k = 128$ bits
M, C, A	Plaintext M, ciphertext C, associated data A (in r_H-bit blocks M_i, C_i, A_i)
$\perp$	Error, verification of authenticated ciphertext failed
$\|x\|$	Length of the bitstring x in bits
$x \| y$	Concatenation of bitstrings x and y
$x \oplus y$	Xor of bitstrings x and y
$S = S_r \| S_c$	The n-bit sponge state S with r-bit outer part S_r and c-bit inner part S_c
$x = \lceil x \rceil^k \| \lfloor x \rfloor_k$	Bitstring x split into the first k bits $\lfloor x \rfloor_k$ (MSB) and last k bits $\lceil x \rceil^k$ (LSB)

Encryption with GIFT-COFB

Round function

Each round of GIFT-128 consists of 3 steps: SubCells, PermBits, and AddRoundKey.

Initialization. The 128-bit plaintext is loaded into the cipher state S which will be expressed as 4 32-bit segments. In the perspective of a 2-dimensional array, the bit ordering is from top-down, then right to left.

$$S = \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} \leftarrow \begin{bmatrix} b_{124} & \cdots & b_8 & b_4 & b_0 \\ b_{125} & \cdots & b_9 & b_5 & b_1 \\ b_{126} & \cdots & b_{10} & b_6 & b_2 \\ b_{127} & \cdots & b_{11} & b_7 & b_3 \end{bmatrix}.$$

The 128-bit secret key is loaded into the key state KS partitioned into 8 16-bit words. In the perspective of a 2-dimensional array, the bit ordering is from right to left, then bottom-up.

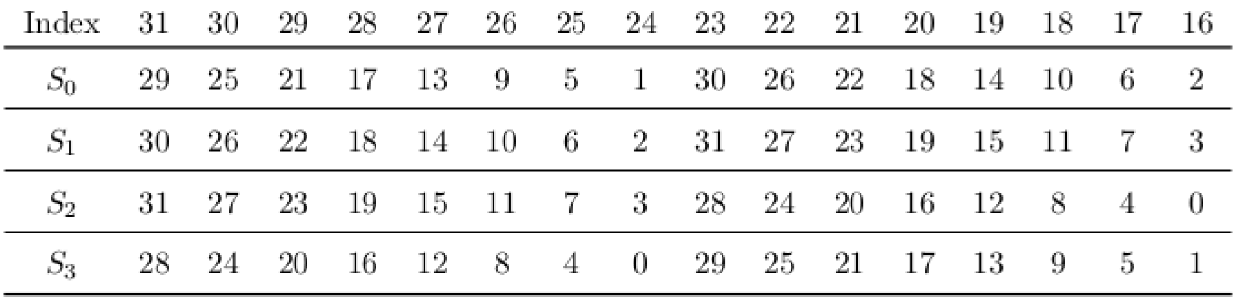
$$KS = \begin{bmatrix} W_0 & \| & W_1 \\ W_2 & \| & W_3 \\ W_4 & \| & W_5 \\ W_6 & \| & W_7 \end{bmatrix} \leftarrow \begin{bmatrix} b_{127} & \cdots & b_{112} & \| & b_{111} & \cdots & b_{98} & b_{97} & b_{96} \\ b_{95} & \cdots & b_{80} & \| & b_{79} & \cdots & b_{66} & b_{65} & b_{64} \\ b_{63} & \cdots & b_{48} & \| & b_{47} & \cdots & b_{34} & b_{33} & b_{32} \\ b_{31} & \cdots & b_{16} & \| & b_{15} & \cdots & b_{2} & b_{1} & b_{0} \end{bmatrix}$$

SubCells. Update the cipher state with the following instructions:

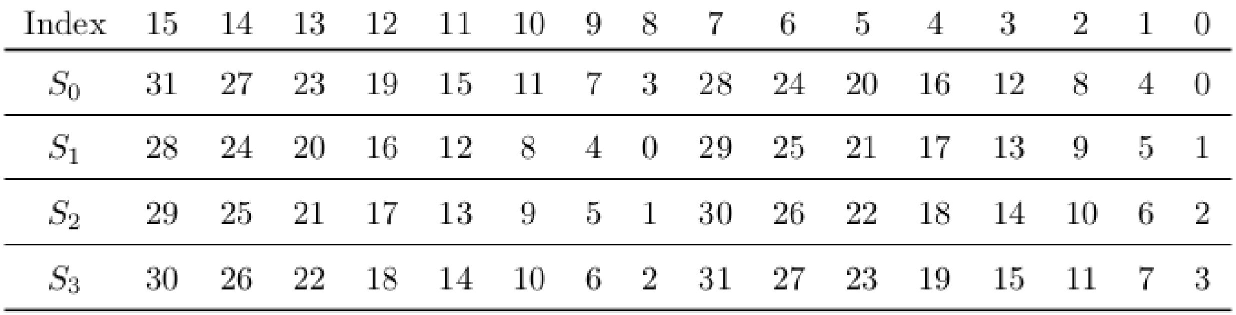
$$\begin{aligned} S_1 &\leftarrow S_1 \oplus (S_0 \ \& \ S_2) \\ S_0 &\leftarrow S_0 \oplus (S_1 \ \& \ S_3) \\ S_2 &\leftarrow S_2 \oplus (S_0 \mid S_1) \\ S_3 &\leftarrow S_3 \oplus S_2 \\ S_1 &\leftarrow S_1 \oplus S_3 \\ S_3 &\leftarrow \sim S_3 \\ S_2 &\leftarrow S_2 \oplus (S_0 \ \& \ S_1) \\ \{S_0, S_1, S_2, S_3\} &\leftarrow \{S_3, S_1, S_2, S_0\}, \end{aligned}$$

PermBits. Different 32-bit bit permutations are applied to each S i independently.

The row “Index” shows the indexing of the 32 bits in all S i ’s and the row “S i ” shows the ending position of the bits. For example, bit 1 (the 2nd rightmost bit) of S 1 is shifted 1 position to the right, to the initial position of bit 0, while bit 0 is shifted 8 positions to the left.



Index	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
S_0	29	25	21	17	13	9	5	1	30	26	22	18	14	10	6	2
S_1	30	26	22	18	14	10	6	2	31	27	23	19	15	11	7	3
S_2	31	27	23	19	15	11	7	3	28	24	20	16	12	8	4	0
S_3	28	24	20	16	12	8	4	0	29	25	21	17	13	9	5	1



Index	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S_0	31	27	23	19	15	11	7	3	28	24	20	16	12	8	4	0
S_1	28	24	20	16	12	8	4	0	29	25	21	17	13	9	5	1
S_2	29	25	21	17	13	9	5	1	30	26	22	18	14	10	6	2
S_3	30	26	22	18	14	10	6	2	31	27	23	19	15	11	7	3

AddRoundKey. This step consists of adding the round key and round constant. Two 32-bit segments U, V are extracted from the key state as the round key. RK = U || V. For the addition of round key, U and V are XORed to S2 and S1 of the cipher state respectively.

S 2 ← S 2 ⨁ U,

S 1 ← S 1 ⨁ V.

For the addition of round constant, S 3 is updated as follows,

S 3 ← S 3 ⨁ 0x800000XY,

where the byte XY = 00c 5 c 4 c 3 c 2 c 1 c 0.

Key schedule and round constants

A round key is first extracted from the key state before the key state update. Four 16-bit words of the key state are extracted as the round key RK = U || V.

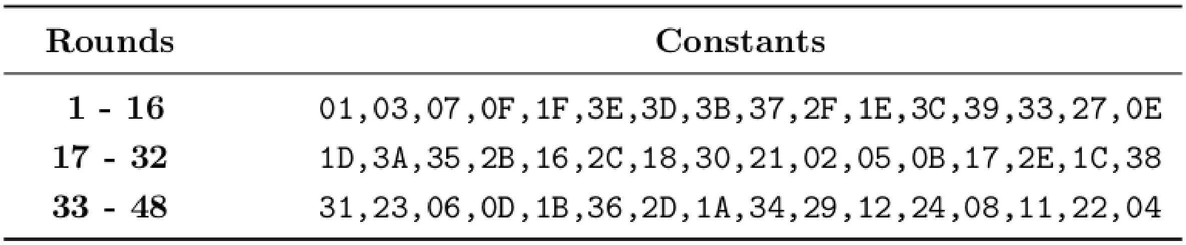
U ← W 2 || W 3, V ← W 6 || W 7.

The key state is then updated as follows,

$$\begin{bmatrix} W_0 & \| & W_1 \\ W_2 & \| & W_3 \\ W_4 & \| & W_5 \\ W_6 & \| & W_7 \end{bmatrix} \leftarrow \begin{bmatrix} W_6 \ggg 2 & \| & W_7 \ggg 12 \\ W_0 & \| & W_1 \\ W_2 & \| & W_3 \\ W_4 & \| & W_5 \end{bmatrix},$$

where ⋙ i is an i bits right rotation within a 16-bit word. The round constants are generated using a 6-bit affine LFSR, whose state is denoted as c5c4c3c2c1c0. Its update function is defined as: c5 || c4 || c3 || c2 || c1 || c0 ← c4 || c3 || c2 || c1 || c0 || c5 ⊕ c4 ⊕ 1.

The six bits are initialized to zero and updated before being used in a given round. The values of the constants for each round are given in the table below, encoded to byte values for each round, with c 0 being the least significant bit.

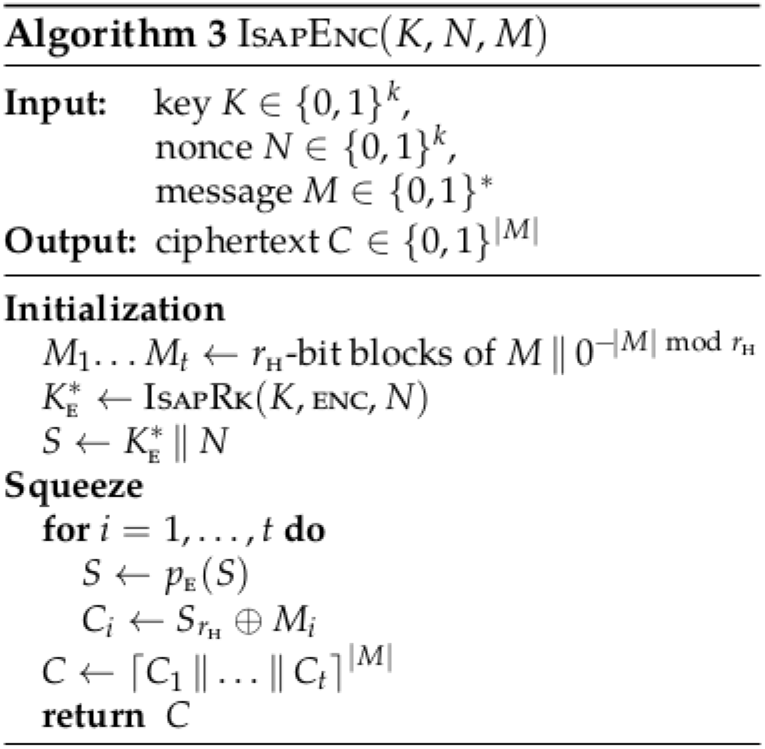


Rounds	Constants
1 - 16	01,03,07,0F,1F,3E,3D,3B,37,2F,1E,3C,39,33,27,0E
17 - 32	1D,3A,35,2B,16,2C,18,30,21,02,05,0B,17,2E,1C,38
33 - 48	31,23,06,0D,1B,36,2D,1A,34,29,12,24,08,11,22,04

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** $(C, T)$
```

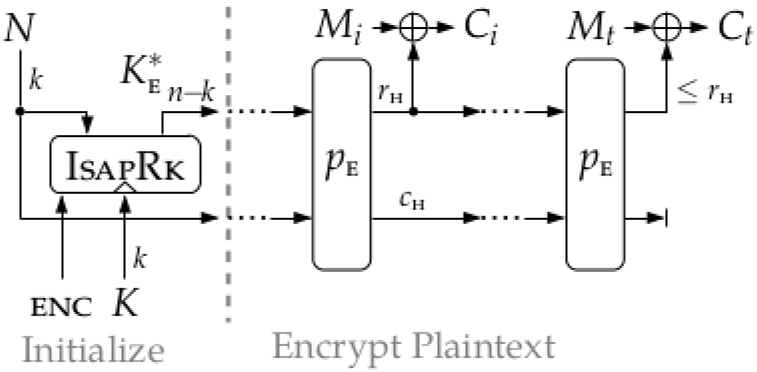
Encryption with ISAP



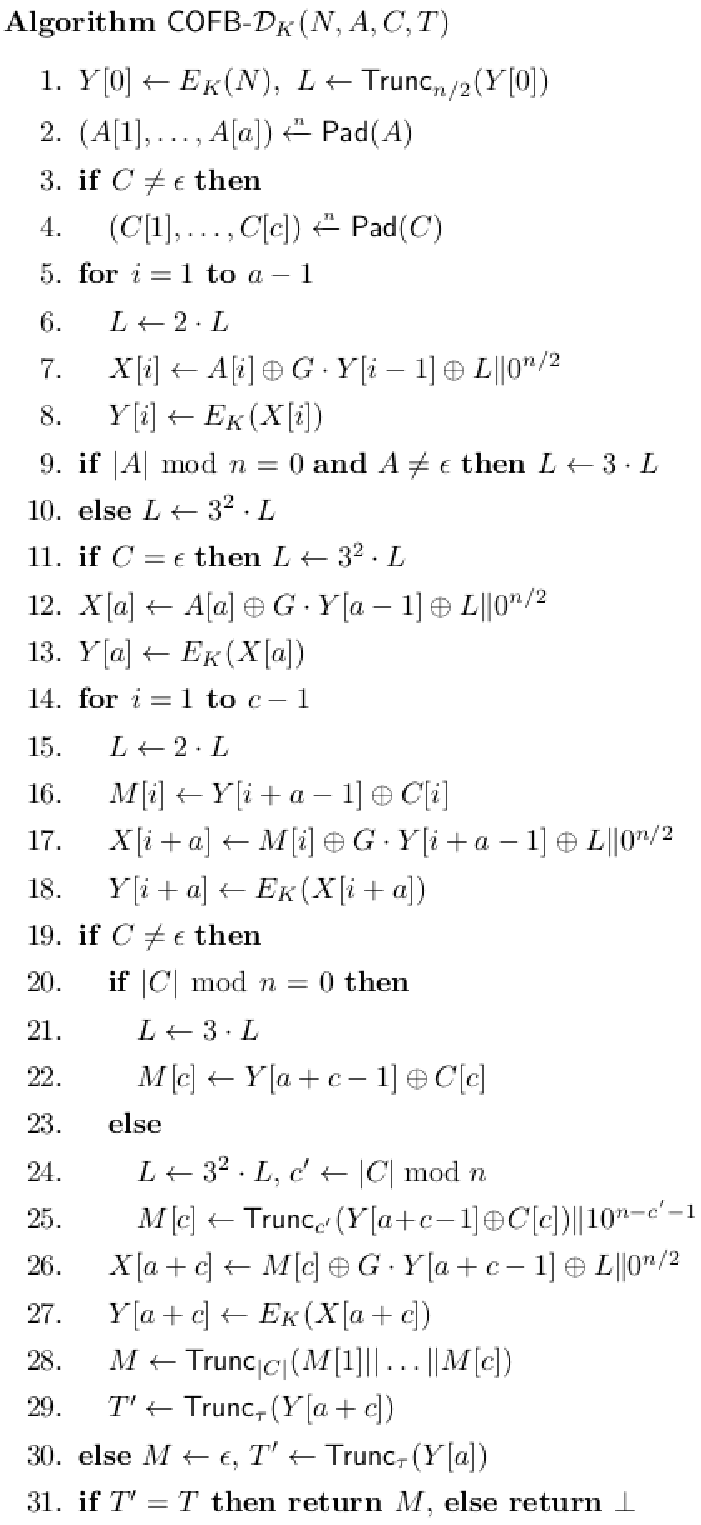
Algorithm 3 IsapEnc(K, N, M)

```
Input:  key $K \in \{0,1\}^k$,
        nonce $N \in \{0,1\}^k$,
        message $M \in \{0,1\}^*$
Output: ciphertext $C \in \{0,1\}^{|M|}$

Initialization
  $M_1 \dots M_t \leftarrow r_H$-bit blocks of $M \,\|\, 0^{-|M| \bmod r_H}$
  $K_E^* \leftarrow \text{IsapRk}(K, \text{ENC}, N)$
  $S \leftarrow K_E^* \,\|\, N$
Squeeze
  for $i = 1, \dots, t$ do
    $S \leftarrow p_E(S)$
    $C_i \leftarrow S_{r_H} \oplus M_i$
  $C \leftarrow \lceil C_1 \,\|\, \dots \,\|\, C_t \rceil^{|M|}$
  return $C$
```



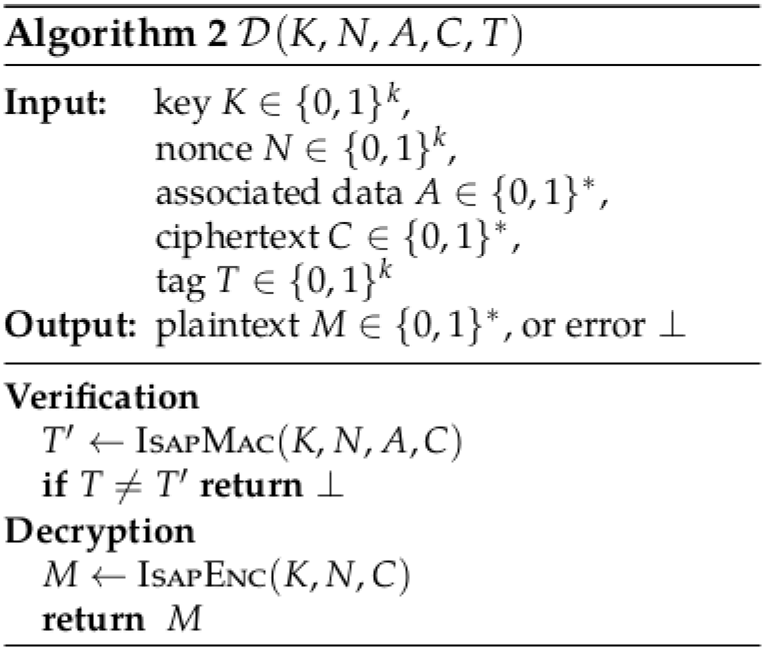
Decryption with GIFT-COFB



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

```
$Y[0] \leftarrow E_K(N),\ L \leftarrow \mathsf{Trunc}_{n/2}(Y[0])$
$(A[1], \ldots, A[a]) \xleftarrow{n} \mathsf{Pad}(A)$
**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** $\perp$
```

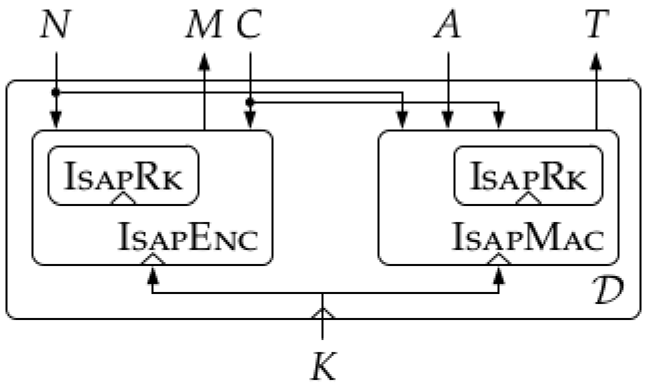
Decryption with ISAP



Algorithm 2 $\mathcal{D}(K, N, A, C, T)$

```
Input:  key $K \in \{0,1\}^k$,
        nonce $N \in \{0,1\}^k$,
        associated data $A \in \{0,1\}^*$,
        ciphertext $C \in \{0,1\}^*$,
        tag $T \in \{0,1\}^k$
Output: plaintext $M \in \{0,1\}^*$, or error $\perp$

Verification
    $T' \leftarrow \textsc{IsapMac}(K, N, A, C)$
    if $T \neq T'$ return $\perp$
Decryption
    $M \leftarrow \textsc{IsapEnc}(K, N, C)$
    return $M$
```



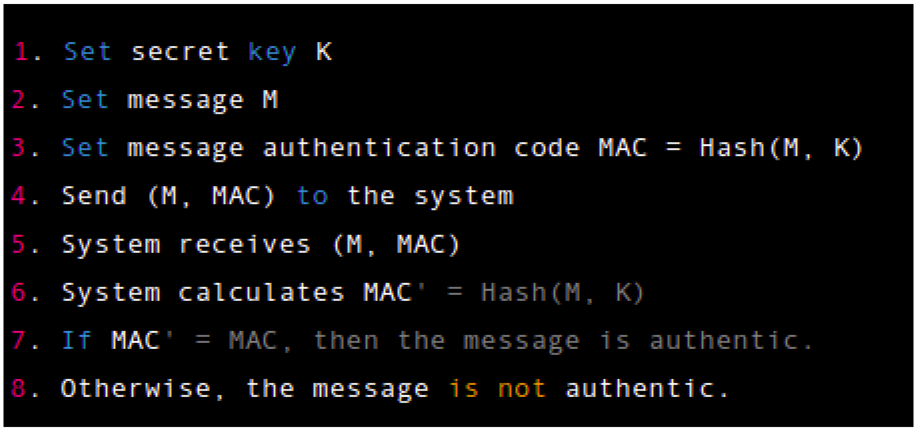
Authentication with GIFT-COFB

Authentication in GIFT-COFB involves verifying the identity of a user or device that is attempting to access a secure system or network. This is typically done by requiring the user to provide some form of credentials, such as a username and password, or by using some other method of identity verification, such as a biometric scan or a security token.

In the case of GIFT-COFB, the authentication process involves the use of a secret key that is shared between the user and the system. The user provides a message, along with a message authentication code (MAC), which is a hash of the message that is generated using the secret key. The system then verifies the MAC by using the same secret key to generate a new MAC for the message. If the two MACs match, the system can be sure that the message is authentic and was not tampered with during transmission.

In addition to authentication, GIFT-COFB also provides confidentiality by encrypting the message using the secret key. This ensures that the message can only be read by someone who has the key and helps to prevent unauthorized access to the data.

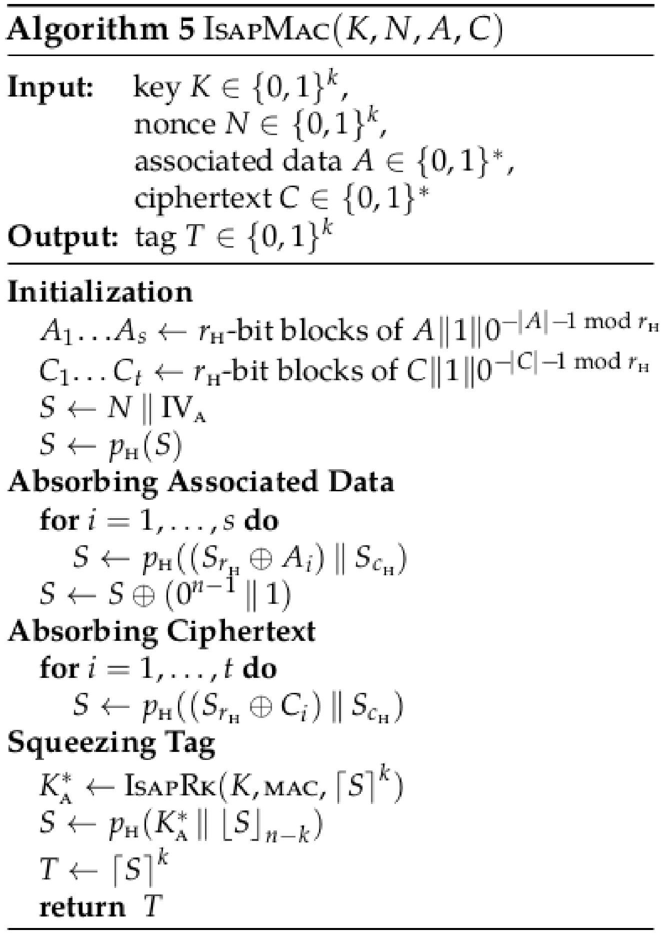
Overall, the authentication process in GIFT-COFB helps to ensure the security and integrity of data transmitted between a user and a secure system and helps to prevent unauthorized access to sensitive information.



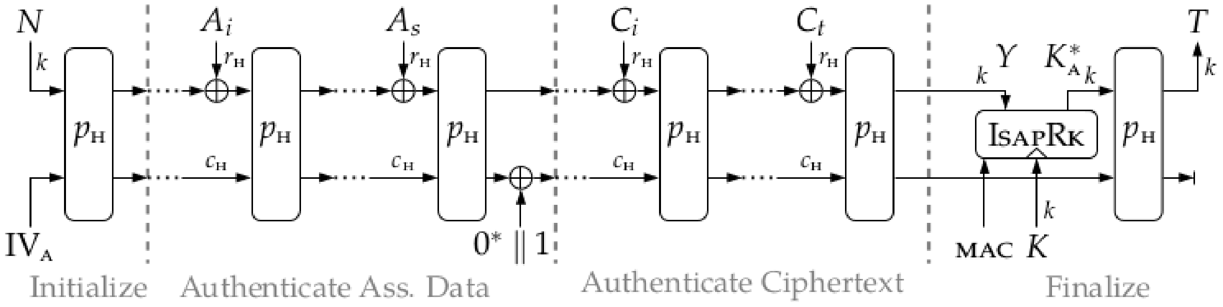
```
1. Set secret key K
2. Set message M
3. Set message authentication code MAC = Hash(M, K)
4. Send (M, MAC) to the system
5. System receives (M, MAC)
6. System calculates MAC' = Hash(M, K)
7. If MAC' = MAC, then the message is authentic.
8. Otherwise, the message is not authentic.
```

This pseudocode shows the basic steps involved in the authentication process for GIFT-COFB. Essentially, the user generates a MAC for the message using the secret key, and then sends the message and the MAC to the system. The system then uses the same secret key to calculate a new MAC for the message and compares it to the original MAC. If the two MACs match, then the system can be sure that the message is authentic and has not been tampered with.

Authentication with ISAP



```
Algorithm 5 $\textsc{IsapMac}(K, N, A, C)$
Input:  key $K \in \{0,1\}^k$,
        nonce $N \in \{0,1\}^k$,
        associated data $A \in \{0,1\}^*$,
        ciphertext $C \in \{0,1\}^*$
Output: tag $T \in \{0,1\}^k$
Initialization
    $A_1 \ldots A_s \leftarrow r_H$-bit blocks of $A \| 1 \| 0^{-|A|-1 \bmod r_H}$
    $C_1 \ldots C_t \leftarrow r_H$-bit blocks of $C \| 1 \| 0^{-|C|-1 \bmod r_H}$
    $S \leftarrow N \,\|\, \mathrm{IV}_A$
    $S \leftarrow p_H(S)$
Absorbing Associated Data
    for $i = 1, \ldots, s$ do
        $S \leftarrow p_H((S_{r_H} \oplus A_i) \,\|\, S_{c_H})$
    $S \leftarrow S \oplus (0^{n-1} \,\|\, 1)$
Absorbing Ciphertext
    for $i = 1, \ldots, t$ do
        $S \leftarrow p_H((S_{r_H} \oplus C_i) \,\|\, S_{c_H})$
Squeezing Tag
    $K_A^* \leftarrow \textsc{IsapRk}(K, \textsc{mac}, \lceil S \rceil^k)$
    $S \leftarrow p_H(K_A^* \,\|\, \lfloor S \rfloor_{n-k})$
    $T \leftarrow \lceil S \rceil^k$
    return $T$
```



```
Authenticated Encryption $\mathcal{E}(K, N, A, M)$
Input:   key $K \in \{0,1\}^k$,
         nonce $N \in \{0,1\}^k$,
         associated data $A \in \{0,1\}^*$,
         plaintext $M \in \{0,1\}^*$
Output:  ciphertext $C \in \{0,1\}^{|M|}$,
         tag $T \in \{0,1\}^k$
Encryption
  $K_E^* = g_1(N, K)$
  $C = ENC_{N,K_E^*}(M)$
Authentication
  $Y = H(N, A, C)$
  $K_A^* = g_2(Y, K)$
  $T = MAC_{K_A^*}(Y)$
  return $C, T$
```

```
Authenticated Decryption $\mathcal{D}(K, N, A, C, T)$
Input:   key $K \in \{0,1\}^k$,
         nonce $N \in \{0,1\}^k$,
         associated data $A \in \{0,1\}^*$,
         ciphertext $C \in \{0,1\}^*$,
         tag $T \in \{0,1\}^k$
Output:  plaintext $M \in \{0,1\}^{|C|}$, or error $\perp$
Verification
  $Y = H(N, A, C)$
  $K_A^* = g_2(Y, K)$
  $T' = MAC_{K_A^*}(Y)$
  if $T \neq T'$ return $\perp$
Decryption
  $K_E^* = g_1(N, K)$
  $M = DEC_{N,K_E^*}(C)$
  return $M$
```

Algorithm 6: Authenticated encryption and decryption procedures.

Security of GIFT-COFB

$$\mathbf{Adv}^{\text{AE}}_{\mathsf{COFB}}((q, q_f), (\sigma, \sigma_f), t) \leq \mathbf{Adv}^{\text{prp}}_{\mathsf{GIFT}}(q', t') + \frac{\binom{q'}{2}}{2^n} + \frac{1}{2^{n/2}} + \frac{q_f(n+4)}{2^{n/2+1}} + \frac{3\sigma^2 + q_f + 2(q + \sigma + \sigma_f) \cdot \sigma_f}{2^n} \qquad (4.1)$$

GIFT-COFB satisfies the requirement of security against 2^112 computations in a single key setting. This comes from the security assumption of GIFT.

GIFT-COFB satisfies the requirement that the input size shall not be smaller than 2^50 −1 bytes under a single key. This comes from the above expression of AE security advantage of COFB (which depicts even a higher bound than 2 50 − 1, when n = 128).

Differential cryptanalysis: Zhu et al. applied the mixed-integer-linear-programming based differential characteristic search method for GIFT-128 and found an 18-round differential characteristic with probability 2^−109, which was further extended to a 23-round key recovery attack with complexity (Data, Time, Memory) = (2^120, 2^120, 2^80). We expect that full (40) rounds are secure against differential cryptanalysis.

Linear cryptanalysis: GIFT-128 has a 9-round linear hull effect of 2^−45.99, which means that we would need around 27 rounds to achieve correlation potentially lower than 2^−128. Therefore, we expect that 40-round GIFT-128 is enough to resist against linear cryptanalysis.

Integral attacks: The lightweight 4-bit S-box in GIFT-128 may allow efficient integral attacks. The bit-based division property is evaluated against GIFT-128 by the designers, which detected an 11-round integral distinguisher.

Meet-in-the-middle attacks: Meet-in-the-middle attack exploits the property that a part of key does not appear during a certain number of rounds. The designers and the follow-up work by Sasaki showed the attack against 15-rounds of GIFT-64 and mentioned the difficulty of applying it to GIFT-128 because of the larger ratio of the number of round key bits to the entire key bits per round; each round uses 32 bits and 64 bits of keys per round in GIFT-64 and GIFT-128, respectively, while the entire key size is 128 bits for both.

Security of ISAP

All Isap family members provide 128-bit security against cryptographic attacks in the notion of nonce-based authenticated encryption with associated data (AEAD): they protect the confidentiality of the plaintext (except its length) and the integrity of ciphertext including the associated data (under adaptive forgery attempts).

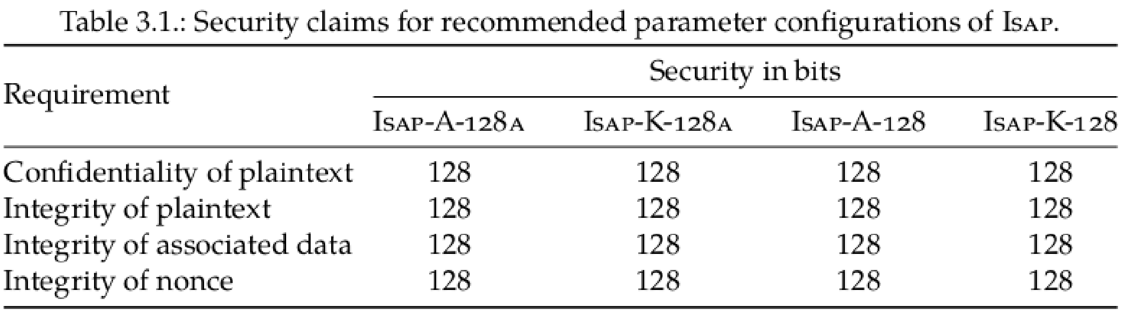


Table 3.1.: Security claims for recommended parameter configurations of Isap.

Requirement	Security in bits			
	Isap-A-128a	Isap-K-128a	Isap-A-128	Isap-K-128
Confidentiality of plaintext	128	128	128	128
Integrity of plaintext	128	128	128	128
Integrity of associated data	128	128	128	128
Integrity of nonce	128	128	128	128

In order to fulfill the security claims stated in above table, implementations must ensure that the nonce is never repeated for two encryptions under the same key, and that the decryption process is only started after successful verification of the final tag. Except for the single-use requirement, there are no constraints on the choice of the nonce. It is possible to use a simple counter. It is beneficial that a system or protocol implementing the algorithm monitors and, if necessary, limits the number of tag verification failures per key. After reaching this limit, the decryption algorithm rejects all tags. Such a limit is not required for the security claims above but may be reasonable in practice to increase the robustness against certain implementation attacks.

Isap is designed to improve robustness against other implementation attacks, including certain fault attacks.

Performance of GIFT-COFB

Hardware

COFB mode was designed with rate 1, that is every message block is processed only once. Such designs are not only beneficial for throughput, but also energy consumption. However, the design does need to maintain an additional 64-bit state, which requires a 64-bit register to additionally included in any hardware circuit that implements it. Although this might not be energy efficient for short messages, in the long run COFB performs excellently with respect to energy consumption. The GIFT-128 block cipher was designed with a motivation for good performance on lightweight platforms. The round key addition for the cipher is over only half the state and the key schedule being only a bit permutation does not require logic gates.

A component-wise breakup of the circuit is given. The power consumed at an operating frequency is 156.3 μW.

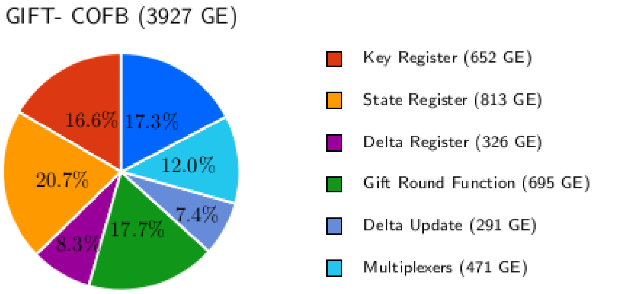
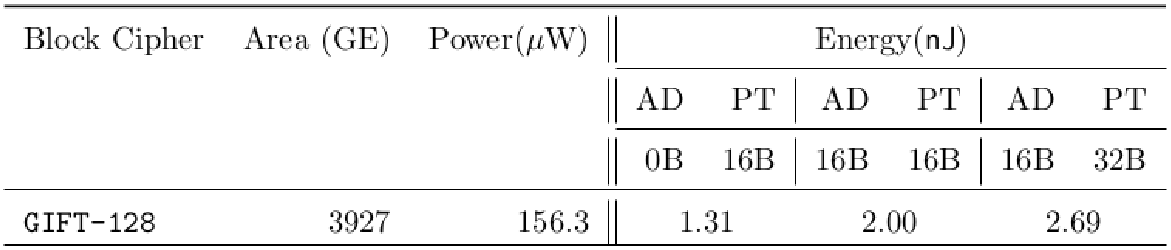


Figure 3.2: Component-wise breakup of the GIFT-COFB circuit

The energy consumption figures for various lengths of data inputs are given in the table.



Block Cipher	Area (GE)	Power(μW)	Energy(nJ)					
			AD	PT	AD	PT	AD	PT
			0B	16B	16B	16B	16B	32B
GIFT-128	3927	156.3	1.31		2.00		2.69	

Table 3.1: Implementation results for GIFT-COFB. (Power reported at 10 MHz)

Performance of ISAP

In terms of performance, ISAP can improve the speed and efficiency of web applications by allowing them to communicate directly with the web server, rather than going through an intermediary. This can result in faster response times and improved scalability, as the web server is able to handle requests more efficiently.

There are a few key factors that can impact the performance of an ISAP application:

Network latency: The speed at which data is transmitted over the internet can impact the performance of an ISAP application.

Server hardware and software: The hardware and software of the server hosting the ISAP application can also impact its performance.

Application design: The design of the ISAP application itself can have a significant impact on its performance. Factors such as the complexity of the application, the number of requests it handles, and the amount of data it processes can all affect its performance.

User workload: The workload of the users of the ISAP application can also impact its performance. If there are a large number of users accessing the application at the same time, it may be slower to respond to requests.

Overall, the performance of an ISAP application will depend on a combination of these factors, and it is important to carefully consider and optimize each of them in order to ensure the best possible performance.

Main differences

ISAP (Internet Server Application Programming Interface) is a technology that enables applications to communicate with web servers using a common set of protocols, while GIFT-COFB (Gradient Index Fiber) is a type of fiber optic cable that is used for transmitting data over long distances.

There are several key differences between these two technologies:

Purpose: ISAP is designed to facilitate communication between applications and web servers, while GIFT-COFB is used for transmitting data over long distances.

Protocols: ISAP uses a set of standardized protocols to facilitate communication between applications and web servers, while GIFT-COFB uses a different set of protocols specifically designed for transmitting data over fiber optic cables.

Performance: ISAP can improve the performance of web applications by allowing them to communicate directly with the web server, while GIFT-COFB can improve the performance of data transmission by providing high bandwidth and low latency.

Applications: ISAP is typically used in web applications, while GIFT-COFB is used in a variety of applications including telecommunications, internet service providers, and cable television.

In summary, ISAP and GIFT-COFB are two different technologies that serve different purposes. ISAP is used for facilitating communication between applications and web servers, while GIFT-COFB is used for transmitting data over long distances.