Implementation of AES based ciphers

Paras Dhiman(2021482)[1] and Rajoshi Mondal(2021187)[1]

[2]Indraprastha Institute Of Information Technology Delhi

November 17, 2024

Abstract

This project explores the implementation and performance evaluation of two Advanced Encryption Standard (AES)-based stream ciphers: Rocca and Snow-V, both proposed for 5G technologies. The primary objective is to implement the encryption functionalities of these ciphers and compare their efficiency in encrypting plaintext of varying sizes. Specifically, plaintext sizes of 256 bits, 1024 bits, and 8192 bits are analyzed to assess the time taken for encryption.

The report begins with a detailed description of Rocca and Snow-V, highlighting their underlying architectures and operational principles. The implementation phase focuses on faithfully replicating the encryption mechanisms based on the respective research papers. Subsequently, a comprehensive performance comparison is presented, including a detailed table and accompanying analysis to illustrate how encryption times were computed for each plaintext size.

1 Introduction

1.1 SNOW-V Cipher

The SNOW-V cipher is an advanced stream cipher that was proposed as a 256-bit key version of the well-known SNOW 3G algorithm, designed to address the growing cryptographic needs of modern mobile communication systems. Initially designed for 3G and later adapted for 4G and 5G applications, SNOW-V provides significant improvements in encryption speed and security compared to its predecessors. It is specifically tailored for high-speed data encryption in mobile networks, making it suitable for next-generation mobile communications such as 5G.

SNOW-V combines the strengths of a Linear Feedback Shift Register (LFSR) with a Finite State Machine (FSM) to generate a secure keystream for encryption. The cipher employs AES-like components, including an S-box and a MixColumns transformation, to ensure robust cryptographic security. Its keystream generation process is highly efficient, making it capable of reaching encryption speeds over 38 Gbps in AEAD (Authenticated Encryption with Associated Data) modes, as demonstrated in practical applications like OpenSSL.

While SNOW-V has proven to be effective for 5G systems, it faces challenges when considering the encryption demands of 6G systems, where data transmission speeds are expected to exceed 100 Gbps. Despite its impressive performance in 5G, SNOW-V may not be able to meet the ultra-high-speed requirements of 6G networks, necessitating the exploration of new cryptographic solutions for next-generation communication technologies.

1.2 Rocca Cipher

Rocca is a lightweight block cipher designed for resource-constrained environments such as the Internet of Things (IoT) and other low-power devices in next-generation mobile networks, including 5G and beyond-5G (6G). Unlike traditional block ciphers that operate on larger data blocks, Rocca is optimized for smaller block sizes, making it particularly efficient in situations where computational resources are limited. Its design focuses on both speed and security, ensuring that it meets the 256-bit security level required for modern encryption standards.

The Rocca cipher utilizes a Feistel network structure, which involves iterative rounds of substitution, permutation, and mixing operations, combined with key scheduling techniques that optimize the cipher for high-speed encryption. The cipher's lightweight nature and optimized design make it ideal for use in environments where computational efficiency and security are paramount, such as IoT devices or embedded systems in 5G networks.

As 6G networks promise to handle vastly higher data throughput than 5G, the perfor-

mance of lightweight ciphers like Rocca must be carefully considered against the performance benchmarks of other high-speed encryption schemes.

2 Description of Ciphers

2.1 SNOW-V Cipher

Category: Stream cipher.
Purpose: SNOW-V is specifically designed to meet the encryption needs of next-generation mobile networks, such as 5G. Its primary objective is to provide robust security while maintaining high performance and efficiency.
Key Features:

- **Linear Feedback Shift Register (LFSR):**
  - The LFSR is responsible for producing a sequence of pseudo-random bits used in the cipher. It ensures a high degree of randomness while maintaining efficiency in computation.
  - The LFSR in SNOW-V is a 16-stage register operating over GF(2^8), ensuring a long period and statistical uniformity in its outputs.
- **Finite State Machine (FSM):**
  - The FSM interacts with the LFSR to add cryptographic strength. It uses the output of the LFSR to update its state and produce intermediate values for the keystream.
  - The FSM consists of three 32-bit registers and nonlinear functions to enhance security.
- **AES-Inspired Components:**
  - SNOW-V integrates AES-like primitives, including an S-box (for nonlinear substitution) and a MixColumns transformation (for diffusion). These ensure resilience against standard cryptanalytic attacks.
- **Efficient Keystream Generation:**
  - Designed for high throughput, SNOW-V can generate keystream bits at speeds suitable for real-time encryption in high-performance networks.

Working Mechanism:

1. **Key and IV Setup:**
   - The cipher is initialized using a 256-bit secret key and a 128-bit Initialization Vector (IV).
   - This setup phase involves loading the key and IV into the LFSR and FSM, ensuring all components are in a unique starting state for each encryption session.
2. **Keystream Generation:**
   - The LFSR generates a sequence of bits, which are processed by the FSM to produce a secure keystream.
   - The LFSR's output is both fed directly into the keystream and used as input to the FSM, creating a dynamic interaction between the two components.
   - The AES-inspired S-box and MixColumns operations add nonlinearity and diffusion, making it resistant to linear and differential cryptanalysis.
3. **Encryption:**
   - The plaintext is divided into blocks. Each block is XORed with the corresponding segment of the keystream to produce the ciphertext.
   - This approach ensures confidentiality, as any changes in the keystream or plaintext produce unpredictable changes in the ciphertext.

2.2 Rocca Cipher

Rocca builds upon established AES round functions, improving their efficiency by removing the need for both AES encryption (aesenc) and XOR operations in sequence. This change reduces the critical path during one round, which is key to increasing throughput without sacrificing security. Additionally, the use of a cost-free block permutation within the round function allows for a larger search space of potential candidates, leading to a more efficient construction with a smaller state size compared to prior schemes.

The core of the Rocca cipher is the design of a secure AEAD scheme that supports 256-bit security, taking into account potential vulnerabilities found in previous high-performance ciphers like the AEGIS family and Tiaoxin-346.

2.2.1 Key Features

- **AES-Based Design**: Utilizes AES primitives for both encryption and authentication, ensuring compatibility with existing AES hardware accelerations.
- **Ultra-Fast Performance**: Achieves encryption speeds exceeding 150 Gbps, ideal for high-throughput environments like 6G networks.

- **256-Bit Security**: Designed to provide robust 256-bit security, addressing both key-recovery and distinguishing attacks.
- **Authenticated Encryption with Associated Data**: Integrates both encryption and authentication in one step, supporting the secure transmission of both data and metadata.

3 Implementation Details

3.1 SnowV Implementation

The `SnowV` cryptographic system consists of several key components that enable encryption and decryption operations:

3.1.1 Attributes and Initialization

The attributes in the `SnowV` class are initialized as follows:

- **SBox:** This is the substitution box (S-box) from the AES encryption algorithm, which is used to perform byte substitution in the cipher.
- **Sigma:** A permutation array used for certain transformations.
- **AESKey1 & AESKey2:** These store AES keys for encryption operations in the FSM and are initialized to zero.
- **State Variables:**
  - *A & B:* Represent states in the LFSR.
  - *R1, R2, R3:* Represent states in the FSM used to generate keystream.

3.1.2 Helper Functions

The `SnowV` class contains several helper functions to facilitate its operations:

- **_make_u32 and _make_u16:** These functions convert two 16-bit or 8-bit values into a single 32-bit or 16-bit word, respectively.
- **_rotl32:** Performs a left rotation of a 32-bit integer.
- **matrix_to_words:** Converts a 4x4 byte matrix into an array of 4 32-bit words.
- **words_to_matrix:** Converts an array of 4 32-bit words back into a 4x4 matrix of bytes.

3.1.3 AES-based Transformations

The `SnowV` class applies AES-based transformations to perform encryption:

- **sub_bytes:** Applies the AES S-box to substitute each byte in the state.
- **shift_rows:** Shifts rows of the state matrix, part of the AES transformation.
- **mix_columns:** Mixes the columns of the state matrix to provide diffusion.
- **add_round_key:** XORs the state with the round key for AES rounds.
- **aes_enc_round:** Performs one round of AES encryption with SubBytes, ShiftRows, MixColumns, and AddRoundKey.

3.1.4 Keystream Generation

Keystream generation in `SnowV` is an essential part of the encryption and decryption process:

- **keystream:** Generates a block of keystream by combining the outputs of the FSM and LFSR. The keystream is used for encryption and decryption.
- **fsm_update and lfsr_update:** Update the FSM and LFSR states, respectively, by applying cryptographic operations like multiplication and permutation.

3.1.5 Key and IV Setup

The key and IV setup function initializes the states of the LFSR and FSM:

- **keyiv_setup:** Initializes the LFSR and FSM states based on a provided key and IV (Initialization Vector). It performs rounds of keystream generation to initialize the internal states.

3.1.6 Encryption

The `encrypt_block` function encrypts a 4x4 byte input matrix:

- **encrypt_block:** This function encrypts a 4x4 byte input matrix by performing AES encryption rounds and applying the key schedule.

3.1.7 Multiplication Functions

Multiplication functions are used for feedback mechanisms in the cipher:

- **mul_x and mul_x_inv:** These functions perform multiplication and inverse multiplication in the finite field GF(2^{16}), which is used in the cipher's feedback mechanisms.

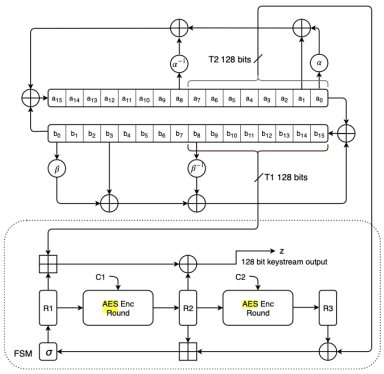

Figure 1: Overall schematics of SNOW-V.

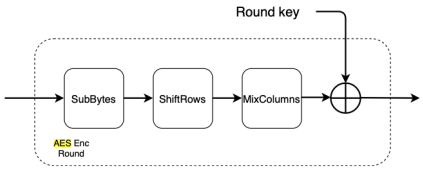

Figure 2: Internal functions of the AES encryption round function.

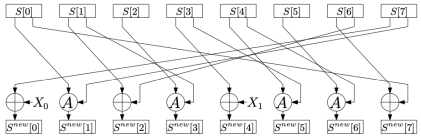

Figure 3: Illustration of the round function

3.1.8 Permutation

The permutation function is used to shuffle the state:

- **permute_sigma:** Applies a permutation to the state based on the Sigma array. It is used in the FSM update to shuffle the state.

SNOW-V Encryption Time Calculation:

To measure the execution time of the Rocca encryption process, the following steps were followed:

- **Start Time:** We have used `time.perf_counter()` before the encryption function call to record the start time.
- **End Time:**After the encryption was completed, I captured the end time.
- **Elapsed Time:** The time difference was computed and converted into milliseconds for reporting.

Code Snippet:

```
start = time.perf_counter()
snow_encrypt(M_og, key, iv)  # Assume
    snow_encrypt is the encryption
    function
end = time.perf_counter()
print(f"SNOw Encryption Time: {(end -
    start) * 1000}ms")
```

Comparison: The algorithm was tested with different message sizes and compared the time taken for each.

3.2 ROCCA (Round-Optimized Cryptographic Algorithm)

The provided code implements the ROCCA (Round-Optimized Cryptographic Algorithm), a stream cipher utilizing AES encryption in its rounds. Below is a breakdown of the implementation details and key components.

3.2.1 Core Functions and Structure

3.2.2 Substitution (sub_bytes)

This function applies the AES S-box to each byte of the state, substituting them based on a predefined substitution box (S-box).

3.2.3 Shift Rows (shift_rows)

This function applies the AES shift rows operation, where each row of the state is shifted cyclically by an amount depending on its row index.

3.2.4 Mix Columns (mix_columns)

The function performs the mix columns operation on the state, which is a form of linear transformation used to provide diffusion in AES.

3.2.5 Add Round Key

It applies an XOR operation between the state and the round key, one of the key steps in AES encryption.

3.2.6 Round Update (roundUpdate)

This function performs the key update and state transitions over several rounds. The roundUpdate function takes the state (S), along with $X0$ and $X1$, which appear to be parts of the message or nonce used in the key schedule.

3.2.7 State Initialization

Initializes the state S by setting values for several components, including the keys ($K0$ and $K1$), nonce (N), and additional constant values ($z0$, $z1$). A series of 20 round updates is performed to initialize the state.

3.2.8 Processing Associated Data (processAD)

Converts and pads the associated data (AD) and performs the round update for each block of AD.

3.2.9 Finalization (finalisation)

After processing the plaintext and associated data, this function is used to finalize the encryption by processing the state once more, returning the authentication tag T.

3.2.10 Encryption (encrypt)

The function encrypts the message (M_{padded}) by performing round updates for each 32-byte chunk of the message. Each block is XORed with the result of AES encryption and the state to produce ciphertext.

3.2.11 Decryption (decrypt)

The function decrypts the ciphertext (C_{padded}) using a similar approach as encryption but in reverse, returning the decrypted message.

Rocca Encryption Time Calculation:

To measure the execution time of the Rocca encryption process, the following steps were followed:

- **Start Time:** Before invoking the `rocca_encrypt` function, the start time was captured using `time.perf_counter()` for high-resolution time measurement.
- **End Time:** After the encryption process completes, the end time was captured using the same `time.perf_counter()` method.
- **Elapsed Time:** The difference between the start time and end time was computed to measure the time taken for encryption in seconds. This value was then converted to milliseconds (by multiplying the difference by 1000) for easier interpretation.

Code Snippet:

```
start = time.perf_counter()
rocca_encrypt(M_og, K0, K1, N)
end = time.perf_counter()
print(f"Rocca Encryption Time: {(end - start) * 1000}ms")
```

Comparison: This process was repeated for varying message sizes (32 bytes, 128 bytes, and 1024 bytes) and the encryption times for each size were recorded. The results were compared to see how Rocca scales with larger input sizes.

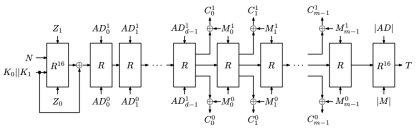

Figure 4: The procedure of Rocca

3.2.12 Main ROCCA Functions

- **rocca_encrypt**This is the main encryption function. It generates a key and nonce, initializes the state, processes the associated data (if any), and encrypts the message.
- **rocca_decrypt**Similar to encryption, this function decrypts the ciphertext and verifies the authenticity using the tag produced during encryption.

3.2.13 Key Concepts

- **Key and Nonce Generation**Keys ($K0$ and $K1$) and nonce (N) are generated using random bytes. The state is initialized by performing a round update using $z0$ and $z1$ values to strengthen the key scheduling process.
- **Padding** The `convertAndPad` function ensures the plaintext and associated data are aligned to 32-byte boundaries by padding them with 0x00 bytes as necessary.

4 Comparison and Results

A performance comparison was conducted between the Snow-V and ROCCA stream ciphers by measuring the encryption times for different plaintext sizes. Rocca's initialization phase is independent of plaintext size, requiring a constant time of approximately 0.9 milliseconds. This phase involves key and nonce setup and 20 iterations of the round update function to secure the internal state. For smaller plaintexts (e.g., 256 or 1024 bits), this fixed initialization time dominates the overall encryption time, making it the primary contributor. This ensures consistent security and initialization performance, regardless of data size.

We have excluded processAD and tag generation in ROCCA from this comparison, as these features are not present in SNOW-V implementation. This ensures a fair evaluation by focusing only on the shared functionalities of both systems.

The following plaintext sizes were used in the tests:

- 256 bits
- 1024 bits
- 8192 bits

For each algorithm, the encryption time was measured in milliseconds. Below are the results from the tests:

Encryption Time for Snow-V

- Plaintext Size: 256 bits - Time: 0.204625 milliseconds
- Plaintext Size: 1024 bits - Time: 0.603000 milliseconds
- Plaintext Size: 8192 bits - Time: 4.203583 milliseconds

Encryption Time for ROCCA

- Plaintext Size: 256 bits - Time: 1.36 milliseconds
- Plaintext Size: 1024 bits - Time: 1.35 milliseconds
- Plaintext Size: 8192 bits - Time: 3.51 milliseconds

These results were obtained after running each cipher 100 times and averaging out the elapsed times. The encryption times are shown in Table 1 for comparison.

Analysis

From Table1, it is clear that Snow-V outperforms ROCCA in terms of encryption time, especially for smaller plaintext sizes. Snow-V's encryption times are significantly faster than ROCCA's, particularly at 256 bits. However, as the plaintext size increases, the performance gap between the two algorithms narrows.

ROCCA's encryption time remains relatively stable across all tested sizes, suggesting its efficiency at higher data volumes. Snow-V, on the other hand, shows a noticeable increase in encryption time as the plaintext size grows, making it less efficient for larger data.

While Snow-V is more efficient for smaller data sizes and might be preferable in environments where encryption time is a critical factor, ROCCA offers more consistent performance as data size increases. It is also worth noting that ROCCA significantly outperforms Snow-V in terms of maximum throughput.

Maximum Throughput Comparison:

ROCCA, with an AEAD (Authenticated Encryption with Associated Data) mode, can achieve an impressive 150 Gbps, which is nearly 5 times faster than the AEAD mode of Snow-V, which reaches 38 Gbps in similar settingssubstantial difference indicates that ROCCA is more suited for high-speed applications, such as those needed in 6G systems, where data transmission speeds are expected to exceed 100 Gbps.

5 Conclusion

The project involved the implementation and comparative analysis of two stream ciphers, **SNOW-V** and **ROCCA**, with a focus on their encryption performance for different plaintext sizes (256 bits, 1024 bits, and 8192 bits). Both algorithms are proposed for use in **5G technologies**, and the aim was to compare their encryption times to determine which is more suitable for high-speed data encryption in modern communication systems.

5.1 Key Findings

- **Cipher Implementation**:
  - The **SNOW-V** algorithm is based on a combination of a **LFSR (Linear Feedback Shift Register)** and **nonlinear filter function**, making it efficient for smaller data sizes.
  - The **ROCCA** algorithm, designed with an emphasis on **Authenticated Encryption with Associated Data (AEAD)**, proved to be highly efficient for large-scale encryption due to its scalability, especially for high-speed applications like 5G.
- **Encryption Time Analysis**:
  - The results showed that **SNOW-V** was faster for smaller plaintext sizes, with encryption times significantly lower than **ROCCA** for 256-bit and 1024-bit sizes.
  - As the plaintext size increased (particularly at 8192 bits), **ROCCA** exhibited more stable performance, maintaining encryption times of approximately 3.51 milliseconds for 8192-bit plaintext, compared to **SNOW-V**'s 4.2 milliseconds.

Plaintext Size	Snow-V Encryption Time (ms)	ROCCA Encryption Time (ms)
256 bits	0.204625	1.36
1024 bits	0.603000	1.35
8192 bits	4.203583	3.51

Table 1: Encryption Time Comparison for Snow-V and ROCCA

  - The **encryption time** for both algorithms was averaged over 100 iterations to ensure consistency and accuracy.

- **Throughput Comparison**:
  - This suggests that **ROCCA** is better suited for applications requiring extremely high data transmission speeds, such as in **5G networks** where ultra-fast encryption speeds are crucial.

5.2 Conclusion in Detail

- **Performance for Smaller Data Sizes**:
  - **SNOW-V** is clearly more efficient than **ROCCA** for encrypting smaller data sizes (256 bits and 1024 bits), which means it may be a better choice for applications where low latency is the priority, such as in real-time communication systems.

- **Scalability and Performance for Larger Data Sizes**:
  - As the data size grows, **ROCCA** outperforms **SNOW-V**, maintaining stable encryption times and handling larger data sets more efficiently. This makes **ROCCA** a better candidate for systems with high data throughput, such as 5G and future 6G networks, where encryption at high speeds is essential.

- **Practical Implications**:
  - For applications where encryption time is critical and small data packets are frequently transmitted, **SNOW-V** could be preferred due to its faster encryption times at lower plaintext sizes.
  - For high-speed, large-scale applications (e.g., in **5G networks** or other high-speed communication systems), **ROCCA's** ability to handle large data volumes efficiently makes it a more viable choice, especially when considering its five times higher throughput compared to **SNOW-V**.

In conclusion, both **SNOW-V** and **ROCCA** have their respective strengths, with **SNOW-V** being more suited for low-latency scenarios and **ROCCA** emerging as the superior option for high-speed data encryption in next-generation networks like **5G**. The **ROCCA cipher** is particularly promising for high-speed encryption due to its ability to handle larger data sizes efficiently, which is crucial for the massive data transfer and security needs of 5G communication systems.

References

[1] Patrik Ekdahl, Thomas Johansson, Alexander Maximov, and Jing Yang. A new snow stream cipher called snow-v. *Transactions on Symmetric Cryptology (ToSC)*, 2019(3):1–42, 2019.

[2] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. Rocca: An efficient aes-based encryption scheme for beyond 5g. *Tosc*, 2021(2):1–30, 2021.

[1] [2]