A logo with a butterfly

Description automatically generated

**CSE 470 CRYPTOGRAPHY AND COMPUTER SECURITY**

**FALL 2023**

**HOMEWORK 2**

**MERT GÜRŞİMŞİR**

**1901042646**

**EXPLANATION OF LIGHTWEIGHT SYMMETRIC ENCRYPTION ALGORITHMS AND NEWLY PROPOSED ALGORITHMS**

Ligthweight cryptography is an ecryption method that is used for small devices (IoT or network of devices because these devices mostly connected to network but not to internet) which has low computational capability. It is a cryptographic algorithm or protocol that is used for RFID tags, sensors, contactless smart cards, health-care devices, etc.

There are a few factors that should be considered while implementing lightweight cryptography which are as follows: size, battery life, latency.

A diagram of a computer hardware system

Description automatically generated

Algorithm is lightweight because data is encrypted by the sensor and sensor is very small device. Data is then decrypted by the server. If you hardened the security, device cost is increased and nobody wants it for small devices. These small devices may not be capable of running encryption algorithms that is designed for phone, computers, etc.

Ligthweight symmetric cryptography uses symmetric key that means it uses same secret key for encryption and decryption. With a system such as a plant or car-control system, it may be possible to embed the secret keys shared by the devices in advance. With this way, both devices use the same key without sharing it that causes an attacker to steal the key. In such a case, secure and efficient data protection can be implemented using symmetric key cryptography alone.

We must focus on confidentially, authentication, access control, privacy, network security, trust, mobile security, secure middleware, and data security while designing a lightweight algorithm.

It is good to mention that lightweight cryptography does not mean short keys. We need lightweight cryptography. In the past there is one computer but many users, then one computer and one users, and today we have many computers and one user (IoT).

Initial lightweight designs mainly focused on low hardware footprint but it is not realistic to have a single cipher to satisfy all needs. To fit within constrained settings, lightweiht ciphers rely on simpler round functions, or minimal key schedules. The simpler structure means the more vulnerable.

There are some lightweight algorithms that are focused on low hardware footprint. They are as follows:

* PRESENT (ISO/IEC Lightweight Block Cipher Standard)
* HIGHT (ISO/IEC Lightweight Block Cipher Standard)
* CLEFIA (ISO/IEC Lightweight Block Cipher Standard)
* LED
* KATAN

There are also designs that low memory consumption on small embedded processors:

* ITUBee
* PRIDE
* SPECK

PRINCE is focused on low latency, Zorro and LS-Designs focus on ease of side-channel protection. Enocoro (key sizes 80 or 128 bits) and Trivium (key size 80 bits) become ISO lightweight stream cipher standards. But 80 bit security is not enough nowadays. We have ISO standards but not NIST standards.

Having a simple algorithm satisfying all of these needs is a very hard process. For this reason, NIST said that we should have a competition.

Evaluation criteria of NIST is as follows:

* Physical
  + Area
  + Memory
  + Implementation
  + Energy
* Performance
  + Latency
  + Throughput
  + Power
* Security
  + Minimum bit security
  + Attack models
  + Side channel resistance

In summary, IoT devices are very different than each other so it is hard to provide standards for all. Current device production does not focus on security. Producers should provide their own security solutions until IoT standards are available. Due to their simplicity, lightweight designs may be weak against attack types that are not discovered yet. Lightweight does not mean shorter key as I have mentioned earlier. Using short keys provides almost no security. There is no need to use keys shorter than 128 bits.

We can write down the general framework for analyzing and comparing lightweight symmetric encryption algorithms:

Performance:

* Speed: How quickly can the algorithm encrypt and decrypt data? This is especially important for devices with limited computing power.
* Code Size: How much space does the algorithm's code take up? Smaller code is crucial for devices with tight memory constraints.
* Memory Usage: How much RAM does the algorithm need? Lightweight devices often have limited memory.
* Energy Consumption: How efficiently does the algorithm use energy, especially for devices running on batteries?

Security:

* Key Length and Strength: How secure is the algorithm in terms of key length and cryptographic strength? It should resist known attacks.
* Resistance to Attacks: How well does the algorithm defend against common cryptographic attacks, like differential and linear cryptanalysis?
* Cryptography Primitives: What cryptographic techniques does the algorithm use, and how secure are they?

Flexibility:

* Adaptability: How well does the algorithm adjust to different scenarios and use cases?
* Key Management: How easy is it to manage keys with the algorithm, and can it support various key establishment mechanisms?
* Algorithm Agility: Can the algorithm be updated or modified easily without compromising security?

Standards Compliance:

* Compliance with Standards: Does the algorithm adhere to established cryptographic standards for interoperability?
* Certifications: Has the algorithm received certifications or validations from recognized cryptographic authorities?

Implementation Considerations:

* Platform Independence: Is the algorithm suitable for different platforms, including both hardware and software implementations?
* Resource Constraints: How well does the algorithm perform under resource limitations common in lightweight devices?

Innovation and Novelty:

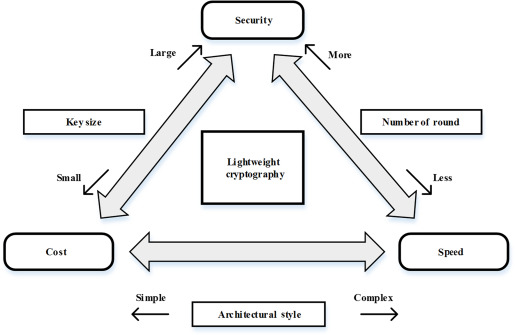
* New Techniques: Does the algorithm introduce new and innovative techniques or approaches?
* Advancements: How does the algorithm address or improve upon the limitations of existing algorithms?

Community Feedback and Adoption:

* Community Support: What are others saying about the algorithm? Check for community feedback, reviews, and general support.
* Adoption: Is the algorithm gaining popularity and being used in real-world applications?

Benchmarking:

* Benchmark Results: Look at independent benchmark results to see how the algorithm's performance compares to others in the same category.



**ASCON**

ASCON was one of the finalist algorithms. It has been selected as new standard for lightweight cryptography in the NIST Lightweight Cryptography competition (2019–2023).

ASCON is a permutation based AEAD (Authenticated Encryption with Associated Data) using sponge construction and hashing scheme. It performs additional key additions during initialization and finalization. ASCON-hash is based on the duplex sponge construction. ASCON was selected as the primary choice for lightweight authenticated encryption in the final portfolio of the CAESAR competition (2014–2019).

* The core of ASCON is a permutation function that processes the data in fixed-size blocks.
* The permutation function is designed to resist various cryptographic attacks, including differential and linear cryptanalysis.

Its block size can be 64 or 128 bits. State size is 320 bits. Key is 128 bits. None is 128 bits. Tag is 128 bits. Algorithm also has 12 rounds. Instead of table look-up, algorithm uses XOR operations. ASCON is also using 5x5 S-box. It does 64 S-box operation.

A diagram of a machine

Description automatically generated

*The encryption of ASCON. pa means the permutation operation p is performed a times. We have a = 12 and b = 6.*

ASCON supports different modes of operation, including a dedicated authenticated encryption mode. The AEAD (Authenticated Encryption with Associated Data) mode allows for both encryption of the plaintext and authentication of the ciphertext and associated data.

ASCON is designed to provide strong security guarantees, including resistance against known cryptographic attacks. The algorithm aims to be secure even in the face of side-channel attacks, where an attacker may gain information from the physical implementation of the algorithm.

My topics are GRAIN128-AEAD and ROMULUS. Although I wanted to mention about ASCON so let’s dive into my topics.

**GRAIN128-AEAD**

Grain128-AEAD (Authenticated Encryption with Associated Data) provides AEAD-only functionality and it is stream cipher based. It is a is a family of lightweight stream ciphers designed for hardware and software implementations in constrained environments. The primary motivation behind GRAIN-128 is to provide a secure and efficient solution for resource-constrained devices, such as RFID tags and sensor nodes.

Grain128-AEAD is a bit-oriented feedback shift register based AEAD scheme optimized for hardware implementations. The AEAD variant of the Grain-128AEAD family is listed below:

A close up of a sign

Description automatically generated

Analysis showed that the secrecy of the key is not fully protected by reintroducing the key.

The AEAD construction typically involves combining the output of the stream cipher with message authentication code (MAC) techniques. This way, it not only encrypts the data but also provides a means to verify the integrity of the decrypted message. GRAIN-128-AEAD is designed to be efficient in terms of computational resources, making it suitable for use in resource-constrained environments where lightweight cryptography is essential.

AEAD requirements are as follows:

* Grain-128AEAD takes a variable-length plaintext, variable-length associated data, a fixed-length nonce (IV) of size 96 bits, and a fixed-length key of size 128 bits. The output is a variable length ciphertext. The plaintext is recovered from a valid ciphertext. An invalid ciphertext does not return a plaintext.
* For a single key, the nonce must be unique. If the nonce is not unique, i.e., it is repeated for the same key, the algorithm leaks information about the two plaintext, and the MAC can be forged.
* The Grain-128AEAD is one algorithm with the only supported parameters are 128-bit key and 96-bit nonce.
* Grain-128AEAD is a bit oriented stream cipher and it thus also allows byte string inputs. The message padding of one ’1’ bit, can in an environment that only operates with bytes, be replaced by a ’1’ followed by seven ’0’s. This will not affect the MAC result.
* Grain-128AEAD has a keystream limitation of 280 bits, i.e., a pre-output stream limitation of 281 bits.

Grain-128AEAD consists of two main building blocks. The first is a pre-output generator, which is constructed using a Linear Feedback Shift Register (LFSR), a Non-linear Feedback Shift Register (NFSR) and a pre-output function, while the second is an authenticator generator consisting of a shift register and an accumulator.

A diagram of a machine

Description automatically generated

*An overview of the initialization in Grain-128AEAD*

The GRAIN-128 stream cipher consists of two main components: the initialization function and the keystream generation function. Description of the algorithm as follows:

* Key and IV setup:
  + The algorithm takes a 128-bit key and a 96-bit initialization vector (IV) as input. The key and IV are used to initialize the internal state of the cipher.
* Initialization function:
  + The initialization function performs a number of steps to set up the internal state. It involves bitwise operations, modular addition, and permutation of the state bits.
* Keystream generation function:
  + The keystream is generated by repeatedly updating the internal state. The state is updated using feedback from different parts of the state. The keystream is extracted from the state and used to encrypt the plaintext.
* Feedback mechanism:
  + The cipher employs a feedback mechanism where parts of the internal state are combined to produce the next state. Nonlinear feedback functions and linear feedback shift registers contribute to the complexity of the keystream generation.
* Cryptographic operations:
  + The algorithm involves bitwise operations, modular addition, and permutation to achieve cryptographic diffusion and confusion.

GRAIN-128 is designed to be a lightweight stream cipher, optimized for efficient implementation on resource-constrained devices. It provides a balance between security and performance in such environments.

**ROMULUS**

Romulus is an AEAD scheme based on the tweakable block cipher SKINNY. Romulus consists of two families: a nonce-based AEAD RomulusN and a nonce misuse-resistant AEAD Romulus-M. Romulus-N uses a rate-1 TBC-based combined feedback mode, and the mode of Romulus-M follows a MAC-then-encrypt approach.

Romulus cipher suite offers following functionalities, which are all based on Skinny-128-384+ tweakable block cipher.

Variant:

* Romulus-H
  + Only cryptographic hash function offered by Romulus cipher suite
  + Given N -bytes input message, this algorithm computes 32 -bytes digest | N >= 0
* Romulus-N
  + A nonce-based authenticated encryption with associated data scheme, which is the primary AEAD candidate of this cipher suite
* Romulus-M
  + A nonce misuse-resistant authenticated encryption with associated data scheme
* Romulus-T
  + A leakage-resistant authenticated encryption with associated data scheme

Romulus-{N, M, T}:

* encrypt: Given 16 -bytes secret key, 16 -bytes public message nonce, N -bytes associated data & M -bytes plain text, the encryption algorithm computes M -bytes cipher text and 16 -bytes authentication tag
* decrypt: Given 16 -bytes secret key, 16 -bytes public message nonce, 16 -bytes authentication tag, N -bytes associated data & M -bytes cipher text, the decryption algorithm computes M -bytes plain text and boolean verification flag. If authentication check fails i.e. boolean verification flag is false, unverified plain text is not released, more explicitly plain text bytes are zeroed.

Romulus consists of Romulus-N, Romulus-M, Romulus-T and RomulusH. Romulus-N implements nonce-based AE (NAE) secure against Nonce-respecting adversaries, and Romulus-M implements nonce Misuse-resistant AE (MRAE) introduced by Rogaway and Shrimpton. Romulus-T implements an updated version of the leakage-resilient AE TEDT. Romulus-H implements a Hash function with standard security properties, such as collision resistance and (2nd) preimage resistance. The name Romulus stands for the set of these members. The primary AEAD member of our submission is Romulus-N, and Romulus-H is our only hash function member.

Romulus-N, Romulus-M and Romulus-T have the following parameters:

• Nonce length nl = 128.

• Message block length n = 128.

• Key length k = 128.

• Counter bit length d = 56.

• Tag length τ = 128.

The encryption algorithm of Romulus-N, Romulus-M and Romulus-T takes a key K, nonce N, associated data (AD) A and message M as input, and returns a ciphertext C and tag T. The decryption algorithm of Romulus-N, Romulus-M and Romulus-T takes (K, N, A, C, T) as input, and returns M or ⊥ indicating rejection.

Romulus is a family of algorithms designed to achieve lightweight hardware performance while maintaining competitive security guarantees and software performance.

A diagram of a state and state

Description automatically generated

**RESOURCES**

<https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/grain-128aead-spec-final.pdf>

<https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/grain-128aead-spec-round2.pdf>

<https://grain-128aead.github.io/C2SI2019_Grain_128AEAD.pdf>

<https://csrc.nist.gov/projects/lightweight-cryptography/finalists>

Status Report on the Second Round of the NIST Lightweight Cryptography Standardization Process

<https://grain-128aead.github.io/>

<https://romulusae.github.io/romulus/>

<https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/romulus-spec-final.pdf>