

T.C

GEBZE TECHNICAL UNIVERSITY

COMPUTER ENGINEERING

CSE 470 Fall 2024

Homework 2024

Due date: Dec 27th, 2024

|  |  |
| --- | --- |
| **Name/Surname** | Ziya Kadir TOKLUOĞLU |
| **Number** | 210104004228 |
| **Department** | Computer Engineering |
| **Advisor** | İbrahim SOĞUKPINAR |

**Outline**

1. **Ascon**
2. **Elephant**
3. **Xoodyak**

**Ascon**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Step** | **Notation** | **Explanation** |
| Initialization | Initialize state SSS with the key K, nonce N, and initialization vector (IV). | S ← IV ∥K∥ N | Combines the key, nonce, and IV into a 320-bit state S. |
|  | Apply 𝑎-round permutation to the state | S ← pa​(S) ⊕ (0320-k∥K) | Strengthens initial state with the key K. |
| Processing Associated Data | Process associated data A in r-bit blocks. | S ← pb ((Sr ​⊕ Ai​)∥ Sc​), i=1,…,si | XOR each block Ai with the rate part Sr apply b-round permutation. |
|  | Add domain separation constant after processing. | S ← S ⊕ (0319 ∥ 1) | Prevents attacks exploiting role confusion between plaintext and associated data. |
| Processing Plaintext | Encrypt plaintext P in r-bit blocks. | Ci ← Sr ​⊕ Pi ​,  S ← pb ((Ci ∥ Sc)), i=1,…,t−1 | XOR plaintext block Pi with Sr , update state S |
|  | Encrypt final plaintext block Pi (if shorter than r bits) |  |  |
| Finalization | Finalize with key K and generate authentication tag T | S ← pa  ​(S ⊕ (0r ∥K∥ 0c -k)),  T←S320−128:320⊕K128 | Ensures integrity of ciphertext and associated data. |

**Notation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **Meaning** | **Symbol** | **Meaning** |
| K | Secret key, up to 160 bits. | P | Plaintext, arbitrary length. |
| N | Nonce, 128 bits. | C | Ciphertext, same length as P. |
| IV | Initialization vector specifying algorithm parameters. | T | Authentication tag, 128 bits. |
| A | Associated data, arbitrary length | S | 320-bit internal state divided into Sr (rate, r bits) and Sc (capacity c bits) |
| pa , pb | Permutations with a and b rounds respectively | | |

Ascon is a lightweight cryptographic algorithm suite designed for authenticated encryption with associated data (AEAD) and hashing, making it ideal for resource-constrained environments such as IoT devices. It was a finalist in the CAESAR competition and has been selected as the primary choice for lightweight encryption. The algorithm achieves 128-bit security against both confidentiality and integrity attacks, ensuring robust protection in scenarios where security is paramount. Ascon's design is highly efficient, leveraging a sponge-based construction that simplifies the implementation of both encryption and hashing functionalities using a unified permutation. Its modular nature and efficient bitwise operations make it particularly well-suited for environments with limited computational resources.

Ascon's key strength lies in its efficiency and adaptability. The algorithm is highly optimized for both hardware and software implementations. For instance, it achieves high throughput in hardware with a small area footprint, requiring less than 10 kGE for encryption. In software, Ascon is designed to take full advantage of 64-bit platforms, enabling fast execution through bitsliced operations and minimal reliance on external memory, which reduces cache accesses. Additionally, Ascon supports multiple parameter sets, such as Ascon-128 and Ascon-128a, allowing users to balance performance and robustness depending on their application requirements. Ascon's small state size and simple design ensure minimal overhead, making it efficient for encrypting short messages, which is a common requirement in lightweight and IoT applications.

One of the key advantages of Ascon is its robustness against misuse scenarios, such as nonce reuse, which can compromise many encryption schemes. Ascon employs a double-keyed initialization and finalization process that enhances its resistance to state recovery and trivial forgeries, even if an internal state is leaked due to implementation vulnerabilities. The algorithm also incorporates features like efficient side-channel protection, low overhead for short messages, and compatibility with countermeasures against timing attacks. These attributes make Ascon a versatile and secure choice for modern cryptographic applications, especially in scenarios requiring lightweight, secure, and efficient encryption and hashing solutions.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Step** | **Notation** | **Explanation** |
| Key Generation | Define the masking function maska,b | maska,b ​=  ϕb2​ ∘ ϕa1​∘P( K ∥ 0n−k) | LFSR-based masking using permutations P, ϕ1 and ϕ2​. |
| Encryption | Split plaintext M into n-bit blocks. | M1, M2 , …….MM | Message is padded and divided into blocks of size n. |
|  | Encrypt each message block Mi. | |  | | --- | | Ci  = Mi ​⊕ P(N∥0n−m ⊕ maski−1,1​) ⊕ maski−1,1​ |  |  | | --- | |  | | XOR message block with masked output of permutation P. |
|  | Concatenate ciphertext blocks | C = C1​ ∥ C2​ ∥…∥ ClM​​ | Produce the complete ciphertext. |
|  | Compute the tag **T** for ciphertext and associated data | T = P(A1​ ⊕ maski−1,0 ) ⊕ maski−1,0​ +  P (Ci​ ⊕ maski−1,2​) | Combines blocks of associated data A and ciphertext C. |
| Decryption | Split ciphertext C into n-bit blocks | C = C1​ ∥ C2​ ∥…∥ ClC | Ciphertext is padded and divided into blocks of size n. |
|  | Decrypt each ciphertext block Ci | Mi​ =Ci ​⊕ P (N ∥ 0n−m ⊕ maski−1,1​) ⊕ maski−1,1​ | XOR ciphertext block with masked output of permutation P. |
|  | Verify the tag T. | T =? Recomputed Tag | Ensures ciphertext integrity and authenticity |

**Notation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **Meaning** | **Symbol** | **Meaning** |
| P | Cryptographic permutation used for encryption and masking. | C | Ciphertext corresponding to M |
| Φ1 , ϕ2 | LFSRs used for masking, where ϕ2 = ϕ1 ⊕ id | T | Authentication tag to verify ciphertext integrity. |
| K | Secret key of size k. | maska,b | Masking function derived from LFSRs and permutation P. |
| N | Nonce of size m | A | Associated data included in tag computation |
| M | Plaintext message of arbitrary length. |  |  |

Elephant is a lightweight authenticated encryption algorithm designed for secure and efficient operation in resource-constrained environments. The algorithm uses an encrypt-then-MAC approach with a counter mode for encryption and a protected counter sum for message authentication. Unlike traditional authenticated encryption schemes that require the inverse of cryptographic primitives, Elephant operates entirely in the forward direction, reducing implementation complexity and computational overhead. This makes it particularly well-suited for environments with strict performance and memory constraints, such as embedded systems and IoT devices. Its design is highly modular and parallelizable, ensuring scalability and adaptability to different use cases.

Elephant comes in three main variants: Dumbo, Jumbo, and Delirium. These variants cater to different security and performance needs, with Dumbo and Jumbo optimized for hardware implementations using the Spongent permutation, and Delirium targeting software platforms with the Keccak-f permutation. The key length for Elephant is 128 bits, and it supports a 96-bit nonce, providing strong security guarantees while minimizing overhead. Each variant offers a balance between security and performance; for instance, Dumbo achieves 112-bit security with a 160-bit permutation, while Jumbo and Delirium provide 127-bit security using 176-bit and 200-bit permutations, respectively. The parallelizable structure of Elephant allows for efficient processing of large data blocks, making it a practical choice for high-throughput applications.

In terms of efficiency, Elephant's lightweight design is optimized for both hardware and software environments. The algorithm's reliance on linear feedback shift registers (LFSRs) for masking eliminates the need for computationally expensive finite field multiplications, further reducing its resource requirements. The cryptographic permutations used in Elephant are small, ensuring a minimal memory footprint. Additionally, Elephant's constant-time masking and resistance to side-channel attacks make it robust against timing and other implementation-specific vulnerabilities. These features, combined with its modular design and proven security, position Elephant as a competitive option in the field of lightweight cryptographic algorithms, particularly in applications requiring secure, efficient, and scalable authenticated encryption.

**Xoodyak**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Step** | **Notation** | **Explanation** |
| Key Generation | Generate a secret key K. | K ∈ {0,1}k | Key size k depends on the security parameter (e.g., 128 bits). |
| Initialization | Combine key and nonce N into state S. | S ← K∥N∥0b−k−m | Initial state S includes key, nonce, and padding to fit the state size b. |
| Processing Associated Data | Process associated data A block by block. | S ← Perm(S)⊕Ai | Permutation is applied to state S, and associated data Ai is XORed into S. |
| Encryption | Divide plaintext P into m-bit blocks Pi | P1​, P2​,…,Pn | Message P is padded and split into blocks. |
|  | Generate ciphertext blocks. | Ci ​= Pi ​⊕ Perm(S) | Each plaintext block Pi is XORed with the permuted state S. |
| Finalization | Generate authentication tag T for C and A | T ← Perm(S) ⊕ f(C,A) | Tag T ensures integrity of ciphertext C and associated data A. |
| Decryption | Recover plaintext by reversing encryption | Pi​ = Ci​ ⊕ Perm(S) | XOR ciphertext blocks Ci​ with the permuted state to recover Pi ​. |
| Tag Verification | Verify the computed tag against the received tag. | Trecv​ =? Tcomputed | Ensures integrity and authenticity of data. |

**Notation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **Meaning** | **Symbol** | **Meaning** |
| K | Secret key | N | Nonce, unique per encryption session |
| A | Associated data, arbitrary length | P | Plaintext, arbitrary length |
| C | Ciphertext, same length as plaintext | T | Authentication tag. |
| S | Internal state maintained during encryption/decryption. | Perm(S) | Cryptographic permutation applied to the state. |
| b | State size in bits. | m | Block size in bits. |

Xoodyak is a lightweight cryptographic scheme designed for hashing, encryption, and authenticated encryption, offering a flexible and efficient solution for symmetric cryptographic operations. It is based on the Cyclist mode of operation, built on the Xoodoo permutation, which uses a compact 384-bit state to perform lightweight yet secure cryptographic computations. Xoodyak is particularly versatile, capable of handling hashing, pseudo-random bit generation, and message authentication. It achieves 128-bit security for all its supported operations, making it suitable for resource-constrained environments like embedded systems and IoT devices. The algorithm's simplicity and modularity allow for a straightforward implementation that minimizes complexity while maintaining strong cryptographic guarantees.

One of the key strengths of Xoodyak is its compactness and efficiency. The algorithm’s lightweight design is achieved by leveraging the Xoodoo permutation, which operates on three planes of 128 bits each, enabling efficient processing on 32-bit processors. Its small memory footprint, requiring only 48 bytes of state, makes Xoodyak highly practical for low-power devices with limited storage and computational resources. Despite its compact design, Xoodyak supports robust protections against side-channel attacks, such as resistance to differential power analysis through mechanisms like key rolling and ratchet functions that ensure forward secrecy. Additionally, its mode of operation enables flexible absorption and squeezing processes, which support incremental hashing and encryption for applications requiring real-time data processing.

Xoodyak's efficiency is further demonstrated by its adaptability to various use cases. The algorithm supports session-based authenticated encryption, enabling the secure exchange of sequences of messages with minimal computational overhead. Its hash mode operates with a customizable output length, offering extendable output functionality similar to SHAKE128, while maintaining 128-bit collision and preimage resistance. In keyed mode, Xoodyak supports authenticated encryption with associated data (AEAD) and message authentication codes (MACs), providing strong guarantees against forgery and tampering. The use of a 128-bit key ensures a high level of security, and the algorithm's structure inherently mitigates risks associated with nonce misuse. These features make Xoodyak an efficient and secure choice for lightweight cryptographic applications, addressing the needs of modern devices while maintaining high levels of security and performance

**Comparative Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | Ascon | Elephant | Xoodyak |
| Purpose | Authenticated Encryption with Associated Data (AEAD) and Hashing | Authenticated Encryption with Associated Data (AEAD) | Authenticated Encryption with Associated Data (AEAD) and Hashing |
| Design Basis | Sponge-based construction with lightweight permutations | Encrypt-then-MAC approach with forward-only cryptographic primitives | Cyclist mode using Xoodoo permutation |
| Security Level | 128-bit security for confidentiality and integrity | 127-bit security (Dumbo/Jumbo) and 112-bit security (Delirium) | 128-bit security for all operations |
| Key Size | Up to 160 bits | 128 bits | 128 bits |
| Nonce Size | 128 bits | 96 bits | Adjustable, depends on the state |
| State Size | 320 bits | Small, optimized for minimal memory use | 384 bits |
| Efficiency | Optimized for both hardware and software; highly efficient in constrained environments | Efficient due to modular and parallelizable design; no finite field multiplications | Compact design with a 48-byte state; optimized for 32-bit processors |
| Hardware Performance | High throughput, less than 10 kGE for encryption | Optimized for hardware with minimal computational overhead | Lightweight, efficient for low-power devices |
| Software Performance | Optimized for 64-bit platforms using bitsliced operations | Efficient and parallelizable for software environments | Adaptable for real-time applications with incremental hashing |
| Robustness | Strong protection against misuse, including nonce reuse | Constant-time masking, resistant to side-channel attacks | Resistant to differential power analysis and supports forward secrecy |
| Applications | IoT devices, embedded systems, secure communications | Embedded systems, high-throughput data encryption | Low-power devices, session-based encryption, real-time data processing |

Ascon is highly efficient for authenticated encryption and hashing, excelling in both hardware and software implementations. Its robust protection against misuse scenarios, such as nonce reuse, ensures high reliability in constrained environments. The scalability of its design makes it adaptable for various IoT and embedded applications.

Elephant focuses on simplicity and modularity, making it highly suitable for resource-constrained environments. Its efficient encrypt-then-MAC approach ensures high performance in hardware, with strong side-channel attack resistance. However, the algorithm's reliance on variants for different use cases adds complexity in selecting the appropriate configuration.

Xoodyak stands out for its compactness and versatility, providing efficient support for hashing, encryption, and real-time operations. Its minimal memory footprint and resistance to side-channel attacks make it particularly suitable for low-power devices. However, its small state size may limit scalability for certain high-throughput applications.

**Conclusion**

Ascon, Elephant, and Xoodyak represent state-of-the-art advancements in lightweight cryptographic algorithms, each designed to balance security, efficiency, and adaptability in resource-constrained environments.

* Ascon excels in its simplicity, sponge-based design, and resilience against misuse scenarios, making it a versatile choice for IoT applications requiring robust AEAD and hashing capabilities.
* Elephant prioritizes implementation simplicity and efficiency, offering strong security through its encrypt-then-MAC structure, which is particularly suited for hardware and high-throughput applications.
* Xoodyak is notable for its compactness and modularity, providing exceptional efficiency for hashing, AEAD, and session-based encryption in low-power devices.

These algorithms cater to various application needs, with Ascon focusing on flexibility, Elephant on ease of implementation and hardware optimization, and Xoodyak on compactness and real-time adaptability. Their adoption will depend on the specific requirements of the system, including memory constraints, throughput demands, and the nature of the cryptographic operations.