DSM – 513: Implementation of Ascon-128 Encryption and Decryption Algorithm

Submitted to:

Bodhisatwa Mazumdar, PhD,

Associate Professor,

Department of Computer Science and Engineering,

Indian Institute of Technology Indore.

Submitted by:

Reetu Shrivastava (2204107036)

Email: msdsm2204107036@iiti.ac.in,

MSDSM Batch – 2,

Indian Institute of Technology Indore.

Implementation of Ascon-128 Encryption and Decryption Algorithm

Introduction

This project report aims to provide comprehensive documentation and explanation of the Ascon-128 encryption and decryption algorithm implemented in Python. Ascon is a family of lightweight cryptographic algorithms, including Authenticated Encryption with Associated Data (AEAD), hash, and MAC algorithms. This report focuses specifically on the Ascon-128 variant, which is a primary variant of the Ascon family.

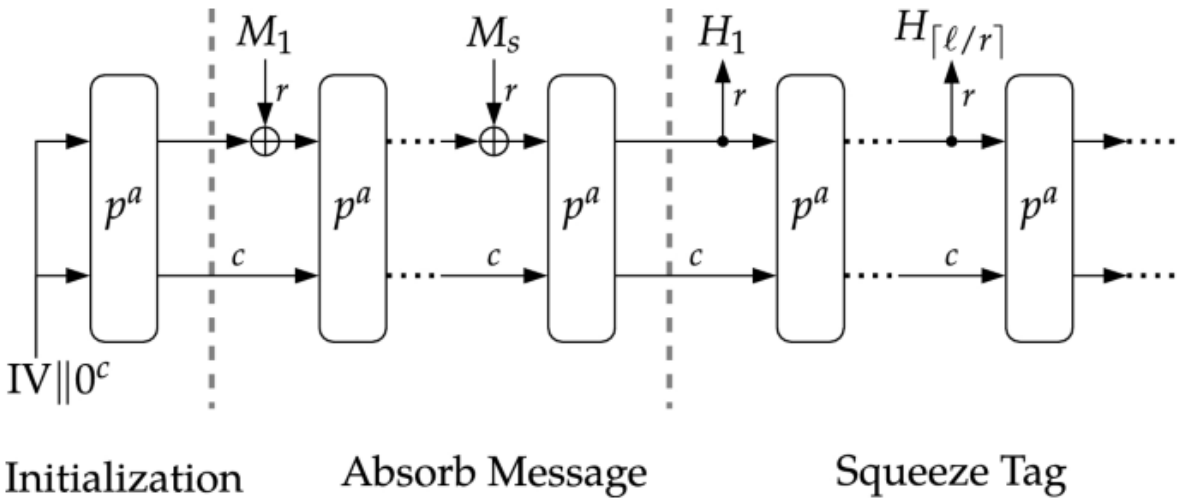
The increasing need for secure communication and data transmission has led to the development of various cryptographic algorithms. Ascon (Authenticated Secret Convergence) is one such algorithm that offers authenticated encryption with associated data (AEAD) functionality. In this project, we present an implementation of the Ascon-128 variant, which is a lightweight cryptographic algorithm suitable for resource-constrained environments.

1.Authenticated Encryption with Associated Data (AEAD):

Authenticated Encryption with Associated Data (AEAD) is a cryptographic technique that provides both confidentiality and integrity assurances for encrypted data, along with the ability to authenticate additional associated data that is not encrypted. In other words, AEAD ensures that the data remains confidential, that it has not been tampered with during transmission, and it allows for the inclusion of additional context or metadata (associated data) that is authenticated but not encrypted. AEAD schemes typically combine a symmetric encryption algorithm with a message authentication code (MAC) to achieve these security goals.

2. Hash Algorithms:

Hash algorithms are cryptographic functions that take an input (or "message") and produce a fixed-size output called a hash value or hash digest. The key properties of hash functions include determinism (the same input always produces the same output), irreversibility (it should be computationally infeasible to reconstruct the input from the hash output), and collision resistance (it should be difficult to find two different inputs that produce the same hash output). Hash functions are commonly used in various cryptographic applications, including data integrity verification, password hashing, digital signatures, and more.



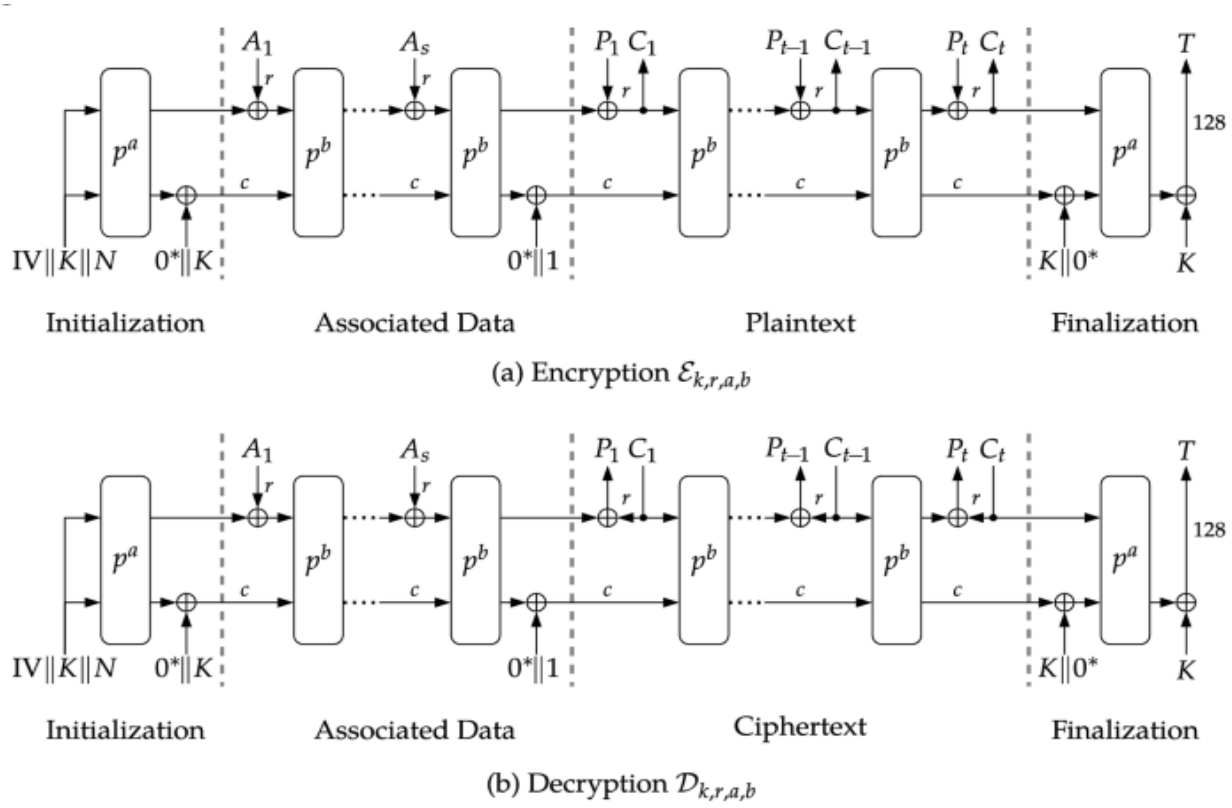
Hashing mode $\mathcal{X}_{h,r,a}$ in ASCON-HASH, ASCON-XOF

3. Message Authentication Code (MAC) Algorithms:

A Message Authentication Code (MAC) is a cryptographic technique used to authenticate the integrity and authenticity of a message. It is generated by applying a MAC algorithm to a message and a secret key, resulting in a fixed-size tag that is appended to the message. The tag serves as a unique fingerprint of the message, allowing the recipient to verify both the integrity (i.e., whether the message has been altered) and the authenticity (i.e., whether it comes from the purported sender) of the message. MAC algorithms can be constructed using symmetric cryptographic primitives such as block ciphers or cryptographic hash functions. They are commonly used in conjunction with encryption schemes to provide authenticated encryption, as in AEAD constructions.

Code Overview

The Python code implements the Ascon-128 algorithm for both encryption and decryption processes. The encryption function includes steps for initialization, processing associated data, processing plaintext, and finalization. Similarly, the decryption function follows steps for **initialization, processing associated data, processing ciphertext, and finalization.**



ASCON 's mode of operation

Initialization: Functions to initialize the state with key and nonce, including padding and permutation steps.

Associated Data Processing: Functions to process associated data, including padding and absorption into the state.

Plaintext Processing: Functions to process plaintext data, including padding, absorption into the state, and ciphertext generation.

Ciphertext Processing: Functions to process ciphertext data, including padding, absorption into the state, and plaintext generation.

Finalization: Function to finalize the encryption process and generate the authentication tag.

Permutation: Function implementing the core permutation step of the Ascon algorithm.

Helper Functions: Additional functions for byte manipulation, bitwise operations, and random byte generation.

Demo and Testing: Demo functions to demonstrate encryption and decryption, and testing functions to verify correctness.

The code structure consists of implementations of key scheduling, permutation, and data processing functions essential for the Ascon-128 algorithm. These functions are integrated into the encryption and decryption processes to ensure secure and efficient data transformation. The initialization steps set up the necessary parameters and state for the encryption or decryption operation. Associated data is processed to incorporate additional information into the encryption or decryption process without revealing the plaintext. Plaintext is processed using the algorithm's cryptographic operations to produce ciphertext in encryption and to recover plaintext from ciphertext in decryption. Finally, the finalization steps ensure the integrity and completeness of the encryption or decryption operation. Together, these components form a comprehensive implementation of the Ascon-128 algorithm in Python, enabling secure data encryption and decryption.

Code Explanation

The code implements the Ascon-128 encryption algorithm following a structured approach. Key components of the code include:

Initialization: Initializes the state with the key, nonce, and fixed initialization vector (IV), setting up the initial conditions for encryption or decryption.

Data Processing: Processes associated data, plaintext, and ciphertext according to the Ascon-128 specification. This involves padding, absorption into the state, permutation, and squeezing.

Permutation: Implements the core permutation step of the Ascon algorithm, comprising adding round constants, a substitution layer, and a linear diffusion layer.

Helper Functions: Provides utility functions for byte manipulation, bitwise operations, and random byte generation, facilitating various tasks within the algorithm.

Demo and Testing: Includes demo functions to showcase encryption and decryption, along with testing functions to verify the correctness of the implementation.

Code Documentation

The code is extensively documented using comments to explain each function's purpose, parameters, and internal operations. Additionally, relevant links to specifications and resources are provided for further reference.

Ascon Parameters

- **State (S):** Ascon state consists of 64 columns and 5 rows, totaling 320 bits.
- **Data Block Size (Rate):** Rate is the size of a data block, set to 64 bits.
- **IV:** Initialization vector, a fixed value specific to Ascon-128.
- **Key Size:** 128 bits.
- **Nonce:** 128 bits.
- **Tag:** Authentication tag, 128 bits.

Functionality Explanation

The code provides functions for both encryption and decryption:

Encryption Function (ascon_encrypt)

- Input: key, nonce, associated data, plaintext
- Parameters: state (S), initial and intermediate rounds (a, b), rate (rate)
- Process: Initialization, processing associated data, processing plaintext, finalization
- Output: Ciphertext (same size as plaintext) + Tag (128 bits)

Decryption Function (ascon_decrypt)

- Takes input parameters similar to encryption.
- Performs parameter validation.
- Initializes the state with the provided key, nonce, and IV.
- Processes associated data and ciphertext.
- Verifies the authentication tag.
- Returns the decrypted plaintext if authentication succeeds; otherwise, returns None.

Initialization Function (ascon_initialize)

- Initializes the state with the key, nonce, and IV.
- Performs state manipulation and permutation for initialization.

Processing Functions (ascon_process_associated_data, ascon_process_plaintext, ascon_process_ciphertext)

- Process associated data, plaintext, and ciphertext, respectively.
- Padding is applied to ensure data size compatibility.
- Data blocks are absorbed into the state and squeezed out to generate ciphertext or plaintext.

Finalization Function (ascon_finalize)

- Generates the authentication tag by finalizing the state and performing XOR operations with the key.

Permutation Function (ascon_permutation)

- Implements the permutation step of the Ascon-128 algorithm.
- Involves adding round constants, a substitution layer, and a linear diffusion layer.

Helper Functions

- **ascon_initialize:** Initializes the Ascon state with the key and nonce.
- **ascon_process_associated_data:** Processes the associated data.
- **ascon_process_plaintext:** Processes the plaintext data.
- **ascon_finalize:** Finalizes the encryption process and generates the tag.

Demonstration and Testing

The code includes functions for demonstration and testing:

- **demo_aead:** Demonstrates encryption and decryption with predefined data.
- **test_ascon_cipher:** Tests encryption and decryption with predefined and random data.
- **proof_of_correctness:** Demonstrates the correctness of encryption and decryption.

Conclusion

The Ascon-128 encryption algorithm implementation presented in this project provides a comprehensive and well-documented solution for lightweight authenticated encryption. The code structure, along with detailed explanations and documentation, ensures clarity, readability, and ease of understanding for developers and researchers interested in cryptographic implementations.

References:

- ***Dobraunig, C., Eichlseder, M., Mendel, F., & Schläffer, M. (2021). ASCON v1.2: Lightweight Authenticated Encryption and Hashing. Journal of Cryptology, 34(3). https://doi.org/10.1007/s00145-021-09398-9.***
- ***https://www.youtube.com/watch?v=RWiH_6UwzzY [Implementation of ASCON in C]***
- ***https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/ascon-spec-final.pdf***
- ***https://www.youtube.com/watch?v=7iFzDInnwig***