ISSN 2007-9737

ASCON-AEAD 128 Implementation in a Software Numerical Platform for Constrained-Resources Devices

Jesús A. Aboytes-González [1],[3], Marco T. Ramírez-Torres[2], Gina Gallegos-García[1,*]

[1] Instituto Politécnico Nacional,
Centro de Investigación en Computación,
Mexico

[2] Universidad Autónoma de San Luis Potosí,
Coordinacion Académica Región Altiplano Oeste,
Mexico

[3] Universidad Politécnica de San Luis Potosí,
Mexico

agustin.aboytes@upslp.edu.mx, tulio.torres@uaslp.mx, ggallegosg@ipn.mx

Abstract. ASCON is a family of lightweight cryptographic algorithms; one of the members is the Authenticated Encryption with Associated Data (AEAD) designed to ensure security while efficiently considering memory and energy usage. In 2023, it was selected as the standard for lightweight cryptography, highlighting its suitability for resource-constrained devices. In this paper we present a comprehensive implementation of the ASCON-AEAD 128 algorithm in a Software Numerical Platform.

The development of this algorithm in such a platform provides programmers and researchers with a versatile platform to simulate, gather metrics, and analyze its performance when implementing ASCON-AEAD 128 in a wide variety of scenarios. Furthermore, this implementation has potential applications in constrained-resource devices such as embedded systems, and IoT devices.

Keywords. ASCON, lightweight cryptography, implementation.

1 Introduction

Nowadays, in the current landscape of information technology, Internet of Things (IoT) and Industry 4.0, data security is a critical aspect that requires specialized applications. The confidentiality and integrity of information have become fundamental issues in an increasingly interconnected world, where each day we have more smart devices with sensitive or personal information connected to the Internet.

Cryptography has emerged as a solution to those issues, by guaranteeing secure communication. In this context, it provides the necessary tools to protect the information during transmission or storage. In fact, encryption algorithms ensure that transmitted data cannot be understood by unauthorized parties.

However, with the proliferation of connected and smart devices, such as Internet of Things (IoT) devices, an additional challenge arises: ensuring security in devices with constrained resources such as memory, energy, and processing power. In this context, one of the main assignments in the field of lightweight cryptography is the development of cryptographic algorithms adapted for devices with limited resources.

Research and development in lightweight cryptography began around 2004 in Europe [10]. In 2017, the National Institute of Standards and Technology (NIST) announced a public call for proposals for a lightweight cryptography standard

[4]. The winner of this call, announced in 2023, was the ASCON family.

It comprises a set of algorithms; one of them is known as Authenticated Encryption with Associated Data, considered for inclusion in the standard document to be published by NIST [5].

Since its standardization, several research groups have contributed to the adaptation of the ASCON algorithm to various programming languages, including C, Java, and Python. These implementations are publicly available through multiple online repositories [2, 3]. The primary objective of these adaptations is to provide developers with a broader set of tools, enabling seamless integration of ASCON into their projects without requiring manual code translation between languages. This flexibility is particularly relevant in constrained environments such as embedded systems and IoT applications, where implementation portability and cryptographic robustness are essential for ensuring secure communication.

In this paper, we present the implementation of the Authenticated Encryption with Associated Data 128 algorithm (ASCON-AEAD 128) in a Software Numerical Platform, which provides a flexible and robust way for the development, simulation, and evaluation of cryptographic algorithms. This implementation serves as a valuable tool for modeling and analyzing the performance of ASCON-AEAD 128 in various scenarios. In addition, there are several applications in embedded systems, IoT devices, secure communications, and other domains where the balance between security and computational efficiency is critical.

It is important to mention that this numerical implementation of ASCON contributes significantly to the scientific and technological community by providing a detailed description of its internal functioning. This implementation can enable the evaluation of performance, security, and optimization strategies in a flexible environment. Furthermore, it supports comparative analysis with other encryption algorithms, facilitating studies on relevant metrics such as efficiency and cryptographic strength. Through a well-documented and modular approach, this work allows the academic cybersecurity community to engage with ASCON more effectively, promoting both educational applications and further research into its internal processes.

The remainder of this paper is organized as follows. Section 2 presents the related work, discussing the original implementation of the ASCON-AEAD 128 algorithm, as well as some currently employed implementations. Section 3 describes our proposed implementation. In Section 4, we present the results obtained by encrypting text using this implementation. Section 5 offers a concise analysis of the outcomes obtained from the implementation. Finally, Section 6 offers conclusions regarding the results and the scope of this study.

2 ASCON-Authenticated Encryption with Associated Data128

The ASCON-AEAD 128 comprises four main phases designed to ensure fast and secure encryption decryption. These phases are Initialization, Associated Data Processing, Encryption/Decryption, and Finalization [6, 7]. In Fig. 1, the ASCON encryption process is illustrated.

In turn, the process consists of four stages: initialization, associated data, encryption/decryption, and finalization. In the initialization stage, the key (K), nonce (N), and initialization constant (IV) are introduced, along with the rate size (r) and the system permutations (a and b), to update the internal state (S) through specific transformations. During the associated data stage, additional information is incorporated while continuing to update the internal state. In the encryption/decryption stage, the internal state is employed to transform the plaintext (P) into ciphertext (C), or vice versa. Finally, in the finalization stage, the key (K) is reintroduced to generate the authentication tag (T), ensuring both data integrity and authenticity.

In the encryption/decryption stage of ASCON-AEAD 128, a fundamental component is the permutation function, which is critical to the algorithm's security. This function transforms the internal state (S) through a series of

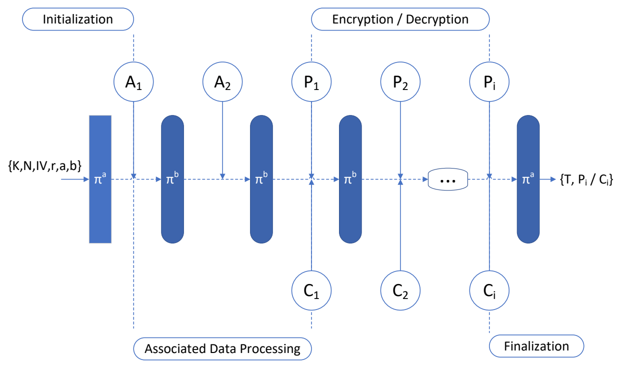

Fig. 1. Block diagram of the ASCON algorithm. The structure illustrates the four stages of the implementation, including the input and output interfaces of each stage, as well as the iterative application of the permutation π

operations designed to provide diffusion and non-linearity—key properties for resistance against cryptanalysis. Although the permutation is not a standalone stage within the algorithm, it is executed throughout multiple phases, playing an essential role in its operation. It performs iterative transformations on the internal state, using a predefined constant added in each round to prevent structural repetition. Diffusion is achieved through a 5-bit S-box, while linear mixing is implemented via XOR operations and bitwise rotations, which propagate the influence of each bit across the state. The number of permutation rounds is determined by the parameters a and b, which are predefined according to the specific phase in which the permutation is applied [6].

The following describes the four main stages of the ASCON-AEAD 128 algorithm, in which the permutation function is applied, and its impact on the cryptographic process.

2.1 Initialization

During the initialization stage, the input parameters are introduced. Those are: the secret key (K), the nonce (N), and the initialization constant (IV), along with the variables r, a, and b, which control the absorption rate, as long as the number of permutation rounds during initialization, in addition to the number of rounds in subsequent phases, respectively.

In this stage, the internal state (S) is modified by applying the permutation function (π), ensuring the proper diffusion and randomness of the data before proceeding with message processing [6].

2.2 Associated Data Processing

The associated data stage integrates the data $(A_1$ and $A_2)$ into the internal state (S) of the ASCON-AEAD 128 algorithm. Although these data are not encrypted, they are embedded within the state to ensure that their integrity and authenticity can be verified during the final stage. This integration is accomplished by updating the state through an XOR operation applied to blocks of r bits. Subsequently, the updated state is subjected to b iterations of the permutation function (π), thereby ensuring secure diffusion of the associated data and reinforcing the cryptographic binding essential for message authentication [6].

2.3 Encryption/Decryption

In the encryption stage of the ASCON-AEAD 128, the plaintext (P_i) is segmented into blocks of r bits and absorbed into the internal state (S), which has been previously updated during the previous phases. Each block is incorporated via an XOR operation with the state, after which the permutation function (π) is applied b times, to non-linearly diffuse the information within the internal state (S). Once the distribution has been updated, a portion of the state is extracted and then combined using another XOR operation with the original plaintext block (P_i) to produce the corresponding ciphertext block (C_i). This process is repeated iteratively until the entire message is processed, thereby ensuring confidentiality, data integrity and authenticity at the same time.

The decryption process is executed symmetrically regarding encryption. In other words, each ciphertext block (C_i) is reintroduced into the internal state (S) following the same scheme of absorption in r-bit blocks. After absorption, the permutation function (π) is applied b times to obtain an updated state, from which a portion is extracted. This extracted portion, when combined via an XOR operation with the corresponding ciphertext block (C_i), enables recovery of the original plaintext block (P_i). Repeating this process for all blocks allows complete reconstruction of the original message while maintaining the confidentiality of the transmitted information [6].

2.4 Finalization

The final stage of the ASCON-AEAD 128 algorithm ensures the integrity and authenticity of encrypted data by generating an authentication tag (T). This tag, computed during the encryption phase is compared with the one obtained during decryption. If both values match, the authenticity of the message is true; otherwise, the message is considered compromised and is discarded.

To generate the authentication tag, the secret key (K) of 128 key-length is reabsorbed in the internal state using an XOR operation, reinforcing the security of the algorithm. Subsequently, the permutation function (π) is applied a times to ensure proper diffusion of information within the internal state. Finally, a portion of the resulting state is extracted and appended as the authentication tag to the ciphertext (C_i).

The verification process during decryption follows the same procedure. After recomputing the tag from the received message and secret key, it is compared to the original tag. If they match, the message is considered valid; otherwise, potential tampering or data corruption is detected. This phase enhances ASCON-AEAD 128 security by providing both confidentiality, integrity, and authenticity within a single scheme, ensuring that any modification to the encrypted or associated data is efficiently detected [6].

3 Implementation in a Software Numerical Platform for Constrained-Resources Devices

Our implementation was made in MatLab as the Software Numerical Platform selected for Constrained-Resources Devices. It includes six main functions,each corresponding to one of the main stages of the ASCON-AEAD 128 algorithm, including the permutation function. It is important to mention that the encryption and decryption stages were implemented as separate functions to simplify the implementation process.

The items in the following list correspond to the functions that implement the main stages of the ASCON-AEAD 128 algorithm. Item 1 performs the initialization stage (Sec. 2.1); Item 2 processes the associated data (Sec 2.2); Item 3 executes the encryption stage, and Item 4 the decryption stage (both in Sec. 2.3); Item 5 carries out the finalization stage (Sec. 2.4). Item 6 implements the permutation function, which is not a stage itself but is invoked throughout multiple stages of the algorithm:

1. `[S]` = ascon_initialize`(K,N)`
2. `[S]` = ascon_process_associated_data`(S,A)`
3. `[S,C]` = ascon_process_plaintext`(S,P)`
4. `[S,P]` = ascon_process_ciphertext`(S,C)`
5. `[T]` = ascon_finalize`(S,K)`
6. `[S]` = ascon_permutation`(S, a or b)`.

In addition, six sub-functions were implemented to manage data coupling between stages. These support functions handle hexadecimal–binary conversion, adjust vector sizes, and perform bitwise rotations to ensure consistent algorithm behavior. Item 1 prepares the initialization vector (IV) for integration into the state; Item 2 applies rightward bit rotations (ROTR); Item 3 inserts zero bytes for padding and vector size adjustment; Item 4 converts integers to byte arrays; Item 5 maps byte arrays into the internal state format; and Item 6 reverses this process by converting byte arrays back into integers:

1. `[IV]` = prepare_iv`(N, K)`
2. `[y]` = rotr`(v, r)`
3. `[y]` = zero_bytes`(r)`
4. `[bArray]` = int_to_bytes`(int, nbts, dir, last)`
5. `[S]` = bytes_to_state`(bytes)`

6. `[v]` = bytes_to_int `(bArray)`.

The repository containing the implementation files is available on GitHub at the following link: https://github.com/AgustinAboytes/ASCON-AEAD-128-Matlab-Implementation and to verify the functionality of our implementation, several tests were performed. The results are presented below.

4 Tests and Results

The encryption and decryption phases are carried out using MATLAB R2018a as the Software Numerical Platform selected for constrained resource devices. They are performed on character messages and string text messages, and the tests are proposed based on previous studies conducted on adaptations of cryptographic algorithms, such as AES, to their MATLAB implementation [8, 9, 1].

4.1 Character message

Table 1 presents a series of characters, specifically the alphabet from letter a to z, along with their encrypted version and the recovered text using our adaptation of the ASCON-AEAD 128 algorithm. In addition, the values for the key (K), nonce (N), tag (T), and associated data (A) are provided.

Table 1. Functionality of our ASCON-AEAD 128 implementation

Text type	Data
Key	1,2,3,4,5,6,7,8,9,10,11,12,13 14,15,16,17,18,19,20,21,22,23 24,25,26,27,28,29,30,31,32,33 34,35,36
Nonce	1,2,3,4,5,6,7,8,9,10,11,12,13 14,15,16
Associated Data	2,3,4,5,6,7,8,9,10,11,12,13,14 15,16,17
Plain	`abcdefghijklmnopqrstuvwxyz`
Cipher	`ÙkÏ#’ë0[{âFÄTî_vJó¨x¡*‘`
Decryped	`abcdefghijklmnopqrstuvwxyz`
Tag	137,103,143,2,25,101,0,83,202, 13,130, 36,145,64,104,205

As presented in Table 1, the ciphertext does not exhibit similarity to the plaintext, demonstrating the effectiveness of the encryption process. Furthermore, the decryption preserves the integrity of the data, confirming the correct implementation of the ASCON-AEAD 128 algorithm.

4.2 String Text message

The following is a sample text that serves as a long plaintext, encrypted using our implementation of the ASCON-AEAD 128 algorithm.

”*Tabatha and Sareth, six and three years old, walked through the twisted trees of the forest, following the breadcrumbs their mother, Kiya, had left the night before. The evening light painted the leaves red and the whispers of the wind seemed to call their names. Suddenly, a silver-furred fox appeared before them, holding Kiya’s embroidered handkerchief in its mouth. Without hesitation, the sisters ran after it, trusting that it would lead them to the truth...*”

As observed, the encrypted version reveals no details of the original plaintext, and the decrypted text is identical to the plaintext, confirming the functionality of the encryption process.

```
ìn-!£ÿ,õS=ì0/Zê°¯jvÓ!^Ud¼«¦îüIìª\%*j È
/K?cg7GÆïE°-ù§ä·Ø@¤r0¡\±?ávÍó¤JiDÆ4è0·Aâk
kgåk¼+ýeõ÷ÔØÇI1CÖabfqS^’Õ¡ß-ô¶"ÏÅ ®dw+XµÂ
É¦ ‘~\$ñpe8N\$FRÍ! váË<¤0"yÅ ý}Ó¸‘Ún¥¬\$
 Íjb¬íñ©¿È*üAãÀë\ jbah & ×ìmrÓc_u®Éñt£
wùÀwF§³ê¿zMõò5tÛÉwý6Â±dÌä¯ ÕãÓu!\.=MNöf
\$N/êTPpÚ÷=FZÈQf)¦Dùg¯¹"q§kS§}Õ¬c1Ýånè8
\$)ço Z¶\$dToq~TuÔ¨"E0Îôk°=áÂ‘D\ûzãkîµòzP
 0iµü^rñ£ØXPÔ¶uCÜ¡Z¸¢³ø!9/õ8Y@ç D¨ÄîAÓ
llel1å5ÌE,¢
```

Table 2. Comparison of execution times (in seconds) across different implementations of the ASCON algorithm, for the encryption and decryption process

Implementation	Time	
	Encryption	Decryption
C	0.000001 s	0.0000001 s
Python	0.000356 s	0.000361 s
Matlab	0.00076 s	0.00092 s

5 Discussion

The functional validation of our implementation was performed by directly comparing the encryption results with those generated by the official ASCON implementations available in the project's GitHub repository, developed in languages such as C and Python. The results were identical in all cases, confirming the full compatibility and correct algorithmic operation of our proposal with respect to the reference versions.

In terms of temporal performance and considering that MATLAB is a Software Numerical Platform, it was observed that our MATLAB implementation, although functionally correct, exhibits lower efficiency compared to C and Python. This is attributed to the fact that MATLAB is not designed as a high-performance execution environment but is intended for numerical analysis and prototyping.

However, the measured execution times are not prohibitive: encryption took 0.00076 seconds and decryption 0.00092 seconds. These values are higher than those recorded for Python (0.000356 s and 0.000361 s, respectively) and significantly slower than those of the C implementation, whose performance is in the microsecond range per operation, according to officially reported cycles-per-byte metrics.

6 Conclusions

The implementation of the ASCON-AEAD 128 algorithm in MATLAB as a software numerical platform for constrained-resources devices presented in this work serves as a useful tool for studying and understanding how this cipher operates. While MATLAB is not as efficient as compiled languages like C, the results show that it delivers acceptable execution times for purposes such as testing, analysis, and education.

During testing, encryption and decryption times were observed in the millisecond range, which is sufficient to validate the algorithm's behavior before deploying it on hardware or in constrained environments like IoT devices. This version also helps reduce development time and facilitates comprehension of the scheme in academic settings.

An important advantage is that MATLAB is compatible with a wide range of systems and can leverage hardware resources such as multicore CPUs, vectorized processing, and GPUs. This allows for simulating more demanding usage scenarios and performing robust testing without switching platforms.

Furthermore, the results confirm that the encryption output is consistent with that of existing reference implementations, which suggests a high level of correctness. This compatibility is particularly valuable, as it enables seamless code migration, testing, and hybrid development across multiple programming environments.

As future work, we plan to improve this implementation to make it faster and more efficient. This includes optimizing the code, using lighter data structures, and applying MATLAB parallel processing capabilities. These improvements aim to bring the performance closer to real world applications while maintaining its value as a platform for analysis, prototyping, and cryptographic education.

Acknowledgments

Aboytes-González is a postdoctoral fellow of SECIHTI (México). This work was funded by SIP-IPN under grant 20250059.

References

1. **Aparna, V. S., Rajan, A., Jairaj, I., Nandita, B., Madhusoodanan, P., Remya, A. A. S. (2019).** Implementation of AES algorithm on text and image using MATLAB. 3rd International Conference on Trends in Electronics and Informatics (ICOEI), IEEE, pp. 1279–1283.

2. **ASCON Developers (2025).** ascon-c: ASCON - lightweight authenticated encryption & hashing. GitHub repository. Accessed on May 11, 2025.

3. **ASCON Developers (2025).** Implementations. Accessed on May 11, 2025.

4. **Bassham, L., Çalık, Ç., McKay, K., Mouha, N., Turan, M. S. (2017).** Profiles for the lightweight cryptography standardization process. NIST draft.

5. **Boutin, C. (2023).** Nist selects 'lightweight cryptography' algorithms to protect small devices. NIST, https://www.nist.gov/news-events/news/2023/02/nist-selects-lightweight-cryptography-algorithms-protect-small-devices.

6. **Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M. (2021).** Ascon v1. 2: Lightweight authenticated encryption and hashing. Journal of Cryptology, Vol. 34, pp. 1–42.

7. **Hernández-Álvarez, L., Bullón Pérez, J. J., Batista, F. K., Queiruga-Dios, A. (2022).** Security threats and cryptographic protocols for medical wearables. Mathematics, Vol. 10, No. 6, pp. 886.

8. **Khan, S., Inayat, K., Muslim, F. B., Shah, Y. A., Atif Ur Rehman, M., Khalid, A., Imran, M., Abdusalomov, A. (2024).** Securing the iot ecosystem: Asic-based hardware realization of ascon lightweight cipher. International Journal of Information Security, pp. 1–12.

9. **Lohit Kumar, D., Reddy, A. R., Jilani, S. A. K. (2016).** T1-implementation of 128-bit AES algorithm in MATLAB. International Journal of Engineering Trends and Technology, , No. 3, pp. 126–129.

10. **Toshihiko, O. (2017).** Lightweight cryptography applicable to various iot devices. NEC Technical Journal, Vol. 12, No. 1, pp. 67–71.

Article received on 21/05/2025; accepted on 22/08/2025.
**Corresponding author is Gina Gallegos-García.*