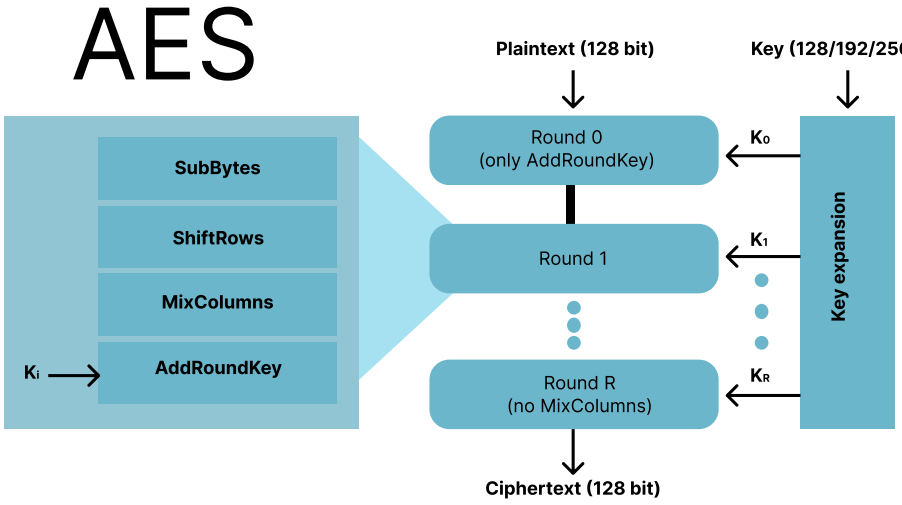
PER2023-034 - Efficient and secure implementation of AES-GCM in Jasmin

Students (Master 2 Cybersecurity)
Noah CANDAELE, Anthony IOZZIA
Université Côte d'Azur, Sophia-Antipolis, France
{noah.candaele,anthony.iozzia}@etu.univ-cotedazur.fr

Supervisors
Sid TOUATI, Benjamin GRÉGOIRE, Jean-Christophe LÉCHENET
Inria research institute, Sophia-Antipolis, France
{sid.touati,benjamin.gregoire,jean-christophe.lechenet}@inria.fr

Cryptography demands special attention in programming due to its unique security requirements. Specialized programming languages address these needs, ensuring secure and efficient cryptographic operations. This research project aims to highlight the AES crypthography algorithm and its AES-GCM extension. These two algorithms are notably robust against quantum computer threats, representing a significant milestone in post-quantum cryptography. Here, we will develop a version of these algorithms in Jasmin, a language dedicated to cryptography, whose purpose is to certify the code produced.

AES



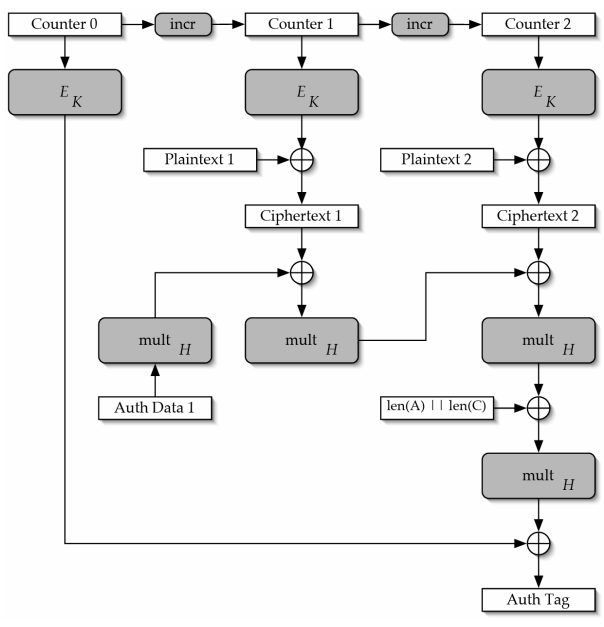
AES stands for Advanced Encryption Standard. It is a symmetric encryption algorithm used to secure sensitive data. A symmetric encryption scheme employs a single key for both encryption and decryption processes. AES was established as a standard by the U.S. National Institute of Standards and Technology (NIST) in 2001.
The AES algorithm operates on blocks of data, with a fixed block size of 128 bits. However, it supports key sizes of 128, 192, and 256 bits.

The encryption process involves multiple rounds of substitution, permutation, and mixing operations, ensuring both diffusion and confusion, which collectively contribute to a high level of security.
One benefit of AES is its ability to rapidly encrypt sizable volumes of data while consuming minimal resources, making it suitable for deployment on compact devices.

Algorithm 1 Pseudocode for AES encryption.

```
1:  procedure AesEncrypt(in, Nr, w)
2:      state ← in
3:      state ← AddRoundKey(state, w[0..3])
4:      for round from 1 to Nr − 1 do
5:          state ← SubBytes(state)
6:          state ← ShiftRows(state)
7:          state ← MixColumns(state)
8:          state ← AddRoundKey(state, w[4 × round..4 × round + 3])
9:      end for
10:     state ← SubBytes(state)
11:     state ← ShiftRows(state)
12:     state ← AddRoundKey(state, w[4 × Nr..4 × Nr + 3])
13:     return state
14: end procedure
```

src: https://doi.org/10.6028/NIST.FIPS.197-upd1



src: The Galois/Counter Mode of Operation (GCM) - NIST

AES-GCM

AES-GCM (Galois/Counter Mode) is a mode of operation for AES that merges encryption and authentication, ensuring both confidentiality and integrity. It operates by dividing data into blocks and encrypting each block using AES in counter mode, where each plaintext block is combined with a unique encrypted counter value. Additionally, AES-GCM includes an authentication tag generated through Galois field multiplications, ensuring data integrity. This algorithm is extensively utilized in secure communications like TLS (Transport Layer Security) to safeguard internet connections, providing efficient and robust protection against eavesdropping and tampering.

$$X_i = \begin{cases} 0 & \text{for } i = 0 \\ (X_{i-1} \oplus A_i) \cdot H & \text{for } i = 1, \ldots, m-1 \\ (X_{m-1} \oplus (A_m^* \| 0^{128-v})) \cdot H & \text{for } i = m \\ (X_{i-1} \oplus C_i) \cdot H & \text{for } i = m+1, \ldots, m+n-1 \\ (X_{m+n-1} \oplus (C_m^* \| 0^{128-u})) \cdot H & \text{for } i = m+n \\ (X_{m+n} \oplus (\text{len}(A) \| \text{len}(C))) \cdot H & \text{for } i = m+n+1. \end{cases}$$

$$H = E(K, 0^{128})$$

$$Y_0 = \begin{cases} IV \| 0^{31}1 & \text{if len}(IV) = 96 \\ \text{GHASH}(H, \{\}, IV) & \text{otherwise.} \end{cases}$$

$$Y_i = \text{incr}(Y_{i-1}) \text{ for } i = 1, \ldots, n$$

$$C_i = P_i \oplus E(K, Y_i) \text{ for } i = 1, \ldots, n-1$$

$$C_n^* = P_n^* \oplus \text{MSB}_u(E(K, Y_n))$$

$$T = \text{MSB}_t(\text{GHASH}(H, A, C) \oplus E(K, Y_0))$$

src: The Galois/Counter Mode of Operation (GCM) - NIST

K: AES secret key	H: hash key
IV: initialization vector	E(): encipher with AES
A: authentication data	m: nb of 128-bit blocks in A
P: plaintext	n: nb of 128-bit blocks in C
C: ciphertext	u: bit length of last block of C
T: authentication tag	v: bit length of last block of A

Jasmin language

Jasmin (github.com/jasmin-lang/jasmin), as a verified programming language and compiler, stands out in the field of high performance and high-assurance cryptography. This platform offers a flexible programming environment that combines both high-level and low-level features. Programmers using Jasmin have the ability to optimize crucial aspects for performance, such as instruction selection and register allocation, while exploiting high-level concepts such as variables, functions, tables and loops to organize and check their code. Jasmin benefits from a formal semantics that facilitates rigorous reasoning on the behavior of programs.

The Jasmin compiler generates an efficient and predictable assembly code that preserves the properties of the source program.
The Jasmin workbench uses the EasyCrypt tool set for formal verification. Jasmin programs can be translated into corresponding EasyCrypt programs to prove functional correction for cryptographic security. Jasmin combines the best of all worlds: high-level language with high performance and high assurance. For these reasons, we will use Jasmin as the programming language for our implementations of cryptographic algorithms.

```
reg u128 c; c = #set0_128(); c = #VPCLMULQDQ(a, b, 0x11);
reg u128 d; d = #set0_128(); d = #VPCLMULQDQ(a, b, 0x00);

reg u64 a0; a0 = 0; a0 = #VPEXTR_64(a, 0);
reg u64 a1; a1 = 0; a1 = #VPEXTR_64(a, 1);
reg u64 xor_a; xor_a = a0; xor_a ^= a1;
reg u64 b0; b0 = 0; b0 = #VPEXTR_64(b, 0);
reg u64 b1; b1 = 0; b1 = #VPEXTR_64(b, 1);
reg u64 xor_b; xor_b = b0; xor_b ^= b1;
reg u128 xor_a_128; xor_a_128 = (128u) xor_a;
reg u128 xor_b_128; xor_b_128 = (128u) xor_b;
reg u128 e; e = #set0_128();
e = #VPCLMULQDQ(xor_a_128, xor_b_128, 0x00);
```

Example of Jasmin code (part of the GHASH function) using low-level instructions for performance

MASTER INFORMATIQUE
UNIVERSITÉ CÔTE D'AZUR
Inria
DIGITAL SYSTEMS FOR HUMANS
GRADUATE SCHOOL AND RESEARCH
POLYTECH NICE-SOPHIA