Flight Software:

Experimental work on secure communication

Author: Arthur Morain
Date: November 2025

Contents

1 Introduction

As cyber threats continue to grow exponentially, enhancing security requirements has become essential to mitigate risks related to confidentiality, integrity, and authentication.

Although the TOLOSAT nanosatellite project does not include propulsion systems or handle confidential communication data, it remains important to integrate basic security mechanisms in compliance with international standards. According to the *Consultative Committee for Space Data Systems (CCSDS) Cryptographic Algorithms Recommended Standard (CCSDS 352.0-B-2)*, secure algorithms should be implemented on mission communication links—whether on the forward space link (telecommand), the return space link (telemetry and science data), or within the ground data network. [1]

In this context, the onboard flight software of TOLOSAT is developed using **STM32H743/753 and STM32H750 advanced ARM®-based 32-bit microcontrollers**. These processors provide significant computational performance and include embedded cryptographic hardware accelerators, such as AES, HASH, and RNG units, enabling efficient implementation of authentication and encryption mechanisms while minimizing power consumption. [2]

Even if TOLOSAT operates as an educational and non-classified mission, it must be considered that satellites can be targeted by spoofing or unauthorized command attempts, especially when flying over countries with active signal transmission infrastructures.

2 Cryptographic processor (CRYP)

The cryptographic processor allows us to encrypt and decrypt data by using multiples algorithm.

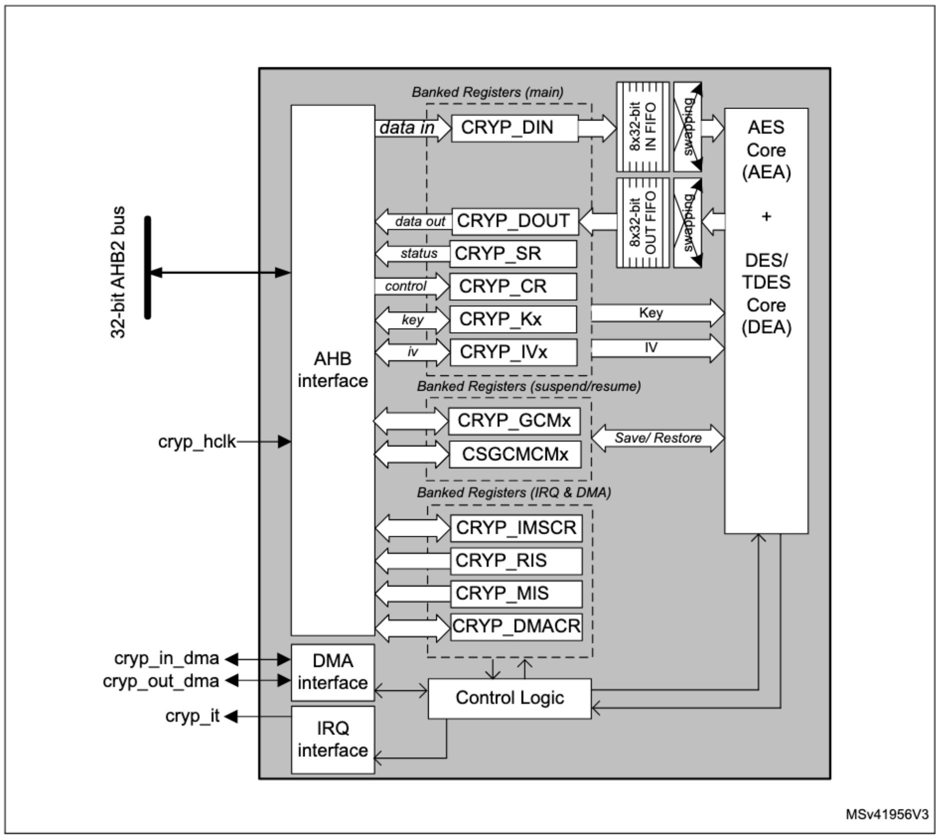

Figure 1: STM32H7 Cryptographic Unit Block Diagram (source: STMicroelectronics, RM0433, 2023)

2.1 Cryptographic Key Storage and Size

According to CCSDS 352.0-B-2 (section 3.2.1) [1], a 256-bit key is required to ensure secure communication links. On the STM32H743/753 and STM32H750 microcontrollers, this key can be directly stored in the hardware cryptographic key registers, denoted as *CRYP_Kx*. Each register holds a portion of the full 256-bit key, allowing the microcontroller to access the key internally without exposing it in main memory.

2.2 Cryptographic Algorithm

With the STM32 microcontroller, several symmetric block cipher algorithms can be considered, including DES, TDES, and AES. DES is no longer considered secure, as it is vulnerable to brute-force attacks due to its short 56-bit key. TDES (Triple DES) applies the DES algorithm three times to increase security, but it remains relatively slow and less efficient for embedded systems. AES, on the other hand, is a modern and highly efficient algorithm, capable of operating with 128-, 192-, or 256-bit keys.

Since CCSDS 352.0-B-2 (section 3.2.1) [1] requires the use of a 256-bit key, AES is the only suitable choice for TOLOSAT. Its combination of strong security and hardware acceleration on the STM32H7 microcontroller makes it ideal for secure satellite communications.

The AES algorithm is specified in the National Institute of Standards and Technology (NIST) Federal Information Processing Standard (FIPS) 197 [3] and ISO/IEC 18033-3 [4], providing an internationally recognized standard for secure symmetric block encryption.

2.3 Authenticated Encryption

Authenticated encryption is a cipher mode that provides **confidentiality, integrity, and authenticity simultaneously**. This approach is also referred to as **Authenticated Encryption with Associated Data (AEAD)**. According to the CCSDS Recommended Standard for Cryptographic Algorithms (CCSDS 352.0-B-2, 2019, section 2.4), performing encryption without authentication is considered insecure and is strongly discouraged for space missions [1].

For the TOLOSAT project, we will use **AES in Galois/Counter Mode (AES-GCM)** on the STM32H743/753/750 microcontrollers. This mode provides both **high performance and strong security guarantees**, and the STM32 hardware cryptographic unit allows efficient computation of AES-GCM directly using the 256-bit key stored in the *CRYP_Kx* registers.

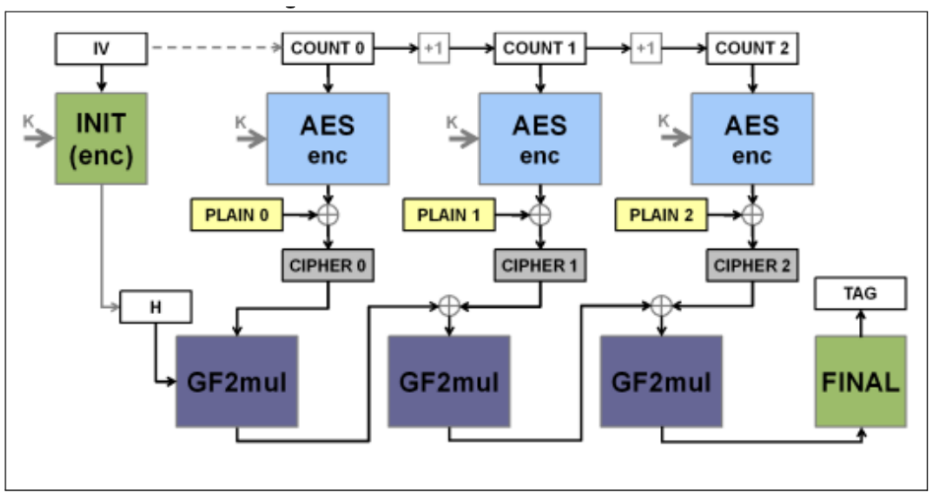

Figure 2: Diagram of the AES Galois/Counter mode (GCM). (Source: STMicroelectronics, RM0433)

According to [2], in Galois/Counter Mode (GCM), the plaintext message is encrypted while a Message Authentication Code (MAC) is computed in parallel. This process produces both the ciphertext and its corresponding MAC, also referred to as the authentication tag, providing simultaneous confidentiality and integrity.

2.4 CRYP flowchart with AES

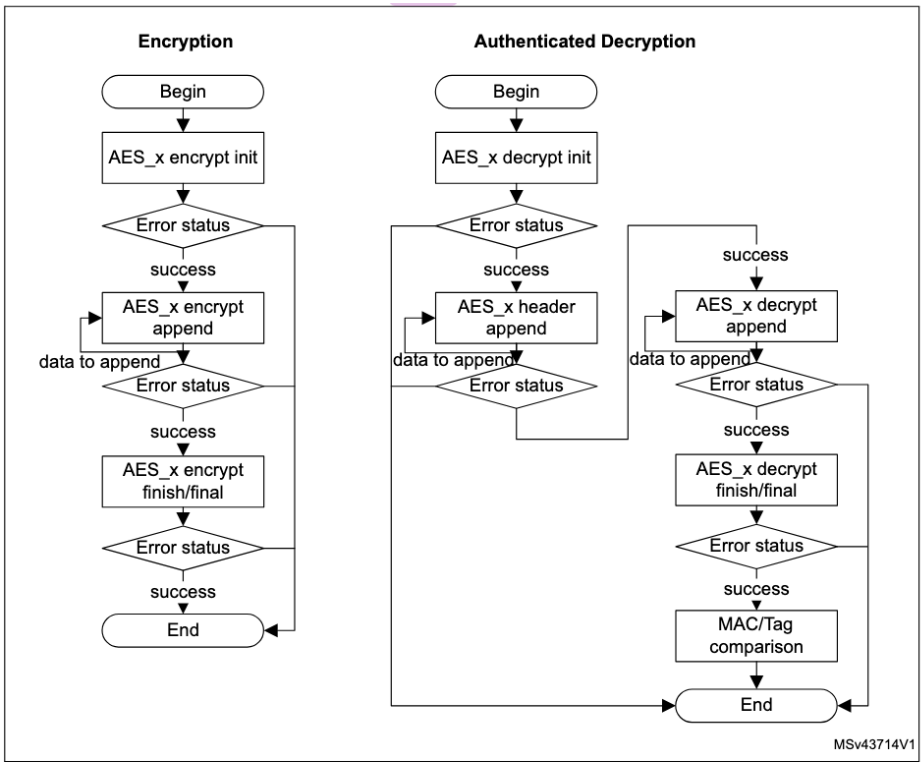

Figure 3: flowchart with AES. (Source: STMicroelectronics, RM0433)

3 Proof of Concept 1: Encrypt and Decrypt a Payload using the Processor

3.1 Use Case: Securing the Communication Between the Ground Station and the Satellite

In this use case, the goal is to demonstrate the feasibility of securing communications between the satellite and the ground segment using the STM32H7 hardware cryptographic processor.

The satellite periodically receives **Telecommands (TC)** from the ground station and transmits **Telemetry (TM)** back. To ensure confidentiality and integrity, each message must be encrypted (for TM) or decrypted (for TC) using the AES-GCM algorithm implemented by the hardware CRYP peripheral.

Scenario

The PoC simulates this communication loop on a single MCU:

1. A payload (e.g., the ASCII message `"HELLO TOLOSAT"`) is stored in RAM.
2. The MCU encrypts this payload using AES-256-GCM, generating both the ciphertext and the authentication tag.

3. The encrypted frame is transmitted over UART, simulating a TM transmission to the ground.
4. The same MCU then receives this encrypted frame through UART (loopback simulation).
5. The CRYP peripheral decrypts the payload and verifies the authentication tag to ensure data integrity.
6. The recovered plaintext is compared with the original message to confirm successful encryption/decryption.

Frame Format

Each message transmitted over UART follows the structure:

```
[Header | IV | Ciphertext | AuthTag]
```

Where:

- `Header` identifies the message type (source, destination, payload length).
- `IV` is a 96-bit nonce, regenerated for each encryption.
- `Ciphertext` is the encrypted payload.
- `AuthTag` (128 bits) ensures message integrity and authenticity.

Expected Result

At the end of the process, the decrypted message in memory must be identical to the original plaintext, and the authentication tag must be successfully verified. This validates that the hardware AES-GCM peripheral can handle both encryption and decryption operations required for secure TC/TM exchanges in flight.

3.2 TC Transmission

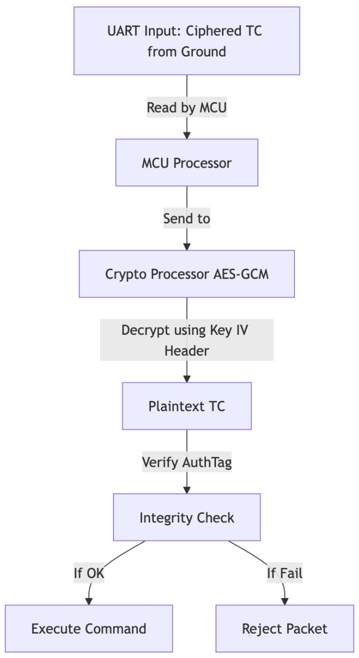

Figure 4: Transmission of a Telecommand (TC) encrypted to the MCU. The MCU decrypts the message using the Crypto Processor and verifies its authenticity with the AuthTag before execution.

3.3 TM Transmission

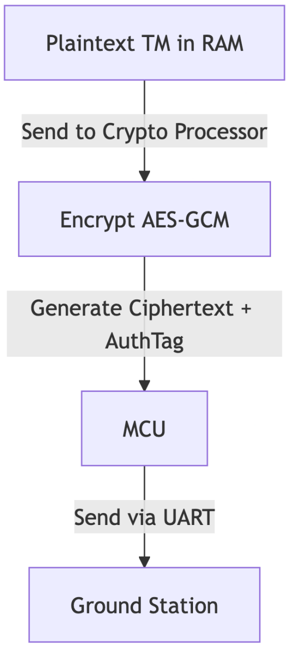

Figure 5: Transmission of Telemetry (TM) encrypted from the MCU to the ground station. The Crypto Processor encrypts the message, generating both the ciphertext and the AuthTag, which are then sent via UART.

3.4 Results

The proof-of-concept demonstrates the secure transmission of data between the ground station and the MCU using AES-GCM encryption. The results for a sample message are summarized below, with each message represented as four 32-bit words in hexadecimal format.

Plaintext	`0x4c4c4548, 0x4f54204f, 0x41534f4c, 0x54000000`
Ciphertext	`0x710ce286, 0x213f4002, 0x9296014b, 0x189df3ee`
Decrypted	`0x4c4c4548, 0x4f54204f, 0x41534f4c, 0x54000000`

Table 1: Hexadecimal representation of plaintext, ciphertext, and decrypted data for the PoC. The decrypted values match the original plaintext.

Confidentiality: The plaintext is transformed into ciphertext using AES-GCM with a 256-bit key and a unique initialization vector (IV) for each message. This ensures that the message content remains secret during transmission.

Integrity: The Galois/Counter Mode (GCM) generates an authentication tag (AuthTag) during encryption. Upon decryption, the MCU verifies this tag. Any modification of the ciphertext or associated data would cause the verification to fail, ensuring the integrity of the message.

Authenticity: The AuthTag also provides authentication, allowing the MCU to confirm that the message originates from a trusted source (i.e., the ground station) and not from an attacker attempting to inject or tamper with messages.

4 Proof of Concept 2: Encrypt and Decrypt a Payload using DMA

4.1 Motivation and Advantage of DMA

Using Direct Memory Access (DMA) for cryptographic operations offloads data transfer and memory management from the MCU. This provides several advantages:

- **Reduced MCU Load:** The MCU can perform other tasks (e.g., telemetry processing, control loops) while data is automatically moved to/from the crypto processor.
- **Faster Data Transfer:** DMA can move large payloads efficiently without manual intervention.
- **Lower Latency:** Continuous streams of data (e.g., TC or TM packets) can be encrypted/decrypted in parallel with other MCU operations.
- **Deterministic Timing:** DMA avoids MCU cycle-by-cycle handling, reducing timing jitter for real-time systems like satellites.

4.2 Use Case: Securing Communications with DMA

In this PoC, we simulate a satellite scenario:

1. The ground station sends a ciphered Telecommand (TC) to the satellite.
2. The MCU sets up a DMA channel to transfer the incoming TC directly to the Crypto Processor (AES-GCM).
3. The Crypto Processor decrypts the message while the DMA handles data movement.
4. The MCU verifies the authentication tag (AuthTag) after decryption.
5. For Telemetry (TM) data, the MCU prepares the plaintext in RAM and uses DMA to move it to the Crypto Processor for encryption.
6. The DMA returns the ciphertext and AuthTag to RAM, ready for transmission via UART.

4.3 Conceptual Diagram: TC Reception via DMA

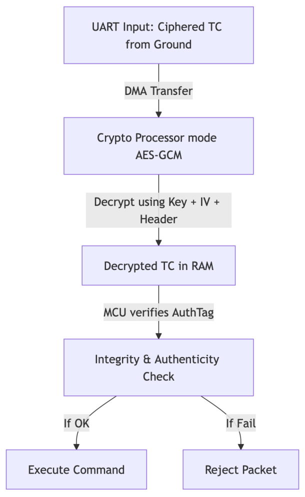
UART Input: Ciphered TC from Ground
DMA Transfer
Crypto Processor mode AES-GCM
Decrypt using Key + IV + Header
Decrypted TC in RAM
MCU verifies AuthTag
Integrity & Authenticity Check
If OK
If Fail
Execute Command
Reject Packet

Figure 6: Decryption of a TC packet using DMA to transfer data to the Crypto Processor. The MCU verifies the AuthTag after decryption.

4.4 Conceptual Diagram: TM Transmission via DMA

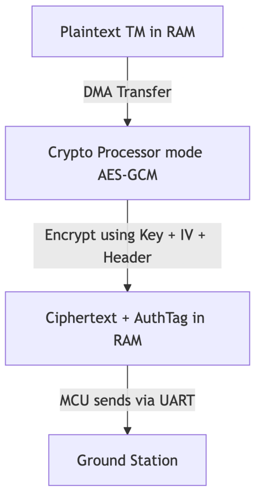
Plaintext TM in RAM
DMA Transfer
Crypto Processor mode AES-GCM
Encrypt using Key + IV + Header
Ciphertext + AuthTag in RAM
MCU sends via UART
Ground Station

Figure 7: Encryption of TM data using DMA. The MCU offloads the transfer to DMA and sends the resulting ciphertext and AuthTag via UART.

4.5 Results and Analysis

The second Proof of Concept (PoC2) aimed to validate the use of the DMA controller to offload the MCU during AES-GCM operations. By transferring the plaintext and ciphertext directly between system memory and the CRYP peripheral, the MCU is released from data movement tasks and only supervises the crypto process through interrupts or polling of completion flags.

Execution behavior. When comparing the two implementations:

- **CPU-based PoC1:** The MCU must actively push and read 32-bit words to and from the CRYP registers, keeping the CPU occupied during the entire encryption process.
- **DMA-based PoC2:** Once the DMA channels are configured, the MCU initiates the operation and can resume other tasks while the DMA engine automatically feeds the CRYP input and collects the output data.

The output logs from UART confirmed correct encryption and decryption sequences using DMA. The generated `AuthTag` matched between encryption and decryption, demonstrating that integrity and authenticity were preserved. No errors were reported in the DMA transfer or CRYP operation, confirming the reliability of the hardware path.

Security verification. As with PoC1, all three core properties were successfully validated:

- **Confidentiality:** The payload is encrypted using AES-256-GCM within the CRYP hardware block.
- **Integrity:** The AES-GCM `AuthTag` detects any data corruption or modification.
- **Authenticity:** Successful tag verification ensures that the message originates from a trusted source.

Conclusion. The DMA-based implementation proved that the STM32H7 cryptographic accelerator can operate efficiently in a fully autonomous data transfer mode. This configuration minimizes CPU overhead while maintaining secure and authenticated communication, which is essential for satellite OBC subsystems such as secure telecommands and telemetry channels.

4.6 Discussion

This PoC demonstrates that using DMA for crypto operations improves efficiency and reduces MCU intervention. For satellites, where CPU time is precious and real-time tasks are critical, DMA allows secure communications without delaying other processes. The confidentiality, integrity, and authenticity checks remain identical to the MCU-only PoC, but the overall system throughput is higher.

5 Implementation Notes and Key Considerations

Throughout the development of the two Proofs of Concept (PoC1 – CPU-driven and PoC2 – DMA-assisted), several important points were identified. These notes apply to both implementations and summarize key lessons for future software integration.

- **Initialization Vector (IV):** The IV must be updated for each message transmission, regardless of the PoC implementation. Reusing the same IV with the same key compromises the AES-GCM security guarantees (confidentiality and authenticity). In a flight context, a hardware RNG or a persistent counter should be used to ensure uniqueness.
- **Key Configuration:** The AES encryption key should be configured directly into the CRYP peripheral registers rather than through higher-level abstractions. This approach guarantees full control of the hardware and simplifies debugging across both PoCs.
- **Header Field:** The GCM header (AAD) should carry relevant metadata such as packet type, sequence number, or timestamp. However, since the source and destination are constant in both PoCs, the header may be simplified or used as a placeholder for authentication validation.
- **DMA Data Size (PoC2 specific):** In the DMA-assisted version, the DMA transfers must use byte-sized data rather than 32-bit words to maintain proper alignment and data flow. This prevents incomplete or corrupted encryption blocks.
- **HAL CRYP Driver Limitation:** The function

  ```
  CRYP_SetDMAConfig(hcryp, (uint32_t)(hcryp->pCrypInBuffPtr),
                    (uint16_t)wordsize,
                    (uint32_t)(hcryp->pCrypOutBuffPtr));
  ```

  provides no return value, making debugging difficult. It is invoked internally by `CRYP_AESGCM_Process_DMA()` and hides DMA configuration or runtime errors. Manual inspection of DMA registers or the CRYP state machine is often required.

Overall, while both PoCs validate the AES-GCM flow (confidentiality, integrity, authenticity), these implementation notes highlight the importance of low-level control when integrating cryptography on embedded systems.

6 Cryptography primer (educational)

6.1 What is AES?

AES (Advanced Encryption Standard) is a symmetric block cipher standardized by NIST. AES works on fixed-size blocks (128 bits) and supports key sizes of 128, 192 and 256 bits. In embedded systems we commonly use AES-128 or AES-256 depending on the required security margin. AES itself provides *confidentiality* — it makes plaintext unreadable without the key.

6.2 What makes a cryptographic algorithm "secure"?

A cryptographic primitive or mode is considered secure when:

- It resists known practical attacks (ex. brute force, differential/linear cryptanalysis) for realistic resource bounds.
- Its security assumptions are well studied and publicly reviewed.
- It is used with *correct* parameters and patterns (unique nonces, proper key management, no key re-use with incompatible modes).
- Implementation side-channels (timing, power, fault) are mitigated when required.

6.3 Why authentication matters

Encryption alone (confidentiality) does not guarantee that the ciphertext was created by an authorised sender or that it wasn't altered. Attackers can flip ciphertext bits, replay messages, or substitute messages. Authentication ensures:

- **Integrity:** any modification of header or ciphertext is detected.
- **Authenticity:** the receiver can verify the message origin (it was constructed under the shared key or authorized private key).
- **Replay protection:** when combined with sequence numbers / nonces.

6.4 Man-in-the-Middle (MITM) – conceptual diagram

An attacker sitting between ground and satellite can:

- intercept commands (read confidentiality),
- modify or inject commands (break integrity),
- replay old commands (break freshness).

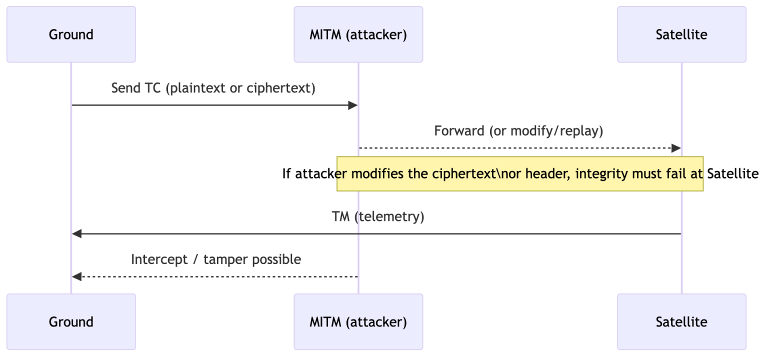
Ground
MITM (attacker)
Satellite
Send TC (plaintext or ciphertext)
Forward (or modify/replay)
If attacker modifies the ciphertext\nor header, integrity must fail at Satellite
TM (telemetry)
Intercept / tamper possible
Ground
MITM (attacker)
Satellite

Figure 8: Man-in-the-Middle (MITM) attack: eavesdrop, modify, inject or replay messages.

6.5 Authenticated Encryption with Associated Data (AEAD) and AES-GCM

AEAD modes (Authenticated Encryption with Associated Data) provide both confidentiality and authentication in one primitive. AES-GCM (Galois/Counter Mode) is an AEAD mode widely used in practice:

- Uses AES in counter mode to encrypt (stream-like): provides confidentiality.
- Uses a GHASH (Galois field) function to compute an authentication tag over ciphertext *and* associated data (AAD/header): provides integrity and authenticity.
- Produces a fixed-length AuthTag (commonly 128 bits). The receiver recomputes the tag and compares it: mismatch ⇒ reject.

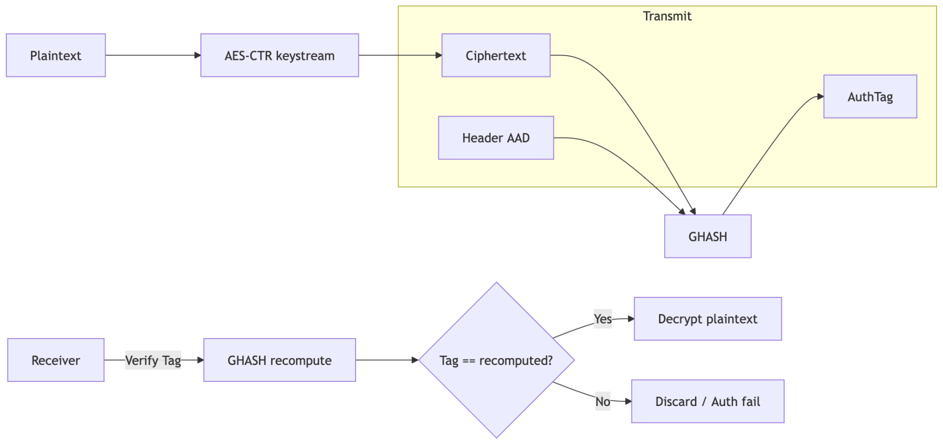

Figure 9: AES-GCM high-level flow: AES-CTR for confidentiality, GHASH for authentication (AAD + ciphertext) producing the AuthTag.

6.6 How AES-GCM mitigates MITM

- Encryption hides plaintext from the eavesdropper.
- The AuthTag protects both the ciphertext and the header (AAD). Any alteration (bit flips, substitutions) changes the tag and is detected.
- Nonce/IV uniqueness prevents replay attacks and keeps counter-based keystreams distinct.

6.7 Practical recommendations (summary)

- **IV / Nonce:** must be unique per message for a given key. Use RNG or a persistent counter (counter is safest for strict uniqueness).
- **Key management:** never hardcode production keys; use secure provisioning and protect them (secure elements, secure boot, encrypted storage).
- **Header / AAD:** include metadata that must be authenticated but not encrypted (source, dest, message type, sequence number).

- **Tag verification:** always verify the auth tag before using decrypted data. On failure, discard and log.
- **Tag length:** 128-bit tags are common; shorter tags reduce security margin.
- **Side-channels:** consider timing/power protections if attacker proximity/faults are plausible.

6.8 Educational note

This section is intentionally concise and educational: for a production system, consult formal standards (NIST SP 800-38D for GCM), and perform a security review (threat model, key lifecycle, fault/side-channel analysis).

References

[1] CCSDS 352.0-B-2, *Cryptographic Algorithms Recommended Standard*, Consultative Committee for Space Data Systems (CCSDS), Washington, D.C., September 2020.

[2] STMicroelectronics, *STM32H743/753 and STM32H750 Advanced ARM®-based 32-bit MCUs Reference Manual (RM0433)*, 2023.

[3] National Institute of Standards and Technology (NIST), *Advanced Encryption Standard (AES)*, Federal Information Processing Standards Special Publication 197, Gaithersburg, Maryland, 2001.

[4] International Organization for Standardization (ISO), *Information Technology—Security Techniques—Encryption Algorithms—Part 3: Block Ciphers*, ISO/IEC 18033-3:2010, 2nd edition, Geneva, 2010.