CISC 468: CRYPTOGRAPHY

LESSON 18: AUTHENTICATED ENCRYPTION

Furkan Alaca

READINGS

- Section 5.1.6: Galois Counter Mode (GCM), Paar & Pelzl
- Padding Oracle Attack, Wikipedia

ENCRYPTION ALONE DOES NOT GUARANTEE INTEGRITY

- Encryption alone does not any integrity or authentication assurances
- The argument that "manipulating the ciphertext will almost always lead to a meaningless message when decrypted" is not always true
- Attackers with knowledge of the message structure may leave intact ciphertext bits corresponding to message fields that the receiver has the ability to validate, and instead manipulate bits that correspond to fields that are more difficult to verify, such as dollar amounts or random data

AUTHENTICATED ENCRYPTION

Combining symmetric encryption with a MAC allows us to obtain the benefits of both:

1. Confidentiality
2. Integrity
3. Source authenticity

AUTHENTICATED ENCRYPTION: ENCRYPT-AND-MAC

- The sender:
  - Encrypts the plaintext: $y = e(x)$
  - MACs the plaintext: $m = MAC(x)$
  - Sends (y, m) to the receiver
- The receiver first decrypts y, and then verifies m

AUTHENTICATED ENCRYPTION: MAC-THEN-ENCRYPT

- The sender:
  - MACs the plaintext: $m = MAC(x)$
  - Encrypts the plaintext and the MAC: $y = e(x||m)$
  - Sends y to the receiver
- The receiver first decrypts y, and then verifies m

PADDING ORACLE ATTACKS

- This attack is most commonly associated with the CBC mode of operation, and takes advantage of:
  - The fact that the receiver must first decrypt y in order to verify m, if using either Encrypt-and-MAC or MAC-then-Encrypt
  - The plaintext padding required by CBC

PKCS#7 PADDING

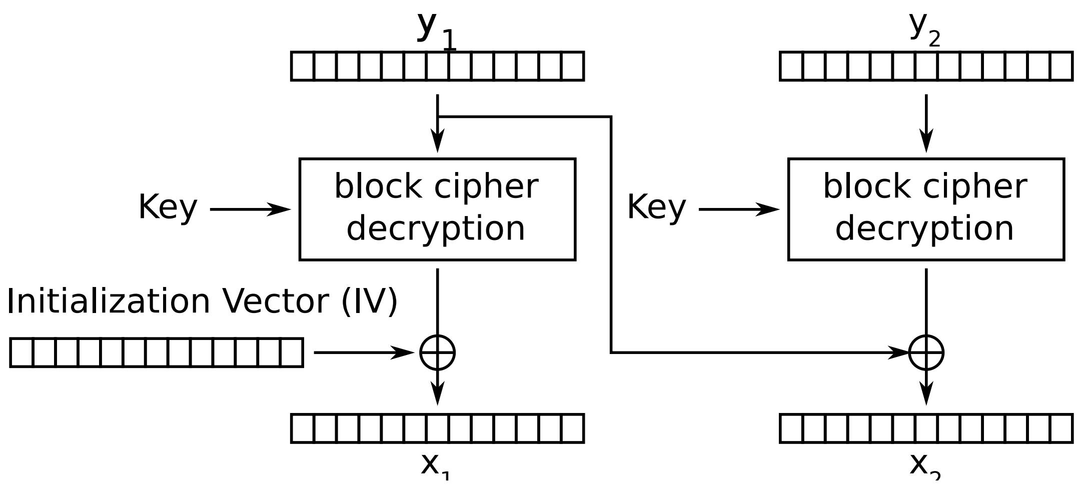
- PKCS#7 padding is most commonly used
- The plaintext is padded to a multiple of a full block length by appending N bytes, each with the value N
  - e.g., if 4 bytes of padding is required, the hex bytes 04040404 are appended
- If the plaintext is an exact multiple of the block length, a full block consisting of the repetition of the value B is added, where B is the block length in bytes
- When the receiver decrypts the ciphertext, it must verify that the padding is valid

PADDING ORACLE ATTACK: PREMISE

- If the receiver needs to decrypt the ciphertext before verifying the MAC, two distinct failure cases may arise during decryption/authentication:
  - Decryption fails due to invalid padding, so the MAC is not checked
  - Decryption succeeds, but the MAC fails
- A padding oracle attack can take place if an attacker can distinguish between these two errors
  - Even if the server doesn't signal which of the two error cases occured, the sender may distinguish between them based on timing

PADDING ORACLE ATTACK: LEVERAGING CBC DECRYPTION

- The attacker constructs y_1' by changing the last byte R of y_1
- Upon decryption, this changes the last byte in x_2, which will almost certainly invalidate the padding



- But the attacker can try again and change R to another value
- If the last byte decrypts to 01, the padding check will succeed but the MAC verification will fail

PADDING ORACLE ATTACK: DECRYPTING BYTES

- After at most 256 attempts, the receiver will signal a MAC verification error
- The attacker then learns that the last byte of $y_1' \oplus d(y_2)$ is 01
  - So, the last byte of $d(y_2) = y_1' \oplus 01$
  - And we decrypt the last byte of $x_2 = d(y_2) \oplus y_1$
- The attacker then sets the last byte of y_1' to $d(y_2) \oplus 02$, which will cause the last byte of y_2 to decrypt to 02
- The attacker can decrypt the entire block y_2 in 4080 attempts

REAL-WORLD PADDING ORACLE ATTACKS

- Original attack was published by Vaudenay in 2002, and soon shown to be applicable to SSL/TLS
  - Implementations were fixed to defend against this attack
- A decade later in 2013, a new variant Lucky Thirteen was discovered
  - It was found in 2016 that the fix for Lucky Thirteen in OpenSSL introduced yet another padding oracle vulnerability
- The POODLE attack discovered in 2014 downgrades the connection to an older protocol version to make the padding oracle attack possible

AVOIDING PADDING ORACLE ATTACKS

"One can simply try to make error responses time-invariant by simulating a MAC verification even when there is a padding error. One can additionally add some random noise in the time delay."

"For future versions of the TLS protocol we recommend to invert the MAC and padding processes: the sender first pad the plaintext then MAC it, so that the receiver can first check the MAC then check the padding if the MAC is valid. This would thwart any active attack in which messages which are received are not authentic."

Source

ENCRYPT-THEN-MAC

- The sender:
  - Encrypts the plaintext: $y = e(x)$
  - MACs the ciphertext: $m = MAC(y)$
  - Sends (y, m) to the receiver
- The receiver first verifies m, and then decrypts y
  - This is good, since it avoids any kind of operations on the message before first determining that it is authentic
  - Some say that doing otherwise inevitably leads to doom

ENCRYPTION AND MAC KEYS

- Both encryption and MAC require a secret key
- With some pairings of encryption and MAC schemes, such as block cipher encryption in CBC mode along with CBC-MAC, are trivial to break if the same key is used for both
- Some pairings may be safe to use with the same key, but by default it is good practice to use a separate key for encryption and for MAC

AUTHENTICATED ENCRYPTION WITH ASSOCIATED DATA (AEAD)

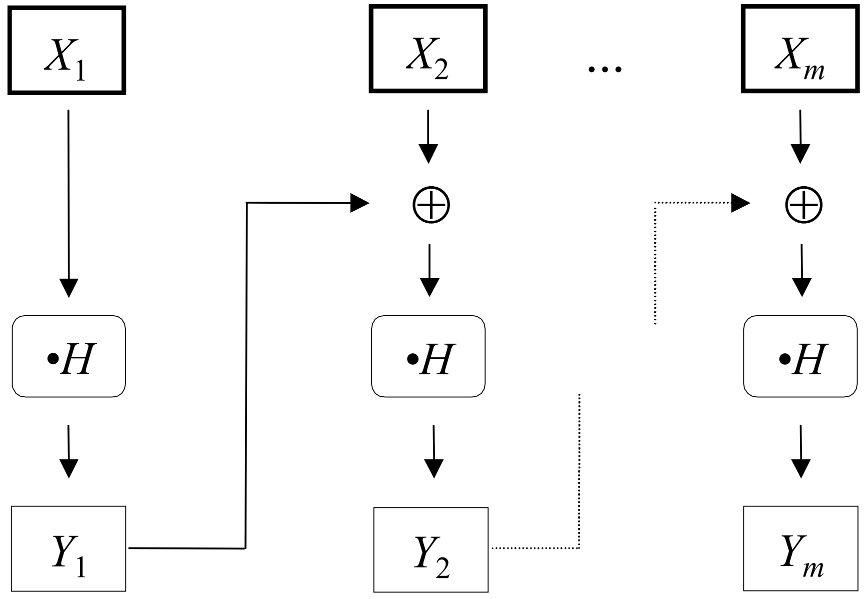
- AEAD algorithms take four inputs:
  - A secret key
  - A nonce (i.e., initialization vector)
  - Plaintext
  - Associated data
- The plaintext is encrypted, but associated data is not
- The MAC is computed on both the ciphertext and associated data
- AEAD (either AES-GCM, AES-CCM, or ChaCha20-Poly1305) is mandatory in TLS 1.3, which is the most recent version of the protocol underlying HTTPS

GALOIS COUNTER MODE (GCM)

- GCM encrypts the plaintext x using CTR mode
- GCM computes an authentication tag over the ciphertext y and a string called *additional authenticated data* (AAD)
  - The algorithm that computes the tag is based on a hash function called GHASH
  - The AAD string includes data that needs to be left unencrypted, such as protocol information (e.g., sequence numbers, source/destination addresses, protocol versions)

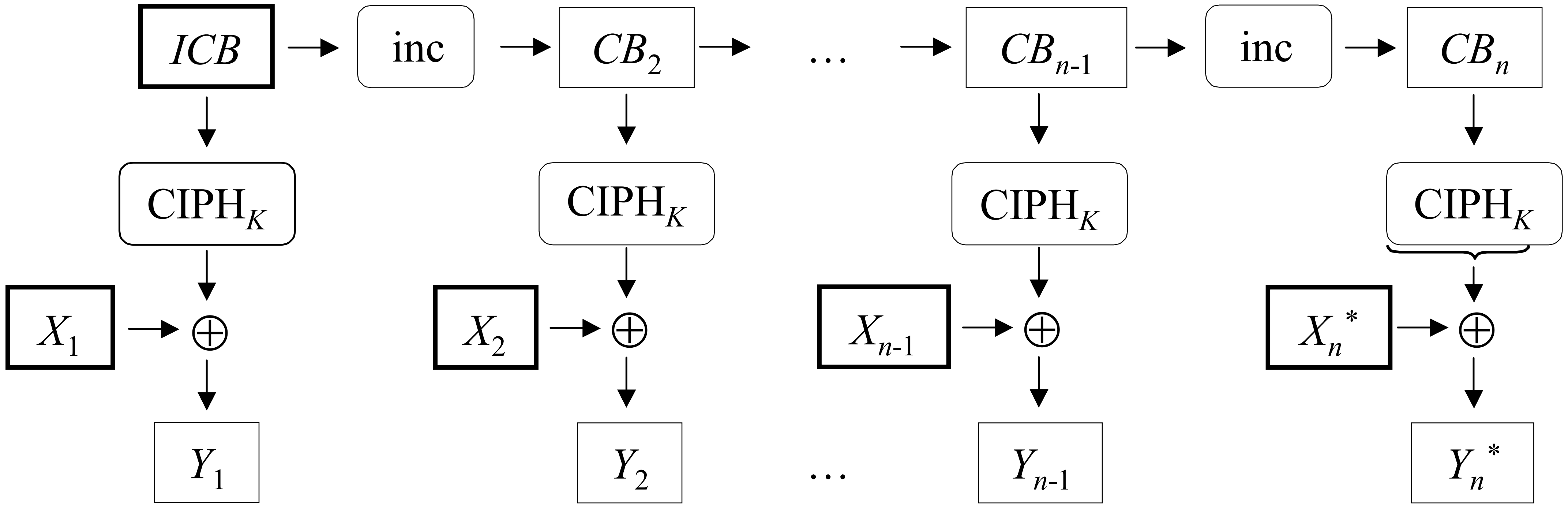
THE GHASH FUNCTION

- The input x is divided into 128-bit blocks $x_1, x_2, \ldots, x_m$
- Let y_0 be the "zero block"
- Compute $y_i = (y_{i-1} \oplus x_i) \cdot H$
  - Addition, multiplication are in the Galois field $GF(2^{128})$
- Return y_m



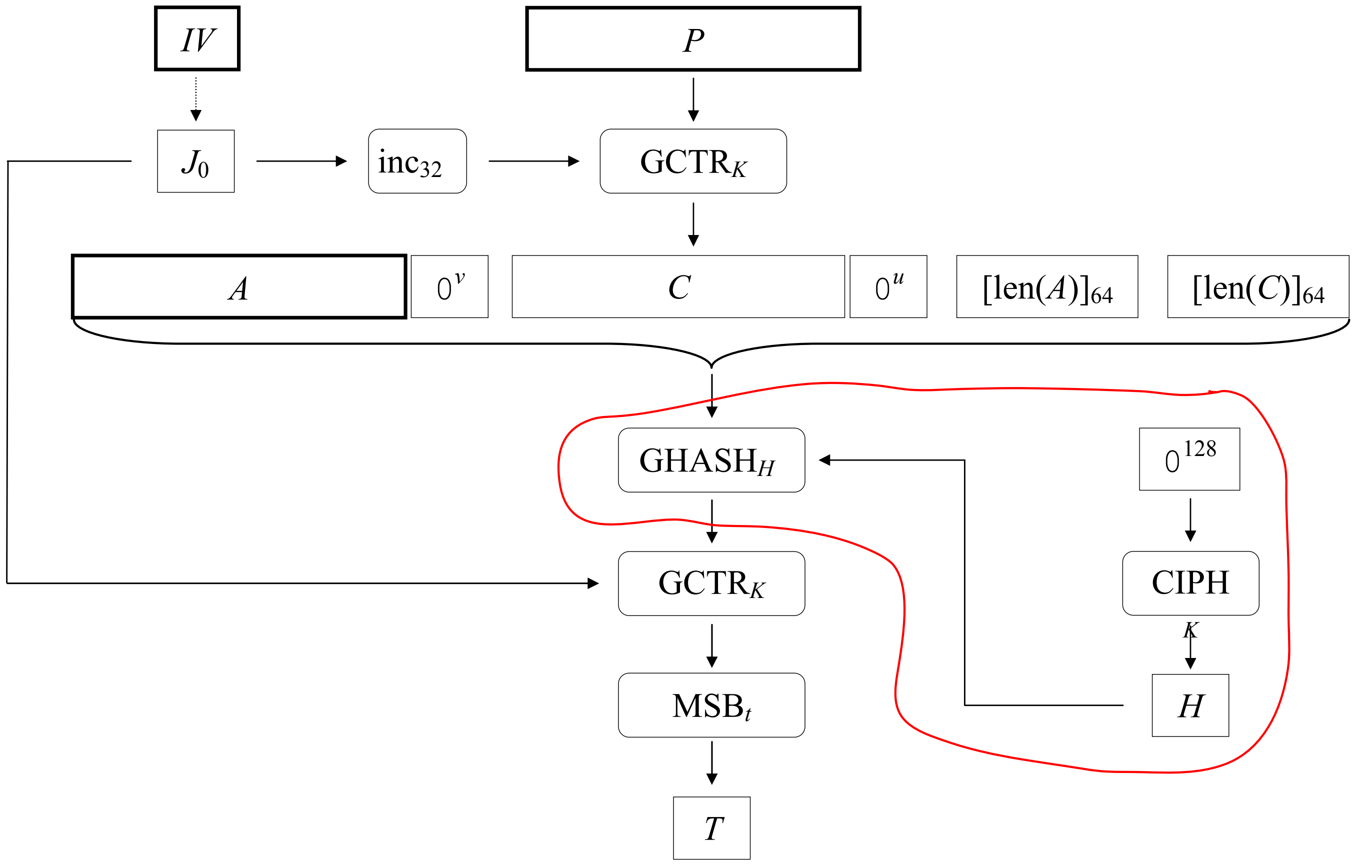
THE GCTR FUNCTION

Uses a block cipher in CTR mode, with left-most 96 bits of ICB serving as a nonce and right-most 32 bits serving as a counter value incremented for each block

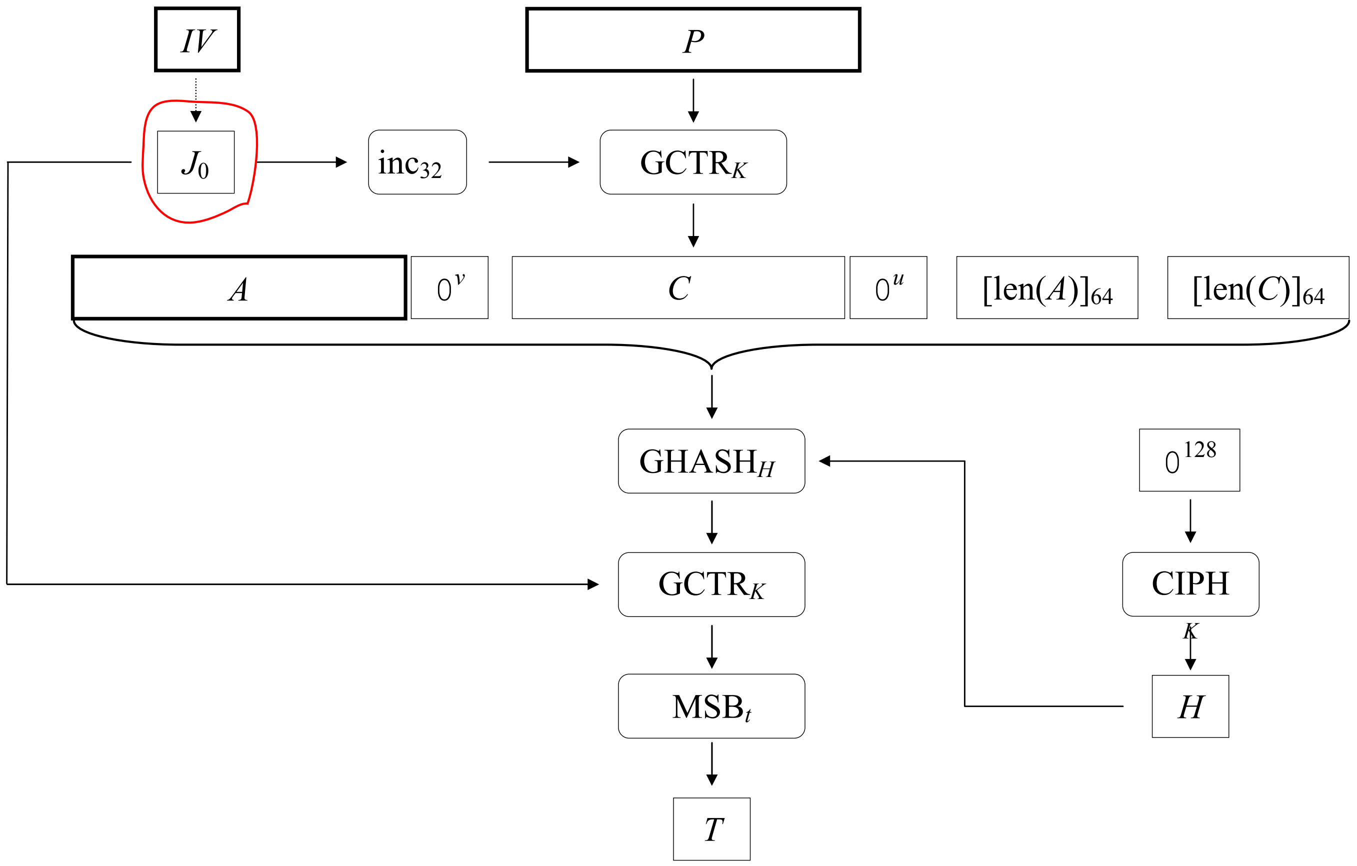


THE GCM MODE OF OPERATION

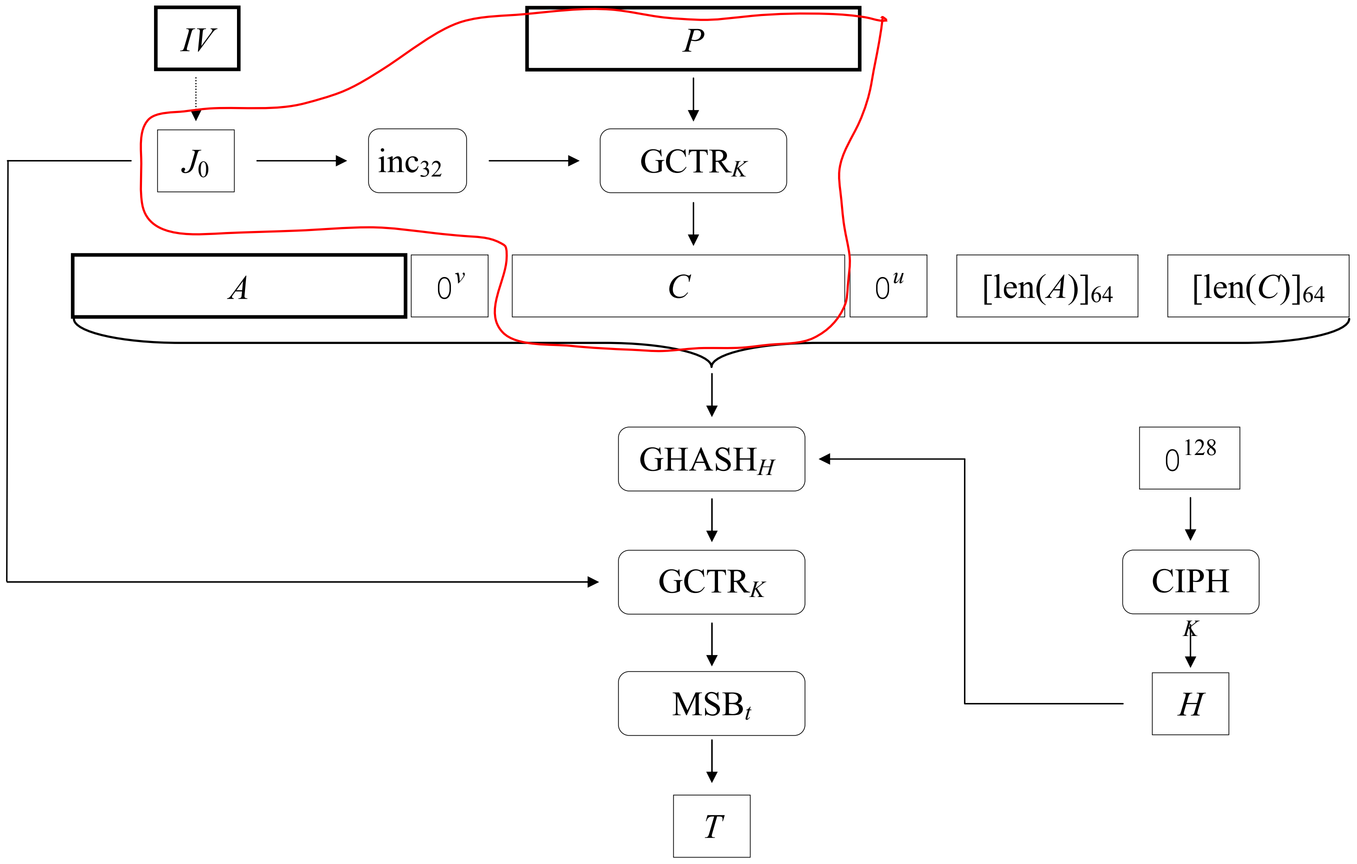
1. The GHASH subkey H is generated by encrypting a zero block with the secret key k



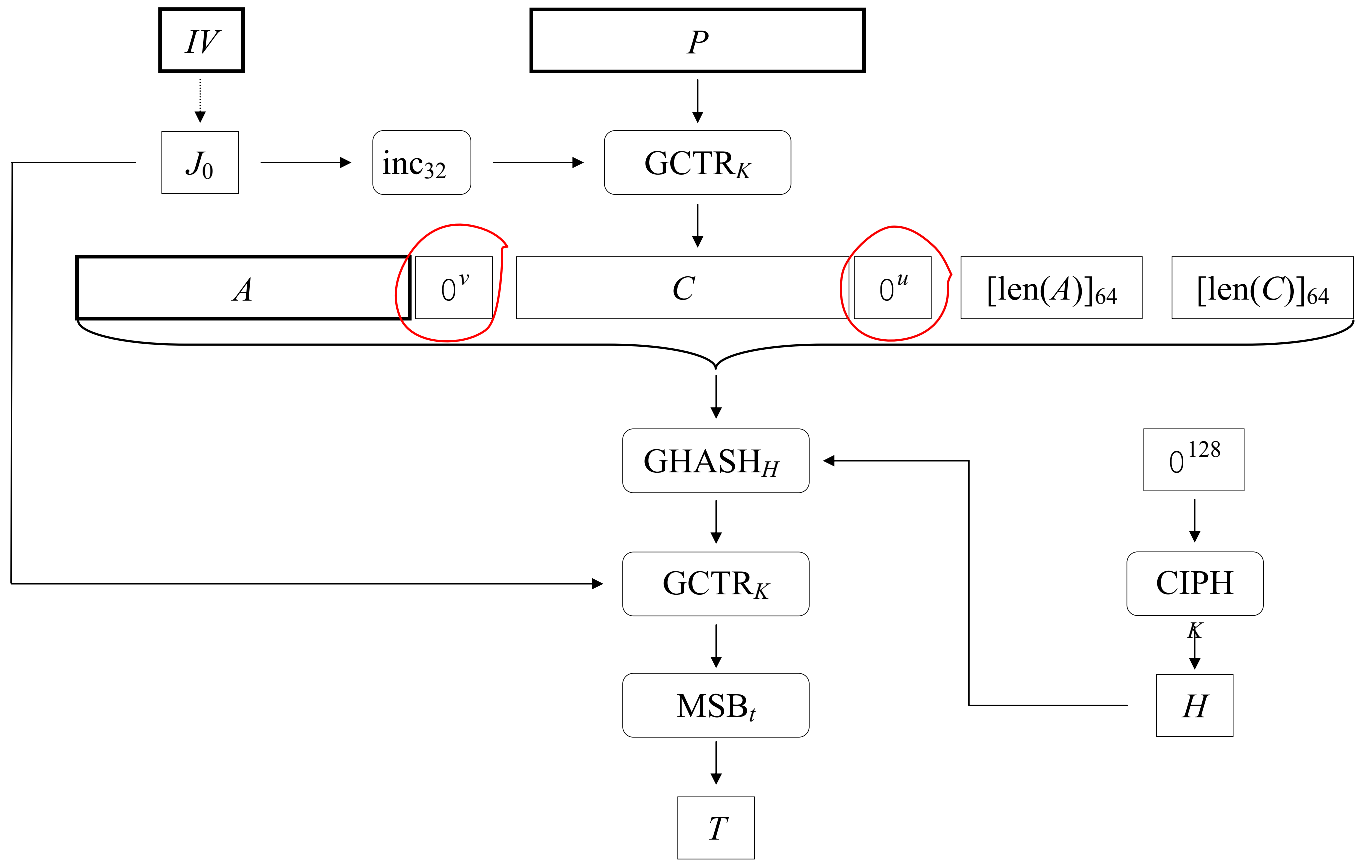
2. The 128-bit block J_0 is the concatenation of a 96-bit IV and 32-bit counter



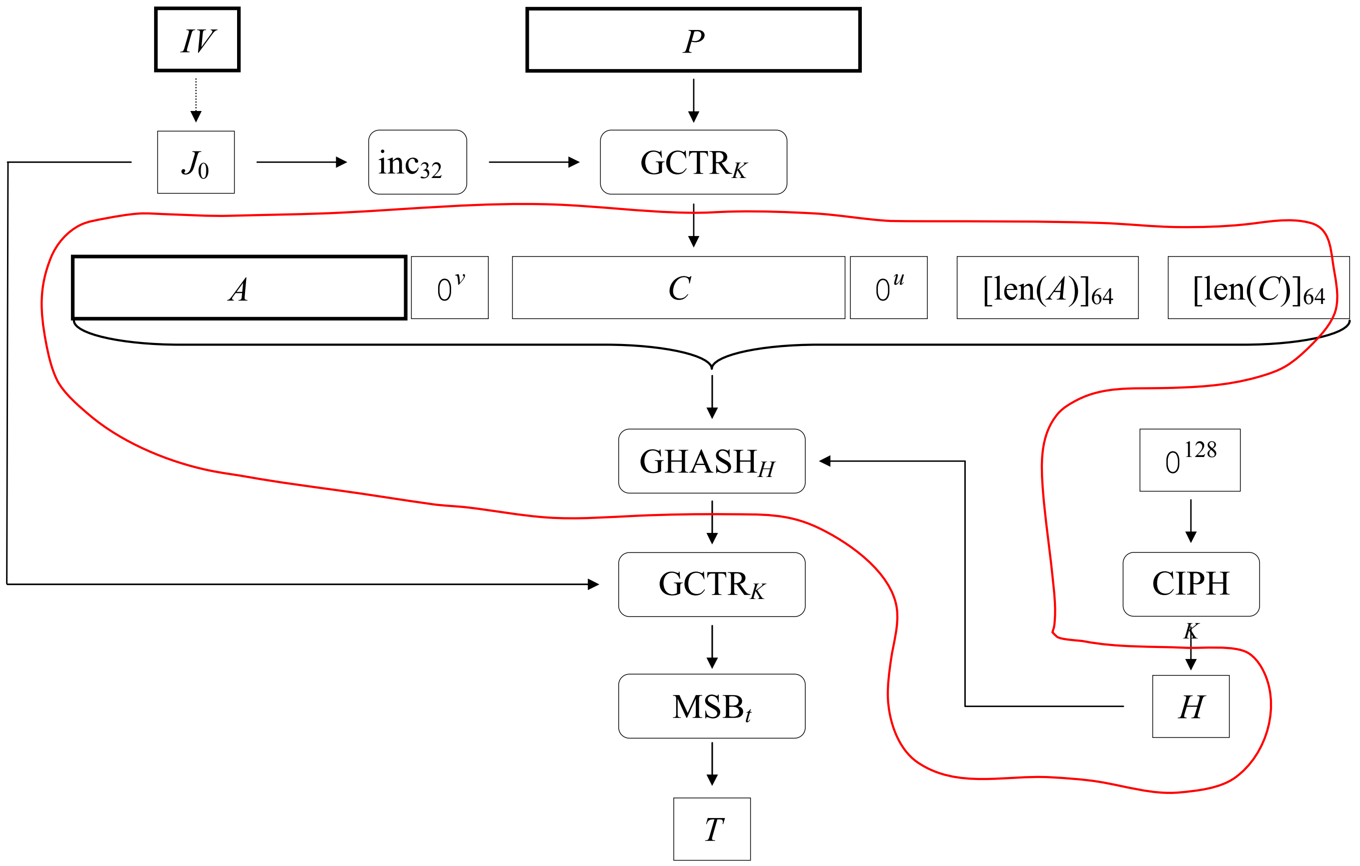
3. The ciphertext C is produced by encrypting the plaintext P with GCTR using the secret key k and initial value J_0.



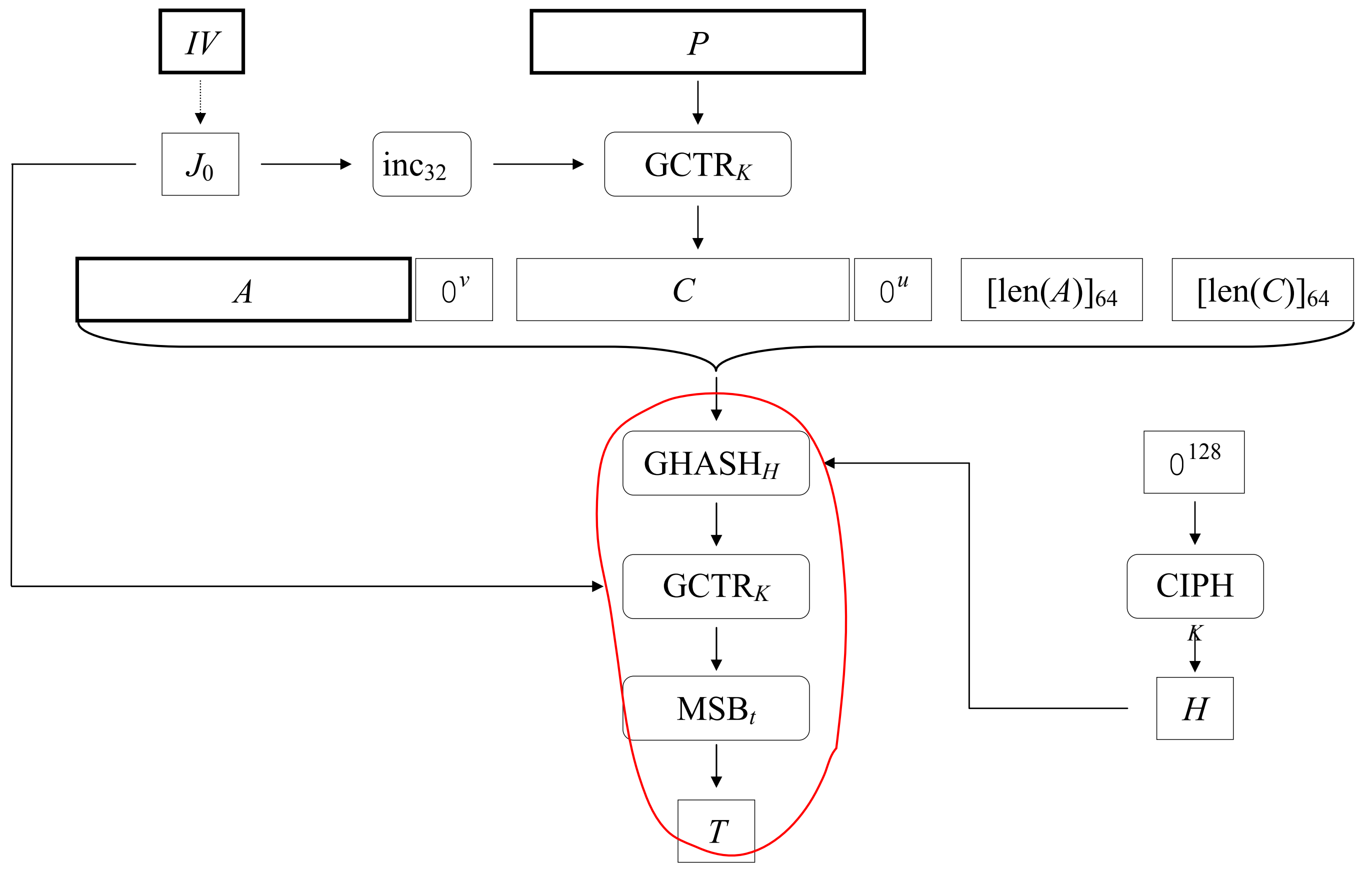
4. Both the AAD and ciphertext are padded with 0 bits to ensure that the resulting strings are a multiple of the block size.



5. The AAD and ciphertext (with padding), the lengths of the AAD and ciphertext, and subkey *H* are passed as input to GHASH.



6. The authentication tag T is generated by encrypting the output of GHASH using GCTR, and truncating the resulting ciphertext block to t bits (typically 128, 120, 112, 104, or 96, or for some applications 64 or 32)


IV
P
J_0
inc_{32}
GCTR_K
A
0^v
C
0^u
$[\text{len}(A)]_{64}$
$[\text{len}(C)]_{64}$
GHASH_H
0^{128}
GCTR_K
CIPH
K
MSB_t
H
T