Integrity and Authenticity

DSC 4110/6111: Modern Cryptography and AI/ML Security

Dr. Laltu Sardar

School of Data Science,
Indian Institute of Science Education and Research Thiruvananthapuram (IISER TVM)

August 5, 2026

Table of Content

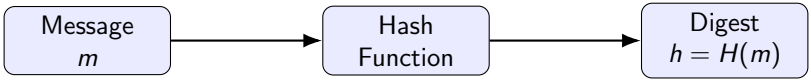
Cryptographic Hash Function

Cryptographic Hash Functions

A cryptographic hash function is a deterministic function

$$H : \{0,1\}^* \to \{0,1\}^n,$$

that maps an arbitrary-length message to a fixed-length digest.



Characteristics

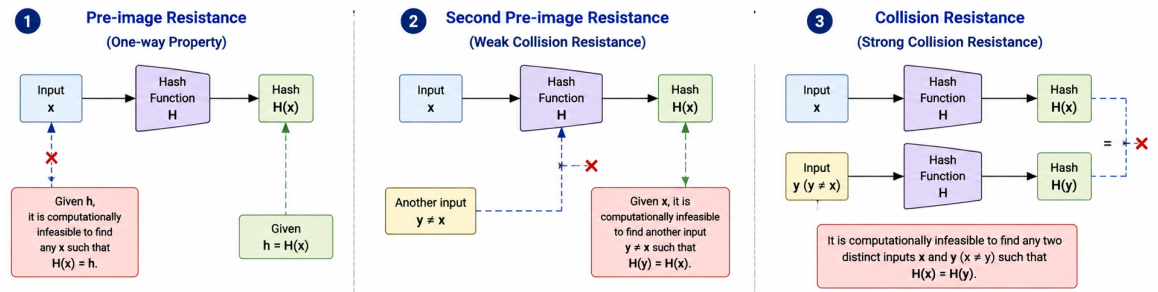
- Input: arbitrary length.
- Output: fixed length (e.g., 256 bits for SHA-256).
- Deterministic and efficiently computable.

Mainly used for Message Integrity

Security Properties of Hash Functions

A secure cryptographic hash function should satisfy

- **Preimage Resistance**
  Given h, it is computationally infeasible to find m such that $H(m) = h$.
- **Second Preimage Resistance**
  Given m, it is computationally infeasible to find $m' \neq m$ such that $H(m) = H(m')$.
- **Collision Resistance**
  It is computationally infeasible to find $m \neq m'$ satisfying $H(m) = H(m')$.



How Are Cryptographic Hash Functions Constructed?

Theoretical Construction

Security can be derived from well-established computational assumptions such as

- Discrete Logarithm Problem (DLP),
- Integer Factorization,
- Lattice problems.

These constructions typically provide provable security but are often computationally expensive.

Modern Construction

Practical hash functions are designed using efficient symmetric operations such as

- compression functions,
- permutations,
- substitution and diffusion,
- ARX operations (Add, Rotate, XOR).

Examples include SHA-2, SHA-3, and BLAKE3.

Key Observation

Unlike public-key cryptography, modern hash functions rely primarily on careful engineering design and extensive cryptanalysis rather than on a single mathematical hardness assumption.

Applications of Hash Functions

Message Digest

$$h = H(m).$$

The digest acts as a compact fingerprint of the message.

Integrity Verification
Sender transmits

$$(m,\ H(m)).$$

The receiver recomputes $H(m)$ and compares the two digests.

Applications

- Message integrity
- Digital signatures
- Message authentication (HMAC)
- Password hashing
- Blockchain
- File verification

Important

A hash function alone provides *integrity checking*, but not authentication. Anyone can compute $H(m)$. Authentication requires a secret key (HMAC) or a digital signature.

Digital Signature Scheme

Digital Signature Scheme

A digital signature scheme consists of three polynomial-time algorithms.

$$(\text{KeyGen}, \text{Sign}, \text{Verify}).$$

$$(pk, sk) \leftarrow \text{KeyGen}(1^\lambda)$$

$$\sigma \leftarrow \text{Sign}_{sk}(m)$$

$$b \leftarrow \text{Verify}_{pk}(m, \sigma)$$

where $b \in \{0, 1\}$.

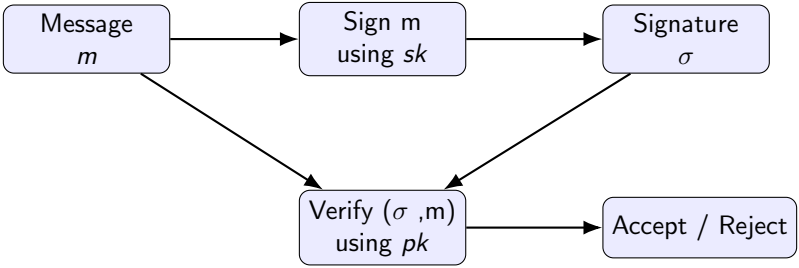
Components

- Public key (pk)
- Secret key (sk)
- Signature (σ)
- Verification algorithm

Purpose

Provide authentication, integrity, and non-repudiation for digital messages.

How Does a Digital Signature Work?



Only the holder of the secret key can generate a valid signature, while anyone possessing the public key can verify it.

A secure signature scheme should satisfy

- **Correctness**
  A valid signature generated using sk must always verify using pk.
- **Existential Unforgeability**
  Without the secret key, no efficient adversary should produce a valid signature on a new message.

Construction of Digital Signature Schemes

Theoretical Construction

Security is based on computational hardness assumptions such as

- Discrete Logarithm Problem,
- Integer Factorization,
- Lattice Problems.

Security is proved by reduction to these hard problems.

Modern Signature Schemes

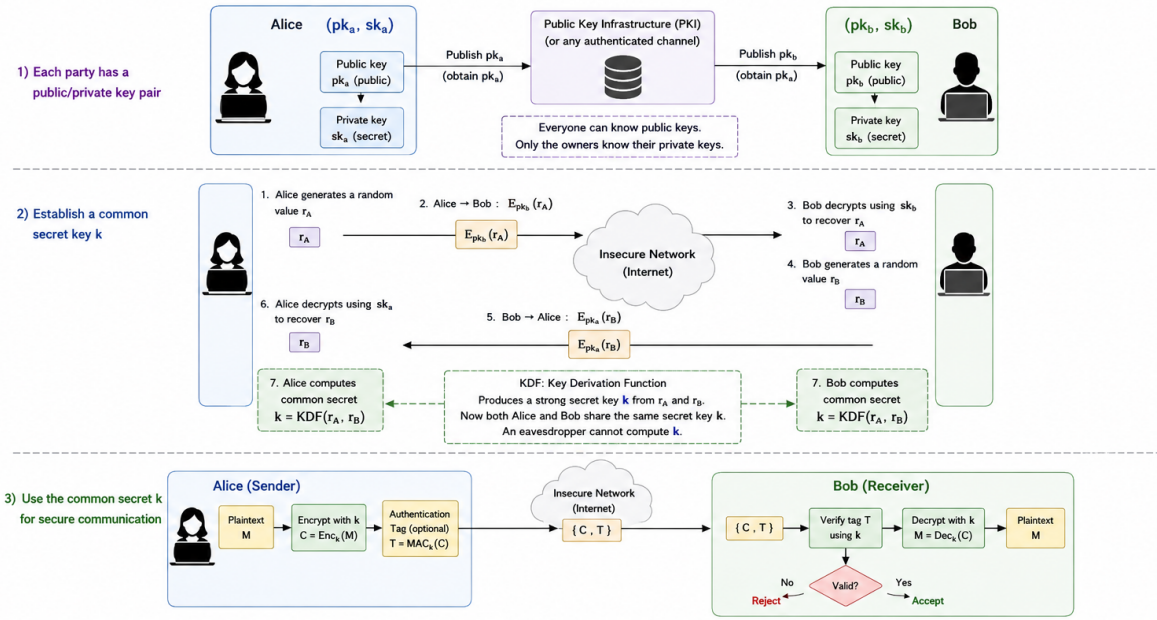
Widely used examples include

- RSA-PSS
- ECDSA
- Ed25519
- ML-DSA (Dilithium)
- Falcon

Post-quantum schemes rely mainly on lattice-based assumptions.

Key Observation

Unlike hash functions, digital signatures derive their security from well-defined mathematical hardness assumptions and are accompanied by formal security reductions.


1) Each party has a public/private key pair
Alice
(pk_a, sk_a)
Public key pk_a (public)
Private key sk_a (secret)
Publish pk_a
(obtain pk_a)
Public Key Infrastructure (PKI)
(or any authenticated channel)
Everyone can know public keys.
Only the owners know their private keys.
Publish pk_b
(obtain pk_a)
(pk_b, sk_b)
Bob
Public key pk_b (public)
Private key sk_b (secret)
2) Establish a common secret key k
1. Alice generates a random value r_A
r_A
2. Alice → Bob : $E_{pk_b}(r_A)$
$E_{pk_b}(r_A)$
Insecure Network (Internet)
3. Bob decrypts using sk_b to recover r_A
r_A
4. Bob generates a random value r_B
r_B
5. Bob → Alice : $E_{pk_a}(r_B)$
$E_{pk_a}(r_B)$
6. Alice decrypts using sk_a to recover r_B
r_B
7. Alice computes common secret $k = KDF(r_A, r_B)$
KDF: Key Derivation Function
Produces a strong secret key k from r_A and r_B.
Now both Alice and Bob share the same secret key k.
An eavesdropper cannot compute k.
7. Bob computes common secret $k = KDF(r_A, r_B)$
3) Use the common secret k for secure communication
Alice (Sender)
Plaintext M
Encrypt with k $C = Enc_k(M)$
Authentication Tag (optional) $T = MAC_k(C)$
Insecure Network (Internet)
{ C , T }
Bob (Receiver)
{ C , T }
Verify tag T using k
Decrypt with k $M = Dec_k(C)$
Plaintext M
Valid?
No
Reject
Yes
Accept

Choosing the Right Primitive

Security Goal	Primitive	Typical Usage
Confidentiality	Block / Stream Cipher	Encrypt messages
Integrity	Hash Function	File verification, fingerprints
Authentication	Digital Signature	Public-key authentication
Authentication	MAC (Keyed Hash)	Shared-key communication

Practical Observation

- Digital signatures are computationally expensive and are typically used only during public-key establishment, certificate verification, software signing, etc.
- Once a shared secret key has been established, authentication is performed using a keyed MAC or, more commonly, an authenticated encryption (AEAD) scheme.

For continuous communication, separately computing a hash (e.g., SHA-256), a MAC, or a digital signature for every message introduces additional computation. Can authentication instead be achieved during block encryption itself?

From Digital Signatures to Authenticated Encryption

Digital signatures provide authentication and integrity, but generating a signature for every encrypted message is computationally expensive.

Similarly, transmitting

$$(m,\ H(m))$$

does not provide authentication, since anyone can compute the hash.

Key Observation

Once a shared secret key has been established, integrity and authentication can be achieved much more efficiently during the encryption process itself, without generating a separate digital signature for every message.

What Can We Do at the **Block Level**?
to achieve
confidentiality, **integrity**, and **authenticity**?

Confidentiality: CBC and CTR are Equally Secure

Assume the underlying block cipher is a secure pseudorandom permutation.

Security Guarantee

Both CBC (with a fresh random IV) and CTR (with a unique nonce/counter) provide **IND-CPA security**.
They achieve essentially the same level of confidentiality.

CBC

- IND-CPA secure
- Sequential encryption
- Random IV

CTR

- IND-CPA secure
- Parallelizable
- Unique nonce/counter

Key Point

Neither mode is more secure than the other with respect to **confidentiality**. The difference lies in efficiency and usability.

Confidentiality Alone is Not Enough

Suppose Alice encrypts

$$m = \$1000.$$

Encryption produces

$$c = \mathrm{Enc}_k(m).$$

An attacker modifies the ciphertext

$$c \longrightarrow c'.$$

Bob decrypts

$$m' = \mathrm{Dec}_k(c').$$

Problem

CBC and CTR cannot determine whether the ciphertext has been modified.
They guarantee confidentiality, but **not integrity or authenticity**.

Why ECB Does Not Provide Integrity

Recall that ECB encrypts each block independently:

$$c_i = E_k(m_i).$$

Suppose an attacker intercepts the ciphertext

$$(c_1, c_2, c_3, c_4)$$

and replaces the second block with another valid ciphertext block c_2' obtained from a different message.

Bob decrypts (c_1, c_2', c_3, c_4) to obtain (m_1, m_2', m_3, m_4), where m_2' is a completely different block.

Since ciphertext blocks can be substituted, reordered, inserted, or deleted without knowledge of the secret key, ECB encryption alone provides

$$\boxed{\text{Confidentiality} \neq \text{Integrity.}}$$

The receiver has no mechanism to detect such modifications.

Why CBC Does Not Provide Integrity I

Suppose the attacker modifies the ciphertext block:

$$c_i' = c_i \oplus e,$$

where e is an arbitrary error (bit-flip) vector.

The decryption of the affected plaintext blocks is:

$$\begin{aligned} m_i' &= D_k(c_i') \oplus c_{i-1} \\ &= D_k(c_i \oplus e) \oplus c_{i-1}. \end{aligned}$$

Since $D_k(\cdot)$ is nonlinear,

$$m_i' \neq m_i,$$

and the plaintext block m_i is completely corrupted.

For the next plaintext block,

Why CBC Does Not Provide Integrity II

$$\begin{aligned} m'_{i+1} &= D_k(c_{i+1}) \oplus c'_i \\ &= D_k(c_{i+1}) \oplus (c_i \oplus e) \\ &= \big(D_k(c_{i+1}) \oplus c_i\big) \oplus e \\ &= m_{i+1} \oplus e. \end{aligned}$$

Thus, the attacker can introduce controlled bit flips into m_{i+1}.

For all subsequent plaintext blocks,

$$m'_j = m_j, \qquad \forall\, j \geq i+2.$$

Hence,

$$\boxed{\begin{array}{ll} m'_i = D_k(c_i \oplus e) \oplus c_{i-1} & \text{(corrupted)}, \\ m'_{i+1} = m_{i+1} \oplus e & \text{(controlled bit flips)}, \\ m'_j = m_j, \quad \forall\, j \geq i+2 & \text{(unchanged)}. \end{array}}$$

Authenticating Block Ciphers
Authenticated Encryption (AE)

Why Do We Need More Than Encryption?

Block cipher modes such as CBC and CTR provide

Confidentiality

but not

Integrity.

An attacker may modify the ciphertext during transmission, and the receiver cannot detect the modification.

Goal

Provide the ability to detect whether a message has been altered during transmission.

A natural solution is to compute a cryptographic digest

$$t = H(m),$$

and transmit (m, t) together.

Can We Authenticate Long Messages Efficiently?

Suppose

$$m = (m_1, m_2, \ldots, m_\ell),$$

contains many blocks.

Although a cryptographic hash produces only a fixed-size digest,

$$H : \{0,1\}^* \to \{0,1\}^n,$$

computing a separate hash or MAC after encryption requires an additional pass over the entire message.

For high-speed communication (TLS, VPNs, video streaming, cloud storage), we would like authentication to be computed while encryption is taking place.

Question

Can integrity be computed block by block during encryption itself?

Universal Hashing to the Rescue

Instead of applying a cryptographic hash to the complete ciphertext, modern AEAD schemes authenticate each ciphertext block using a **keyed universal hash function**.

For ciphertext blocks

$$c_1, c_2, \ldots, c_\ell,$$

compute

$$t_i = h_k(c_i),$$

and combine them iteratively into a single authentication tag

$$t = t_1 \oplus t_2 \oplus \cdots \oplus t_\ell.$$

Advantages

- Authentication is computed together with encryption.
- Only one final authentication tag is transmitted.
- Universal hashing is significantly faster than general-purpose cryptographic hashing.

Relationship Between the Block Cipher Key and the Hash Key

AES-GCM uses a single secret key K. From this key, the universal hash key is derived as

$$\boxed{H = E_K(0^{128}),}$$

where 0^{128} denotes the all-zero 128-bit block.

Thus,

Key Observation

- No additional secret key is exchanged.
- The encryption key and the universal hash key are cryptographically linked.
- AES uses K for encryption, while GHASH uses the derived key K' for authentication.

Other schemes employ similar approaches, in which no separate key exchange is required to obtain the key for universal hashing.

AES GCM Encryption

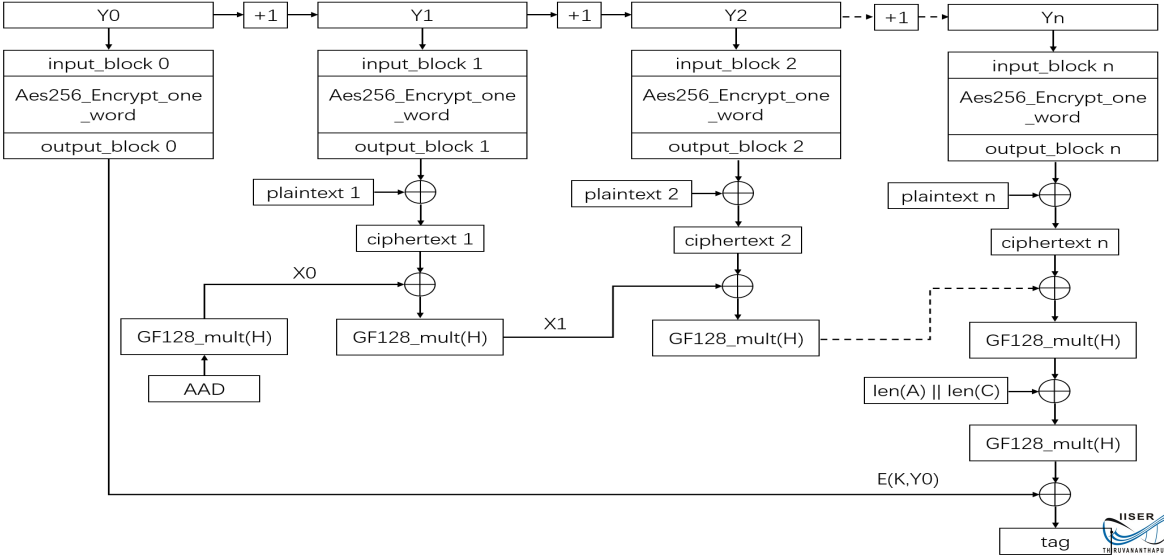


Figure source: https://xilinx.github.io/Vitis_Libraries/security/2019.2/guide_L1/internals/gcm.html

AES GCM Decryption

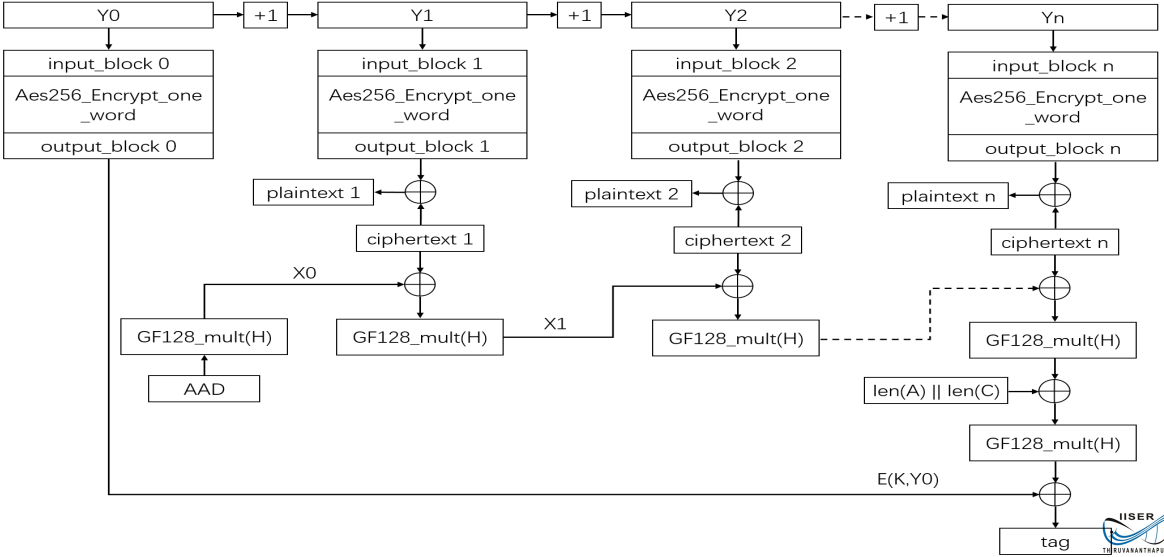


Figure source: https://xilinx.github.io/Vitis_Libraries/security/2019.2/guide_L1/internals/gcm.html

Authenticating Associative Data

Authenticated Encryption with Associative Data(AEAD)

*For references only

Why Do We Need Associated Data?

Authenticated Encryption (AE) protects the plaintext by providing

$$\boxed{\text{Confidentiality} + \text{Integrity} + \text{Authentication}.}$$

However, many communication protocols contain additional information that

- should remain visible,
- should *not* be encrypted,
- but must still be protected against modification.

Examples include

$$\boxed{\text{Sequence Number, Packet Header, Sender ID, Protocol Version}.}$$

Question

How can these fields be authenticated without encrypting them?

Authenticated Encryption with Associated Data (AEAD)

AEAD extends authenticated encryption by introducing **Associated Data (AAD)**.

$$(K, N, A, M) \longrightarrow (C, T)$$

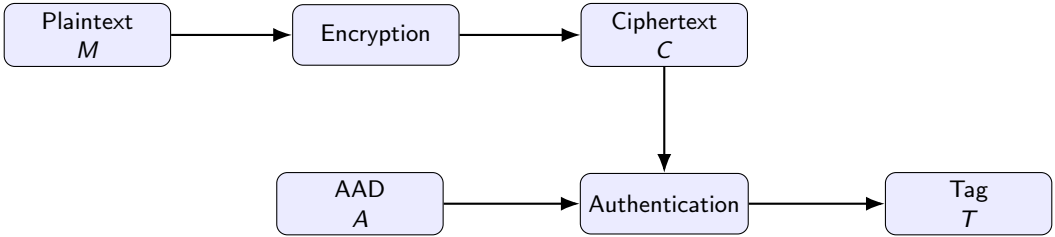
where

- K : secret key,
- N : nonce,
- A : associated data (AAD),
- M : plaintext,
- C : ciphertext,
- T : authentication tag.

Security Guarantee

- Plaintext M is encrypted.
- Ciphertext C is authenticated.
- Associated data A is authenticated but not encrypted.

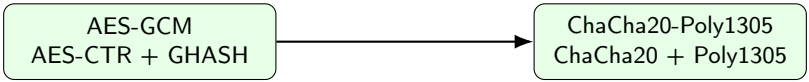
How AEAD Uses Associated Data



Key Idea

AAD is never encrypted. It is processed together with the ciphertext during authentication. Any modification to either the AAD or the ciphertext causes tag verification to fail.

Modern AEAD Schemes



Both schemes

- encrypt using a stream of pseudorandom bits,
- authenticate using a keyed universal hash,
- support associated data (AAD),
- are widely deployed in TLS 1.3, QUIC, SSH, Signal, and many other modern protocols.

Implementation of AES-GCM

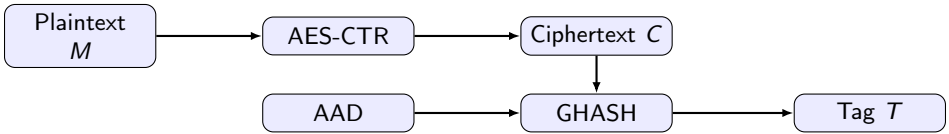
AES-GCM combines

AES-CTR

for encryption and

GHASH

for authentication.



Key Components

- AES-CTR encrypts the plaintext.
- GHASH authenticates both the ciphertext and AAD.
- Final output: (C, T).

Implementation of ChaCha20-Poly1305

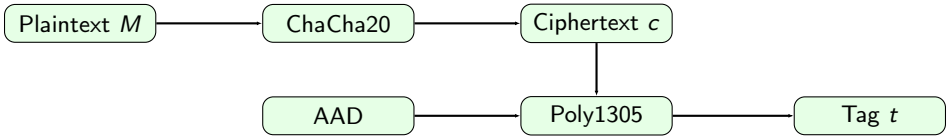
ChaCha20-Poly1305 combines

ChaCha20

for encryption and

Poly1305

for authentication.



Key Components

- ChaCha20 generates the keystream for encryption.
- Poly1305 authenticates both the ciphertext and AAD.
- Final output: (C, T).

References

Textbook:

1. Douglas R. Stinson and Maura Paterson. *Cryptography: Theory and Practice*. 4th Edition. Chapman & Hall/CRC Press, 2019. ISBN: 9780367148591.

Acknowledgment:

- The figures in this work were generated with the assistance of ChatGPT (OpenAI).

Dr. Laltu Sardar, Assistant Professor, IISER Thiruvananthapuram
laltu.sardar@iisertvm.ac.in, laltu.sardar.crypto@gmail.com
Course webpage: https://laltu-sardar.github.io/courses/26_crypto.html.