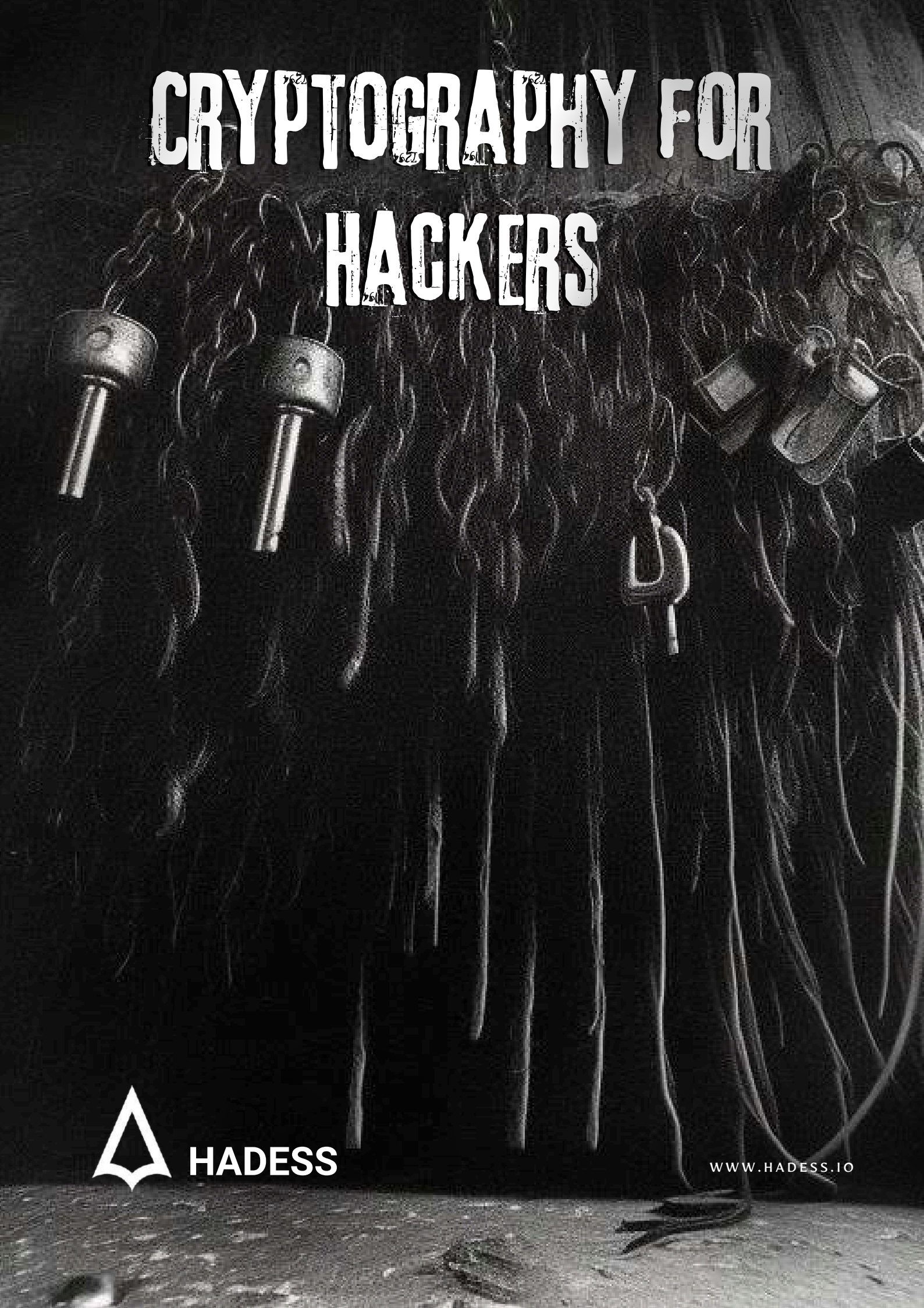
# CRYPTOGRAPHY FOR HACKERS



HADESS

WWW.HADESS.IO

IF YOU REVEAL YOUR SECRETS TO THE WIND, YOU SHOULD NOT BLAME THE WIND FOR REVEALING THEM TO THE TREES.

KAHLIL GIBRAN

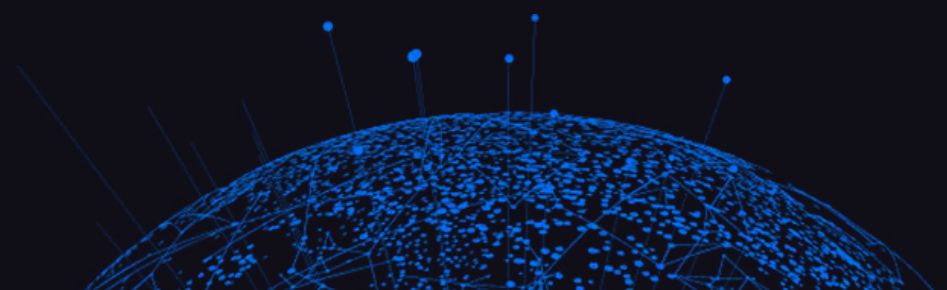
GIAMBATTISTA DELLA PORTA

# TABLE OF CONTENT

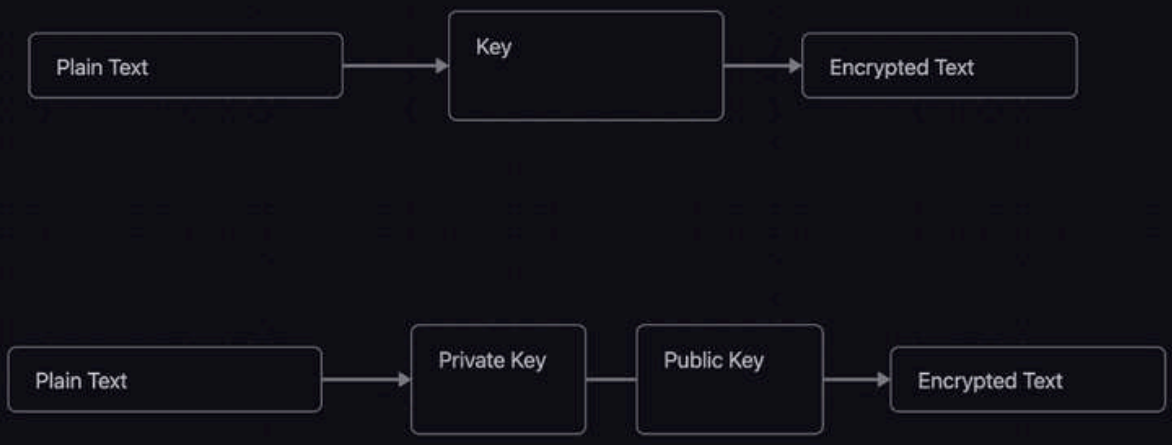
# Cryptography for Hackers

Cryptography, the art of securing information, is foundational in the world of cybersecurity and hacking. It involves various methods and techniques to convert readable data into an unreadable format, known as ciphertext, to protect sensitive information from unauthorized access. One of the most widely used cryptographic methods is **symmetric encryption**, where a single key is used for both encryption and decryption. Examples include AES (Advanced Encryption Standard) and DES (Data Encryption Standard). Symmetric encryption is fast and efficient, making it ideal for encrypting large volumes of data. However, the main challenge lies in securely sharing the key between the communicating parties.

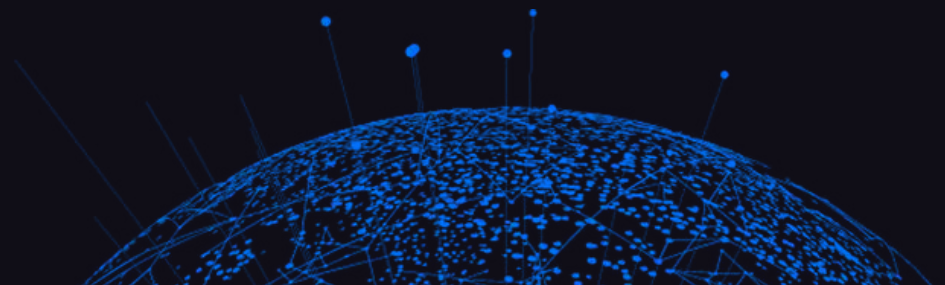
In contrast, **asymmetric encryption** uses a pair of keys: a public key for encryption and a private key for decryption. This method, utilized in protocols like RSA (Rivest-Shamir-Adleman) and ECC (Elliptic Curve Cryptography), enhances security by eliminating the need for key exchange. Asymmetric encryption is typically used for securing data transmissions, digital signatures, and ensuring the authenticity of messages. However, it is slower than symmetric encryption and is often combined with symmetric encryption in practical applications to optimize both security and performance. Cryptographic hashing, another vital method, generates a fixed-size hash value from data, providing integrity checks and playing a crucial role in password storage, digital signatures, and blockchain technology.



# Encryption and Decryption in Cryptography

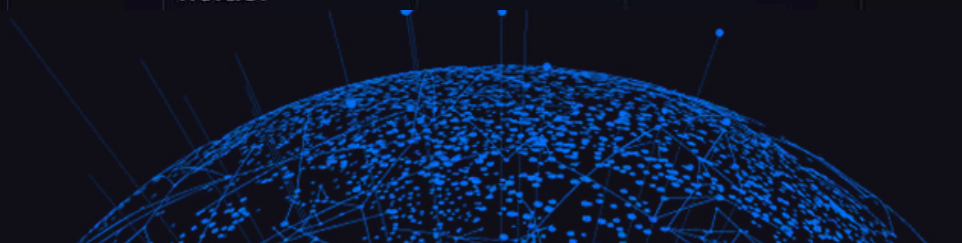



Encryption and decryption are fundamental processes in cryptography, ensuring the confidentiality of data by transforming readable information (plaintext) into an unreadable format (ciphertext) and vice versa. **Symmetric encryption** uses the same key for both encryption and decryption, offering speed and efficiency, while **asymmetric encryption** employs a pair of keys (public and private) to enhance security, albeit at a performance cost. These techniques are vital for secure communication, data protection, and authentication in various digital applications.

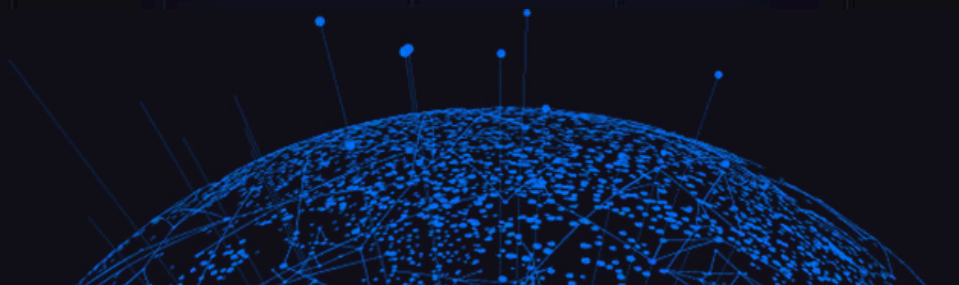


## Popular Cryptographic Methods

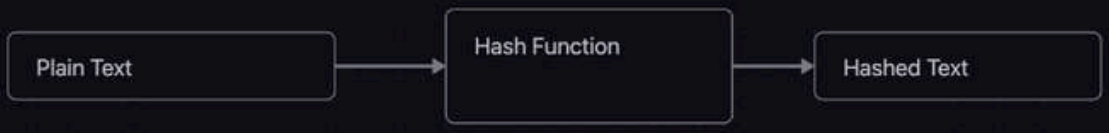
| ID | Method | Description | Type | Usage | Known Attacks |
|---|---|---|---|---|---|
| 1 | AES | Advanced Encryption Standard; widely used symmetric key encryption standard. | Symmetric | Data encryption, VPNs, wireless security | Side-channel attacks |
| 2 | RSA | Rivest-Shamir-Adleman; popular asymmetric encryption algorithm based on integer factorization. | Asymmetric | Secure data transmission, digital signatures | Timing attacks, padding oracle attacks |
| 3 | SHA-256 | Secure Hash Algorithm 256-bit; part of the SHA-2 family, producing a 256-bit hash. | N/A (Hash Function) | Data integrity, blockchain technology | Collision attacks (theoretical) |
| 4 | ECC | Elliptic Curve Cryptography; asymmetric encryption leveraging elliptic curves over finite fields. | Asymmetric | Mobile devices, SSL/TLS certificates | Side-channel attacks |



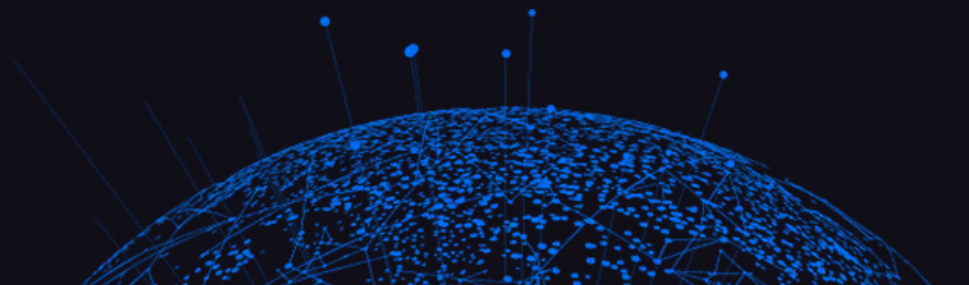
| | | | | | |
|---|---|---|---|---|---|
| 5 | 3DES | Triple Data Encryption Standard; applies DES cipher algorithm three times to each data block. | Symmetric | Legacy systems, ATM encryption | Meet-in-the-middle attacks |
| 6 | Blowfish | Symmetric-key block cipher designed for fast performance. | Symmetric | Password hashing, file encryption | Key schedule attacks |
| 7 | MD5 | Message-Digest Algorithm 5; produces a 128-bit hash value. | N/A (Hash Function) | Checksums, data integrity (legacy) | Collision attacks |
| 8 | Diffie-Hellman | Key exchange method allowing secure sharing of keys over a public channel. | Asymmetric | VPNs, secure key exchange | Man-in-the-middle attacks |
| 9 | DES | Data Encryption Standard; an older symmetric-key method now considered insecure. | Symmetric | Legacy systems | Brute-force attacks |



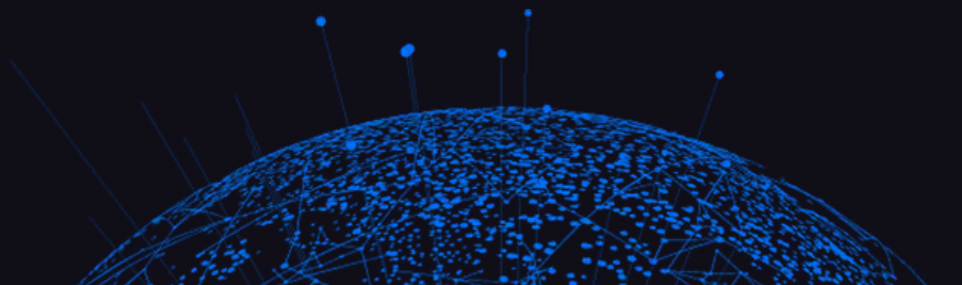
# Hashing in Cryptography



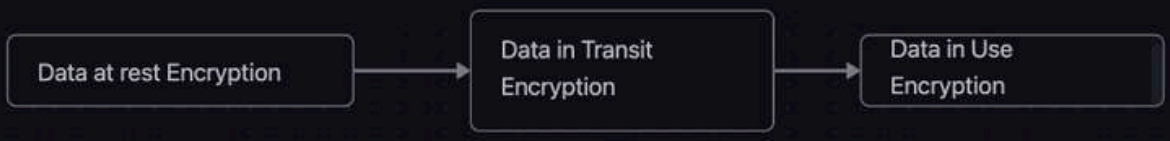

Hashing is a cryptographic technique that converts data into a fixed-size string of characters, which is typically a digest that uniquely represents the input. Hash functions are designed to be fast, deterministic, and irreversible, ensuring that even a small change in input produces a significantly different hash. This property makes hashing essential for verifying data integrity, securely storing passwords, and ensuring the authenticity of digital signatures. Commonly used hash functions include MD5, SHA-1, and the SHA-2 family, with SHA-256 being particularly popular for its balance of security and efficiency.



| ID | Hash Method | Description | Usage | Attacks |
|---|---|---|---|---|
| 1 | SHA-256 | Part of the SHA-2 family, producing a 256-bit hash. | Blockchain, digital certificates, password hashing. | Resistant to known attacks; currently secure. |
| 2 | SHA-3 | Latest SHA family, designed as a backup for SHA-2. | Cryptographic applications, security protocols. | Resistant to known attacks; very secure. |
| 3 | SHA-1 | Older hash method producing a 160-bit hash. | Legacy systems, old digital certificates. | Vulnerable to collision attacks; not recommended for new use. |
| 4 | MD5 | Early hash function with a 128-bit hash. | Checksums, legacy systems. | Vulnerable to collision attacks; broken for cryptographic use. |
| 5 | RIPEMD-160 | A 160-bit hash function from the RIPEMD family. | Cryptographic applications, digital signatures. | More secure than MD5 and SHA-1, but less common. |
| 6 | BLAKE2 | A fast and secure hash function, designed as an alternative to MD5 and SHA. | General cryptographic applications, file verification. | Resistant to known attacks; highly secure. |
| 7 | WHIRLPOOL | Produces a 512-bit hash, based on a modified AES. | Archiving, file verification. | No known practical attacks; very secure. |

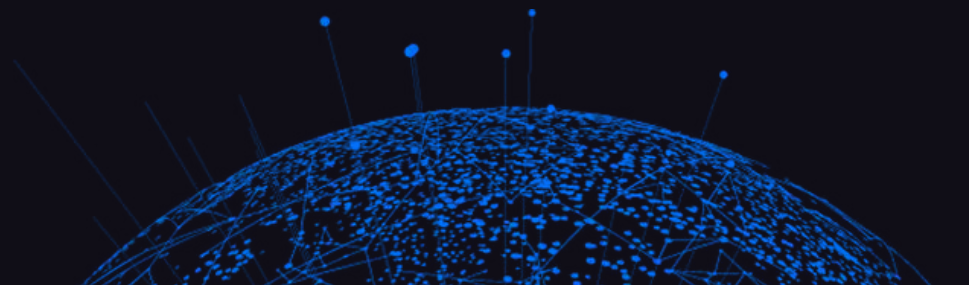


# Difference Between Data-at-Rest and Data-in-Transit Cryptography




**Data-at-Rest** cryptography refers to the encryption methods used to protect data stored on devices or in databases. This includes data on hard drives, SSDs, and backup tapes. The goal is to prevent unauthorized access to stored data, even if the storage media are physically stolen or accessed. Common methods include **full-disk encryption** (e.g., BitLocker, FileVault), **database encryption**, and **file-level encryption**. The focus is on securing the data from breaches or unauthorized access while it is not actively being transmitted.

**Data-in-Transit** cryptography, on the other hand, protects data as it moves across networks, from one location to another, such as when data is being sent over the internet, between devices, or across internal networks. This includes email, file transfers, and web traffic. Techniques like **TLS/SSL** (used in HTTPS), **VPNs**, and **end-to-end encryption** (e.g., in messaging apps) are common. The primary objective is to protect the data from interception, eavesdropping, or man-in-the-middle attacks while it's being transmitted.

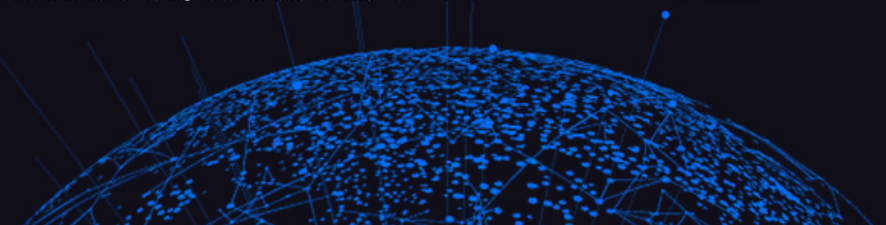


# Finding the Cryptography Algorithm from Encrypted Text

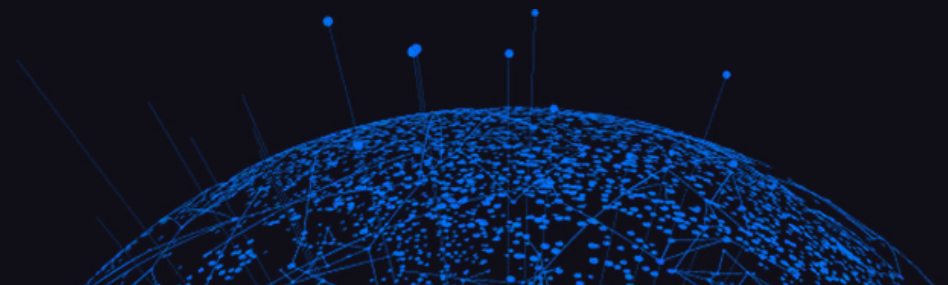
Identifying the cryptographic algorithm used to encrypt a piece of ciphertext requires a methodical approach. This process involves analyzing the structure and characteristics of the ciphertext, performing pattern recognition, and applying various cryptographic techniques to deduce the algorithm. Below is a step-by-step guide, along with an ASCII mindmap and character review conditions.

## Step-by-Step Process:

1. **Initial Ciphertext Analysis:**
    - **Character Set:** Determine the type of characters used in the ciphertext (e.g., alphanumeric, symbols, binary).
    - **Length:** Check if the length is a multiple of common block sizes (e.g., 64, 128, 256 bits).
    - **Base Encoding:** Determine if the ciphertext is encoded in a format like Base64, Hexadecimal, etc.
2. **Pattern and Structure Recognition:**
    - **Repetition Patterns:** Identify repeating patterns or blocks which might indicate block ciphers or certain modes like ECB.
    - **Entropy Assessment:** Measure the randomness of the ciphertext. High entropy suggests strong encryption.
3. **Algorithm Matching:**
    - **Hash Length Comparison:** Compare the length and structure with known hash functions (e.g., MD5 - 128 bits, SHA-256 - 256 bits).
    - **Block Cipher Analysis:** Consider the block size (e.g., 64 bits for DES, 128 bits for AES) and structure.

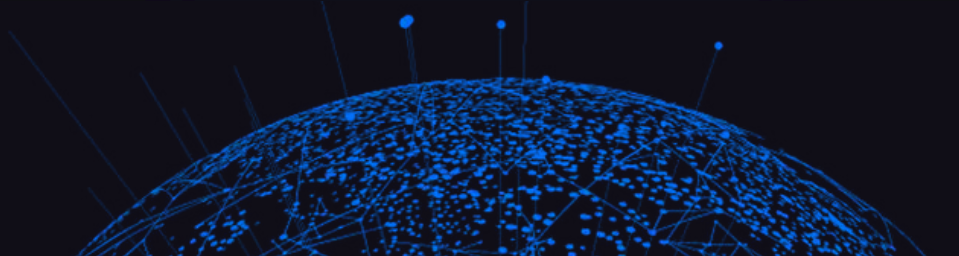


4. **Frequency Analysis:**
   - **Character Frequency Distribution:** Perform frequency analysis to detect non-random distributions, hinting at substitution ciphers or weak encryption.
   - **Alphabetic Characters:** Check if the ciphertext maintains a pattern similar to language-based text, indicating possible simpler ciphers.
5. **Test Decryption:**
   - **Known-Plaintext Attack:** Use any known plaintext to attempt partial decryption, testing against different algorithms.
   - **Brute Force/Dictionary Attacks:** Apply brute force or dictionary attacks if possible, especially for weaker or simpler algorithms.
6. **Utilize Cryptographic Tools:**
   - **Cryptanalysis Tools:** Employ cryptographic tools or libraries (e.g., Hashcat, John the Ripper) to automate algorithm identification and attack attempts.



## Step-by-Step Character Review and Conditions:

1. **Character Set Analysis:**
   - **Condition:** If the ciphertext contains only hexadecimal characters (0-9, A-F), it might be encoded or could indicate a hash function.
   - **Condition:** If the ciphertext is Base64 (alphanumeric + `/`, `+`, `=`), it suggests encoding often used after encryption.
2. **Length and Block Size:**
   - **Condition:** If the ciphertext length is a multiple of 64, 128, or 256 bits, it could be indicative of a block cipher like DES, 3DES, or AES.
   - **Condition:** If the ciphertext length is fixed and small (e.g., 128 or 256 bits), it might be a hash output (e.g., MD5, SHA-256).
3. **Repetition Patterns:**
   - **Condition:** Repeating blocks every 8 or 16 characters might indicate ECB mode of a block cipher (e.g., AES in ECB mode).
   - **Condition:** Lack of repetition with consistent length might suggest CBC mode or another encryption mode that avoids patterns.
4. **Entropy and Randomness:**
   - **Condition:** If the ciphertext appears highly random with no discernible patterns, it likely involves a strong encryption algorithm.
   - **Condition:** Low entropy might suggest simpler or broken encryption methods, possibly vulnerable to certain attacks.
5. **Character Frequency:**
   - **Condition:** If frequency analysis shows a distribution similar to English text, it may suggest a substitution cipher or weak encryption.
   - **Condition:** Random or flat frequency distribution suggests a modern, strong encryption method.



## Scenario: Received Encrypted Text

You receive the following encrypted text:

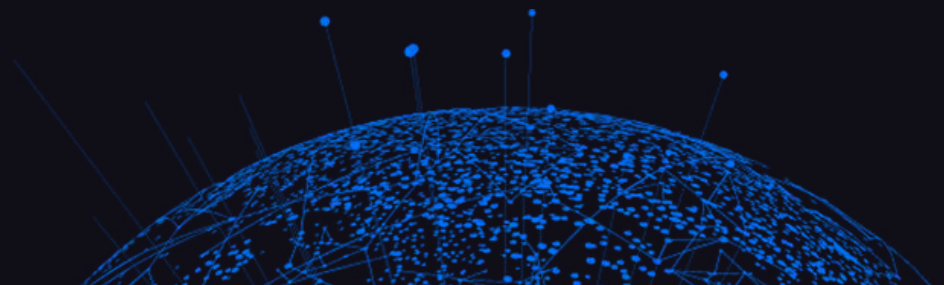
```
QmFzZTY0RW5jb2RlZFRleHQ=
```

You need to identify which cryptographic algorithm or method was used to create this ciphertext.

## Step-by-Step Example

### 1. Initial Ciphertext Analysis

- **Character Set:**
  - Example: The text `QmFzZTY0RW5jb2RlZFRleHQ=` includes uppercase and lowercase letters, numbers, and special characters like `=`.
  - **Condition:** These characters suggest Base64 encoding, a common post-encryption encoding method.
- **Length:**
  - Example: The length of the ciphertext is 24 characters.
  - **Condition:** The length is not directly indicative of a specific block size, but since Base64 encodes in chunks, this length can help deduce the original plaintext size.



- **Base Encoding:**
  - Example: Recognizing the `=` padding and character set, you identify this as Base64.
  - **Condition:** Base64 is used to encode binary data in ASCII format, commonly seen after encryption.

2. **Pattern and Structure Recognition**

- **Repetition Patterns:**
  - Example: No repetition of blocks is observed in the Base64-encoded string.
  - **Condition:** If repetition existed, it might indicate a block cipher in ECB mode. The absence suggests either a non-repeating pattern or that Base64 hides repetition in the underlying binary data.
- **Entropy Assessment:**
  - Example: The encoded text appears random at a glance, but further entropy analysis is needed.
  - **Condition:** High entropy in the Base64-encoded string hints at strong encryption in the underlying data.

3. **Algorithm Matching**

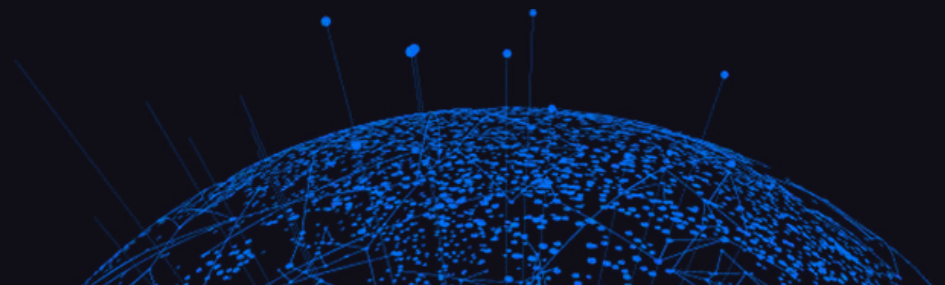
- **Hash Length Comparison:**
  - Example: The Base64 string length of 24 characters translates to 18 bytes of binary data, which does not directly match standard hash lengths (e.g., MD5, SHA-256).
  - **Condition:** This indicates it is likely not a hash function output but possibly a short message encrypted and encoded.
- **Block Cipher Analysis:**
  - Example: Given no clear multiple of a common block size (e.g., 16 bytes for AES), it's challenging to match a block cipher directly.
  - **Condition:** The non-matching block size suggests this might be a smaller data piece or a short plaintext encrypted with a block cipher and then encoded.

4. Frequency Analysis

- Character Frequency Distribution:
  - Example: Since the text is Base64 encoded, normal frequency analysis on the encoded string won't be useful.
  - **Condition:** If the underlying data was a simple cipher, a noticeable pattern would emerge upon decoding.
- Alphabetic Characters:
  - Example: Decoding `QmFzZTY0RW5jb2RlZFRleHQ=` gives `Base64EncodedText`.
  - **Condition:** This reveals that the text was not encrypted but simply encoded. If it were encrypted, decoding would yield non-readable text.

5. Test Decryption

- Known-Plaintext Attack:
  - Example: Suppose you expect the plaintext could be "Base64EncodedText". Decoding reveals this directly.
  - **Condition:** If a plaintext guess succeeds, it confirms the method (in this case, Base64 encoding).
- Brute Force/Dictionary Attacks:
  - Example: If decoding fails, you might try known encryption algorithms (AES, DES) with typical passwords.
  - **Condition:** A successful decryption attempt would confirm the algorithm used.

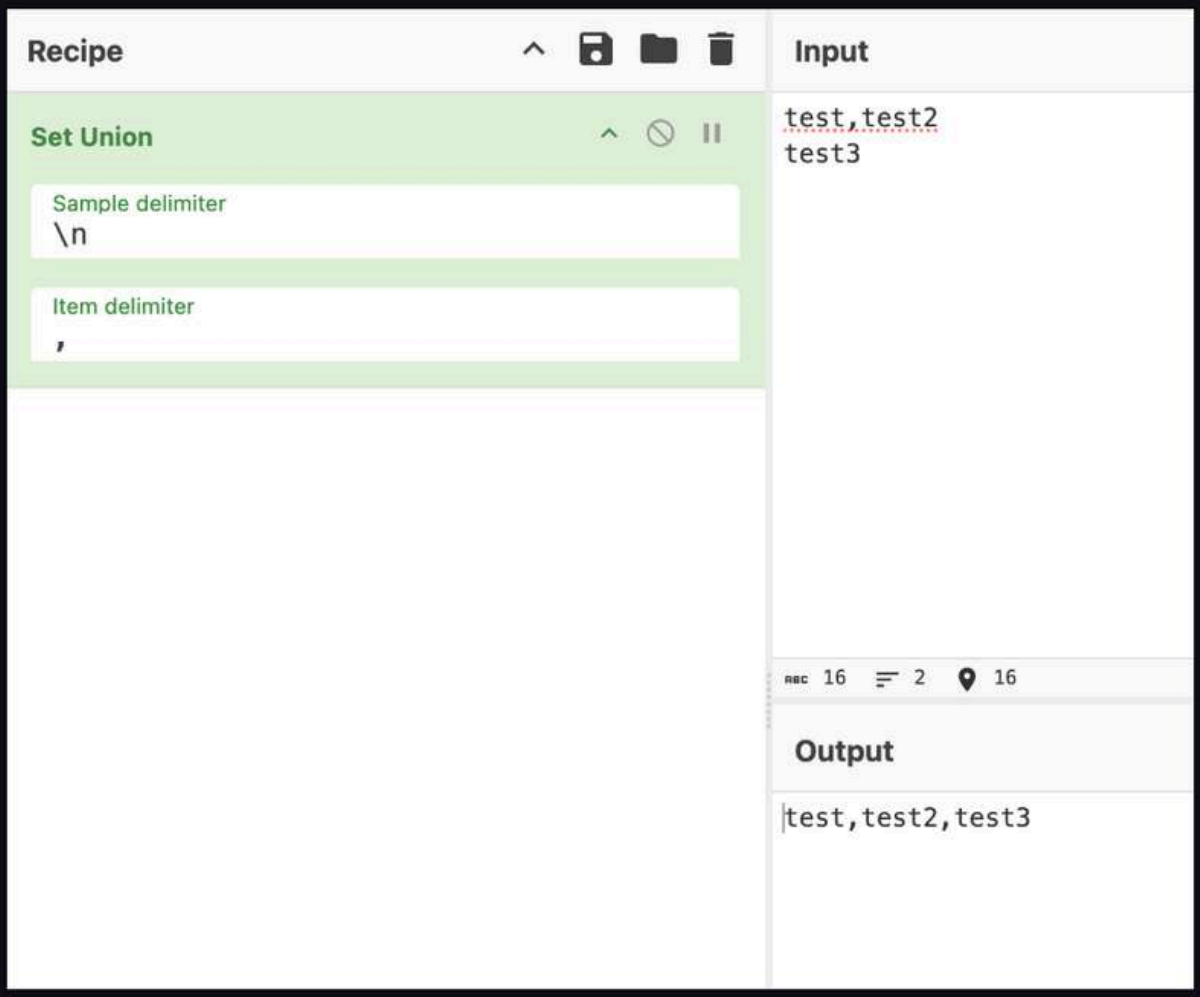


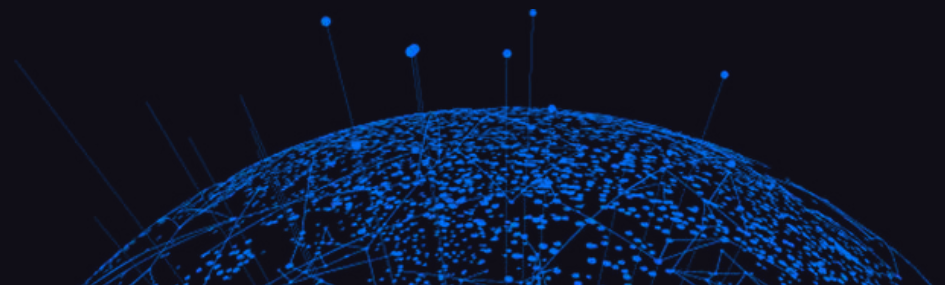
# Arithmetic and Logic Operations

Arithmetic and logic operations are fundamental to computer science, mathematics, and cryptography. Below is an explanation of each operation, along with an example test case for clarity.

## Set Operations

### 1. Set Union



- **Description:** Combines all elements from two sets, removing duplicates.
  - **Example:**
    - **Sets:** A = {1, 2, 3}, B = {3, 4, 5}
    - **Union:** A ∪ B = {1, 2, 3, 4, 5}

2. **Set Intersection**

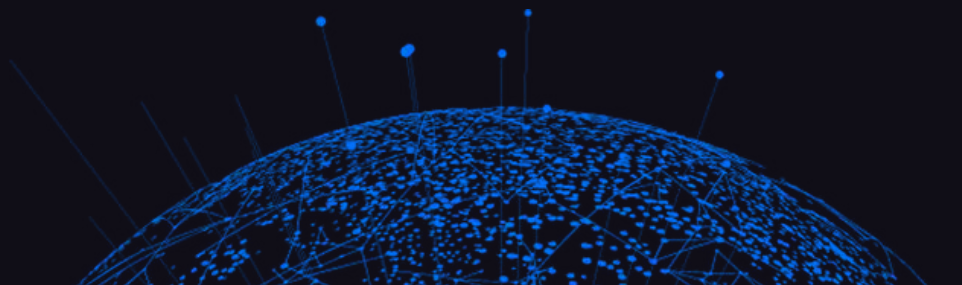
   - **Description:** Yields elements common to both sets.
   - **Example:**
     - **Sets:** A = {1, 2, 3}, B = {3, 4, 5}
     - **Intersection:** A ∩ B = {3}

3. **Set Difference**

   - **Description:** Elements in one set but not the other.
   - **Example:**
     - **Sets:** A = {1, 2, 3}, B = {3, 4, 5}
     - **Difference:** A - B = {1, 2}

4. **Symmetric Difference**

   - **Description:** Elements in either set, but not in both.
   - **Example:**
     - **Sets:** A = {1, 2, 3}, B = {3, 4, 5}
     - **Symmetric Difference:** A Δ B = {1, 2, 4, 5}



## Logic Operations

### 7. XOR (Exclusive OR)

- **Description:** Outputs true when inputs differ.
- **Example:**
  - **Inputs:** A = 1010, B = 1100
  - **XOR Result:** A ⊕ B = 0110

### 8. XOR Brute Force

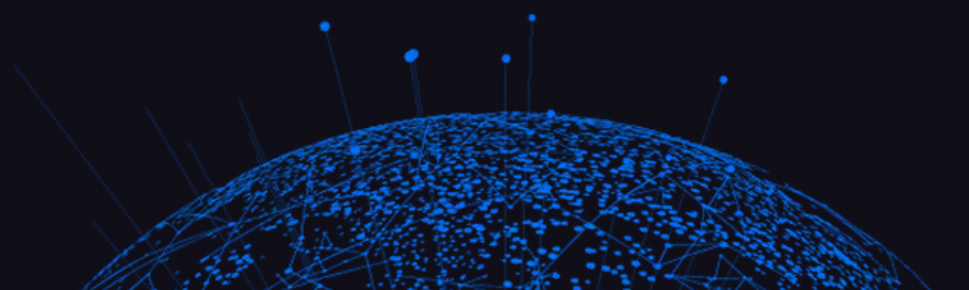
- **Description:** Testing possible keys in a brute-force attack by XORing ciphertext with keys.
- **Example:**
  - **Ciphertext:** C = 0110
  - **Brute Force Keys:** Key1 = 0011, Key2 = 1001, etc.
  - **Result (Key1):** C ⊕ Key1 = 0101

### 9. OR

- **Description:** Outputs true if at least one input is true.
- **Example:**
  - **Inputs:** A = 1010, B = 1100
  - **OR Result:** A | B = 1110

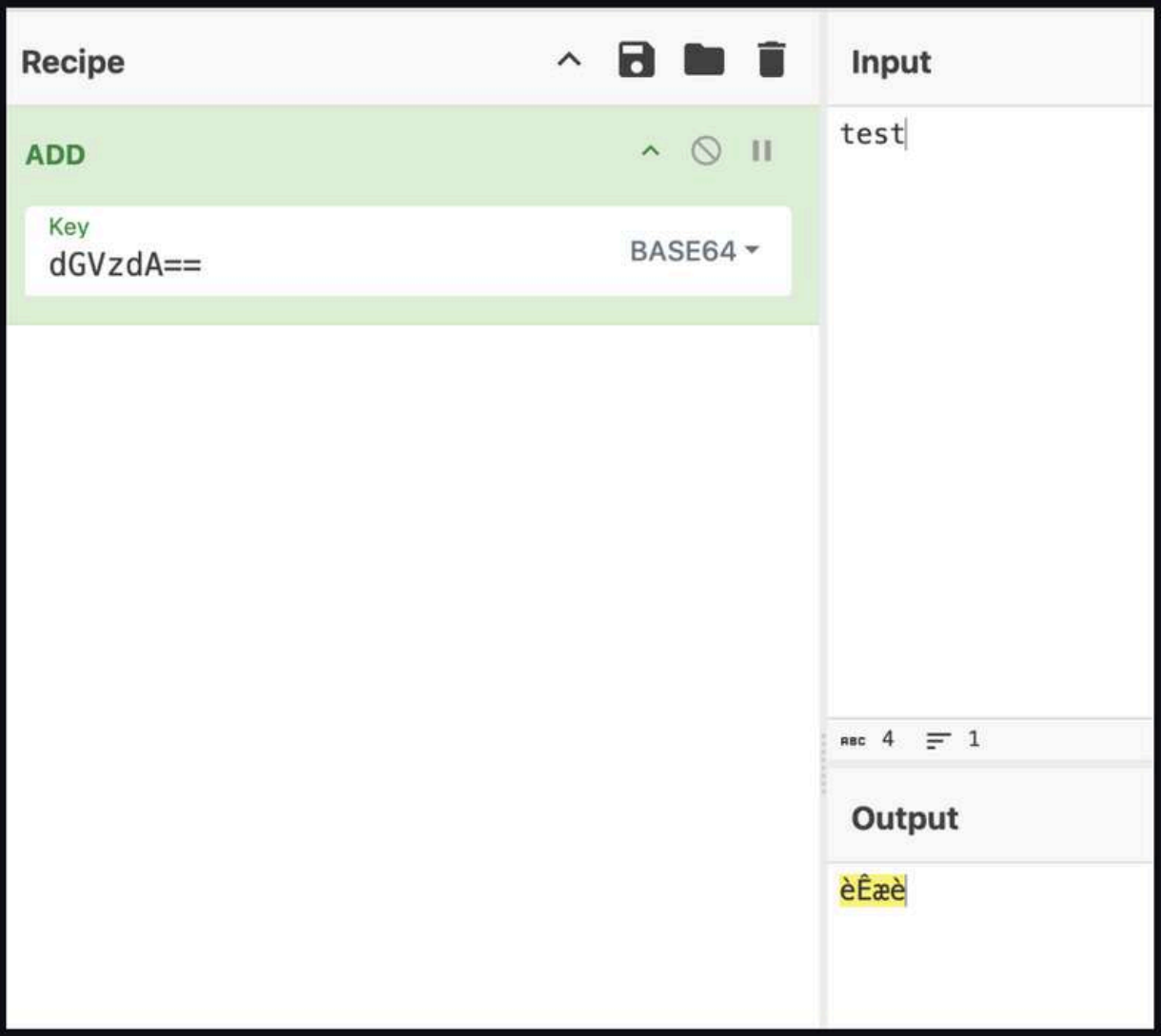
### 10. NOT

- **Description:** Inverts the input.
- **Example:**
  - **Input:** A = 1010
  - **NOT Result:** ¬A = 0101

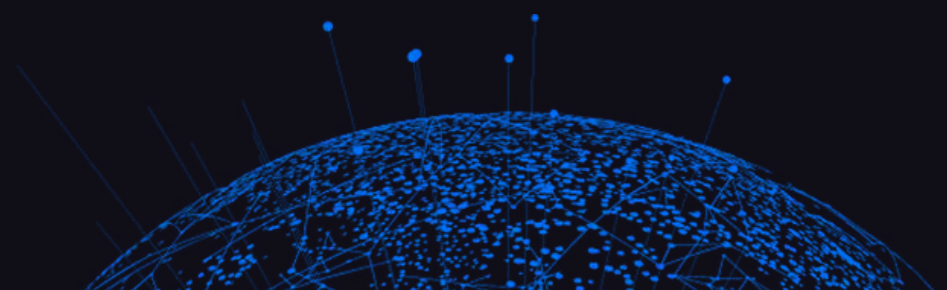


## Arithmetic Operations

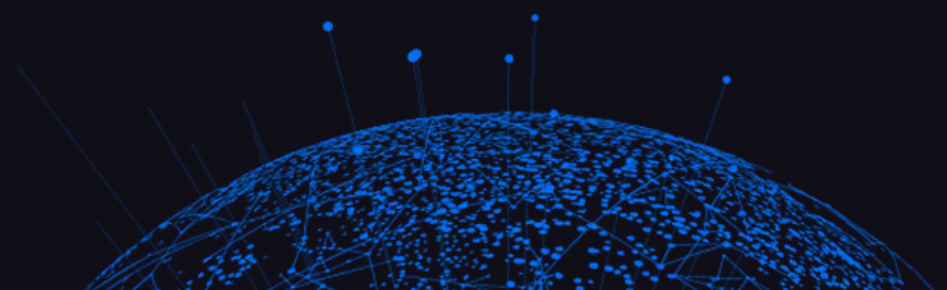
### 12. ADD




- **Description:** Adds two numbers.
  - **Example:**
    - **Inputs:** A = 7, B = 3
    - **Addition Result:** A + B = 10

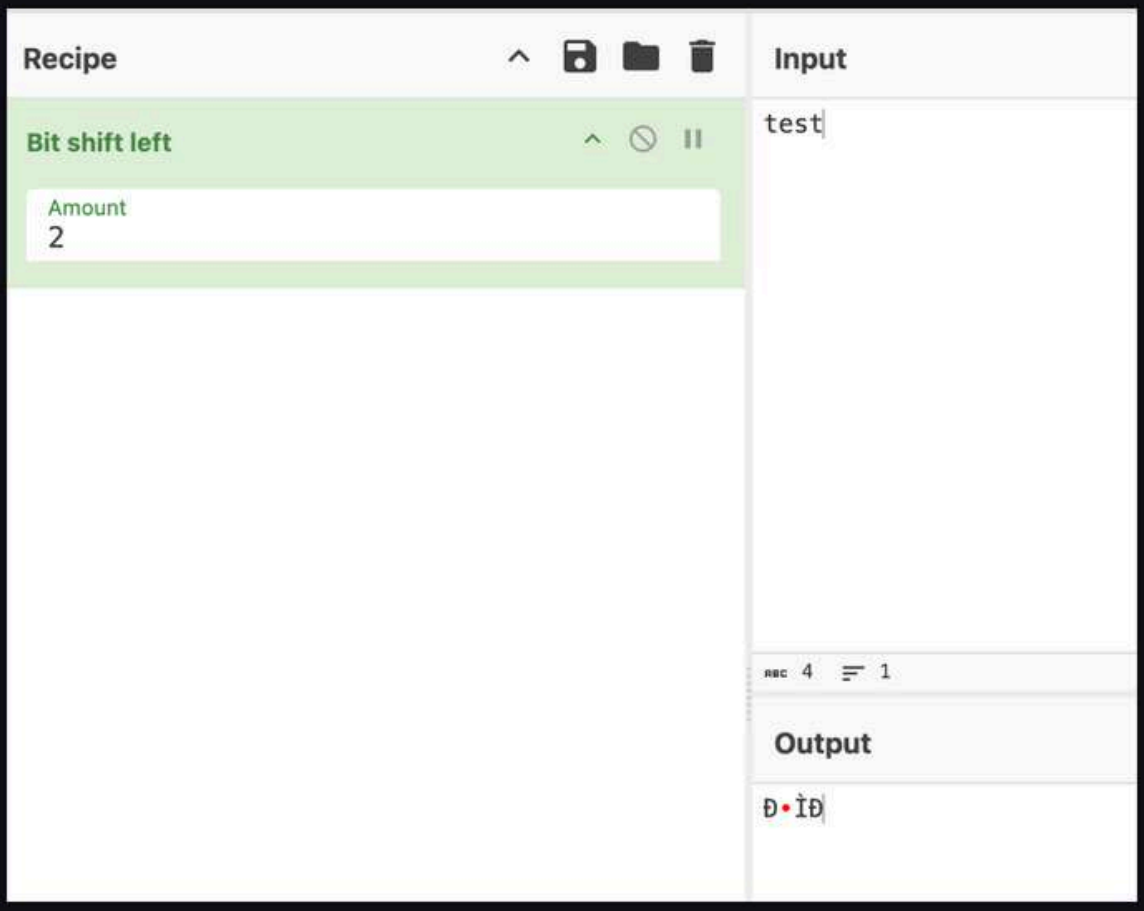


13. **SUB (Subtract)**
    - **Description:** Subtracts one number from another.
    - **Example:**
        - **Inputs:** A = 7, B = 3
        - **Subtraction Result:** A - B = 4
14. **Multiply**
    - **Description:** Multiplies two numbers.
    - **Example:**
        - **Inputs:** A = 7, B = 3
        - **Multiplication Result:** A * B = 21
15. **Divide**
    - **Description:** Divides one number by another.
    - **Example:**
        - **Inputs:** A = 9, B = 3
        - **Division Result:** A / B = 3
16. **Sum**
    - **Description:** Sums a list of numbers.
    - **Example:**
        - **List:** {1, 2, 3, 4}
        - **Sum Result:** 1 + 2 + 3 + 4 = 10
17. **Subtract (List)**
    - **Description:** Subtracts all elements in a list from a starting value.
    - **Example:**
        - **List:** {10, 2, 3, 1}
        - **Subtraction Result:** 10 - 2 - 3 - 1 = 4



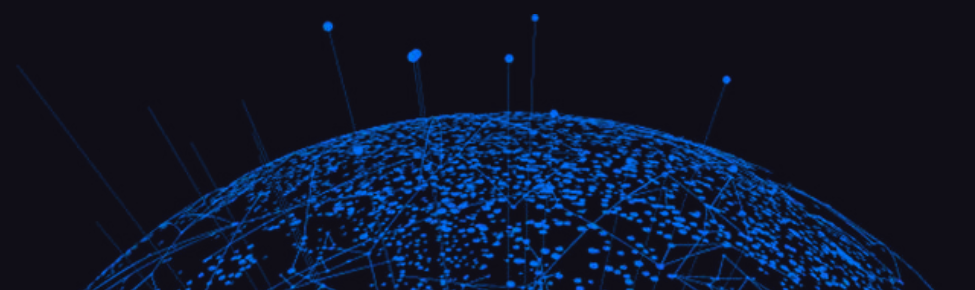
## Bitwise Operations

### 21. Bit Shift Left



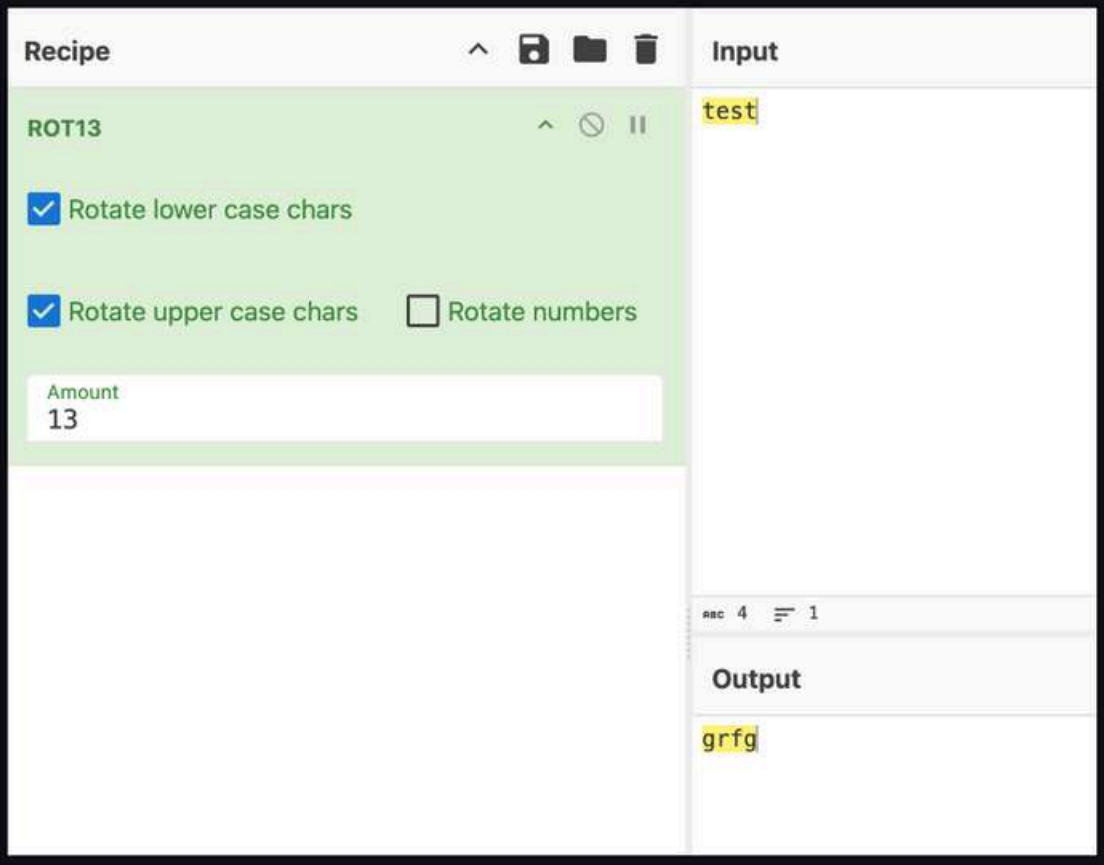

- **Description:** Shifts bits to the left, filling with 0s.
  - **Example:**
    - **Input:** A = 0011
    - **Shift Left 1:** A << 1 = 0110

22. **Bit Shift Right**
    - **Description:** Shifts bits to the right, filling with 0s.
    - **Example:**
        - **Input:** A = 0011
        - **Shift Right 1:** A >> 1 = 0001
23. **Rotate Left**
    - **Description:** Rotates bits to the left, wrapping around.
    - **Example:**
        - **Input:** A = 0011
        - **Rotate Left 1:** Rotate(A, 1) = 0110
24. **Rotate Right**
    - **Description:** Rotates bits to the right, wrapping around.
    - **Example:**
        - **Input:** A = 0011
        - **Rotate Right 1:** Rotate(A, -1) = 1001



## ROT Cipher Operations

### 25. ROT13



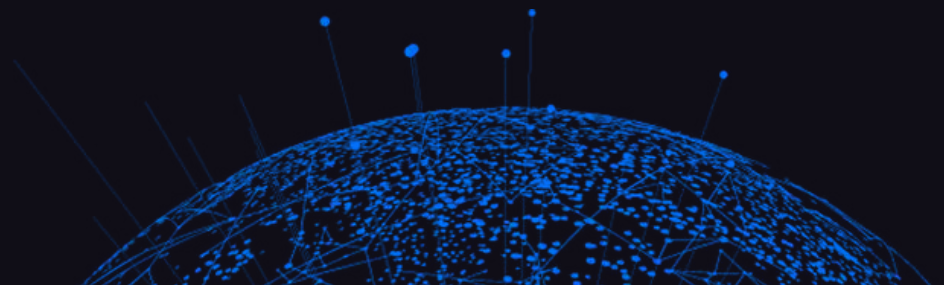

- **Description:** Substitutes each letter with the one 13 places after it in the alphabet.
    - **Example:**
        - **Input:** "HELLO"
        - **ROT13 Result:** "URYYB"

## Compression and Decompression Techniques

Compression is the process of reducing the size of data, while decompression restores the data to its original size. Various algorithms and formats are used for different purposes, from efficient storage to fast access. Here's an overview of each compression and decompression method with example test cases.

## Compression and Decompression Methods

1. **Compression**
    - **Description:** General term for algorithms that reduce data size by encoding information more efficiently.
    - **Example Test Case:**
        - **Input Data:** "hello hello hello"
        - **Compressed Output:** "h2e2l6o3"
2. **Raw Deflate**
    - **Description:** Compresses data using the DEFLATE algorithm without adding headers or checksums.
    - **Example Test Case:**
        - **Input Data:** "hello hello hello"
        - **Compressed Output:** Binary stream, e.g., `0x78 0x9c ...`
3. **Raw Inflate**
    - **Description:** Decompresses data compressed with Raw Deflate.
    - **Example Test Case:**
        - **Compressed Input:** Binary stream, e.g., `0x78 0x9c ...`
        - **Decompressed Output:** "hello hello hello"



## Block and Stream Ciphers

11. **LS47 Encrypt**
    - **Description:** LS47 is a symmetric block cipher.
    - **Example Test Case:**
        - **Input Data:** "hello world", **Key:** "somekey"
        - **Encrypted Output:** Binary data, e.g., `0x8f9c5d1e...`
12. **LS47 Decrypt**
    - **Description:** Decrypts data encrypted with LS47.
    - **Example Test Case:**
        - **Encrypted Input:** Binary data, e.g., `0x8f9c5d1e...`
        - **Decrypted Output:** "hello world"
13. **RC2 Encrypt**
    - **Description:** RC2 is a variable-key block cipher designed for encryption in the 1990s.
    - **Example Test Case:**
        - **Input Data:** "hello world", **Key:** "somekey"
        - **Encrypted Output:** Binary data, e.g., `0x3c1d4e7f...`
14. **RC2 Decrypt**
    - **Description:** Decrypts data encrypted with RC2.
    - **Example Test Case:**
        - **Encrypted Input:** Binary data, e.g., `0x3c1d4e7f...`
        - **Decrypted Output:** "hello world"

# Hashing Algorithms and Checksum Methods

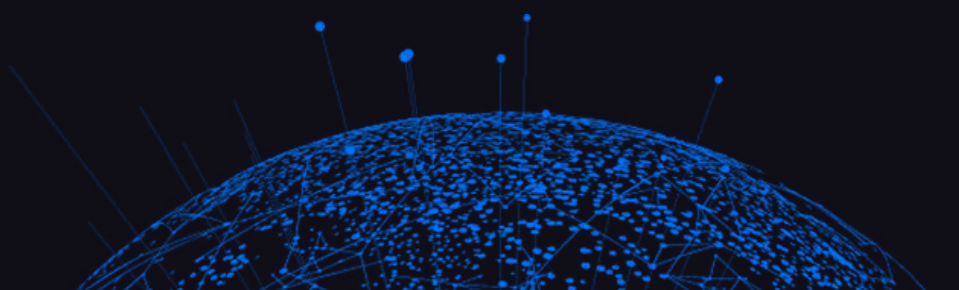
Below is a detailed overview of various hashing algorithms and checksum methods with their technical analysis and example test cases.

## 1. MD2

- **Description:** MD2 is a cryptographic hash function designed by Ronald Rivest. It produces a 128-bit hash value.
- **Technical Analysis:** Known for its simplicity but slower compared to modern algorithms. It is rarely used today due to vulnerabilities.
- **Example Test Case:**
  - **Input Data:** "hello"
  - **MD2 Hash Output:** `04e0f4e3e32f0b9ecfa4d8efba6bb9b2`

## 2. MD4

- **Description:** MD4 is a cryptographic hash function designed by Ronald Rivest. It generates a 128-bit hash value.
- **Technical Analysis:** Vulnerable to collision attacks and is largely replaced by MD5 and more secure algorithms.
- **Example Test Case:**
  - **Input Data:** "hello"
  - **MD4 Hash Output:** `d7a8fbb9c2c5d7e7f6a5a8b1b4f6e6b7`



# Technical Analysis of Cryptographic Operations

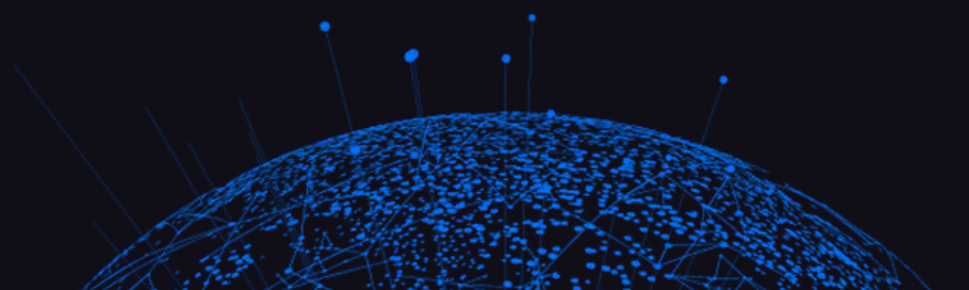
Below is a detailed overview of various cryptographic operations with their technical analysis and example test cases.

## 1. Public Key

- **Description:** A public key is part of a public-private key pair used in asymmetric encryption. It is used to encrypt data or verify signatures.
- **Technical Analysis:** The public key can be distributed openly, while the private key is kept secret.
- **Example Test Case:**
  - **Input Data:** Public Key (PEM format)
  - **Operation:** Verify the format and compatibility with encryption algorithms.
  - **Output:** Valid public key for use in encryption or signature verification.

## 2. Parse X.509 Certificate

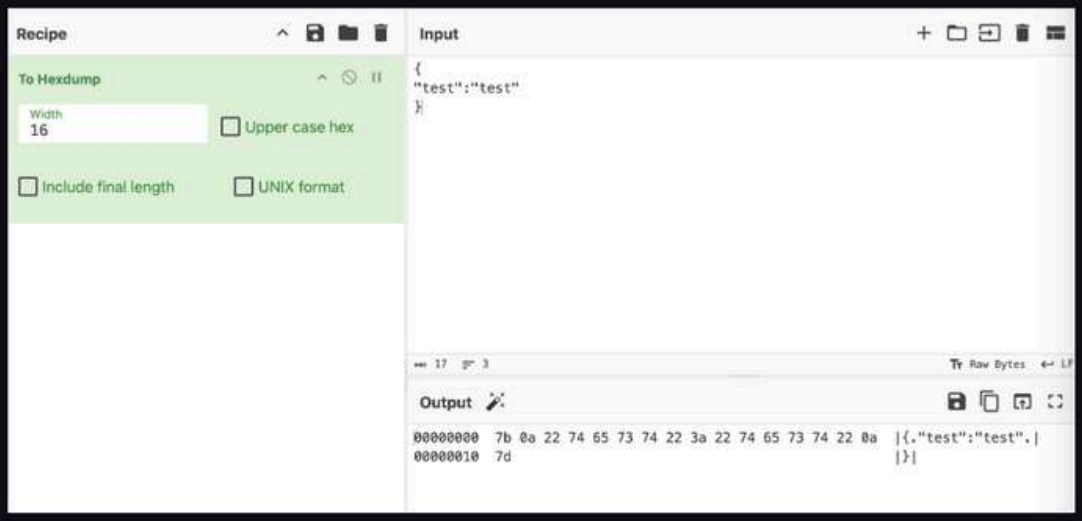
- **Description:** X.509 certificates are digital certificates that use the X.509 standard to provide information about the certificate holder and issuer.
- **Technical Analysis:** Parsing involves extracting information like issuer, subject, validity period, and public key from the certificate.
- **Example Test Case:**
  - **Input Data:** X.509 Certificate (PEM format)
  - **Operation:** Extract certificate details.
  - **Output:** Certificate details such as issuer, subject, and validity dates.



# Data Format Conversion Algorithms

Here's a detailed breakdown of various data format conversions, including technical analysis and example test cases for each item.

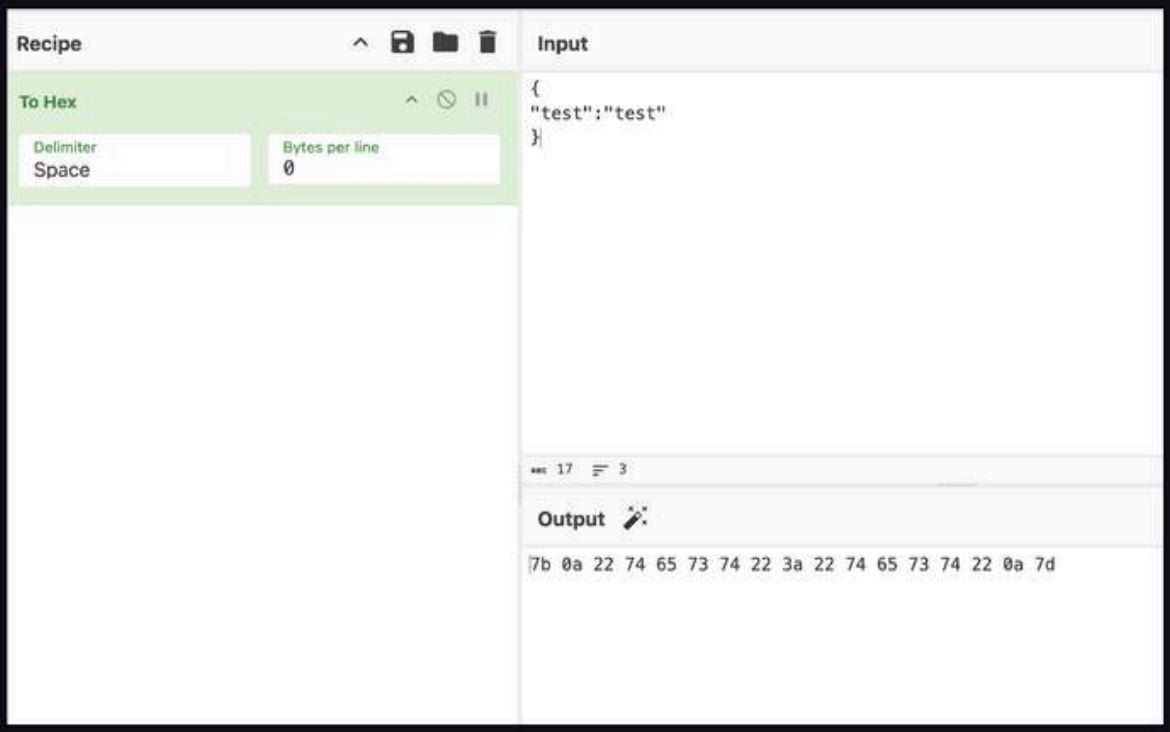
## 1. To Hexdump




- **Description:** Converts binary data into a hex dump format where each byte is represented by its hexadecimal value.
- **Technical Analysis:** This format is useful for debugging and visualizing binary data. Each byte is converted into a two-digit hexadecimal representation, often accompanied by an ASCII representation of the data.
- **Example Test Case:**
  - **Input Data:** Binary data `b'\x01\x02\x03'`
  - **Operation:** Convert to hex dump.
  - **Output:**

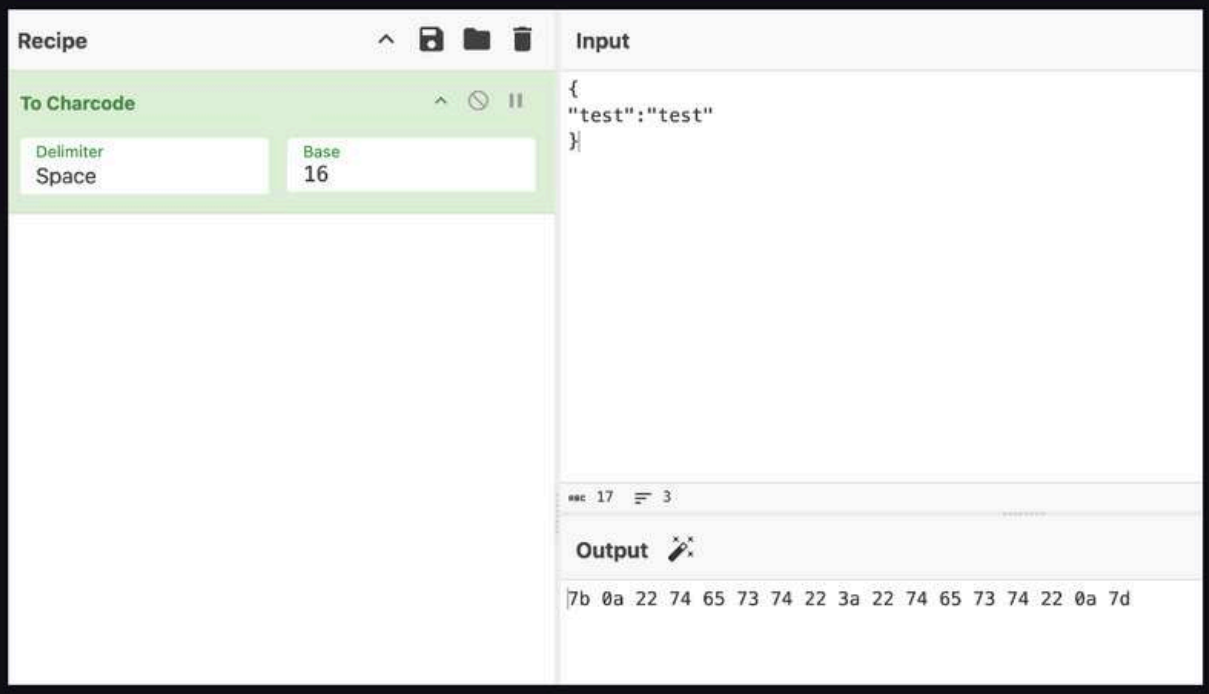
```
00000000  01 02 03
```

## 3. To Hex



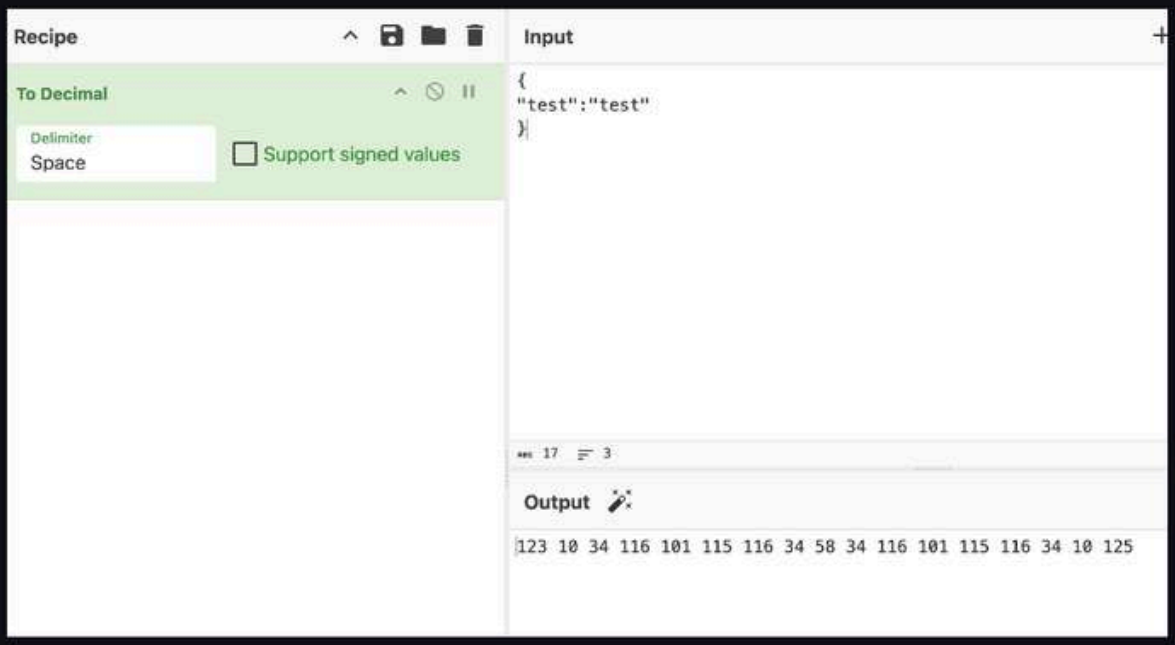

- **Description:** Converts binary data into a hexadecimal string.
- **Technical Analysis:** Each byte of binary data is converted to its two-digit hexadecimal representation.
- **Example Test Case:**
  - **Input Data:** Binary data `b'\x10\x20\x30'`
  - **Operation:** Convert to hex.
  - **Output:** `102030`

## 5. To Charcode



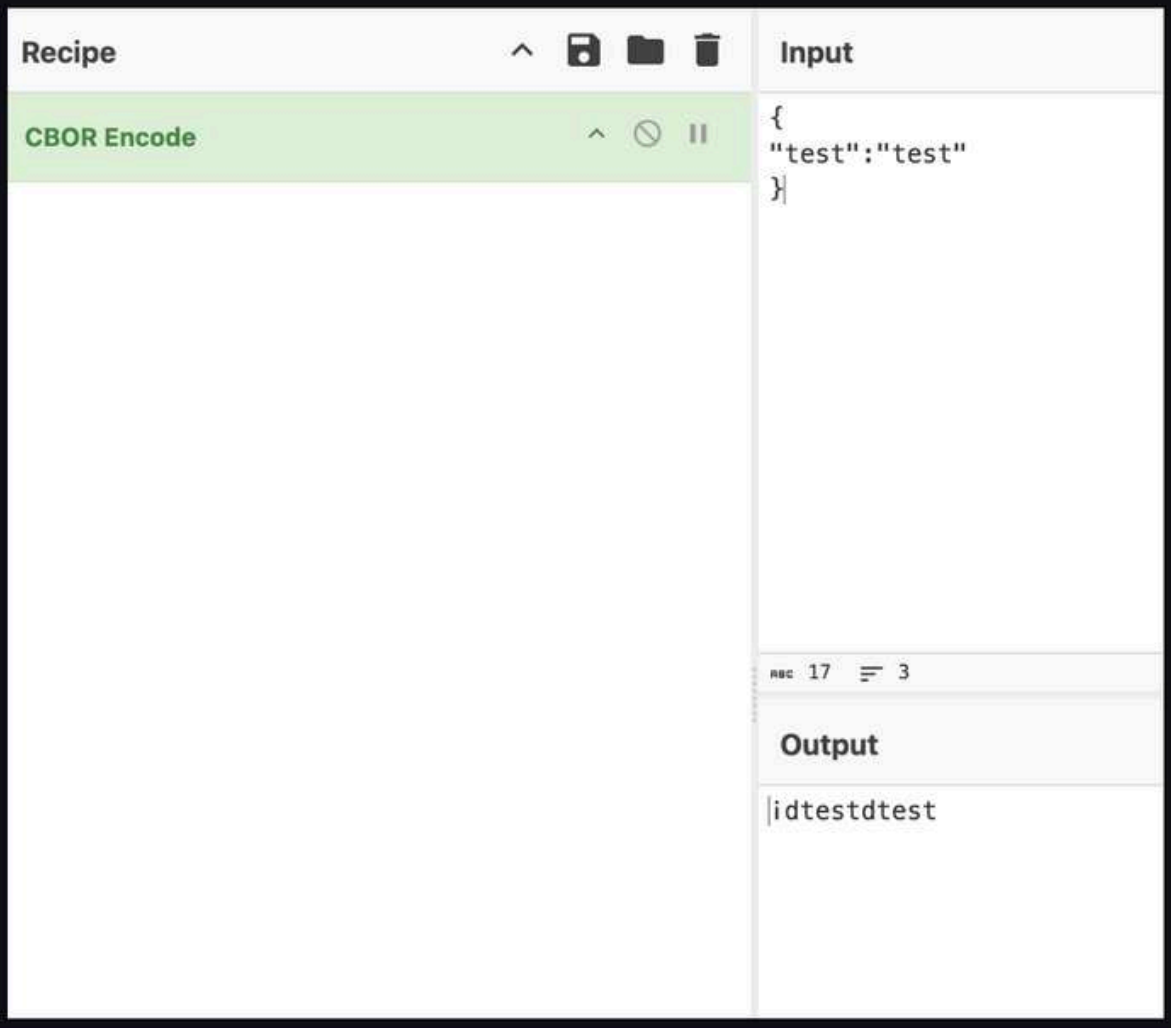

- **Description:** Converts text into a sequence of character codes (e.g., ASCII or Unicode).
- **Technical Analysis:** Each character in the text is translated into its corresponding numeric code.
- **Example Test Case:**
  - **Input Data:** Text `ABC`
  - **Operation:** Convert to charcodes.
  - **Output:** `[65, 66, 67]` (ASCII codes for `A`, `B`, `C`)

## 7. To Decimal

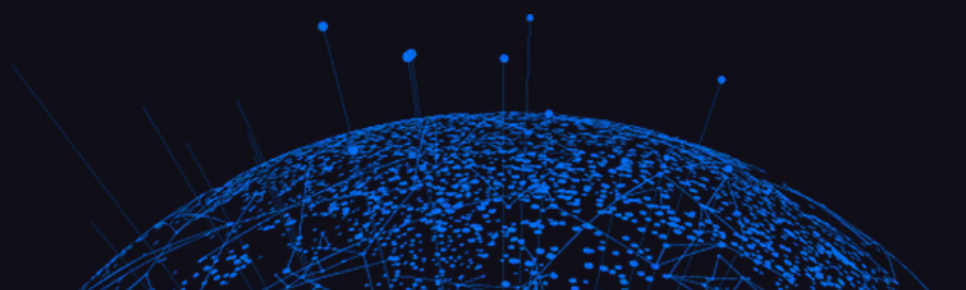



- **Description:** Converts binary or other numeric representations into decimal numbers.
- **Technical Analysis:** This involves interpreting the binary or other number format as a decimal number.
- **Example Test Case:**
  - **Input Data:** Binary `1010`
  - **Operation:** Convert to decimal.
  - **Output:** `10`

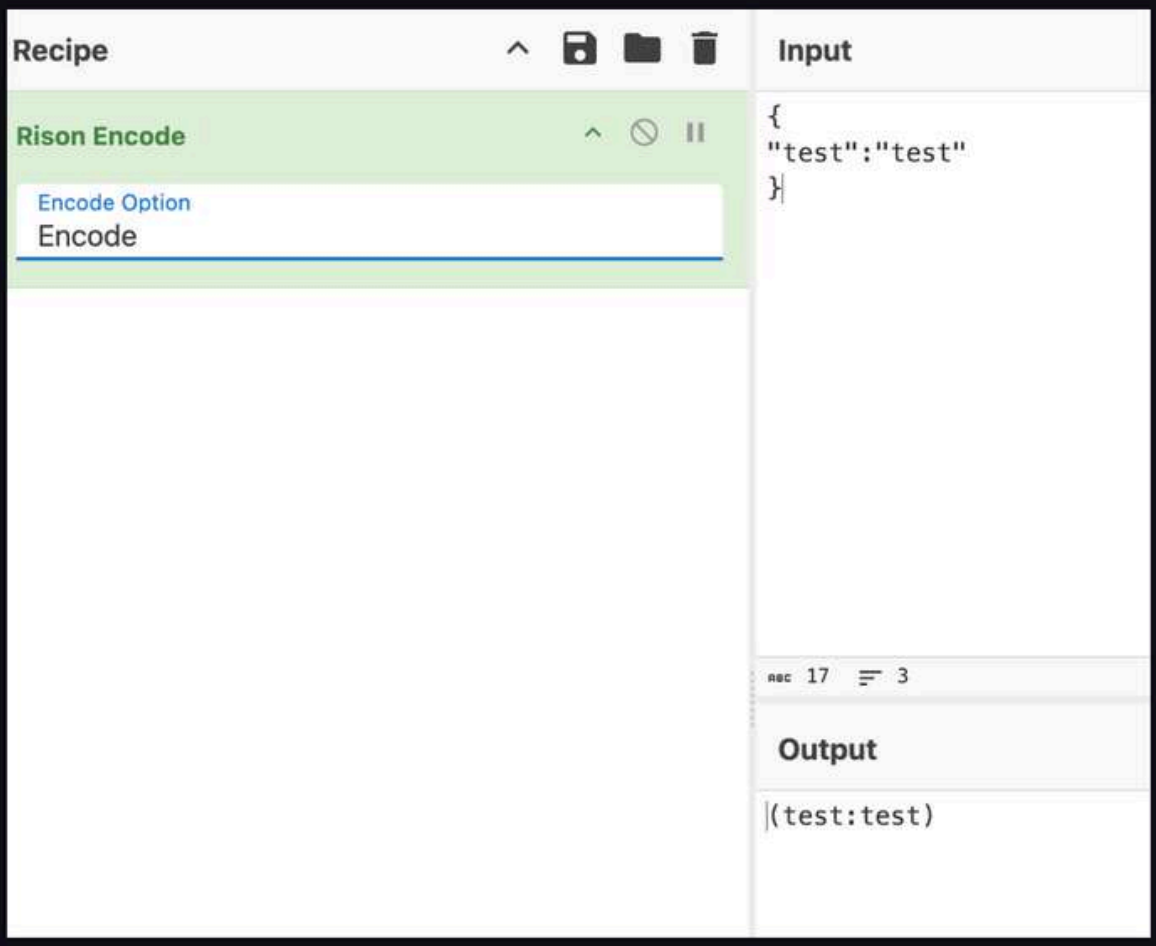
## 65. CBOR Encode



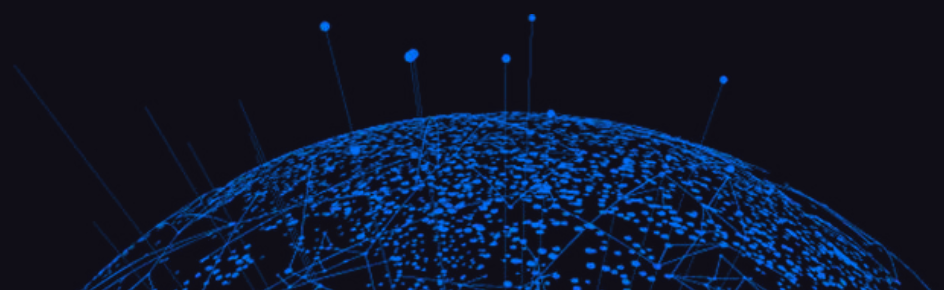

- **Description:** Encodes data into Concise Binary Object Representation (CBOR) format.
- **Technical Analysis:** CBOR is a binary format similar to JSON but more compact.
- **Example Test Case:**
  - **Input Data:** Data `{ "name": "John", "age": 30 }`
  - **Operation:** Convert to CBOR.
  - **Output:** CBOR binary data.

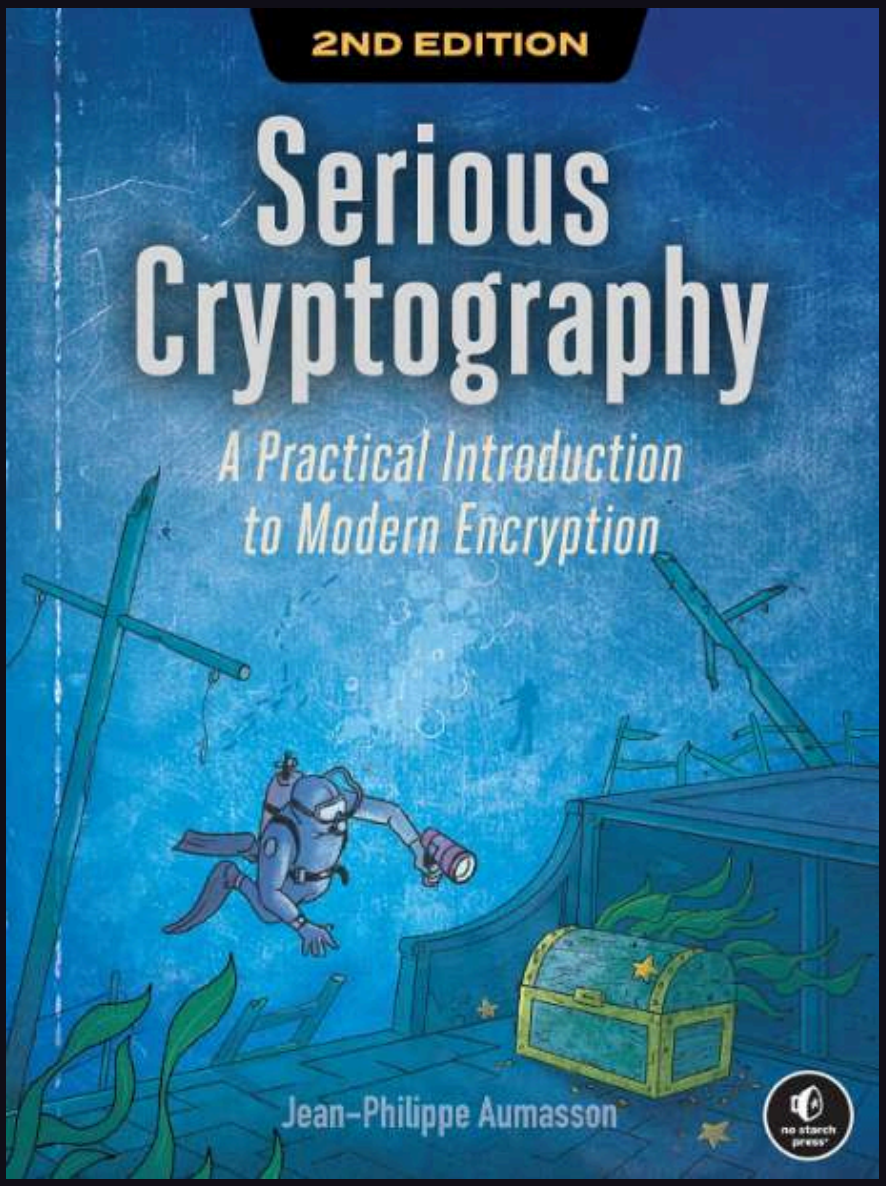


## 68. Rison Encode




- **Description:** Encodes data into Rison format, a compact JSON-like format.
- **Technical Analysis:** Rison encoding provides a more compact representation of JSON data.
- **Example Test Case:**
  - **Input Data:** Data `{ "name": "John", "age": 30 }`
  - **Operation:** Convert to Rison.
  - **Output:** Rison encoded string.

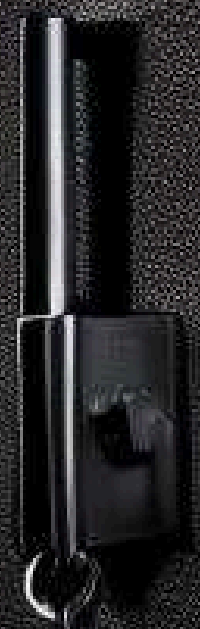



2ND EDITION
Serious Cryptography
A Practical Introduction to Modern Encryption
Jean-Philippe Aumasson
no starch press

# HADESS

## cat ~/.hadess

"Hadess" is a cybersecurity company focused on safeguarding digital assets and creating a secure digital ecosystem. Our mission involves punishing hackers and fortifying clients' defenses through innovation and expert cybersecurity services.




Website:
**WWW.HADESS.IO**

Email
**MARKETING@HADESS.IO**


To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.