Secure Laboratory Image Encryption

Cybersecurity Project

Group Members:

Mayssam Hassen
Sabrine Ayadi
Sayda Ouaddar
Aya Hafsi

Major: Business Analytics

Minor: Information Technology

Contents

1 **Introduction:** 3

1.1 Problem Statement . . . 3

2 **Existing solutions:** 4

2.1 AES-Only Encryption . . . 4

2.2 RSA-Only Encryption . . . 4

2.3 DICOM with Basic TLS Encryption . . . 5

2.4 Simple XOR Encryption . . . 5

3 **Proposed Solution** 6

3.1 ChaCha20 Encryption: . . . 6

3.2 AES + RSA Hybrid Encryption : . . . 7

4 **Technical Part:** 8

4.1 AES encryption: . . . 8

4.1.1 Algorithm Breakdown: . . . 8

4.1.2 Main steps in AES: . . . 9

4.1.3 Mathematical representation: . . . 9

4.1.4 AES encryption part . . . 9

4.1.5 AES decryption part . . . 10

4.2 RSA encryption . . . 11

4.2.1 Key Features: . . . 11

4.2.2 Key Generation . . . 11

4.2.3 Encryption and Decryption Process . . . 12

4.2.4 RSA Mathematical Foundations . . . 12

4.2.5 Matrix Representation of the State . . . 12

4.3 Hybrid Encryption (AES + RSA) . . . 13

4.3.1 Functional Flow . . . 13

4.3.2 Detailed Technical Principles . . . 14

4.3.3 Use Case Application . . . 16

4.4 ChaCha20 encryption: . . . 17

4.4.1 Key Features . . . 17

4.4.2 State Initialization . . . 17

4.4.3 The Quarter Round Function . . . 18

4.4.4 Full Rounds . . . 19

4.4.5 ChaCha20 for Image Encryption: . . . 19

5 **Conclusion** 21

1 Introduction:

In today's digital world, technology is playing a vital role in improving healthcare. Laboratories now use digital tools to handle medical images like X-rays, MRI scans, and lab results. Instead of using physical copies, these images are processed and shared electronically, making communication faster and more efficient.

This shift to digital has become the norm, with over 90% of hospitals in developed countries using electronic health records and digital imaging. It has helped improve diagnosis, speed up workflows, and support remote collaboration between medical professionals.

1.1 Problem Statement

Sending such sensitive information online can be risky. If medical images are not protected properly, they can be accessed by hackers or unauthorized users. This can harm patient privacy, damage the lab's reputation, and even lead to legal problems.

How can laboratories protect sensitive medical images during digital transmission?

To address this concern, cryptography offers a robust framework for securing digital data. It relies on mathematical algorithms to transform sensitive information into a format that is unreadable to unauthorized users. One of the most essential techniques within cryptography is encryption, which involves converting plaintext data—such as a medical image—into ciphertext, a scrambled version that appears meaningless without the proper decryption key.

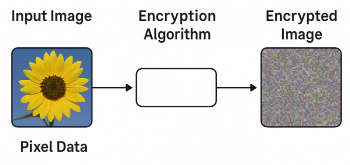


Figure 1: Image Encryption Process

2 Existing solutions:

Despite advancements in digital encryption, many existing solutions for protecting medical images during transmission suffer from security gaps, limited scalability, or performance inefficiencies. Here are some of the commonly used methods and their shortcomings:

2.1 AES-Only Encryption

Description: AES (Advanced Encryption Standard) is a symmetric key algorithm widely used for encrypting data including medical images.

Weaknesses:

- **Symmetric encryption requires secure key exchange, which can be compromised during transmission.**
- **Once the AES key is exposed, all encrypted data becomes vulnerable. .**
- **Vulnerable to brute force attacks if key sizes are small or reused.**
- **Does not support non-repudiation (no way to prove who encrypted the data).**

2.2 RSA-Only Encryption

Description: RSA is an asymmetric cryptographic algorithm often used to encrypt small chunks of data or symmetric keys.

Weaknesses:

- **Very slow for large files like medical images.**
- **High computational cost on lower-end devices. .**
- **Not suitable for real-time processing or large-scale data sets.**
- **More susceptible to quantum attacks in the future**

2.3 DICOM with Basic TLS Encryption

Description: DICOM (Digital Imaging and Communications in Medicine) is a standard in medical imaging. Often secured using Transport Layer Security (TLS) during transmission. .

Weaknesses:

- **Security is only active during transmission (not at rest).**
- **TLS can be compromised by MITM (man-in-the-middle) attacks if not properly configured.**
- **No encryption once the file is stored locally or shared via third-party apps.**
- **No multi-layered encryption to safeguard against internal threats.**

2.4 Simple XOR Encryption

Description: XOR-based encryption applies a logical XOR operation between the image pixels and a key stream.

Weaknesses:

- **Too simplistic; easily broken using frequency analysis or known-plaintext attacks..**
- **Lacks complexity and randomness.**
- **No Not suitable for securing sensitive healthcare data.**

3 Proposed Solution

MEDICRYPTIS

To address the limitations in current systems for transmitting medical images we propose a dedicated web application for laboratories.This web app allows medical professionals to upload ,encrypt ,and decrypt MRI scans ,lab results ,and X-ray images, them ensuring data privacy and compliance with healthcare regulations

One of the key features of our solution is that it provides two encryption methods:

3.1 ChaCha20 Encryption:

modern stream cipher designed for high-speed, secure encryption. Unlike traditional block ciphers, it operates on a continuous stream of data and avoids common vulnerabilities like timing attacks or issues related to block padding. It uses a 256-bit key and a nonce to generate a keystream that is XORed with the image data.

Strengths:

- High speed and strong security
- Efficient performance on any device
- High resistance to cryptanalysis and side-channel attacks.
- Easy integration with image files

- **Designed to be constant time to avoid timing attacks**

When to use :

- **Internal sharing between lab technicians and doctors. .**
- **Fast encryption/decryption where high throughput is needed.**
- **Scenarios where performance matters as much as privacy**

3.2 AES + RSA Hybrid Encryption :

combines two powerful algorithms: AES (Advanced Encryption Standard) for encrypting the actual image, and RSA (Rivest-Shamir-Adleman) for encrypting the AES key. AES is a symmetric cipher known for its speed and security, widely used in everything from secure communications to file storage. RSA, an asymmetric algorithm, allows secure key exchange, ensuring that only the intended recipient can decrypt the AES key and thus the image.

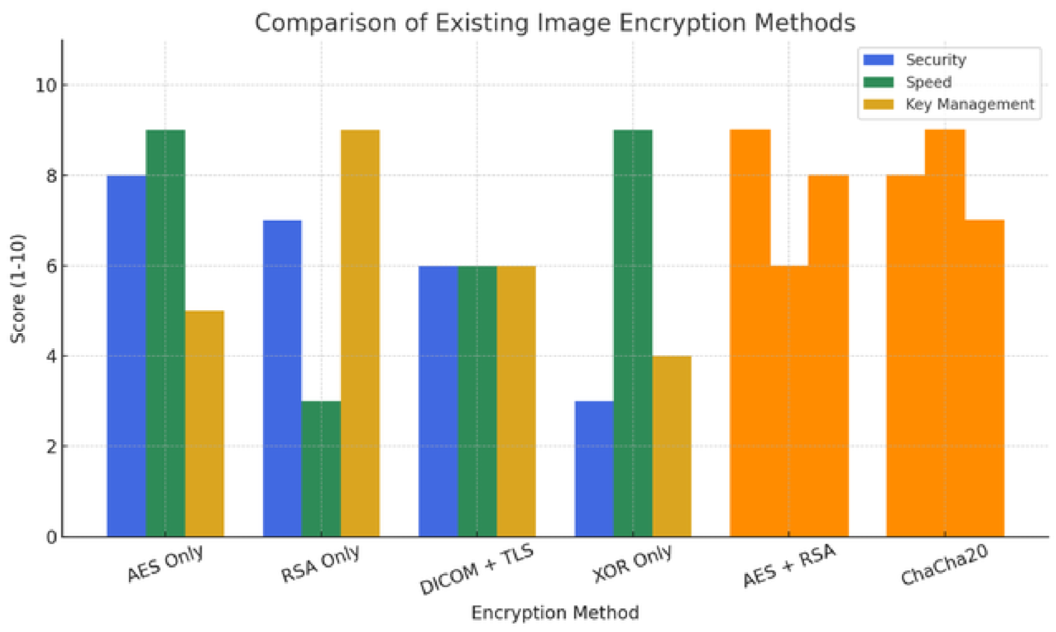
Strengths:

- **AES for encrypting the image content quickly**
- **RSA for encrypting the AES key, ensuring only authorized users can decrypt the file**

When to use :

- **Sending sensitive data to external parties or hospitals.**
- **Long-term storage where keys may change or rotate.**
- **Use cases that require public-key infrastructure and non-repudiation**

Both methods offer customizable security levels to meet different user needs. The app features a user-friendly interface for non-technical users and is accessible from any secure browser. It is designed to follow healthcare privacy standards, including HIPAA and GDPR, ensuring the secure transmission of sensitive data.



4 Technical Part:

4.1 AES encryption:

AES is a symmetric encryption algorithm, meaning it uses the same key for both encryption and decryption. It's widely used for securing data due to its speed, strength, and efficiency.

4.1.1 Algorithm Breakdown:

Block cipher: AES encrypts data in fixed-size blocks of 128 bits.
Key sizes: Supports 128-192 or 256-bit keys.
Rounds: The number of processing rounds depends on the key size:

10 rounds for 128-bit keys
12 rounds for 192-bit keys
14 rounds for 256-bit keys

4.1.2 Main steps in AES:

1)SubBytes: Each byte is substituted using a predefined S-box.

2)ShiftRows: Rows of the state matrix are shifted.

3)MixColumns: Columns are mixed using linear transformation.

4) AddRoundKey: The current state is XORed with a round key.

4.1.3 Mathematical representation:

StateS:4×4 byte matrix representing 128-bit block.

Round Key K:Derived from the main key using the key schedule.

AddRoundKey:

$$S \leftarrow S \oplus K$$

SubBytes:

$$S_{i,j} \leftarrow \text{S-box}(S_{i,j})$$

ShiftRows:Cyclically shift each row partie

MixColumns:Matrix multiplication over GF(2):

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

4.1.4 AES encryption part
Let's define:

- **Initial Round**:

$$S \leftarrow S \oplus K_0$$

- **Nr 1 Rounds (Nr = 10/12/14)**:

$$S \leftarrow \text{MixColumns}(\text{ShiftRows}(\text{SubBytes}(S))) \oplus K_i$$

- **Final Round**:

$$S \leftarrow \text{ShiftRows}(\text{SubBytes}(S)) \oplus K_{Nr}$$

4.1.5 AES decryption part

Decryption in AES is the inverse of encryption. Each round applies the inverse transformations in reverse order.

The main inverse operations are:

- **InvShiftRows**: Right cyclic shifts of rows.
- **InvSubBytes**: Apply the inverse S-box to each byte.
- **InvMixColumns**: Apply the inverse mixing matrix over GF(2^8).**AddRoundKey** : $XORwiththecorrespo$

1. **Initial Round:**

$$S \leftarrow S \oplus K_{Nr}$$

   where K_{Nr} is the round key for the last round.

2. **Nr - 1 Main Rounds:**

$$S \leftarrow InvMixColumns(InvShiftRows(InvSubBytes(S))) \oplus K_i$$

   for rounds $i = Nr - 1$ down to 1.

3. **Final Round (without InvMixColumns):**

$$S \leftarrow InvShiftRows(InvSubBytes(S)) \oplus K_0$$

- **InvSubBytes**: Apply the inverse S-box, i.e., replace each byte $s_{i,j}$ with $InvSBox(s_{i,j})$.
- **InvShiftRows**:
  - Row 0: No shift

  - Row 1: Right shift by 1 byte
  - Row 2: Right shift by 2 bytes
  - Row 3: Change to the right by 3 bytes

- **InvMixColumns**: Each column is multiplied by the following matrix over GF(2^8) :

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}$$

4.2 RSA encryption

RSA (Rivest–Shamir–Adleman) is a foundational asymmetric encryption algorithm based on number theory, providing secure data transmission by leveraging the difficulty of factoring large prime numbers. It is widely used for encrypting sensitive information, especially for securely exchanging symmetric keys like AES keys.

4.2.1 Key Features:

- **Public-Key Cryptography**: Uses separate keys for encryption (public) and decryption (private).
- **Security Basis**: Depends on the computational difficulty of factoring the product of two large primes.
- **Scalability**: Key sizes typically range from 2048 to 4096 bits to ensure security.
- **Non-Repudiation**: Guarantees that a sender cannot later deny the authenticity of their message.

4.2.2 Key Generation

The RSA key generation process involves:

1. **Selecting Two Prime Numbers**: p and q are large, random primes.
2. **Computing the Modulus**: $n = p \times q$

3. **Calculating Euler's Totient Function**: $\varphi(n) = (p-1)(q-1)$
4. **Choosing the Public Exponent**: e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$
5. **Calculating the Private Exponent**: $d \equiv e^{-1}\varphi(n)$ (modular inverse of e modulo $\varphi(n)$).

4.2.3 Encryption and Decryption Process

Encryption: Given a plaintext message M, the ciphertext C is calculated as:

$$C = M^e n$$

Decryption: To recover the original message M:

$$M = C^d n$$

Here, (e, n) is the public key, and (d, n) is the private key.

4.2.4 RSA Mathematical Foundations

- **Prime Multiplication**: Easy - multiplying two large primes.
- **Prime Factoring**: Hard - factoring their product n back into p and q.
- **One-Way Function**: Fundamental to RSA security.

Example:

$$p = 61, \quad q = 53n = 3233, \quad \varphi(n) = 3120e = 17d = 2753$$

4.2.5 Matrix Representation of the State

$$\text{State Matrix} = \begin{bmatrix} p & q & n & \varphi(n) \\ e & d & & \\ M & C & & \\ & & & \end{bmatrix}$$

4.3 Hybrid Encryption (AES + RSA)

The hybrid encryption algorithm combines two techniques: symmetric encryption (AES) and asymmetric encryption (RSA) to offer both speed and security in transmitting sensitive data like medical images.

- AES is highly efficient at encrypting large amounts of data due to its block-based structure and lightweight operations. It provides speed.
- RSA, while slower for large data, excels at securely encrypting small pieces of data like encryption keys. It provides security for key distribution.

By encrypting the actual image with AES and encrypting the AES key with RSA, we ensure that:

- Only authorized recipients can decrypt the image.
- The transmission remains fast and scalable even for large files.

4.3.1 Functional Flow

The complete process is broken down into four main stages:

Key Generation

- The receiver generates an RSA key pair: a public key for encryption and a private key for decryption.
- Simultaneously, a fresh, random AES session key is generated for each new medical image.

Encryption Phase

- The medical image is encrypted using the generated AES session key, applying AES encryption block by block.
- The AES session key itself is then encrypted using the receiver's RSA public key.

Transmission Phase

- Both the encrypted image and the RSA-encrypted AES key are packaged together and transmitted to the receiver.

Decryption Phase

- Upon reception, the receiver uses their RSA private key to decrypt the AES session key.
- With the recovered AES key, the receiver decrypts the image, restoring the original medical data.

Thus, the data remains protected even if intercepted during transmission — without access to the private RSA key, decryption is impossible.

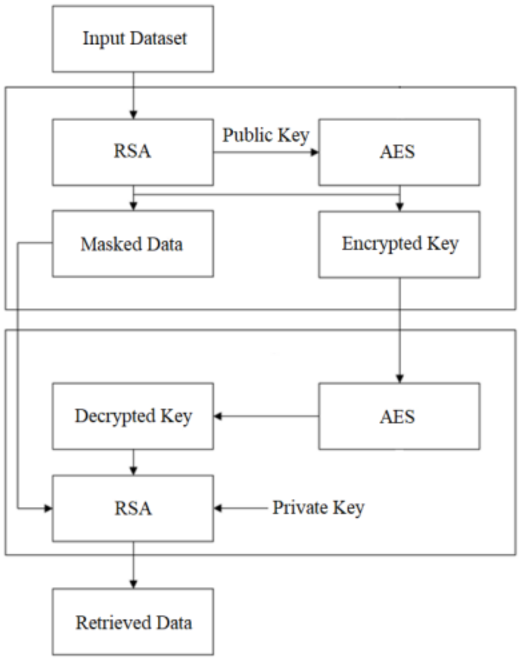


Figure 4 Process of RSA and AES Hybrid Model

4.3.2 Detailed Technical Principles

The encryption flow can be broken down into a sequence of dependent steps, where each cryptographic operation builds upon the previous one to ensure confidentiality and integrity.

Step 1: AES Session Key Generation:

Before encrypting the image, a fresh, random AES session key K_{AES} is generated. Typically, a 128-bit or 256-bit key is created using a cryptographically secure random number generator.

Example:

$$K_{AES} = \texttt{0x2b7e151628aed2a6abf7158809cf4f3c}$$

This key will be used for encrypting the full medical image.

Step 2: AES Image Encryption:

The medical image is processed and encrypted block by block using AES.

- The image is divided into fixed-size blocks (typically 128 bits each).
- Each block undergoes AES encryption rounds using the key K_{AES}.
- An appropriate AES mode of operation is used to enhance security.

$$CiphertextBlocks = AES - Encrypt(ImageBlocks, K_{AES})$$

Thus, the full encrypted image C_{image} is obtained.

Step 3: RSA Encryption of the AES Key:

To protect the AES key itself during transmission, it is encrypted with the receiver's RSA public key (e, n).

Mathematically, if the AES key K_{AES} is represented as an integer m, then:

$$c = m^e n$$

Step 4: Transmission:

The system transmits two elements to the receiver:

- C_{image}: the AES-encrypted medical image
- c: the RSA-encrypted AES session key

Both are typically sent together in a secure message package.

Step 5: Decryption on Receiver's Side

Upon receiving the package:

1. The receiver uses their RSA private key d to decrypt the AES key:

$$m = c^d n$$

2. The receiver then decrypts the medical image:

$$ImageBlocks = AES - Decrypt(C_{image}, K_{AES})$$

   Restoring the original medical data.

This hybrid design ensures:

- **Efficiency**: AES encrypts large images quickly.
- **Security**: RSA protects the AES key during transmission.
- **Scalability**: New AES keys can be generated per image for enhanced security.

4.3.3 Use Case Application

The hybrid AES + RSA encryption system is particularly well-suited for the secure transmission of medical images between laboratories, hospitals, and healthcare providers.

- **Confidentiality:** Ensures that sensitive patient data remains private and inaccessible to unauthorized parties during transmission and storage.
- **Integrity:** Maintains the integrity of medical images by preventing unauthorized alterations or corruptions during transit.
- **Controlled Access:** Guarantees that only authorized personnel with the correct decryption keys can access the original medical data.
- **Long-term Storage and Key Rotation:** Supports secure archiving of medical images with periodic key rotation, ensuring compliance with evolving security standards and protecting against potential future threats.

4.4 ChaCha20 encryption:

ChaCha20 is a stream cipher, a symmetric encryption algorithm that generates a pseudorandom keystream to encrypt data via XOR. It was designed by Daniel J. Bernstein as an improvement over Salsa20. It's used in many secure applications like TLS 1.3, OpenSSH, and WireGuard VPN due to its speed and resistance to timing attacks

Algorithm Breakdown

4.4.1 Key Features

- **Key (256-bit):** Provides military-grade security equivalent to AES-256, resistant to brute-force attacks. Divided into 8 words for keystream generation
- **Nonce (96-bit):** Unique ”message number” that enables safe key reuse. Prevents catastrophic keystream repetition when encrypting multiple images
- **Counter (32-bit):** Block sequence number that ensures uniqueness within a single image's encryption. Rolls over after 256GB of datas
- **Block (64-byte):** Standard processing unit matching the algorithm's 4×4 word matrix. Efficient for modern hardware
- **Rounds (20 total):**Alternating column/diagonal transformations that thoroughly mix the state. Provides a security margin beyond the 8 rounds needed for full diffusion.

4.4.2 State Initialization

The state initialization in ChaCha20 defines how the cipher transforms a 256-bit key, 96-bit nonce, and 32-bit counter into a 512-bit (64-byte) initial state matrix. This matrix serves as the foundation for keystream generation.

$$\text{State} = \begin{bmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ b & n_0 & n_1 & n_2 \end{bmatrix}$$

Figure 4: 4×4 matrix of 32-bit words

The state is populated with three components

1. **Constants (4 words, 128 bits)** Fixed ASCII string ”expand 32-byte k” to ensure diffusion and break symmetry
2. **Key (8 words, 256 bits)** The 256-bit encryption key is split into eight 32-bit words (k to k)
3. **Counter + Nonce (4 words, 128 bits)**)

4.4.3 The Quarter Round Function

The Quarter Round (QR) is the core nonlinear operation that gives ChaCha20 its cryptographic strength. It transforms four 32-bit words (a, b, c, d) through a carefully designed sequence of modular addition, XOR, and bit rotations.

1. Addition
2. Bitwise XOR
3. Left Rotation

$$
\begin{aligned}
a &\leftarrow a + b \\
d &\leftarrow (d \oplus a) \lll 16 \\
c &\leftarrow c + d \\
b &\leftarrow (b \oplus c) \lll 12 \\
a &\leftarrow a + b \\
d &\leftarrow (d \oplus a) \lll 8 \\
c &\leftarrow c + d \\
b &\leftarrow (b \oplus c) \lll 7
\end{aligned}
$$

Figure 5: mathematical representation

4.4.4 Full Rounds

ChaCha20 applies 20 rounds of transformations to its internal state to generate cryptographically secure pseudorandom output. These rounds alternate between two distinct patterns to ensure thorough mixing of the state

Round Types

- Column Rounds (Even-numbered: 0,2,4...18)
- Diagonal Rounds (Odd-numbered: 1,3,5...19)

4.4.5 ChaCha20 for Image Encryption:

1.Pre-Processing

Images are converted to a standardized byte stream, handling formats like BMP, PNG, and JPEG while preserving metadata. A 256-bit key is derived using HKDF, and a unique 96-bit nonce is generated for each image to prevent keystream reuse

2.Core Encryption

ChaCha20 processes images in 64-byte blocks, applying 20 rounds of transformations (alternating column and diagonal operations) to generate a keystream. Each round uses modular addition, XOR, and bit rotations for strong diffusion. The keystream is XORed with pixel bytes for encryption

3.Format Handling

- **BMP:**Encrypts raw pixel data directly.
- **PNG:**Targets compressed IDAT chunks, requiring decompression/recompression.
- **JPEG:**Encrypts DCT coefficients to maintain structure.
- **TIFF:**Processes strips individually for partial decryption.

4.Security Features

- **Visual Obfuscation:**Eliminates patterns and equalizes histograms.
- **Unique Keystreams:**Nonce + counter ensures no repetition.

5.Decryption

Decryption mirrors the encryption process exactly, leveraging the XOR operation's self-inverting property. The system requires the same key, nonce, and initial counter value used during encryption. When properly implemented, decryption restores the original image bit-for-bit, including perfect preservation of alpha channels and color profiles.

5 Conclusion

As the medical field continues its shift toward digitalization, ensuring the secure transmission of sensitive medical images becomes increasingly vital. While traditional encryption methods like AES and RSA have their merits, they fall short in terms of scalability, speed, and adaptability to modern cyber threats. Our proposed solution—a web application offering both ChaCha20 and AES+RSA hybrid encryption—bridges this gap by combining performance with robust security tailored to healthcare needs. Looking forward, integrating advanced techniques such as homomorphic encryption or quantum-resistant algorithms could further strengthen data protection, ensuring long-term privacy and compliance in an evolving digital landscape.

References

[1] Daniel J. Bernstein, *ChaCha, a variant of Salsa20*, 2008. Available online: `https://cr.yp.to/chacha/chacha-20080128.pdf`

[2] Yaron Nir and Adam Langley, *ChaCha20 and Poly1305 for IETF Protocols*, RFC 8439, Internet Engineering Task Force (IETF), 2018. Available online: `https://datatracker.ietf.org/doc/html/rfc8439`

[3] D. Kauchak, *Introduction to Encryption*, Computer Science Department, Pomona College. Available online: `https://cs.pomona.edu/~dkauchak/classes/s17/cs52-s17/handouts/encryption.pdf`