Ministry of Higher Education
and Scientific Research

University of Tunis

Tunis Business School

وزارة التعليم العالي و البحث العلمي

جامعة تونس

المعهد العالي للأعمال بتونس

Project Report

Theoretical Foundations of File Encryption, Decryption, and Hashing

Elaborated By :

Fatma BEN REJEB
Roua ELKAMEL
Haifa AMARA
Ons SAID
Mariem MHADHBI

Supervisor :

Prof. Manel ABDELKADER

Academic year : 2024/2025

Contents

GitHub Link

1 Introduction

In today's interconnected digital world, where sensitive data is constantly exchanged and stored, securing information has become a critical priority. File encryption, decryption, and hashing are fundamental techniques in cybersecurity, ensuring confidentiality, integrity, and authenticity of data. This report aims to provide a comprehensive understanding of these core concepts by exploring their theoretical foundations, functional flow, and mathematical principles. It also examines standard protocols, security requirements, and existing solutions currently used in the industry. Serving as a foundation for the development of a user-friendly application, this report not only highlights the strengths and limitations of these technologies, but also sets the stage for practical implementation. The insights provided here will guide the secure processing of files and promote better protection of digital information.

2 Core Concepts

2.1 Encryption

Encryption is the process of converting readable data (plaintext) into an unreadable format (ciphertext) using a mathematical algorithm and a key. This ensures that only authorized users with the correct decryption key can access the original information.

- **Main Components:**
  - Plaintext: Original, readable data
  - Ciphertext: Encrypted, unreadable output
  - Encryption Algorithm: The mathematical function used to encrypt data
  - Key: A secret value used to perform encryption (and decryption)
- **Types of Encryption:**
  - **Symmetric Encryption:** Uses **the same key** for both encryption and decryption
    - **Advantages:** Fast and efficient, especially for bulk data
    - **Disadvantages:** Key distribution can be difficult

- **Asymmetric Encryption:** Uses **a pair of keys**
  - **Public Key:** For encryption
  - **Private Key:** For decryption
  - **Advantages:** Solves key distribution problem, enables digital signatures
  - **Disadvantages:** Slower than symmetric encryption

Symmetric Encryption	Asymmetric Encryption
AES (Advanced Encryption Standard): Most widely used; secure and efficient	**RSA (Rivest-Shamir-Adleman):** Based on factoring large prime numbers. Used for secure key exchange and digital signatures
3DES (Triple DES): More secure than DES but slower	**ECC (Elliptic Curve Cryptography):** Provides strong security with smaller keys. Efficient for lightweight devices
Blowfish: Flexible and fast, good for software applications	**DSA (Digital Signature Algorithm):** Used for authentication
ChaCha20: Optimized for mobile and embedded systems	**ElGamal:** Offers encryption and signing; used in PGP

2.2 Decryption

Decryption is the process of converting ciphertext back into its original, readable format (plaintext) using the correct decryption key and algorithm.

2.3 Key Management

Key management involves generating, distributing, storing, using, and disposing of cryptographic keys in a secure manner. It is essential to the effectiveness of encryption.

- **Key Management Activities:**
  - Key generation
  - Secure key exchange (especially critical for symmetric encryption)
  - Secure storage
  - Key rotation and expiration
  - Key disposal

- **Key Derivation Functions (KDFs):**
  - PBKDF2 (Password-Based Key Derivation Function 2)
  - Argon2: Memory-hard function, highly secure and efficient for password hashing

2.4 Hashing

Hashing is a one-way process that converts readable data into a fixed-size, unique string of characters called a hash value or digest.

- **Key Features:**
  - Deterministic: Same input always produces same output
  - Fast computation
  - Pre-image and collision resistant
  - Avalanche effect
- **Common Algorithms:**
  - SHA-256, SHA-3
  - BLAKE2, Whirlpool
  - Argon2 (for passwords)
  - MD5, SHA-1 (deprecated)

3 Functional Flow

3.1 File Selection

User selects a file of any type for processing.

3.2 Operation Selection

User chooses to:

- Encrypt
- Decrypt
- Generate a hash

3.3 Key Input

Required for encryption/decryption:

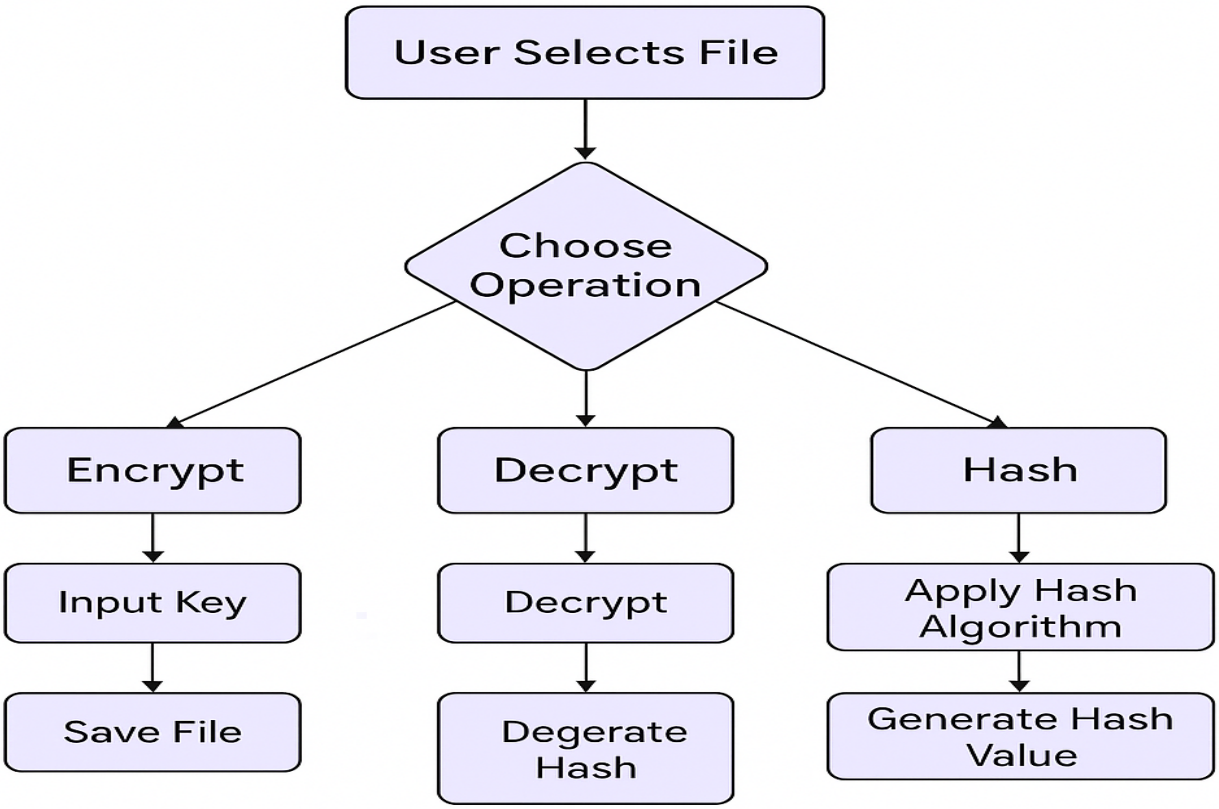
- **Symmetric:** Same key for both operations
- **Asymmetric:** Public key for encryption, private key for decryption
- Not required for hashing.

3.4 Process Execution

- **Encryption:** File is read, encrypted (e.g., AES-256), and saved with a secure extension.
- **Decryption:** Encrypted file is read, decrypted with the provided key, and the original file is restored.
- **Hashing:** File is read, hashed (e.g., SHA-256), and a digest is generated.

3.5 Output

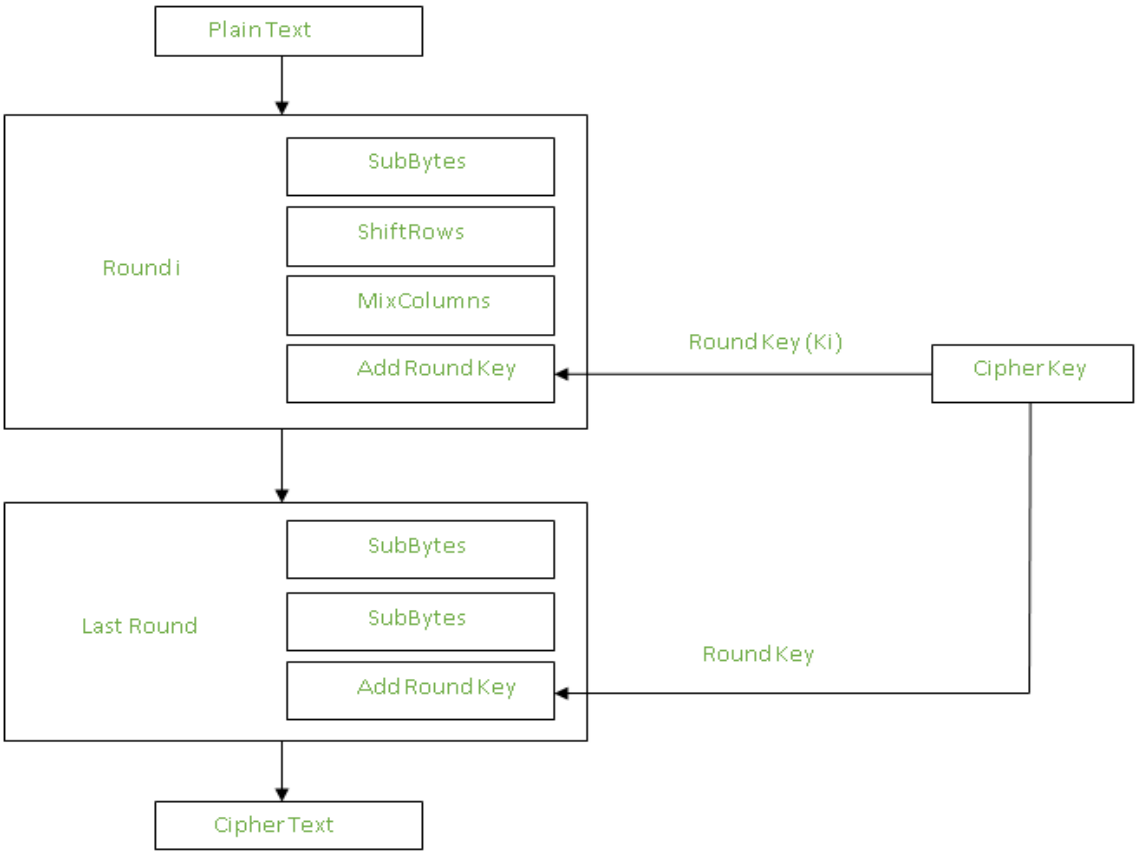
Encrypted file, decrypted file, or hash value generated.



4 Mathematical foundations

4.1 AES (Advanced Encryption Standard/Symmetric Encryption)

- Based on substitution-permutation network.
- Operates on 128-bit blocks: That means it takes 128 bits as input and outputs 128 bits of encrypted cipher text.
- Key sizes: 128, 192, or 256 bits.
- The encryption process consists of multiple rounds depending on key size (10 for 128-bit keys, 12 for 192-bit, and 14 for 256-bit).
- Each round consists of:
  - **SubBytes:** A non-linear substitution step.
  - **ShiftRows:** A transposition step.
  - **MixColumns:** A mixing operation.
  - **AddRoundKey:** Each byte of the state is combined with a round key.



4.2 RSA (Rivest-Shamir-Adleman/Asymmetric Encryption)

- Based on modular arithmetic and large prime numbers.
- Key generation:

1. Generate two large primes p and q.
2. Compute $n = p \times q$ and $\phi(n) = (p-1)(q-1)$.
3. Choose public exponent e (e.g., 65537).
4. Compute private exponent d where $e \times d \equiv 1 \mod \phi(n)$.

- **Public key:** (e, n)
- **Private key:** (d, n)
- **Encryption:** C = M^e mod n
- **Decryption:** M = C^d mod n
- The idea of RSA is based on the fact that it is difficult to factorize a large integer. The Public Key is (n, e), where n and e are publicly known, while the Private Key is (n, d). Since only the receiver knows the value of d, only they can decrypt the message. **But is it possible to find the value of d using n and e?** We know that **(d * e) ≡ 1 mod** $\Phi(n)$, so if we can calculate the value of $\Phi(n)$, we can find the value of d. But $\Phi(n)$ **= (p − 1) * (q − 1).** So, we need the value of p and q. Now, one might think that it's quite easy to find the value of p and q as **n = p * q** and n is already publicly known but RSA Algorithm takes the value of p and q to be very large which in turn makes the value of n extremely large and factorizing such a large value is computationally impossible. Therefore encryption strength lies in the values of **p** and **q**. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024-bit keys could be broken shortly. But till now it seems to be an infeasible task.

4.3 SHA-256 Hashing

- Merkle-Damgård construction.
- Processes 512-bit blocks with compression functions.
- 64 rounds of compression.
- Uses logical functions (Ch, Maj, ,).
- Final output is a 256-bit fixed hash.

5 Existing Solutions

In this section we survey the major file-encryption and hashing tools currently available, grouping them by deployment context (commercial suites, open-source utilities, cloud offerings, development libraries, and OS-native features). For each solution we summarize key characteristics, advantages, and limitations.

5.1 Commercial & Enterprise Software

- **Symantec Endpoint Encryption**
  - **Use case:** Whole-disk and removable media encryption with centralized policy management for large organizations
  - **Advantages:** Scalable, integrates with enterprise key stores, compliance reporting
  - **Limitations:** High licensing costs; complex initial configuration

- **McAfee Complete Data Protection**
  - **Use case:** File- and folder-level encryption, access control, and DLP integration
  - **Advantages:** Granular policy enforcement; unified console for endpoint security
  - **Limitations:** Steep learning curve; may impact endpoint performance
- **AxCrypt**
  - **Use case:** User-friendly file encryption for small businesses and individuals
  - **Advantages:** Simple GUI; AES-256; integrates with popular cloud storage
  - **Limitations:** Requires paid subscription for team features; no hashing/verification tool
- **BitLocker (Windows Pro/Enterprise)**
  - **Use case:** Full-disk encryption via TPM, native to Windows
  - **Advantages:** Transparent to the user; minimal overhead; seamless OS integration
  - **Limitations:** Limited to Pro/Enterprise editions; closed-source; key recovery can be burdensome

5.2 Open-Source Tools

- **VeraCrypt**
  - **Scope:** Full-disk, volume and hidden-container encryption
  - **Advantages:** AES/Serpent/Twofish cascades; plausible deniability via hidden volumes
  - **Limitations:** No cloud-sync integration; initially complex UI for beginners
- **GnuPG (GPG)**
  - **Scope:** OpenPGP-standard encryption & signing; key-management for email and files
  - **Advantages:** RFC4880 compliance; extensive front-end ecosystem; both symmetric & asymmetric modes

  - **Limitations:** Steep CLI learning curve; GUI front-ends vary in quality

- **OpenSSL**
  - **Scope:** Toolkit for SSL/TLS plus general cryptographic operations (enc, hashing, keygen)
  - **Advantages:** Industry-standard; supports broad algorithm suite; commercial-friendly license
  - **Limitations:** Poorly organized docs; risk of misconfiguration; doesn't cover all modern standards

  Example Command: `openssl enc -aes-256-cbc -salt -in file.txt -out file.enc`

- **Cryptomator**
  - **Scope:** Client-side folder encryption for cloud storage
  - **Advantages:** Easy drag-and-drop vaults; cross-platform; open-source
  - **Limitations:** Manual key-backup; file-level only (no full-disk)

5.3 Cloud-Focused Encryption

- **Boxcryptor**
  - **Use case:** End-to-end encryption layer atop Dropbox, Google Drive, OneDrive, etc.
  - **Advantages:** Zero-knowledge design; desktop & mobile apps; shareable encrypted folders
  - **Limitations:** Freemium model caps number of providers; subscription required for teams

- **SpiderOak One Backup**
  - **Use case:** "No-knowledge" encrypted backup & sync service
  - **Advantages:** Strong privacy guarantees; versioning; cross-platform
  - **Limitations:** Slower throughput vs. native clients; paid only

5.4 Developer Libraries & APIs

- **cryptography (Python)**
  - **Algorithms:** AES, ChaCha20, RSA
  - **Advantages:** High-level API; secure defaults; supports modern standards
  - **Limitations:** Slower than native C libraries; dependency-heavy for minimal tasks

```python
from cryptography.fernet import Fernet
key = Fernet.generate_key()
cipher = Fernet(key)
encrypted = cipher.encrypt(b"secret data")
decrypted = cipher.decrypt(encrypted)
```

- **Bouncy Castle (Java/C#)**
  - **Algorithms:** AES, RSA, SHA, and more
  - **Advantages:** Extensive algorithm coverage; mature and actively maintained
  - **Limitations:** Verbose API; steep learning curve for newcomers
- **crypto (Node.js module)**
  - **Algorithms:** AES, RSA, HMAC
  - **Advantages:** Built-in to Node.js; efficient for web tokens and REST APIs
  - **Limitations:** Limited configurability; basic cryptographic operations only
- **System.Security.Cryptography (.NET)**
  - **Algorithms:** AES, RSA, SHA
  - **Advantages:** Native integration with Windows; excellent documentation
  - **Limitations:** Primarily Windows-centric; limited flexibility across platforms

- **libsodium (NaCl)**
  - **Algorithms:** Xsalsa20, Poly1305, Curve25519
  - **Advantages:** Modern, high-speed primitives; simple API; strong default practices
  - **Limitations:** Lacks support for some legacy algorithms; steep C-based integration for some users

5.5 OS-Native Encryption Features

- **Windows EFS (Encrypting File System:** Per-file NTFS encryption, transparent in Explorer; limited to NTFS and domain-joined machines, with key-recovery challenges.
- **macOS FileVault:** Full-disk AES-XTS encryption tied to user password; no per-folder control, limited to Apple hardware.
- **Linux LUKS (cryptsetup):** Standard for partition encryption; integrates

Comparison of Existing Solutions

Solution	Type	Encryption	Hashing	Ease of Use	Cost
VeraCrypt	Desktop	AES, Serpent, Twofish	No	Medium	Free
AxCrypt	Desktop	AES-256	SHA-1	High	Freemium
GPG	CLI	Multiple	SHA-256	Low	Free
OpenSSL	CLI	Multiple	Multiple	Low	Free
HashTab	Desktop	No	Multiple	High	Free
BitLocker	OS	AES	No	Medium	Included
Cryptomator	Desktop	AES	SHA-256	Medium	Freemium

6 Standards and requirements

These protocols ensure cryptographic robustness and interoperability.

6.1 Relevant Standards

Several established standards guide the secure implementation of encryption, decryption, and hashing techniques:

- **FIPS PUB 197:** Defines the Advanced Encryption Standard (AES), a widely adopted symmetric encryption algorithm.
- **FIPS 180-4:** Specifies the Secure Hash Standard (SHA), including SHA-1, SHA-2, and SHA-3 families.
- **PKCS #1:** Establishes the RSA Cryptography Standard, detailing the implementation of RSA encryption and digital signatures.
- **NIST SP 800-38:** Provides recommendations for block cipher modes of operation (e.g., CBC, GCM) to ensure secure encryption practices.

6.2 Security Requirements

To ensure robust and resilient cryptographic operations, the following security requirements must be addressed:

- **Key Management:** Secure generation, storage, and exchange of cryptographic keys to prevent unauthorized access.
- **Protection Against Known Attacks:** Implementation must mitigate risks from attacks such as padding oracle attacks, timing attacks, and brute-force attacks.
- **Secure Implementation:** Systems must be designed to resist side-channel attacks (e.g., timing, power analysis).
- **Proper Algorithm Configuration:** Careful selection of modes of operation (e.g., GCM for AES) and proper padding schemes are essential to prevent vulnerabilities.

7 Conclusion

In conclusion, file encryption, decryption, and hashing are essential components of modern cybersecurity, providing critical protection for sensitive data. Our application serves as a practical, secure, and educational tool for managing files, demonstrating the real-world application of cryptographic techniques. This deliverable has defined the core cryptographic concepts, outlined the functional workflows, and surveyed leading tools and libraries, while establishing a robust threat model and comparing the advantages and limitations of different approaches. Moving forward, the next steps will focus on refining the system design, selecting appropriate algorithms, ensuring secure implementation, and validating the prototype to strengthen its practical application in securing digital information.

8 Bibliography

- **NIST. (2001). FIPS PUB 197:** Advanced Encryption Standard (AES).
- **Rivest, R., Shamir, A., & Adleman, L. (1978):** A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.
- **NIST. (n.d.). SP 800-185:** SHA-3 Standard.
- **Bruce Schneier. (2017):** Applied Cryptography.
- **Stallings, W. (2020):** Cryptography and Network Security: Principles and Practice.
- **OpenSSL Documentation:** Available at: https://www.openssl.org/docs/