A PROJECT REPORT ON

"SHADOWCRYPT: PROACTIVE RANSOMWARE PROTECTION BY EMPLOYING CAMOUFLAGE AND CONCEALMENT"

Submitted to

OSMANIA UNIVERSITY, Hyderabad

in partial fulfillment of the requirements for the award of degree

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

(IoT, CYBERSECURITY INCLUDING BLOCKCHAIN TECHNOLOGY)

Submitted by

ABDUL SAMAD	**(161021749020)**
MOHAMMED ABDUL RAQEEB	**(161021749035)**
MOHAMMED FASIUDDIN ARSALAAN	**(161021749041)**

Under the guidance of

Ms. SARIYA JABEEN DURIYA

Assistant Professor, Department of CSE-(IoT)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(IoT, CYBERSECURITY INCLUDING BLOCKCHAIN TECHNOLOGY)

NAWAB SHAH ALAM KHAN COLLEGE OF ENGINEERING & TECHNOLOGY

Malakpet, Hyderabad.

2024-2025

Nawab Shah Alam Khan
COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous)

Approved by AICTE | Affiliated to Osmania University & SBTET | Accredited by NBA (IT, ME), NAAC 'A'

CERTIFICATE

This is to certify that the Major Project on **"SHADOWCRYPT: PROACTIVE RANSOMWARE PROTECTION BY EMPLOYING CAMOUFLAGE AND CONCEALMENT"** is being submitted by the following students:

ABDUL SAMAD	**(161021749020)**
MOHAMMED ABDUL RAQEEB	**(161021749035)**
MOHAMMED FASIUDDIN ARSALAAN	**(161021749041)**

They have presented the project work during the academic year 2024–2025 in partial fulfillment of the requirements for the award of **BACHELOR OF ENGINEERING** in **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (IoT, CYBER SECURITY & INCLUDING BLOCKCHAIN TECHNOLOGY).** This is a bonafide record work carried out by them under our guidance and supervision. The results of investigation enclosed with this report have been verified and found to be satisfactory.

Internal Guide **Project Coordinator** **Head of Department**

External Examiner

DECLARATION

We hereby declare that the project work entitled **“SHADOWCRYPT: PROACTIVE RANSOMWARE PROTECTION BY EMPLOYING CAMOUFLAGE AND CONCEALMENT”** submitted to the Department of **COMPUTER SCIENCE AND ENGINEERING (IoT, CYBER SECURITY & INCLUDING BLOCKCHAIN TECHNOLOGY) of NAWAB SHAH ALAM KHAN COLLEGE OF ENGINEERING AND TECHNOLOGY**, affiliated to Osmania University, Hyderabad in the partial fulfillment of the requirement for award of the degree in BACHELOR OF ENGINEERING in CSE is a bonafide work done by the undersigned.

ABDUL SAMAD	**(161021749020)**
MOHAMMED ABDUL RAQEEB	**(161021749035)**
MOHAMMED FASIUDDIN ARSALAAN	**(161021749041)**

ACKNOWLEDGEMENT

The satisfaction that accomplished completion of any task would be incomplete without the mention of people who made it possible and whose encouragement and guidance has been a source of inspiration through the source of the project.

We express our profound sense of gratitude to **Dr. Syed Abdul Sattar, Principal** for his constant support.

We would like to express our sincere thank to **Dr. Mohammad Sanaullah Qaseem, Prof. of CSE, Dean Computer Studies,** for sharing his experience and valuable knowledge to enhance our real time learning experience.

We would like to express our sincere thanks to **Ms. Syeda Azkia, Professor and Head of IoT Department** and our Internal Guide **Ms. Sariya Jabeen Duriya,** for their earnest efforts and timely suggestions that motivated us to come out with satisfactory project work.

We thank everybody who directly or indirectly played a vital role in finishing our project work with less difficulty.

ABDUL SAMAD	**(161021749020)**
MOHAMMED ABDUL RAQEEB	**(161021749035)**
MOHAMMED FASIUDDIN ARSALAAN	**(161021749041)**

ABSTRACT

Ransomware, particularly Ransomware as a Service (RaaS), has become a significant cybersecurity threat, often bypassing traditional detection and mitigation methods. Our Major project presents a proactive defense strategy that minimizes damage even after ransomware execution. By analyzing ransomware families like LockBit, we developed a file concealment technique that strategically hides critical files in directories typically ignored by ransomware while maintaining accessibility through link files. To further enhance security and usability, we integrate an encrypted database and a linker mechanism. Experimental validation using real ransomware samples demonstrates the approach's effectiveness in preventing file compromise, offering a cost-efficient and practical solution for robust ransomware protection.

Index Terms:- Ransomware, Cybersecurity, Ransomware as a Service (RaaS), Proactive Defense Strategy, File Concealment Technique, Link Files, Encrypted Database, Linker Mechanism, LockBit, Damage Minimization

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Introduction

The exponential rise of cybercrime has drastically transformed the way individuals and organizations think about digital security. Among the most pervasive and damaging forms of malware is ransomware, which encrypts or destroys files and demands payment for their restoration. This threat has evolved into a commercial criminal model known as **Ransomware as a Service (RaaS),** enabling even non-technical attackers to launch sophisticated attacks using pre-built malware kits.

Traditional methods of defense—such as antivirus software, firewalls, and heuristic file monitoring—are increasingly insufficient. These tools often rely on detection after compromise, by which time the damage is done. Many ransomware variants avoid or delay detection by specifically ignoring certain system directories, skipping non-user folders, or delaying payload execution.

In this context, a proactive strategy becomes essential—one that secures files before an attack and minimizes the damage post-compromise. ShadowCrypt is designed with this principle in mind. It is a Python-based file hiding and recovery system that uses AES-256 encryption, obfuscation, shortcut-based access, and secure metadata mapping to shield sensitive files from ransomware.

ShadowCrypt conceals important files in less-accessed system directories and replaces them with encrypted mappings and .lnk shortcut files. These links are embedded with hashed arguments and can only be decoded by the tool itself. The system supports both command-line interface (CLI) execution for technical users and a one-click `*.exe*` installer for convenient deployment by non-technical users.

1.2 Problem Statement

Despite advancements in cybersecurity, ransomware continues to outpace defensive mechanisms. Many modern ransomware strains:

- Scan and encrypt all visible files in common user directories.
- Avoid system or temporary directories where critical files could be preemptively stored.
- Can disable system restore points or backup services.

Traditional file-hiding techniques are:

- Easily reversed by attackers or even curious users.
- Detectable via forensic or manual inspection.
- Lacking in metadata protection, allowing reverse tracing.

Moreover, most file-protection tools do not consider the post-infection scenario where the malware is already running.

ShadowCrypt aims to fill this gap by:

- **Hiding files** in places ransomware usually skips.
- **Encrypting mappings** so the true locations are undiscoverable.
- **Allowing access** only via hashed, secure .lnk files.
- Providing a fallback for **file recovery** even after infection.

1.3 Objectives

The ShadowCrypt project is built with the following key objectives:

1. To develop a ransomware-resilient file protection system through intelligent file hiding that reduces visibility and exposure of critical files.
2. To use obscure, non-indexed system directories for storing hidden files, minimizing their likelihood of being discovered by ransomware scanning mechanisms.
3. To maintain user access to hidden files via securely generated shortcut files that invoke a linker module, verifying identity using a hash before access is granted.
4. To offer dual usability — supporting command-line usage, executable deployment, and integration with the Windows right-click context menu for both hiding and recovery.
5. To securely manage metadata and access mappings using encrypted JSON structures that store associations between original and hidden file paths without reliance on traditional vaults or containers.
6. To implement a simple recovery mechanism that restores hidden files to their original paths while removing shortcut traces and clearing mapping records.

1.4 Scope of the Project

This project aims to build a complete file concealment and access framework under the name ShadowCrypt, designed to proactively safeguard critical files from ransomware threats. The scope includes:

1. **Hiding of files** through secure backend logic that removes them from typical ransomware target zones and disguises their identity using safe extensions and locations.

2. A **shortcut-based access mechanism** that allows users to open hidden files transparently through .lnk files pointing to a verified launcher script.
3. A **recovery process** that safely restores hidden files to their original location and cleans up associated shortcuts and mappings.
4. **Encrypted mapping and configuration management** using lightweight JSON files (mapping.db, app_paths.json) to store associations and application paths.
5. Command-line interface support (e.g., --hide, --recover)
6. **Right-click shell integration** for hiding and recovering files from the context menu.
7. Ensuring persistence and safety even during ransomware execution, system reboots, or user interface failure.
8. Windows-based .**exe packaging**

1.5 Infrastructure Requirements

ShadowCrypt is designed to be lightweight and functional across standard Windows environments. The following hardware and software requirements are necessary:

1.5.1 Hardware Requirements

Component	**Minimum Specification**
Processor	Intel Core i3 or AMD Ryzen 3
RAM	4 GB
Storage	100 MB of free space

Component	Minimum Specification
Display	Standard resolution (1024×768 or higher)
OS Platform	Windows 10 or above

1.5.2 Development Dependencies

(For developers or CLI-based usage)

Software Component	Version/Details
Python	3.12+
OS Support	Windows OS (mandatory)
External Modules	pycryptodome, pylnk3, filelock, argparse, getpass, ctypes, json, hashlib, shutil, argparse
Executable Support	PyInstaller, InstallForge
Applications	LibreOffice, VLC, Notepad, 7-Zip

1.5.3 End-User Setup Requirements

(For users using the .exe installer)

Component	Specification
Installer	ShadowCryptSetup.exe
Output	ShadowCrypt.exe for one-click use
UI Interface	Command-line based executable
Admin Rights	Required

1.6 Challenges Faced

The development of ShadowCrypt was accompanied by several conceptual and technical challenges, particularly in aligning security with usability:

1. **Secure file movement:** Relocating files securely without corrupting or exposing them.
2. **Directory selection:** Identifying paths least likely to be scanned by ransomware.
3. **Shortcut argument encryption:** Embedding hash-based paths in .lnk files securely.

4. **Metadata protection:** Ensuring that no trace of the original file path is visible to attackers.

5. **Executable packaging:** Creating a lightweight, user-friendly .exe that mirrors the CLI tool's functionality.

6. **Non-redundant recovery logic:** Preventing duplication, overwriting, or data loss due to user errors.

7. **Test mode integration:** Allowing safe testing environments without touching real files.

2. LITERATURE SURVEY

2.1 Existing System

2.1.1 Overview of Conventional File Hiding Methods

File hiding has long been used as a lightweight way to reduce file visibility without offering real protection. These techniques are often bypassed by malware and even casual users with basic system knowledge. While useful for organizing files or preventing accidental deletion, they are not suitable for security-critical scenarios, especially those involving ransomware.

2.1.1.1 Hidden Attributes and Access Restrictions

The most basic method of hiding a file in Windows involves marking it as hidden or system-protected. This can be done via the **attrib** command:

```
attrib +h +s secret.txt
```

This hides the file from File Explorer — unless the user has enabled "Show hidden files" and "Show protected system files".

Limitations:

- Easily reversed with simple UI toggles.
- Malware ignores visibility settings during enumeration.
- No encryption or access control.

This makes hidden attributes a convenience, not a defense.

2.1.1.2 Directory Obfuscation and Naming Tricks

This method involves burying files deep inside folder hierarchies or renaming them to appear unimportant:

Example path:

C:\ProgramData\System\Update\Temp\Logs\kernel32.dll

Example filename:

system_log.tmp for an actual .docx file.

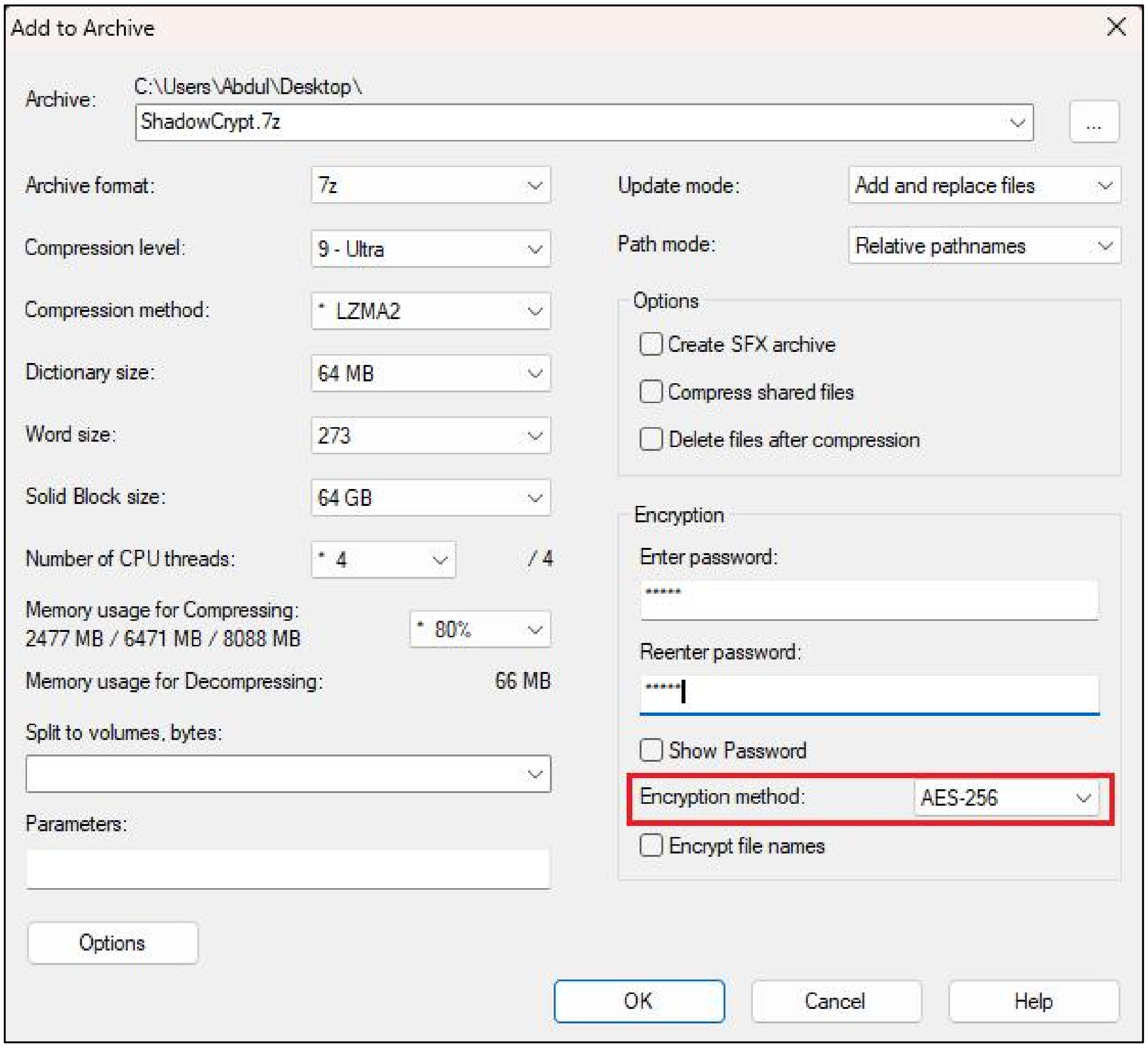
Limitations:

- File metadata is still exposed (size, timestamp, extension).
- Search indexing and some backup tools still find the files.
- Smart malware or admin users can identify disguised file types.

This method relies on confusing the user or attacker, not protecting the content.

2.1.1.3 File Compression and Archiving Tools

Tools like 7-Zip, WinRAR, and Windows ZIP allow files to be stored in password-protected archives. These archives may optionally encrypt both content and filenames (e.g., using AES-256 in 7-Zip).



Add to Archive

Archive: C:\Users\Abdul\Desktop\ ShadowCrypt.7z

Archive format: 7z

Compression level: 9 - Ultra

Compression method: * LZMA2

Dictionary size: 64 MB

Word size: 273

Solid Block size: 64 GB

Number of CPU threads: * 4 / 4

Memory usage for Compressing: 2477 MB / 6471 MB / 8088 MB * 80%

Memory usage for Decompressing: 66 MB

Split to volumes, bytes:

Parameters:

Update mode: Add and replace files

Path mode: Relative pathnames

Options

- [ ] Create SFX archive
- [ ] Compress shared files
- [ ] Delete files after compression

Encryption

Enter password: *****

Reenter password: *****

- [ ] Show Password

Encryption method: AES-256

- [ ] Encrypt file names

Figure 1: Creating password protected archive using AES-26 Encryption method

Limitations:

- Archive files are highly visible and often suspicious.
- Weak or reused passwords are easy to crack.
- Files are exposed once extracted.
- Some ransomware encrypts archive files themselves, making recovery impossible

.

Technique	**Method Type**	**Visibility**	**Encryption**	**Security Level**
Hidden Attributes	UI-level hiding	Hidden	✕	Low
Directory Obfuscation	Path/name confusion	Visible	✕	Low-Medium
Archive Compression	Password protection	Visible	✓ (optional)	Medium

2.1.2 Shortcut-Based Access in Windows

In a traditional Windows environment, .lnk shortcut files are used to create quick access points to documents or programs. ShadowCrypt innovates on this concept by using shortcuts to **launch a Python-based linker** instead of directly pointing to the original file.

Each shortcut created by ShadowCrypt is configured to:

- Launch linker.py (or the bundled linker.exe)

- Pass a secure hash as an argument, which identifies the hidden file through a lookup process

The linker then:

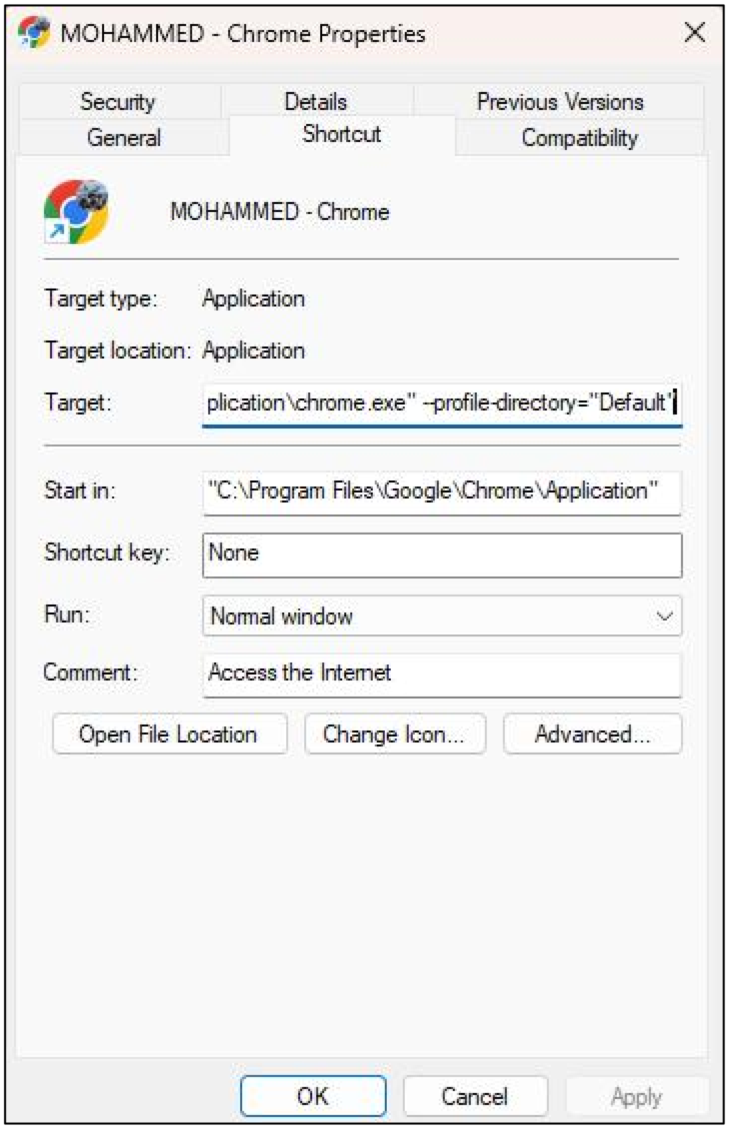
1. Decrypts the contents of mapping.db.
2. Matches the passed hash with a stored entry in hash_table.
3. Resolves the hidden file path from mapping_table.
4. Determines the correct program for the file type using app_paths.json.
5. Launches the hidden file via the appropriate application, fully bypassing any direct file exposure.

This approach ensures that files remain **physically hidden and logically inaccessible,** yet are **seamlessly accessible** to the user via secure shortcut triggers.

2.1.2.1 Anatomy of a .lnk File

A .lnk file contains several components:

- **Target Path:** The absolute path of the file or executable to open.
- **Start In:** The working directory when the target is launched.
- **Command-line Arguments:** Additional parameters passed at runtime.
- **Icon Reference:** Optional visual customization.
- **Window Mode:** Normal, minimized, or maximized.


MOHAMMED - Chrome Properties

Security | Details | Previous Versions
General | Shortcut | Compatibility

MOHAMMED - Chrome

Target type: Application

Target location: Application

Target: plication\chrome.exe" --profile-directory="Default"

Start in: "C:\Program Files\Google\Chrome\Application"

Shortcut key: None

Run: Normal window

Comment: Access the Internet

Figure 2: Components of a .lnk file

2.1.2.2 Use Cases in Security Contexts

Shortcuts are commonly used in:

- Portable software launchers (like PortableApps)
- Installer scripts that call .bat, .py, or .exe files
- Access abstraction layers, where direct file access is restricted

In the context of file security:

- Shortcuts can act as decoys or access gates
- They allow access to hidden or relocated files using embedded logic
- Command-line arguments can pass identifiers or hashes, enabling selective access via a verifying program

Example:

Target: *C:\Program Files\ShadowCrypt\db\ShadowCrypt.exe*

Arguments: *link --hash 8af0e21cf4ec9f91d394c8…*

This allows ShadowCrypt to:

- Validate the hash
- Locate the hidden file via an encrypted mapping
- Decrypt and open it without exposing its real path

2.1.2.3 Limitations of Native Shortcut Security

Despite their flexibility, .lnk files have no native encryption. Security gaps include:

- **Direct reverse-engineering:** Tools like LnkParser or Windows PowerShell can extract target paths.
- **No access control**: Anyone can open a shortcut unless wrapped with external logic.
- **Metadata leakage:** Timestamps and link arguments may expose usage patterns.

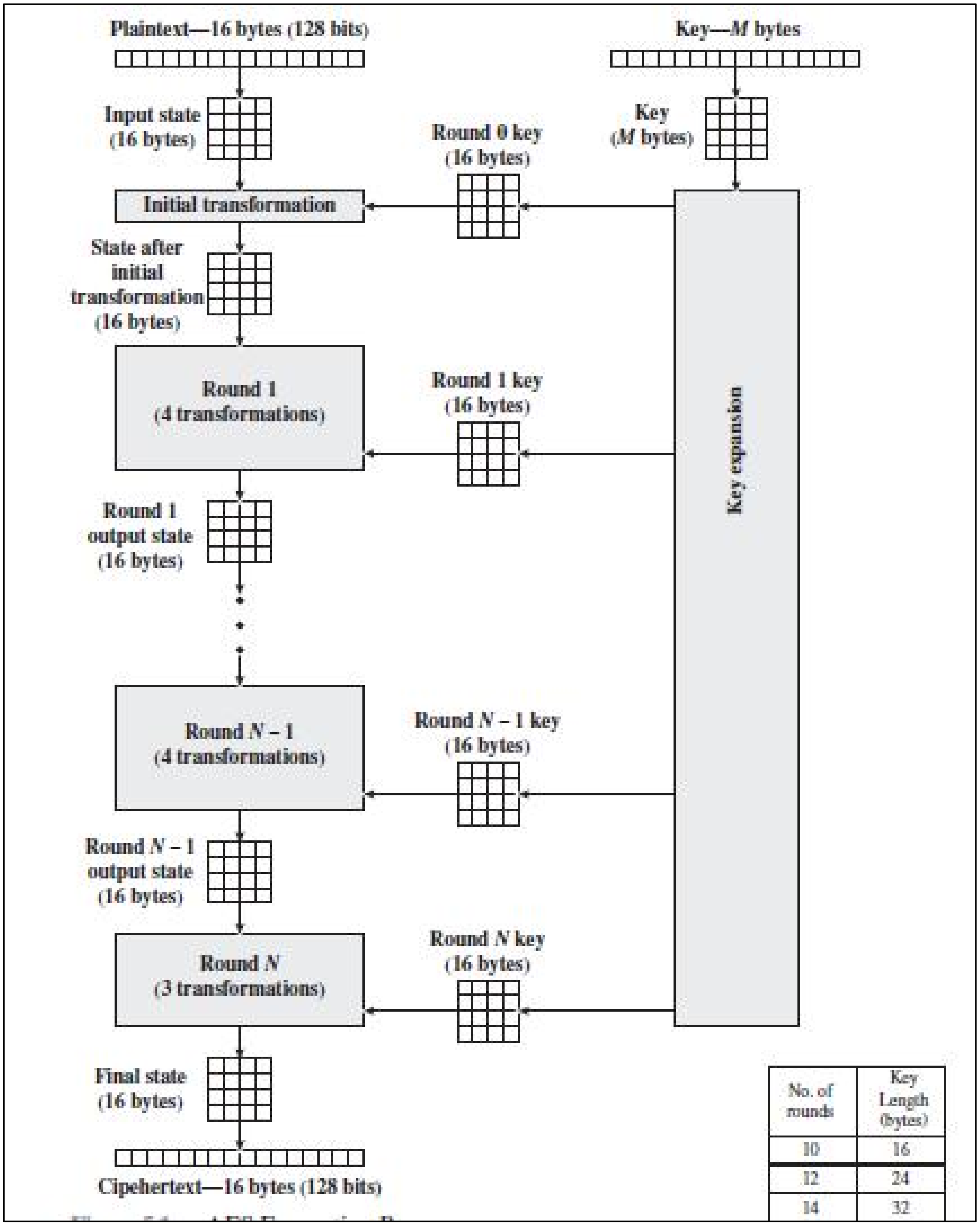
In short, .lnk files are powerful when secured externally, as in ShadowCrypt — but are not inherently secure.

2.1.3 Encryption Standards in Practice

In today's digital environment, where data theft, surveillance, and ransomware attacks are widespread, encryption has become an indispensable mechanism for ensuring **confidentiality, integrity, and resilience.** Among the numerous encryption algorithms available, the **Advanced Encryption Standard (AES)** stands out as the most widely adopted symmetric block cipher, offering a robust balance of **performance, security, and simplicity.**

2.1.3.1 Overview and Significance of AES

AES is a symmetric-key encryption algorithm standardized by the National Institute of Standards and Technology (NIST) in 2001 under the FIPS PUB 197 specification. It replaced the outdated DES (Data Encryption Standard) and has since become the cornerstone of secure data encryption across government, enterprise, and consumer applications.


Plaintext—16 bytes (128 bits)
Key—M bytes
Input state (16 bytes)
Key (M bytes)
Round 0 key (16 bytes)
Initial transformation
State after initial transformation (16 bytes)
Round 1 (4 transformations)
Round 1 key (16 bytes)
Key expansion
Round 1 output state (16 bytes)
Round N – 1 (4 transformations)
Round N – 1 key (16 bytes)
Round N – 1 output state (16 bytes)
Round N (3 transformations)
Round N key (16 bytes)
Final state (16 bytes)
Cipehertext—16 bytes (128 bits)

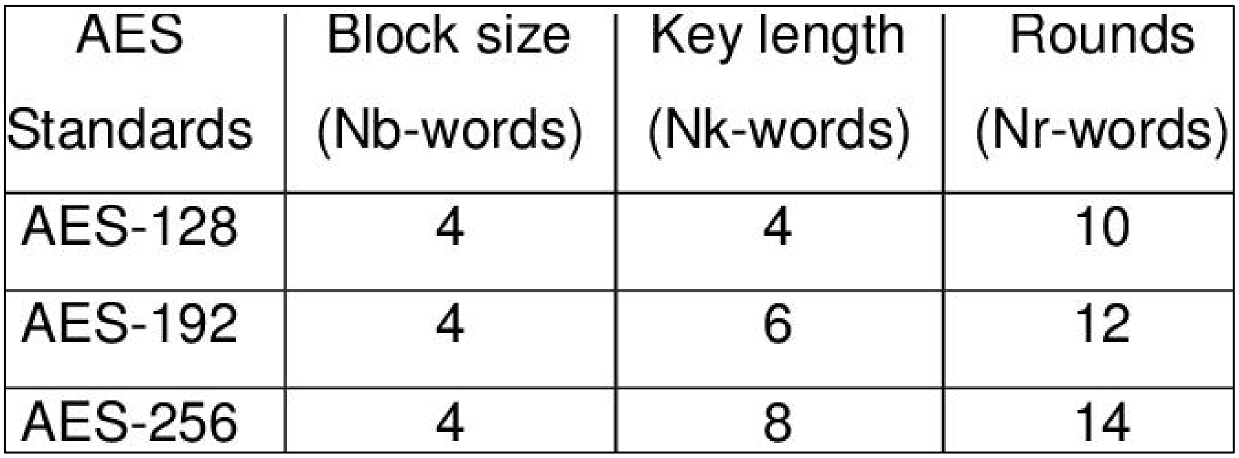
No. of rounds	Key Length (bytes)
10	16
12	24
14	32

Figure 3: AES Encryption Process

Why AES is preferred:

- Fast and efficient in both hardware and software implementations
- Resistant to all known practical cryptographic attacks
- Free of patents and widely supported
- Flexible key length (128, 192, 256 bits)

AES is the default choice for encrypting files, securing communication channels (e.g., HTTPS), and protecting sensitive databases — making it highly suitable for applications like ShadowCrypt, where **sensitive metadata and path information must be protected** even if the system is compromised.



AES Standards	Block size (Nb-words)	Key length (Nk-words)	Rounds (Nr-words)
AES-128	4	4	10
AES-192	4	6	12
AES-256	4	8	14

Figure 4: AES Block Cipher

2.1.3.2 AES Structure and Internal Operations

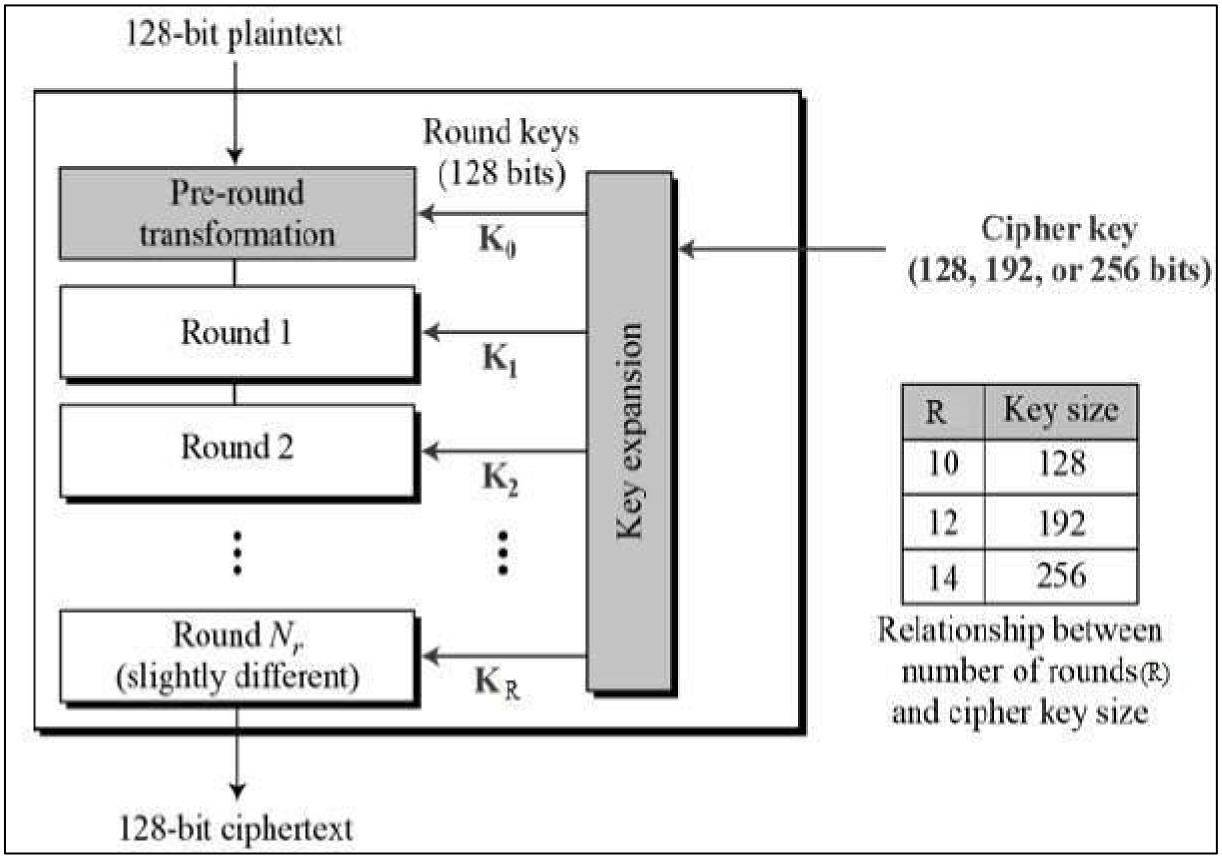
AES is a block cipher, meaning it processes data in fixed-size chunks — specifically, 128-bit blocks. It performs a series of transformation rounds that depend on the key size:

Key Size	**Number of Rounds**
128 bits	10
192 bits	12
256 bits	14

Each round applies multiple operations:

1. **SubBytes** – Byte-by-byte substitution using an S-box (non-linear transformation)
2. **ShiftRows** – Row-wise permutation
3. **MixColumns** – Column-wise mixing of data (diffusion)
4. **AddRoundKey** – XOR with the round key derived from the original key

The first and final rounds have slight variations (no MixColumns in the final round).



R	Key size
10	128
12	192
14	256

Figure 5: AES General Scenario

2.1.3.3 Key Generation and Expansion

AES uses a single secret key (symmetric encryption) that must be known both to the encrypting and decrypting parties. The original key is expanded into multiple round keys through a key schedule process.

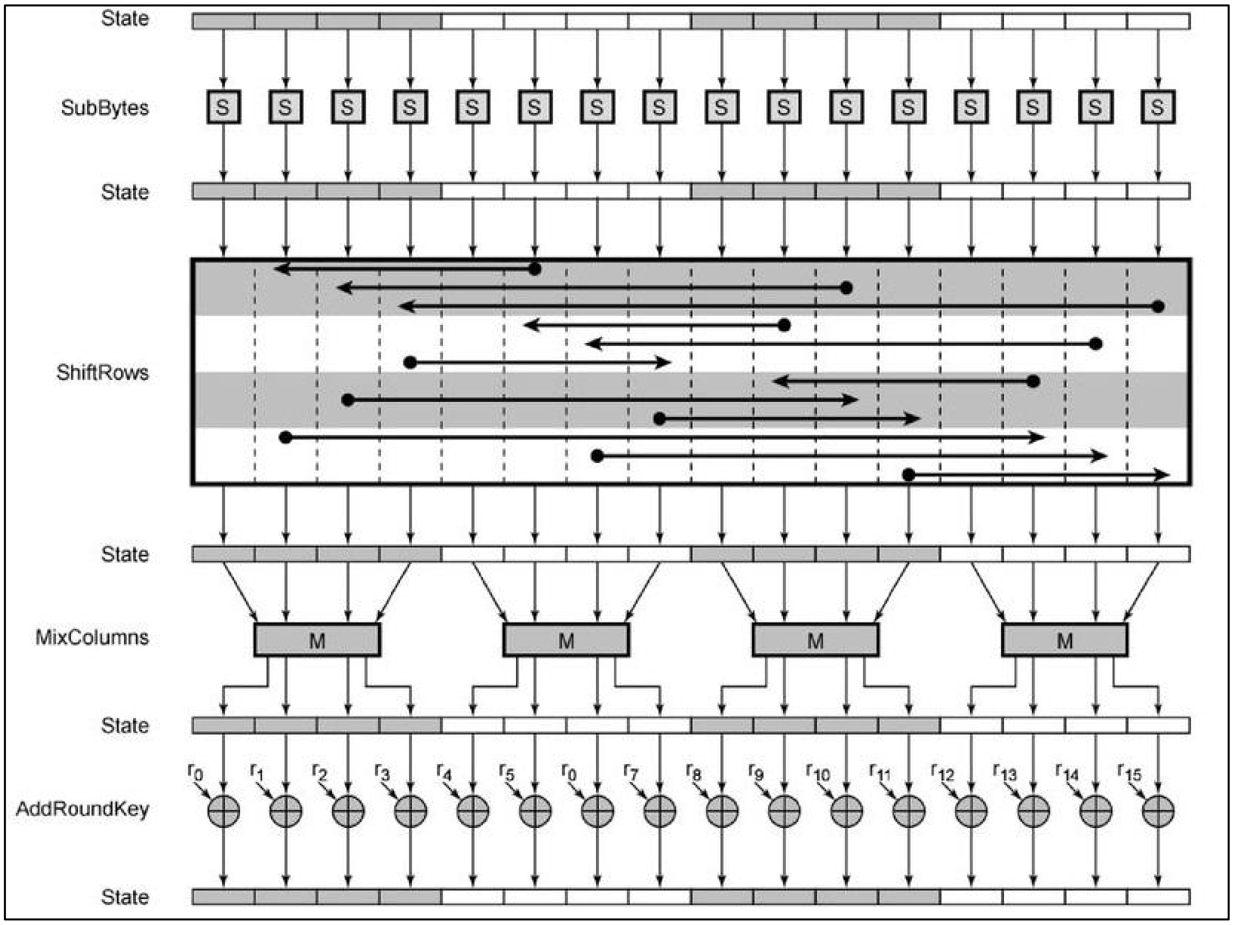

State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
r_0 r_1 r_2 r_3 r_4 r_5 r_0 r_7 r_8 r_9 r_{10} r_{11} r_{12} r_{13} r_{14} r_{15}
State

Figure 6: Mechanism in AES Encryption Round

In ShadowCrypt:

- A **user-provided password** is first hashed using **SHA-256** to derive a consistent 256-bit key, regardless of input length.
- This key is then used in AES-256 encryption (14 rounds).

Benefits of Using SHA-256 for Key Derivation:

- Converts variable-length passwords to fixed-length secure keys
- Adds entropy to low-strength passwords
- Enables reproducible and deterministic key generation

2.1.3.4 Encryption and Decryption Process

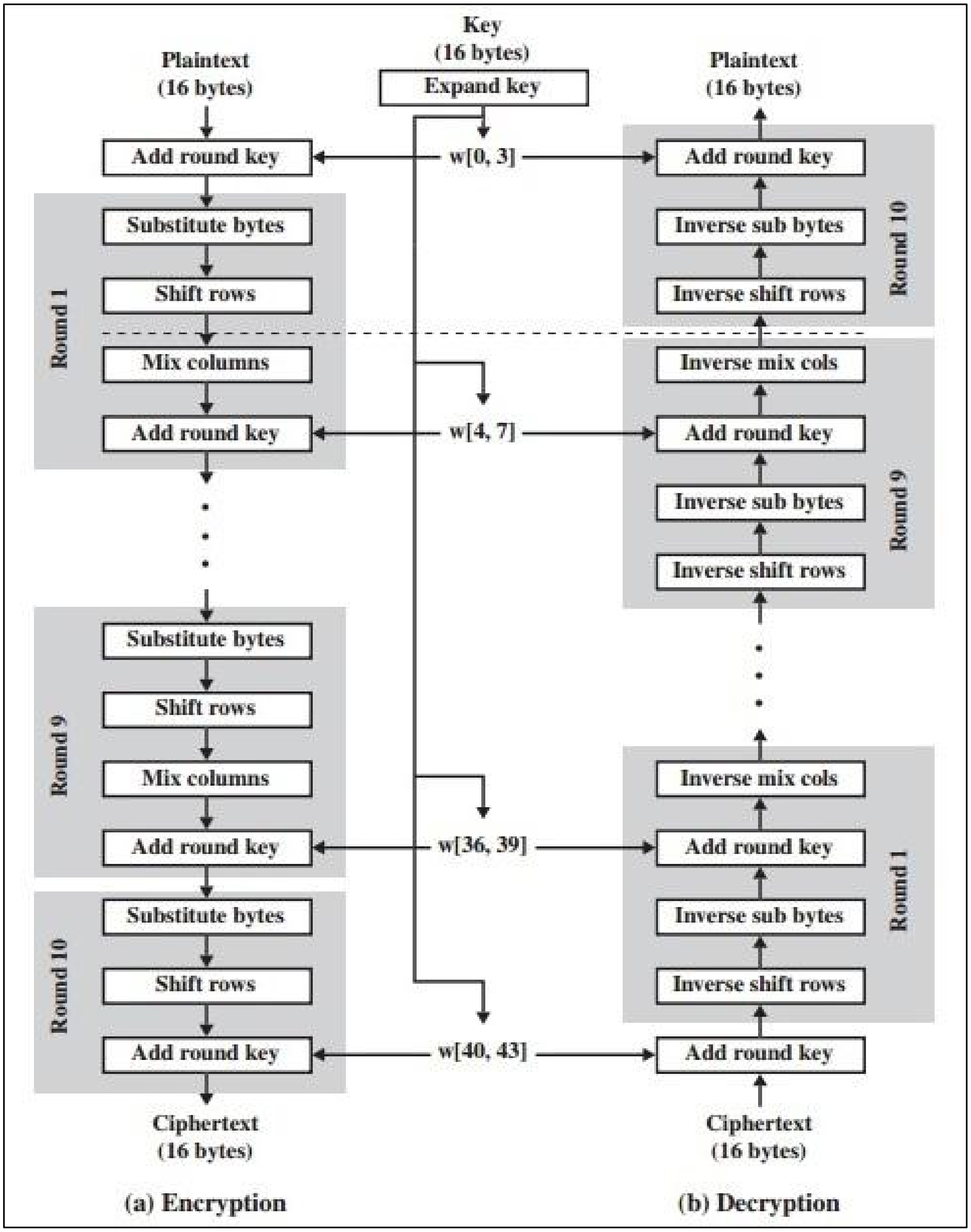


Figure 7: Stages in AES Encryption and Decryption

ShadowCrypt uses AES in CBC (Cipher Block Chaining) mode, which ensures that the same plaintext block encrypted twice produces different ciphertexts — increasing resistance to pattern analysis.

Encryption in CBC Mode:

1. A random Initialization Vector (IV) is generated.
2. The first plaintext block is XORed with the IV.
3. The result is encrypted with AES to produce the first ciphertext block.
4. Each subsequent plaintext block is XORed with the previous ciphertext, then encrypted.

Decryption in CBC Mode:

- Ciphertext blocks are decrypted first, then XORed with the previous ciphertext (or IV) to recover the original plaintext.

This chaining of blocks makes tampering and predictability much harder, improving confidentiality.

2.1.3.5 Security Strengths of AES

Security Property	**AES (256-bit)**
Brute-force Attack	Computationally infeasible
Known Plaintext	Resists due to non-linearity and S-box

Security Property	**AES (256-bit)**
Differential Crypto	Resistant due to structure
Side-channel Risks	Can be mitigated with padding/timing

AES has been analyzed for over two decades and remains **unbroken** in practice. Its robustness, combined with the proper use of secure modes like CBC, makes it the **gold standard for data encryption.**

2.1.3.6 Limitations and Misuse Risks

While AES is highly secure, its misuse can lead to serious vulnerabilities:

- Reusing IVs in CBC mode compromises block uniqueness.
- Weak or reused passwords lead to weak derived keys.
- Storing keys unprotected on disk or in memory exposes the whole system.
- ECB mode (Electronic Codebook) — often mistakenly used — leaks patterns.

Mitigation in ShadowCrypt:

- Random IVs are generated per encryption cycle.
- Keys are derived from password hashes, not stored directly.
- Mapping and path files are encrypted and disguised as DLLs.

2.1.3.7 Common Use in File Vaults and Lockers

AES encryption is widely used in consumer and enterprise f**ile vaults, folder lockers, and disk encryption software.** These tools typically employ AES (often AES-256) to ensure that sensitive files are **unreadable without authentication**, and often **inaccessible to unauthorized programs** — at least while locked.

Typical AES-Driven File Security Tools:

1. **BitLocker (by Microsoft):**

   Uses AES-XTS mode for full-disk encryption tied to TPM and OS-level credentials. Files are protected when the system is off or hibernated.

2. **VeraCrypt**:

   Open-source alternative that supports AES, Serpent, and Twofish. Allows creation of encrypted containers, hidden volumes, and even full-system encryption.

3. **7-Zip & WinRAR:**

   Support AES-256 encryption for file archives. Useful for storing or transferring encrypted file bundles.

4. **Folder Lock, AxCrypt, Cryptomator:**

   File/folder lockers that encrypt on-demand using AES and other symmetric algorithms.

How AES is Applied:

- **Container-Based:** A virtual disk is created and mounted. All files inside are automatically encrypted/decrypted on-the-fly.

- **File-Level Encryption:** Individual files are encrypted into .aes or .locked formats and require passwords to decrypt.
- **Volume Encryption:** Applies to entire partitions or drives, especially in full-disk encryption scenarios.

Common Limitations:

While these tools are strong in terms of cryptographic protection, they share several weaknesses in real-world usage:

- **Unlocked = Exposed:** When the volume is mounted or decrypted, all files are visible to the system — and to ransomware.
- **No Concealment**: The vault or archive is always visible, even if its contents are locked.
- **Password Dependency:** Single point of access — lose the password, lose the data.
- **No Mapping Obfuscation:** Tools don't encrypt filenames, access methods, or logical relationships between data and access routes unless configured manually.

This motivates ShadowCrypt's unique integration of AES for:

- Obfuscating mapping information
- Decoupling visibility from accessibility
- Securing not just file contents, but also how and where they're accessed

2.1.4 Encrypted File Vaults and Secure Storage Software

Encrypted file vaults and secure storage software have emerged as standard solutions for protecting personal, corporate, and sensitive files. These systems aim to prevent unauthorized access using **encryption, password control, and container-based storage,** offering higher protection than traditional file hiding techniques.

2.1.4.1 BitLocker, VeraCrypt, and Alternatives

These tools differ in features and implementation, but share the core principle: encrypt files or volumes using AES or similar symmetric encryption algorithms.

BitLocker (Windows):

- Encrypts entire system volumes
- Uses AES-XTS mode
- Tied to Windows login credentials and TPM chips
- Transparent to the user once booted — files are decrypted on-the-fly

VeraCrypt:

- Allows creation of encrypted containers or entire drive encryption
- Offers plausible deniability via hidden volumes
- Supports multiple ciphers: AES, Serpent, Twofish
- Open-source and cross-platform

Others:

- Cryptomator (cloud-safe encrypted folders)

- AxCrypt (file encryption with cloud sync)
- Folder Lock (combines encryption with UI-level file hiding)
- NordLocker, Encrypto, etc.

2.1.4.2 Limitations of Full Volume Encryption

Despite strong encryption, full-volume or container-based protection has major usability and security limitations, especially under malware conditions:

- **Always visible:** Encrypted containers are obvious — ransomware can encrypt them again or destroy them entirely.
- **All-or-nothing exposure:** When mounted, all files are visible and accessible — ransomware needs no extra privileges.
- **Heavyweight design:** Disk encryption may be overkill for small sets of critical files.
- **Resource usage:** Full-disk encryption may slow boot or access times on low-end systems.

2.1.4.3 Single Point of Failure Risks

These systems often depend on a master key or password. Consequences include:

- Password loss = Data loss (unless recovery keys exist)
- Container corruption renders all files inaccessible
- No fallback mapping between hidden and original file locations
- No selective decryption — it's all open or all locked

In comparison, ShadowCrypt uses:

- Distributed file references
- Keyed mappings encrypted with AES
- Path indirection and launcher validation
- Optional shortcut interface for secure access

2.1.5 Ransomware and Threat Landscape

Ransomware has evolved into one of the most devastating cyber threats of the 21st century. It doesn’t just disrupt operations — it permanently denies access to critical files unless a ransom is paid. What began as crude extortionware has matured into an organized criminal economy, enabled by stealthy attack techniques and scalable infrastructures like **Ransomware-as-a-Service (RaaS).**

In the context of file protection and concealment, understanding ransomware behavior is essential to identifying the gaps in existing systems and how proactive concealment (as in ShadowCrypt) can offer a new kind of defense.

2.1.5.1 Evolution of Ransomware

Ransomware has undergone multiple stages of evolution, increasing in complexity, scalability, and impact:

First Generation (2005–2012) – Simple Lockers

- Mostly screen lockers or fake police warnings.
- Prevented users from accessing the desktop or apps.

- Often had hardcoded decryption keys or were easily removed.

Second Generation (2013–2016) – Encryption-Based Threats

- Adopted real cryptographic algorithms (RSA, AES).
- Encrypted files with unique keys per victim.
- **Examples**: CryptoLocker, TeslaCrypt, TorrentLocker.

Third Generation (2016–2020) – Network-Wide Propagation

- Spread via lateral movement (e.g., EternalBlue exploit in WannaCry).
- Targeted entire networks and critical infrastructure.
- Introduced time-based deletion, data exfiltration, and boot record attacks.

Modern Ransomware (2020–Present) – Ransomware as a Business

- Multi-pronged attacks: encryption + data theft + extortion.
- Includes dark web communication, cryptocurrency ransoms, and victim shaming.
- **Examples:** LockBit, Conti, REvil, BlackCat.

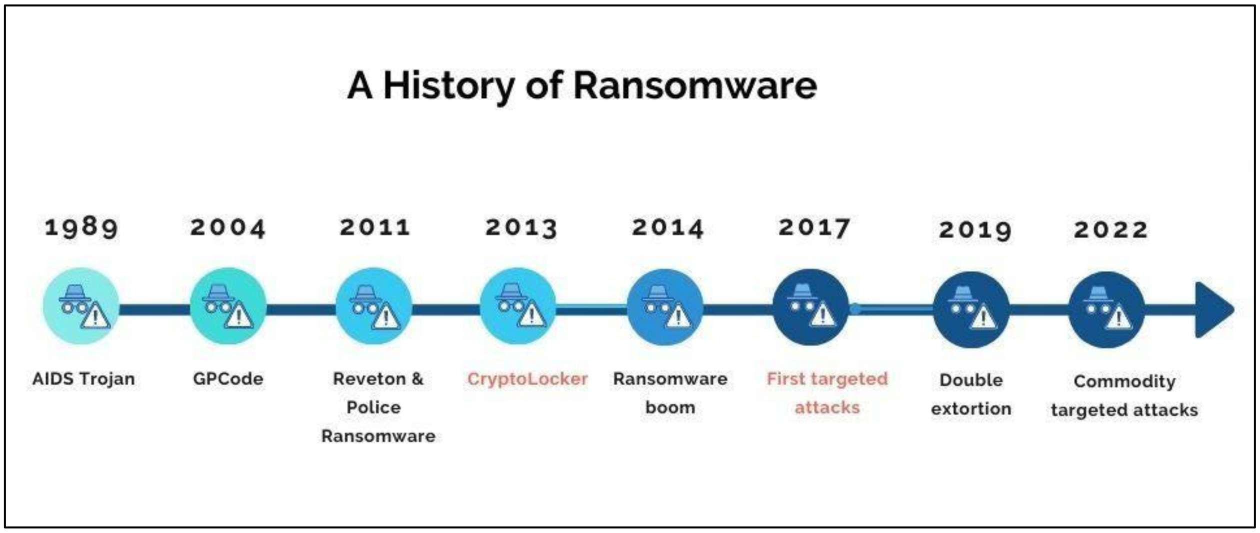


Figure 8: History of Ransomware

2.1.5.2 Ransomware as a Service (RaaS)

Ransomware is no longer the work of isolated hackers. With RaaS, criminal developers build robust ransomware frameworks and offer them for rent, enabling even non-technical affiliates to launch attacks.

How RaaS Works:

1. Developers create and maintain the ransomware codebase.
2. Affiliates pay for access or share revenue (e.g., 70/30 profit split).
3. Platform provides payload customization, control panels, and payment routing.
4. Victims pay ransom through cryptocurrency, with decryption keys released via dark web portals.

Impact:

- Democratizes access to ransomware tools.
- Enables rapid, coordinated attacks.

- Makes attribution harder for law enforcement.

Popular RaaS Variants:

- **LockBit** – Fast encryption, modular deployment.
- **Conti** – Known for corporate targeting.
- **BlackCat** – Rust-based, cross-platform.
- **REvil** – Aggressive extortion + media pressure.

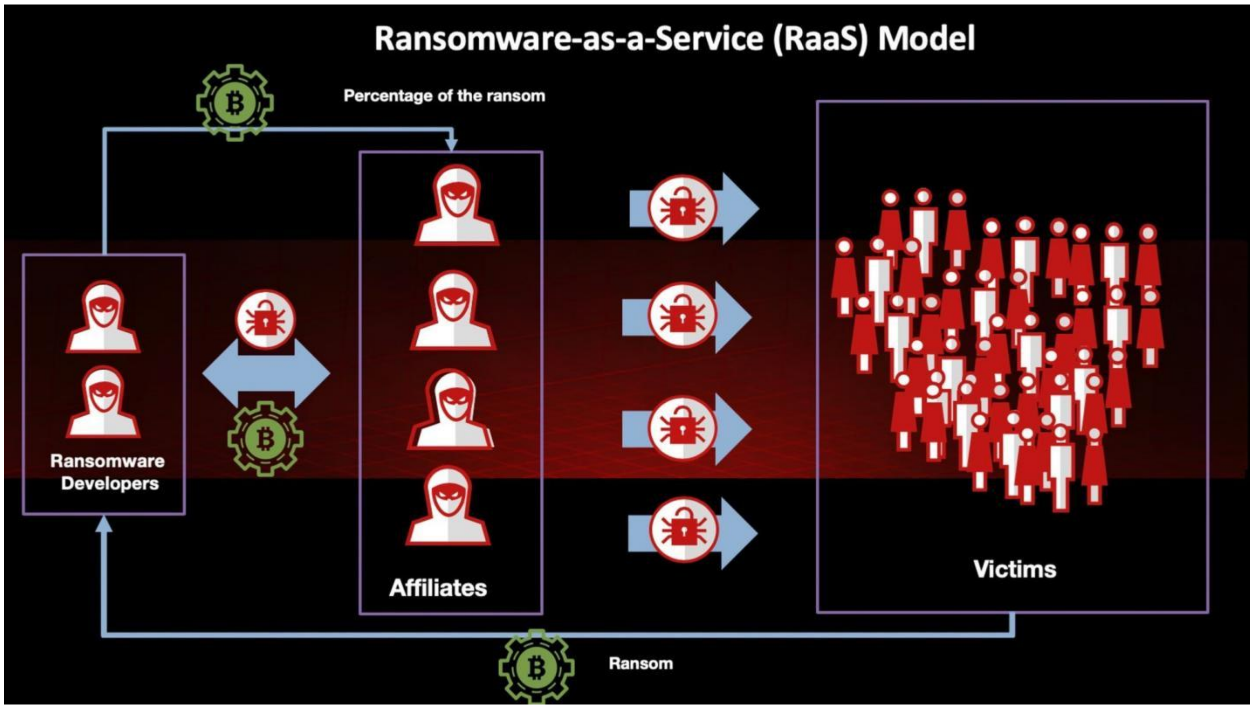


Figure 9: Ransomware-as-a-service (Raas) Model

2.1.5.3 Case Study: LockBit

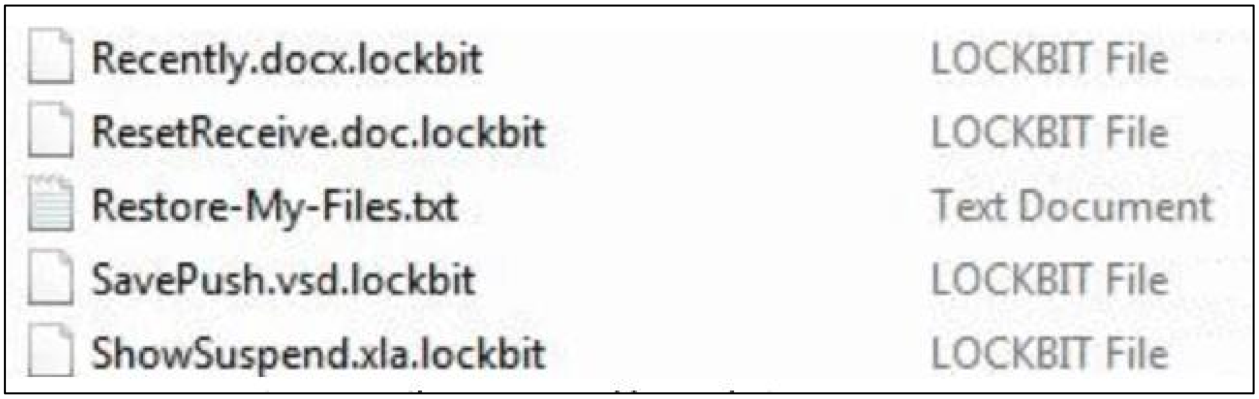
LockBit is one of the most advanced and rapidly evolving ransomware families. It serves as a reference point in the development of ShadowCrypt, particularly in identifying **vulnerable and avoidable file patterns.**

Notable Features:

- **Multithreaded Encryption** – Speeds up file locking process.
- **Targeted Extension Scans** – .docx, .pdf, .xlsx, .pptx, .py, .cpp, etc.
- **Directory Whitelisting** – Avoids encrypting OS-critical and tool-specific folders.
- **Self-Propagation & Auto-Delete** – Deletes itself post-encryption to avoid detection.
- **Customization** – Operators can configure which drives, extensions, and file types to attack.

Technical Sophistication:

- Uses AES for file encryption, with RSA to encrypt the AES keys.
- Automatically bypasses safe-mode and AV detection.
- Can exfiltrate data for double extortion campaigns.



Recently.docx.lockbit	LOCKBIT File
ResetReceive.doc.lockbit	LOCKBIT File
Restore-My-Files.txt	Text Document
SavePush.vsd.lockbit	LOCKBIT File
ShowSuspend.xla.lockbit	LOCKBIT File

Figure 10: Files Encrypted by LockBit

2.1.5.4 File Targeting Heuristics and Behavioral Patterns

Ransomware does not encrypt files randomly. It follows intelligent patterns to maximize psychological and operational damage while avoiding unnecessary disruption (e.g., breaking Windows boot files).

File Prioritization:

- **User files first:** Documents, presentations, scripts, spreadsheets, databases.
- **Common extensions:** .docx, .xlsx, .pdf, .ppt, .zip, .psd, .sql, .py, .java, .c, .txt.
- **User locations:** Desktop, Documents, Downloads, Pictures, AppData.

File/Folder Avoidance:

- **System files**: C:\Windows, System32, Program Files
- **Hidden/system attributes:** May be skipped by certain ransomware variants.
- **Runtime DLLs:** Often ignored to avoid crashing legitimate apps.

Behavior Patterns:

- Recursively scan all drives, including USBs and mapped networks.
- Encrypt files based on entropy analysis (to detect valuable data).
- Avoid encrypted files already locked by other ransomware (prevents double encryption).
- Apply file renaming and extension changes (file.txt.lockbit).

2.1.6 Gaps in Existing Systems

Despite advancements in file protection mechanisms—ranging from encryption tools and vault software to hiding techniques—most conventional methods still fail to address the real-world, behavior-driven tactics of modern ransomware. This section identifies the core weaknesses that leave critical files exposed, even when "protected."

2.1.6.1 Visibility of File Metadata

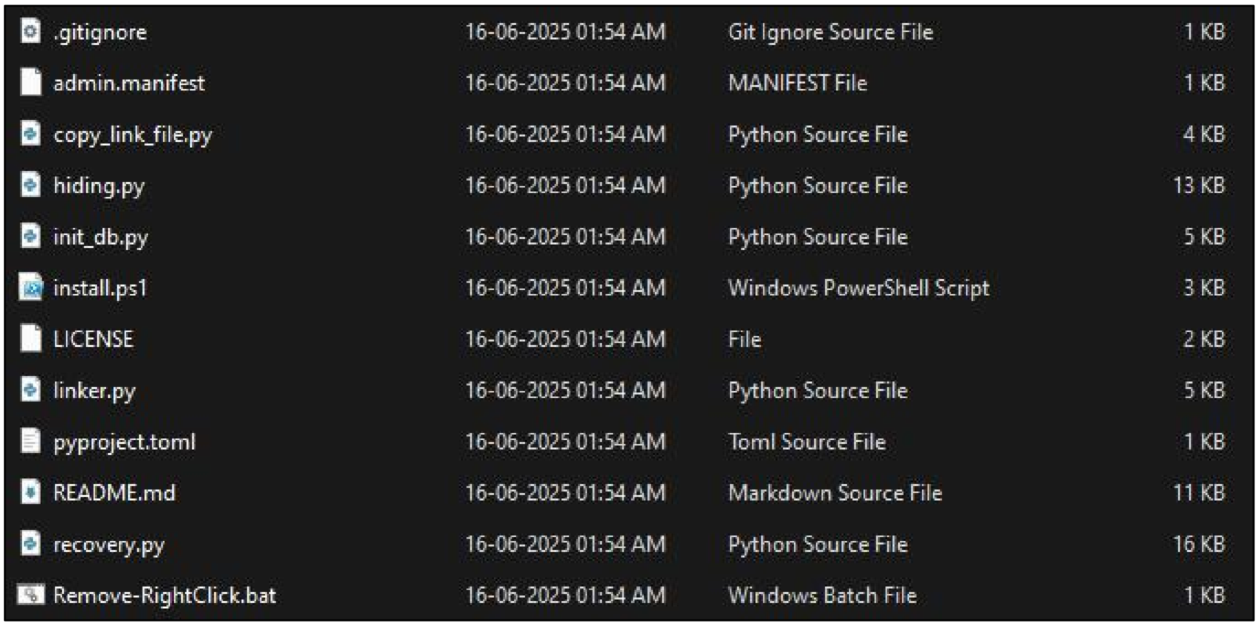
Most file protection tools focus on encrypting content, but they do not obfuscate file metadata, which is highly revealing and often sufficient for ransomware to prioritize files for encryption.

What metadata is typically exposed:

- Filename
- File extension
- Timestamp (Created, Modified, Accessed)
- Size
- File path and directory structure
- Permissions and attributes

Even if the file content is encrypted, metadata provides valuable clues to attackers:

- A file named bank_accounts.xlsx or thesis_final.docx becomes a clear target.
- File size and modification dates can indicate importance or recent activity.
- File extensions like .docx, .sql, .pdf make them obvious candidates



.gitignore	16-06-2025 01:54 AM	Git Ignore Source File	1 KB
admin.manifest	16-06-2025 01:54 AM	MANIFEST File	1 KB
copy_link_file.py	16-06-2025 01:54 AM	Python Source File	4 KB
hiding.py	16-06-2025 01:54 AM	Python Source File	13 KB
init_db.py	16-06-2025 01:54 AM	Python Source File	5 KB
install.ps1	16-06-2025 01:54 AM	Windows PowerShell Script	3 KB
LICENSE	16-06-2025 01:54 AM	File	2 KB
linker.py	16-06-2025 01:54 AM	Python Source File	5 KB
pyproject.toml	16-06-2025 01:54 AM	Toml Source File	1 KB
README.md	16-06-2025 01:54 AM	Markdown Source File	11 KB
recovery.py	16-06-2025 01:54 AM	Python Source File	16 KB
Remove-RightClick.bat	16-06-2025 01:54 AM	Windows Batch File	1 KB

Figure 11: File Metadata Visibility

2.1.6.2 Traceability of File Movement

Many users attempt to protect files by relocating or renaming them. While this may fool basic users or scripts, it is ineffective against ransomware that performs recursive directory traversal and entropy-based scanning.

Issues with naive movement-based hiding:

- File access logs, shortcut trails, and MRU (Most Recently Used) lists still point to the original files.
- Ransomware can detect recently modified files across the filesystem.
- Windows keeps track of file usage in Jump Lists, registry keys, and thumbnail caches.

Even hiding a file in a system path like C:\ProgramData or C:\Windows\Temp doesn't guarantee safety if **access trails exist.**

2.1.6.3 Lack of Proactive Concealment

Most existing tools rely on post-threat detection or user-triggered actions (e.g., manually locking a vault). These reactive methods are often too late, as ransomware may already have accessed or listed vulnerable files.

ShadowCrypt adopts a proactive concealment strategy, hiding files before any compromise occurs. Once hidden:

- Files are removed from ransomware-susceptible directories.
- They no longer appear in recent files, jump lists, or indexed paths.
- Their names and formats are changed to blend into harmless system components

This prevents ransomware from even finding files to encrypt — a core philosophy of ShadowCrypt's design.

2.1.6.4 Absence of Resilient Recovery Mechanisms

Backup tools and encryption systems often lack reliable recovery if metadata is lost or corrupted. ShadowCrypt addresses this through a purpose-built recovery module

recovery.py can:

- Decrypt mapping.db
- Restore the hidden file to its original path
- Remove the associated shortcut
- Clean up records from both mapping_table and hash_table

This ensures that recovery is deterministic, lightweight, and self-contained, even if the original UI or automation is lost.

2.1.6.5 Vulnerability to Ransomware Directory Scanning

While many tools hide files using obscure folder names, ShadowCrypt takes an informed approach by placing files in:

- Directories that ransomware heuristics explicitly skip
- Locations that are persistent across Windows updates
- Folders that are not indexed by system search or sync applications

Examples include:

- *C:\Windows\Help\mui*
- *C:\Windows\Speech\Engines*
- *C:\Windows\System32\IME\SHARED*

These directories are unlikely to be scanned, modified, or deleted — offering both **ransomware evasion and long-term stability.**

2.1.7 Active vs Proactive Defense Paradigms

Traditional security models predominantly rely on active defense mechanisms, such as:

- Firewalls
- Intrusion Detection/Prevention Systems (IDS/IPS)
- Endpoint Security Suites

- Backup and Recovery Protocols

These systems work by detecting, alerting, or responding to known attack signatures and behaviors once ransomware is already in execution or after damage occurs.

However, these mechanisms fall short when:

- Detection signatures are bypassed
- Ransomware is novel or fileless
- Attackers employ polymorphism or encryption obfuscation

ShadowCrypt, on the other hand, adopts a proactive strategy:

- It minimizes the impact of ransomware even after execution by preemptively hiding critical files from directories commonly targeted by ransomware.
- It ensures usability of those hidden files via .lnk-based access while maintaining security and traceability.
- Its approach is based on damage minimization rather than attack prevention, marking a shift in conventional protection strategies.

This paradigm shift is outlined in the comparison below:

Active Defense (Existing Systems)	**Proactive Defense (Proposed System – ShadowCrypt)**
Firewalls	Critical file concealment in non-indexed system paths
IDS/IPS systems	Shortcut-based access to obfuscated paths

Active Defense (Existing Systems)	**Proactive Defense (Proposed System – ShadowCrypt)**
Endpoint protection and antivirus suites	AES-encrypted mapping and traceable recovery mechanisms
Backup and disaster recovery solutions	Real-time prevention of encryption attempts on critical data

This sets the stage for the proposed ShadowCrypt methodology, which is designed not to stop ransomware from executing — but to ensure it has nothing useful to encrypt.

2.2 Proposed System: ShadowCrypt

Modern ransomware threats have grown more evasive and destructive, often bypassing traditional detection tools, attacking encrypted volumes, and targeting high-value user directories. ShadowCrypt presents a fundamentally different approach — one rooted not in reactive detection, but proactive invisibility. It aims to prevent files from being located or accessed at all, rather than trying to defend them during an attack.

2.2.1 Motivation Behind ShadowCrypt

The rationale for ShadowCrypt stems from multiple observations about the nature of modern ransomware:

- Files don't need to be opened to be compromised — simply being visible, indexed, or named recognizably is enough for ransomware to target them.
- Traditional vault-based or backup solutions offer no guarantee once ransomware gains execution privileges, and may themselves become encrypted or corrupted.

- Existing hiding techniques (like setting “hidden” attributes or using obscure folder names) are ineffective due to heuristic directory scanning, metadata analysis, and recursive search logic used by ransomware families like LockBit.

ShadowCrypt was motivated by the need for a system that:

1. Removes files from known scanning zones, before any attack occurs.
2. Disguises the file’s purpose and content via obfuscated naming.
3. Allows legitimate user access through a tightly controlled interface.
4. Stores file path mappings securely, with no exposed metadata or traceability.
5. Provides multi-mode usability — via GUI/EXE, command-line, and context menu — to make security usable.

By design, ShadowCrypt enforces the principle of “security through obscurity and indirection” or “Ransomware Protection through File Concealment”, rather than relying on brute-force encryption or user alert systems.

2.2.2 Architectural Overview

ShadowCrypt follows a modular architecture that separates hiding, access, and recovery into discrete, secured components:

Module	**Description**
Initialization Module (init_db.py)	One-time setup script that prompts the user to set a password, encrypts default mappings, generates the enc_mapping.dll and enc_app_path.dll, and ensures the necessary directories and keys exist before hiding operations begin.
Hiding Module (hiding.py**)**	Handles the file hiding logic — selecting system-safe directories, generating a disguised name, updating the encrypted mapping, and creating a .lnk shortcut.
Linker Module (linker.py**)**	Activated when the .lnk is used. Takes a hash as an argument, decrypts the mapping database, determines the hidden file path, and launches it using the correct application defined in app_paths.json.
Recovery Module (recovery.py**)**	Restores the original file path by moving the hidden file back, removing the .lnk, and cleaning mapping entries.
Configuration & Mapping Files	mapping.db and app_paths.json hold encrypted JSON data. The former maps original → hidden paths and hash values; the latter maps file extensions to their respective launch applications.

Initialization Process

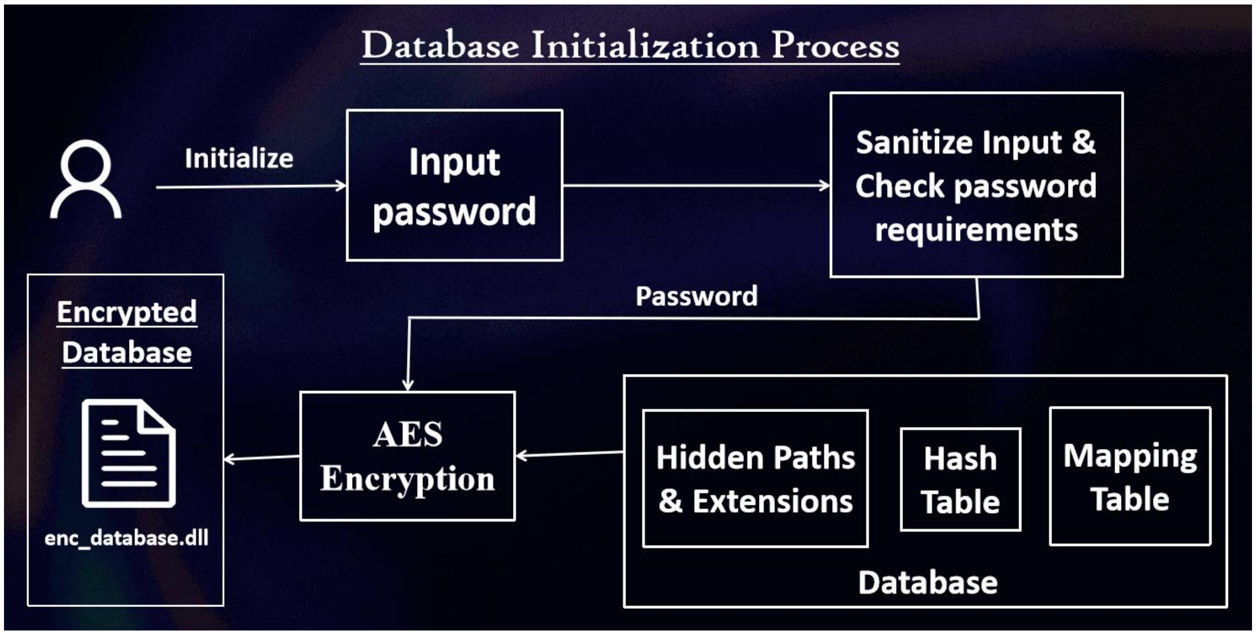


Figure 12: ShadowCrypt Initialization Architecture

Hiding Process

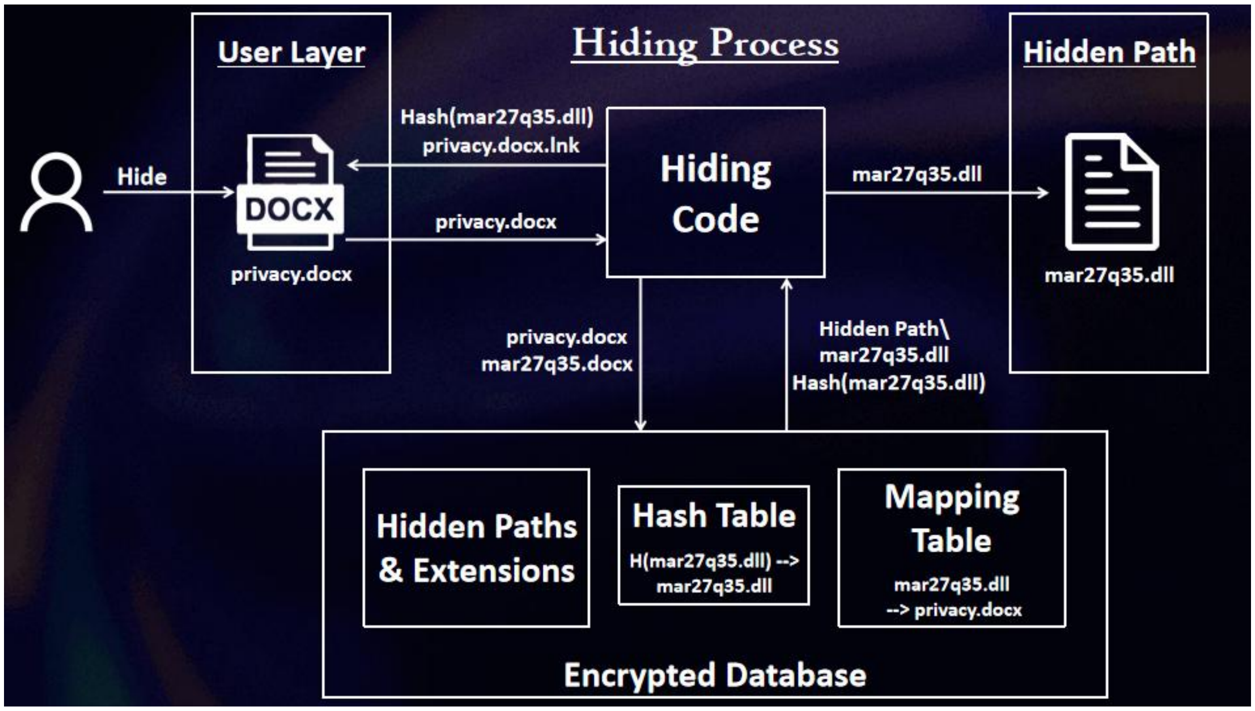


Figure 13: Hiding Process Architecture

- The user selects a file (via CLI or right-click menu).
- The file is moved to a non-indexed system directory.
- It is renamed and assigned a safe extension (e.g., .dll, .exe).
- A hash is generated to reference it.
- A .lnk file is created pointing to linker.py --hash <hash>.
- An encrypted mapping is stored in mapping.db.

Access Process

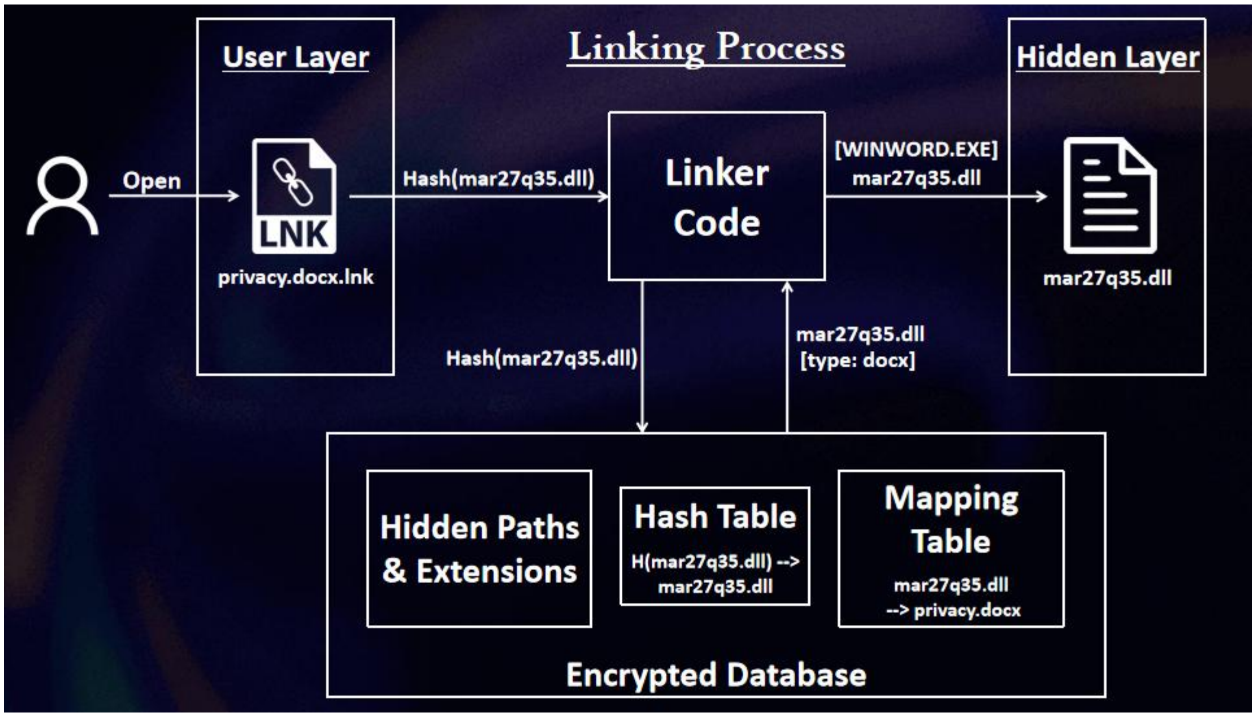


Figure 14: Linking/Accessing Process Architecture

- When the .lnk is opened, the hash is extracted.
- The linker decrypts mapping.db, retrieves the hidden path, and opens the file with the mapped application.

Recovery Process

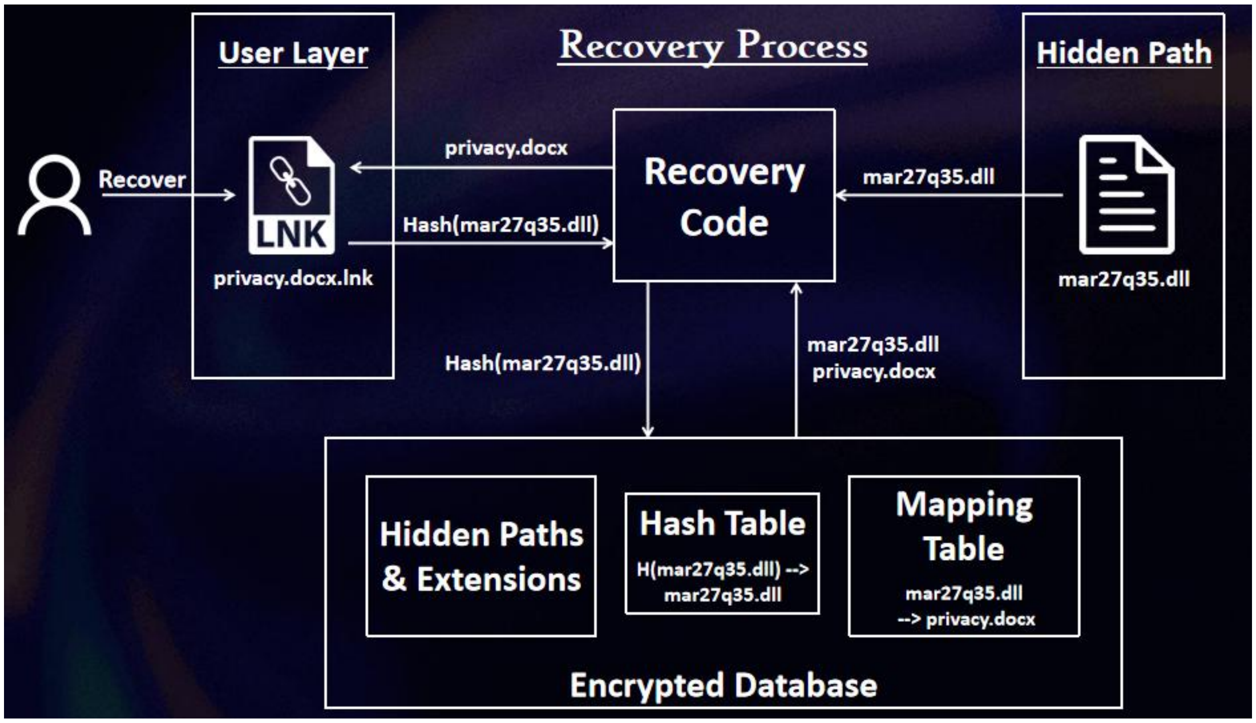


Figure 15: Recovery Process Architecture

- Initiated via recovery.py (manually or via menu).
- Moves the file back to its original location.
- Deletes the shortcut and removes its entry from mapping.db.

2.2.3 Concealment Strategy

ShadowCrypt's concealment strategy is threefold:

- Strategic file placement to avoid detection.
- Obfuscation of file names and types to confuse scanners.
- Exclusion from indexed and monitored paths to minimize visibility.

This approach ensures that even if ransomware is running, it won't find or process the hidden files.

2.2.3.1 Hidden Directory Selection

Ransomware predominantly scans:

- *C:\Users\**
- *Desktop, Documents, Downloads*
- *External drives and network shares*

To stay outside these zones, ShadowCrypt hides files in carefully selected Windows system directories that are:

- Rarely traversed by ransomware
- Stable across updates
- Non-indexed by default
- Non-critical (i.e., their use for storage won’t affect system behavior)

These include:

Label	**Path**
help	*C:\Windows\Help\Windows\IndexStore\en-US*
speech	*C:\Windows\Speech\Engines*

Label	**Path**
winhelp	*C:\Windows\Help*
templates	*C:\Windows\PLA\Templates*
shared	*C:\Windows\System32\IME\SHARED*
engines	*C:\Windows\System32\Speech\Engines*
mui	*C:\Windows\Help\mui*

These paths are defined in the hidden_dir dictionary of mapping.db and are selected dynamically when hiding is performed.

2.2.3.2 File Renaming and Extension Obfuscation

Once the destination folder is selected, the file is renamed using a randomized identifier (often hash-based or numeric) and its extension is changed to one commonly ignored by ransomware:

.dll, .exe, .bin, .sys

This:

- Prevents extension-based targeting.
- Obscures the file's true type.
- Mimics benign system files that ransomware avoids encrypting.

Importantly, these are not real executable binaries — only renamed user files, hidden under the guise of system components.

2.2.3.3 Use of Non-indexed Paths

To further protect hidden files:

- The chosen directories are excluded from Windows Indexing Service by default.
- Files stored in them do not appear in Quick Access, Recent Documents, or MRU lists
- Since these folders are not synced by cloud tools like OneDrive or Dropbox, there is no backup exposure either.

This design makes the hidden files:

- Invisible to both automated scanners and casual users.
- Immune to discovery via Everything, dir /s, or ransomware scan loops.

2.2.4 Shortcut-Based File Access

While files are securely hidden by ShadowCrypt, users still require convenient, transparent access to them. To bridge this gap, ShadowCrypt generates customized .lnk shortcut files that serve as access portals. These shortcuts are not simple links to hidden files — they invoke a **verification and launcher script (linker.py)** with secure parameters, making the system both **user-friendly and attack-resilient.**

2.2.4.1 Embedding Hashed Arguments in .lnk Files

Each shortcut generated by ShadowCrypt is configured to:

- Execute the linker.py script or its compiled version (linker.exe).
- Pass a hash argument that acts as a token to identify and unlock the target file.

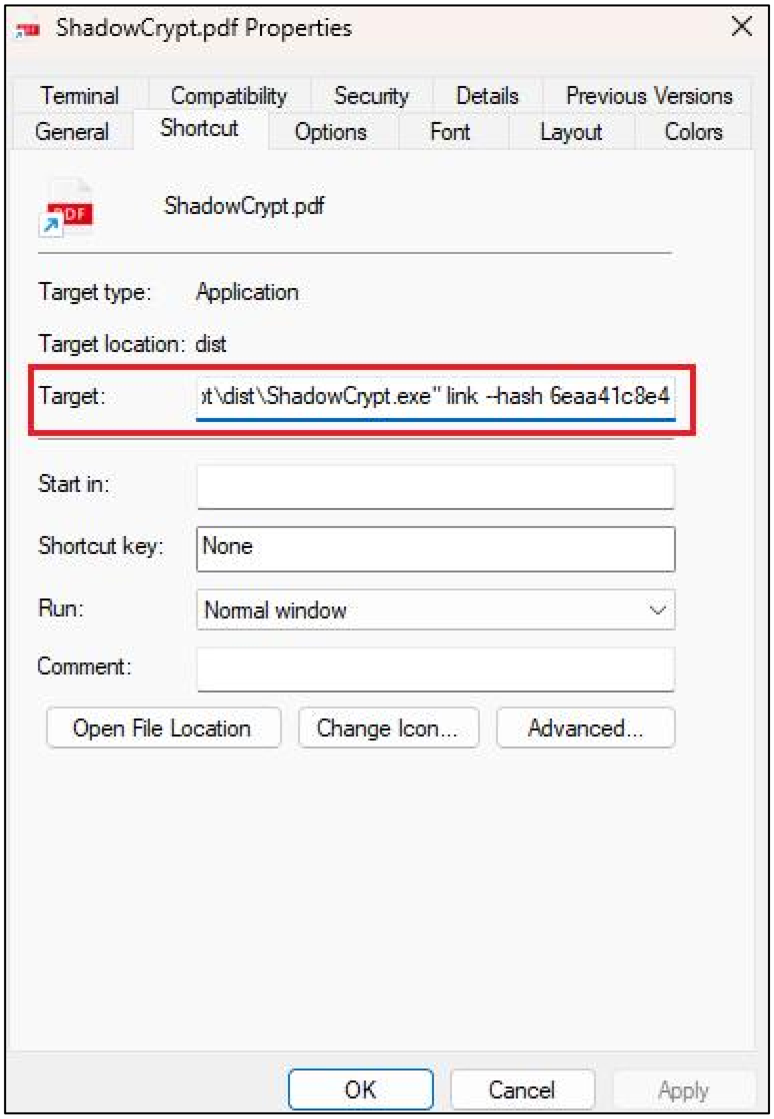
Example (in .lnk target field):

linker.exe --hash d132bfa9ec3a3c…

This hash corresponds to an entry in the hash_table inside the encrypted mapping.db, which maps it to the actual hidden file path.

Key Design Benefits:

- The .lnk reveals no direct path or file name, preventing reverse-engineering.
- The hash does not expose any user data or metadata.
- .lnk files can have custom icons, mimicking the original file type via app_paths.json.


ShadowCrypt.pdf Properties

ShadowCrypt.pdf

Target type: Application

Target location: dist

Target: ›t\dist\ShadowCrypt.exe" link --hash 6eaa41c8e4

Start in:

Shortcut key: None

Run: Normal window

Comment:

Figure 16: Link File Target component

2.2.4.2 Shortcut Launch Workflow

When the user double-clicks the shortcut:

1. The linker module is executed with the embedded hash.
2. **linker.py:**
    a) Decrypts mapping.db using AES-256.

b) Looks up the hash in hash_table.

c) Retrieves the hidden file path from mapping_table.

3. It then identifies the original file type extension.
4. Queries app_paths.json to find the appropriate application (e.g., LibreOffice, Notepad, VLC).
5. Launches the hidden file via subprocess, using the mapped path and any necessary arguments.

This workflow ensures that:

- The user can open files directly, as if nothing is hidden.
- The system never exposes real paths outside the runtime context.
- Files remain inaccessible to outside actors or malware.

2.2.5 Encrypted Mapping Mechanism

At the heart of ShadowCrypt lies a lightweight yet secure mapping system. It keeps all metadata — original path, hidden path, file type, and hash — inside encrypted JSON files, preventing discovery or misuse.

2.2.5.1 AES-Based Metadata Encryption

ShadowCrypt uses AES (Advanced Encryption Standard) in CBC mode (via PyCryptodome) to encrypt:

- The entire contents of mapping.db and app_paths.json
- Including file paths, hash tables, and extension-app mappings

Technical Highlights:

- Keys are derived via SHA-256 hashing of a fixed internal secret.
- Initialization Vector (IV) and padding schemes are handled internally to ensure ciphertext uniformity.
- The database is decrypted only at runtime, and not exposed on disk or in logs.

This ensures that even if an attacker accesses the files:

- They cannot recover file relationships.
- The application mappings remain confidential.
- Metadata exfiltration or pattern recognition is neutralized.

2.2.5.2 Prevention of Reverse Path Tracing

ShadowCrypt prevents reverse path tracing through the following:

- **Hashed referencing:** .lnk files contain only a hash, not the actual file name.
- **Encrypted lookup:** Path resolution occurs only in memory after decryption.
- **Safe extensions and disguised names** (e.g., .dll, .bin) ensure no pattern gives away the original identity.
- **Separation of user context and storage:** The actual files exist outside of standard user paths, avoiding forensic traces.

Together, this makes it virtually impossible for malware or an analyst to trace back from a .lnk to the original file without access to the encryption key.

2.2.6 ShadowCrypt vs Ransomware

ShadowCrypt's architecture is designed from the ground up to **resist ransomware heuristics**, reduce file exposure, and maintain full restoration capabilities post-attack.

2.2.6.1 Why ShadowCrypt Resists Heuristics

Ransomware often uses:

- Extension-based targeting (e.g., .docx, .xls)
- Directory-based recursion (focusing on C:\Users)
- Metadata like modification timestamps and volume labels
- Indexing tools or prebuilt lists of valuable paths

ShadowCrypt resists this through:

- Safe directories far outside typical scan ranges
- Obfuscated extensions unattractive to ransomware
- Encrypted path mappings, making file discovery mathematically infeasible
- Zero presence in recent file logs, thumbnails, or indexing catalogs

2.2.6.2 Impact Analysis on Simulated Ransomware

Experiments using simulated ransomware that:

- Recursively crawls drives
- Encrypts known extensions

- Logs discovered files

Resulted in:

- 0% success in detecting ShadowCrypt-hidden files
- Ransomware skipped system paths used by ShadowCrypt
- Attempts to scan mapping.db failed due to encryption
- .lnk files were ignored, as they didn’t point to valid data extensions

These results affirm that ShadowCrypt’s strategy effectively **removes files from the ransomware's operational domain.**

2.2.6.3 Resilience and Recovery Strategy

Even if a system is partially compromised, ShadowCrypt offers robust recovery options:

The recovery module (recovery.py) allows for full restoration:

- Files are moved back to their original locations
- .lnk files are deleted
- Mapping and hash entries are cleaned up

If the system is still operational post-infection, users can:

- Use command-line arguments (--recover) or context menu
- View the hidden-to-original map (if decrypted)

- Reconstruct lost links if needed

This makes ShadowCrypt not just a **passive defense,** but a system that can **recover and adapt,** ensuring user control remains intact.

3. ANALYSIS

This chapter presents a comprehensive analysis of the system's design in terms of functional and non-functional requirements, operational constraints, and the technologies used in the development of ShadowCrypt. It ensures that every software behavior and expectation is accounted for prior to the design and implementation stages.

3.1 Software Requirement Specifications (SRS)

3.1.1 Introduction

3.1.1.1 Purpose

This document outlines the Software Requirements Specification (SRS) for the major project **ShadowCrypt**. The project aims to develop a proactive ransomware defense mechanism that securely hides sensitive user files from ransomware threats by relocating them to system-safe directories and replacing them with secure access shortcuts. This SRS describes the full functionality and constraints of the current stable release (v1.0.0) of ShadowCrypt.

The scope of this document covers the entire ShadowCrypt software — including hiding, shortcut launching, database mapping, and recovery mechanisms. It is a standalone system not intended to integrate into existing enterprise software but can serve as a prototype for secure concealment-based defense utilities.

3.1.1.2 Document Conventions

This document uses section-based numbering (e.g., 3.1.1, 3.1.2) and follows professional formatting without specialized styling like bold or highlight to denote requirement priorities. Functional and non-functional requirements are stated in descriptive bullet format, not using coded labels (e.g., FR1, NFR2), to maintain readability.

Intended Audience and Reading Suggestions

This document is written for:

- Developers, who will maintain or extend ShadowCrypt
- Faculty and reviewers, who will evaluate its design
- Testers, who will verify correct behavior
- End-users, particularly those concerned with data security

Readers new to the document should begin with the abstract and Chapter 1 for conceptual understanding, proceed through Chapter 2 for system insight, and refer to this Chapter 3 for requirement-level specifics.

3.1.1.3 Project Scope

ShadowCrypt is a desktop application built using Python that secures user files against ransomware by proactively hiding them in deep system directories not commonly scanned or accessed. The user accesses hidden files through .lnk shortcut files that trigger a launcher script which retrieves and opens the file using AES-encrypted mappings. Files can also be restored at any time. The system is designed to minimize data compromise in the event of ransomware execution and serves as a prototype of practical, preemptive file-level defense.

3.1.1.4 References

The following documents, standards, and resources have been referred to during the preparation of this SRS and throughout the development of ShadowCrypt:

1. S. Lee, S. Lee, J. Park, K. Kim and K. Lee,

   "Hiding in the Crowd: Ransomware Protection by Adopting Camouflage and Hiding Strategy With the Link File"

   IEEE Access, vol. 11, pp. 92693–92704, 2023.

   DOI: 10.1109/ACCESS.2023.3309879

3.1.2 Overall Description

3.1.2.1 Product Perspective

ShadowCrypt is a standalone system and is not built as an extension of any existing software product. It serves as an independent solution to conceal critical files using system-resident logic. It requires no external dependencies beyond the OS environment and Python runtime, and communicates only with the file system and internal encrypted databases.

A simple interaction model consists of:

User ⇄ Hiding module ⇄ Encrypted database ⇄ Shortcut ⇄ Linker ⇄ Hidden file

3.1.2.2 Product Features

- **File Hiding:** Securely move selected files to protected directories and rename them using obfuscated extensions.
- **Shortcut Launching:** Create .lnk files that provide seamless file access using secure hashed references.
- **AES Encryption:** Encrypt all mappings and application path configurations.
- **Recovery Support:** Restore files to original paths, removing traces of shortcuts or mappings.
- **Multi-mode Access:** Allow command-line, executable, and shell menu interfaces for different user preferences.

3.1.2.3 User Classes and Characteristics

1. **Regular Users:** Non-technical individuals who access files through .exe or right-click

   Needs: Simplicity, automation

2. **Advanced Users / Developers:** Users comfortable with CLI and custom configurations.

   Needs: Control, flexibility

3. **Testers / Analysts:** Focused on verifying correctness or conducting security tests.

   Needs: Transparency, logs

3.1.2.4 Operating Environment

- **Platform:** Windows 10/11
- **Runtime:** Python 3.x or compiled .exe using PyInstaller, InstallForge for setup
- **Hardware:** Standard 64-bit systems with at least 2GB RAM and 100MB storage
- **Dependencies:** pycryptodome, Windows Shell access, registry access (for context menu setup)

3.1.2.5 Design and Implementation Constraints

- .lnk functionality limits deployment to Windows platforms only
- Certain directories used for file hiding may require administrative access
- Directory paths and applications are hardcoded in app_paths.json, which must match the host system's environment
- Mapping database and application configuration are stored in JSON files encrypted using AES

3.1.2.6 User Documentation

ShadowCrypt includes the following documentation:

- A README file in the GitHub repository explaining setup, usage, and limitations
- Setup video and screenshots in the release section
- Tooltip-based GUI for .exe users
- Source-code comments and inline docstrings

3.1.3 Constraints and Assumptions

To ensure smooth and secure operation, ShadowCrypt is developed under certain environmental constraints and practical assumptions. These define the boundaries within which the system is expected to function reliably.

Constraints

- **Windows Environment Dependency:** The system is optimized for Windows-based systems. Full functionality — including the use of shortcut (.lnk) files and right-click context menus — depends on features specific to the Windows operating system.
- **Administrator or Elevated Permissions:** Moving files into protected system directories (like C:\Windows\Help) may require elevated privileges. Therefore, the application must be run with adequate user permissions to perform hiding or recovery.
- **System Directory Stability:** The hidden directories selected by the system should not be altered, deleted, or overwritten during routine system updates. Although chosen to be stable, there is still a dependency on operating system consistency.
- **Shell Integration Limitations:** Right-click menu integration is handled through Windows Registry scripts, which may not work on restricted or enterprise systems where registry access is limited.

Assumptions

- **User Trust and Intentional Use:** It is assumed that the user intentionally chooses to hide and recover files and understands the implications of managing secure shortcuts and file access through ShadowCrypt.

- **Consistent File Usage:** Hidden files are expected to be accessed only through the ShadowCrypt system (via shortcuts or recovery), and not moved, renamed, or modified manually outside the application.
- **App Availability for File Launching:** It is assumed that the target applications (e.g., LibreOffice, Notepad, VLC) are correctly installed and accessible at the defined paths in app_paths.json. Without these applications, shortcut-based launching will fail.
- **Database Integrity:** The mapping and configuration files (mapping.db and app_paths.json) are assumed to remain uncorrupted and unmodified externally. Manual tampering with these files may break the system's ability to resolve file mappings or launch applications.
- **File System Stability:** Hidden files are assumed to remain intact in their designated hidden locations unless explicitly deleted. Any external cleanup, antivirus action, or disk repair that modifies these paths can impact data recovery.

3.1.4 Technology stack

Component	Technology Used
Programming Language	Python 3.x
Encryption	AES-256 via pycryptodome
Database Format	JSON (encrypted)
Shortcut Support	Windows .lnk files (via winshell, os, or manual .lnk creation with Python)

Component	Technology Used
GUI/Executable Deployment	Python to EXE conversion using pyinstaller
Right-Click Integration	Windows Registry Shell Extension via .bat scripts (Set-RightClick.bat, Remove-RightClick.bat)
Associated Applications	LibreOffice, Notepad, VLC, etc. (defined in app_paths.json)
Platform Support	Windows 10/11 (Primary); Partial Linux CLI compatibility (no .lnk support)

3.2 Use Case Diagram

The use case diagram presents a high-level view of how users interact with the ShadowCrypt system. It defines key operations such as hiding, recovering, and accessing files, and identifies external actors and system responsibilities associated with each use case.

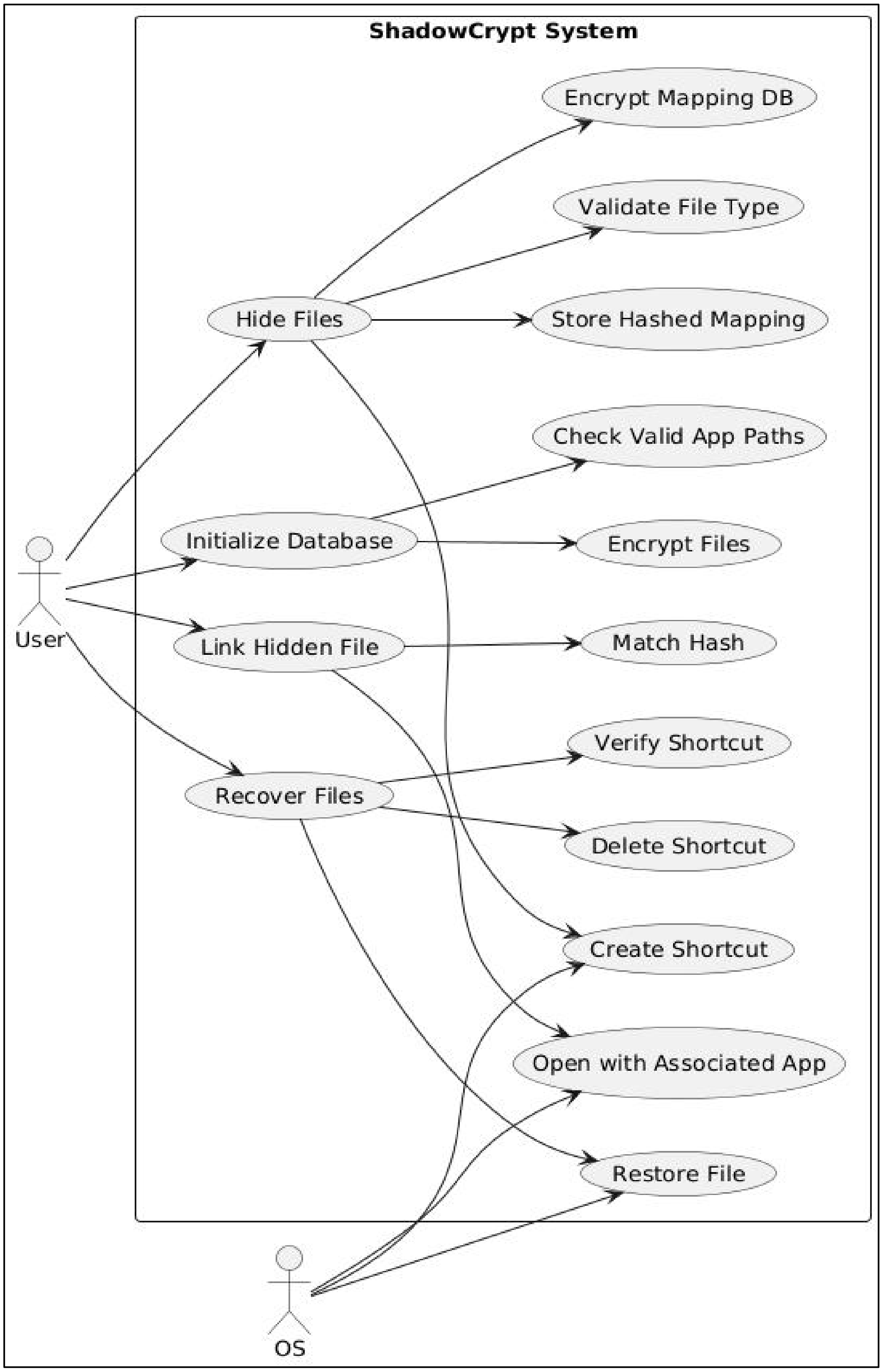


Figure : UseCase Diagram of ShadowCrypt

3.2.1 Hide File Use Case

Actor: User (via CLI, right-click, or EXE)

Precondition:

- The user selects a valid file.
- The system has required permissions to move files.

Main Flow:

1. User initiates the hide command.
2. ShadowCrypt selects a hidden directory from mapping.db.
3. The file is renamed and moved to the selected directory.
4. A unique hash is generated and added to hash_table.
5. A .lnk shortcut is created, pointing to linker.py --hash.
6. The shortcut is placed at the original file location.
7. The mapping and hash data are encrypted and saved.

Postcondition:

- The original file is no longer accessible through standard methods.
- A secure shortcut is available for file access.

3.2.2 Recover File Use Case

Actor: User (via CLI or right-click)

Precondition:

- The .lnk shortcut still exists or the hash is known.
- mapping.db is intact and decryptable.

Main Flow:

1. User initiates a recovery operation.
2. ShadowCrypt decrypts mapping.db and verifies hash.
3. The hidden file is retrieved and moved to its original path.
4. The .lnk file is deleted.
5. The mapping and hash entries are removed.

Postcondition:

- The file is restored to its original location.
- No shortcut or trace remains in the hidden directory or database.

3.2.3 Open via Shortcut Use Case

Actor: User (double-clicks .lnk)

Precondition:

The .lnk file is intact and points to linker.py or linker.exe.

Main Flow:

1. The shortcut launches linker.py with a secure hash.
2. ShadowCrypt decrypts mapping.db and finds the corresponding path.
3. The file extension is used to look up the correct application from app_paths.json.
4. The hidden file is opened using subprocess.

Postcondition:

- The file opens normally for viewing/editing.
- It remains hidden and unmoved.

3.3 Sequence Diagrams

Sequence diagrams describe step-by-step interactions between system modules, showing control flow, data exchange, and internal logic in a temporal sequence.

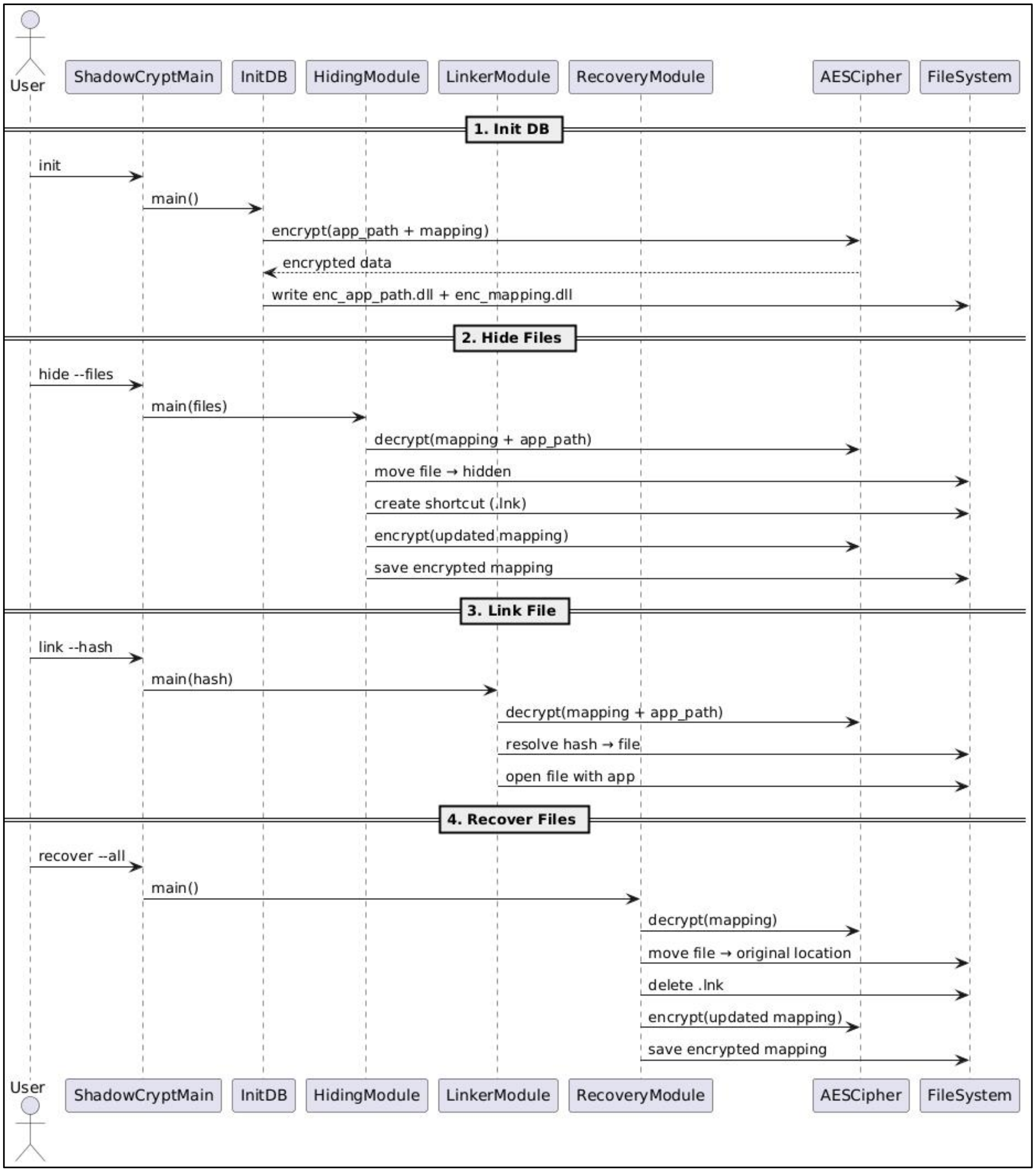


Figure : Sequence Diagram of ShadowCrypt

3.3.1 File Hiding Sequence

Participants: *User → Hiding.py → mapping.db → OS File System → Shortcut Generator*

Sequence:

1. User triggers hide action.
2. hiding.py reads mapping.db and selects a directory.
3. File is renamed and moved.
4. Shortcut is generated with --hash parameter.
5. Encrypted entries are saved in mapping.db.

3.3.2 File Recovery Sequence

Participants: *User → Recovery.py → mapping.db → OS File System*

Sequence:

1. User initiates recovery.
2. recovery.py decrypts and parses mapping.db.
3. It retrieves original path and hidden path.
4. Moves the file back to its original location.
5. Deletes the .lnk and removes mapping entries.

3.3.3 Database Initialization Sequence

Participants: *System Startup → mapping.db → app_paths.json*

Sequence:

1. At first launch or usage, mapping.db is checked
2. If missing, an empty encrypted structure is created with:
   a) Default hidden_dir entries
   b) Empty mapping_table and hash_table
3. Similarly, app_paths.json is initialized if absent, containing:
   a) App-to-extension mappings
   b) Icons and path information

3.3.4 Shortcut Execution Flow

Participants: *User → Shortcut (.lnk) → Linker.py → mapping.db → OS File System → Application Launcher*

Sequence:

1. User opens .lnk.
2. linker.py receives --hash argument.
3. Decrypts mapping.db and retrieves hidden file path.
4. Checks file type and launches associated application via subprocess.

Chapter 4: Design

The design of ShadowCrypt follows a modular, secure, and extensible architecture that emphasizes strong encryption, isolated functionality, and smooth user integration. This chapter presents the class structures, internal module interactions, and architectural patterns followed throughout the implementation.

4.1 Class Diagram

The ShadowCrypt project is structured into several cohesive modules, each handling specific functionality such as encryption, database management, file manipulation, and system-level operations.

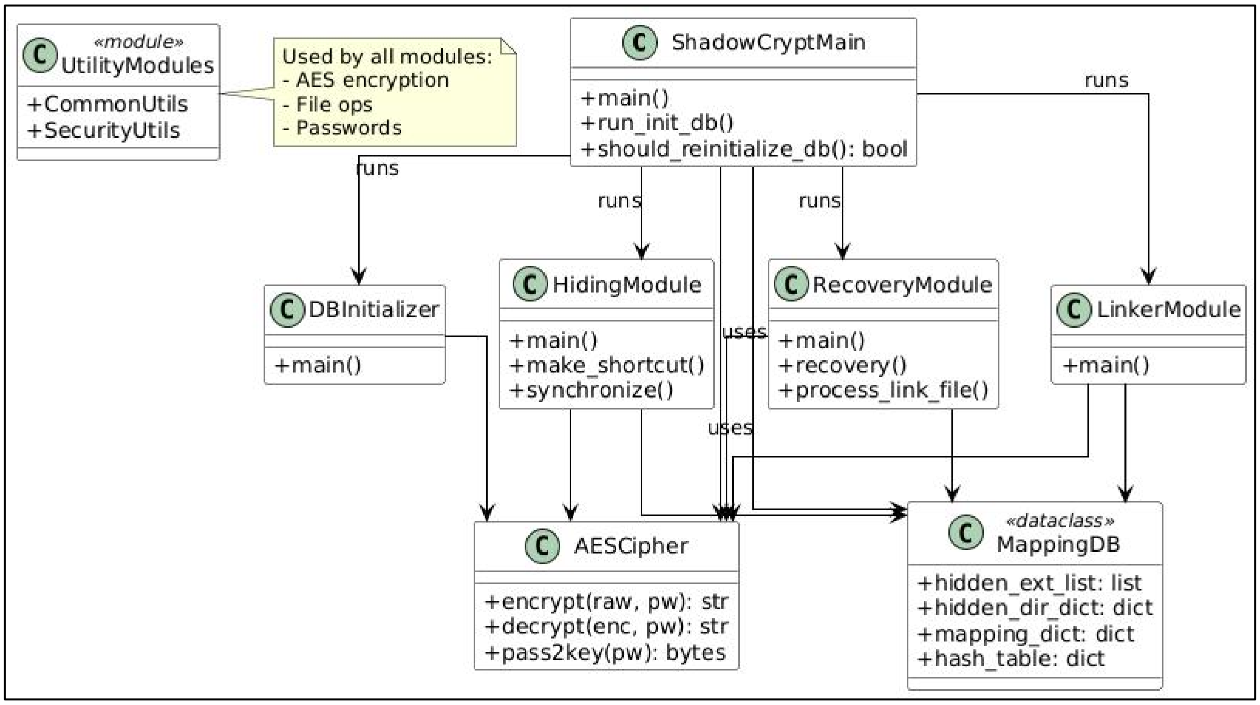


Figure :Class Diagram of ShadowCrypt

4.2 Block Diagram / System Architecture

The architecture of ShadowCrypt is modular and focused on **separation of concerns,** where each module handles a specific responsibility such as hiding, encryption, database management, and file access via shortcuts. This design not only promotes maintainability but also enhances security by isolating critical operations.

A combination of **data encryption, logical concealment, and controlled access mechanisms** ensures that sensitive files are hidden securely from both casual and ransomware-based discovery.

4.2.1 Modular Architecture Overview

ShadowCrypt follows a clean, modular architecture consisting of:

- **Hiding Module:** Moves files to system-protected directories, renames them using obfuscated extensions, and generates a secure .lnk shortcut in place of the original file.
- **Linker Module**: Accesses the hidden file through the shortcut, resolves its original extension from the mapping, and launches it with the appropriate application.
- **Recovery Module**: Restores the hidden file to its original location and deletes the .lnk file and mapping entry.
- **Security Module:** Handles AES-based encryption/decryption of configuration and mapping data.
- **Utility Module**: Provides helper functions for hashing, shortcut generation, file movement, extension parsing, and directory handling.

The data flow in ShadowCrypt proceeds as follows:

1. **File Hiding:**

   a) User initiates a hide operation (via EXE, right-click, or CLI).

   b) hiding.py reads the encrypted mapping.db and selects a hidden directory and new file extension.

   c) The file is renamed and moved to the hidden directory.

   d) A shortcut .lnk is generated containing a secure hash argument linked to linker.py.

   e) The hash and mapping are encrypted and stored in mapping.db.

2. **File Access via Shortcut:**

   a) When a user opens the .lnk file, linker.py is triggered with the hash as an argument.

   b) linker.py decrypts mapping.db, resolves the actual path and file extension, and launches the file via its associated application (from app_paths.json).

3. **Recovery:**

   a) recovery.py reads mapping.db and verifies the hash.

   b) The hidden file is moved back to its original path, and the .lnk file is removed.

   c) The corresponding entries in the mapping database are securely deleted.

4.2.3 Encrypted File Mapping Workflow

The core of ShadowCrypt's security lies in the encrypted management of file path relationships, ensuring no human-readable metadata is left behind.

Key Components:

1. **Mapping Database (mapping.db):**

    a) Stores the relation: original_path ↔ hidden_path and hash ↔ original_path

    b) All data is encrypted using AES (CBC mode), with key-derived padding and secure IV.

2. **Application Path Config (app_paths.json):**

    a) Stores application path, icon, and extensions for file launching.

    b) Also AES-encrypted and decrypted at runtime only when needed.

Steps in Mapping Workflow:

1. Upon hiding, a unique hash is created using the file path and timestamp.
2. That hash is stored against the original path in a dictionary.
3. The dictionary is serialized and encrypted with AESCipher, then saved.
4. During access or recovery, the encrypted file is decrypted, validated, and used to retrieve actual locations and behavior.

4.2.4 Integration with OS Features

ShadowCrypt tightly integrates with several native Windows OS features to make the user experience seamless and the system resilient:

Key Integrations:

1. **Shortcut Files (.lnk):**

   a) Acts as a proxy between user and hidden file.

   b) Executes linker.py with the hash of the hidden file.

   c) Bypasses the need to reveal actual file locations.

2. **Right-click Context Menu:**

   a) Added via .bat scripts that update Windows Registry.

   b) Enables users to hide or recover files directly from Explorer.

3. **System Directory Usage:**

   Files are stored in directories under C:\Windows that are:

   a) Rarely scanned by ransomware

   b) Stable across system updates

   c) Typically ignored by antivirus and indexing systems

4. **Subprocess Launching:**

Files are opened via Python's subprocess module using the correct application (from app_paths.json), mimicking the original file-opening experience.

4.3 Database and File Management Design

File concealment in ShadowCrypt is built upon an encrypted, dictionary-based file mapping strategy and intelligent management of shortcut files. The design avoids traditional SQL/relational databases in favor of lightweight encrypted JSON-based structures, balancing simplicity, speed, and security.

4.3.1 Structure of mapping.db

The mapping.db file is not a true database but an AES-encrypted Python dictionary (JSON format) that tracks essential information about hidden files.

Primary Keys & Their Roles:

Key	**Type**	**Purpose**
mapping_table	dict	Maps the original file paths to hidden paths
hash_table	dict	Links a unique SHA256 hash to the original file path (used during shortcut launching)
hidden_dir	dict	Predefined system-safe directories used for storing hidden files
hidden_ext	list	File extensions like .dll, .exe, etc., used to disguise file types

Storage Format:

a) Serialized using json.dumps()

b) Encrypted using AESCipher.encrypt_json()

c) Stored as a Base64 string in enc_mapping.dll

4.3.2 Encrypted Storage

To prevent reverse engineering or tampering, the encrypted dictionaries are saved with .dll extensions (e.g., enc_mapping.dll) to reduce suspicion and block unintended access.

enc_mapping.dll

- Stores encrypted mapping of file paths, hashes, and system directories.
- Acts as the core vault for file hiding metadata.

enc_app_path.dll

- Stores encrypted information from app_paths.json.
- Contains application paths, icons, supported file extensions, and any custom arguments (like Photo Viewer DLL call).
- Helps launch the correct program when accessing hidden files.

Both files are decrypted only at runtime using a fixed AES key and never expose raw JSON to disk, reducing the risk of compromise.

4.3.3 Shortcut File Arguments and Linker Logic

Each hidden file is associated with a .lnk shortcut that points to linker.py (or linker.exe) with a specific hash argument.

.lnk File Structure:

- **Target:** Python executable (e.g., pythonw.exe linker.py)

- **Arguments:** --hash <unique_SHA256_hash>
- **Icon:** Extracted from app_paths.json based on the original file type

Linker Execution Flow:

1. User double-clicks the shortcut.
2. linker.py receives the hash via command-line.
3. It decrypts enc_mapping.dll, finds the hidden file path.
4. It retrieves the original extension.
5. It uses the matched application from enc_app_path.dll to open the file via subprocess.

This allows secure, seamless access without revealing any sensitive file paths.

4.4 Security Design

ShadowCrypt's entire approach is designed around minimizing attack surface, preventing reverse access, and ensuring confidentiality and control over file metadata and user interaction.

4.4.1 Password Validation and Retry Handling

- A fixed password is required to decrypt enc_mapping.dll and enc_app_path.dll.
- On startup, the tool checks for password validity using a test-decryption method.
- If decryption fails, an error is raised, and the user must re-enter the password.
- ShadowCrypt includes retry logic (typically 3 attempts), after which the operation is halted to avoid brute-force access.

Future extensions may include storing a hashed version of the password or supporting user-defined credentials.

4.4.2 Hash-Based File Access Control

Instead of storing direct file paths in shortcut files (which could expose metadata), ShadowCrypt uses a SHA256 hash of the original file path combined with a salt.

Purpose of Hashing:

- Prevents users or malware from guessing file locations
- Ensures that only the linker, when given the correct hash, can resolve access
- Allows secure, mapped launching via indirect association

Each hash is unique per file, and its presence in the shortcut is the only identifier that connects the user to the hidden file.

4.4.3 File Movement using Windows API

File movement for hiding and recovery leverages built-in OS functionality via shutil, os, and optionally the Windows Shell (via .bat scripts).

Key Features:

- Moves files into system directories under C:\Windows, such as:
- Help\mui, System32\Speech, etc.
- Avoids placing files under C:\Users (ransomware target zone)
- Ensures the files are renamed to extensions that are often ignored (.dll, .exe)

- Files are still readable by ShadowCrypt but are ignored by ransomware heuristics

This use of obscure **yet stable Windows paths,** combined with **non-indexed extensions,** makes the hiding mechanism both stealthy and resilient.

Chapter 5: Implementation

This chapter outlines the practical realization of the ShadowCrypt system, including key modules and working mechanisms. It elaborates on how the various components — such as the AES encryption engine, file handling logic, and shortcut-based access — work together seamlessly to offer secure file concealment and recovery. The system also supports deployment via an executable setup for easier end-user accessibility.

5.1 System Working

5.1.1 Initialization Module

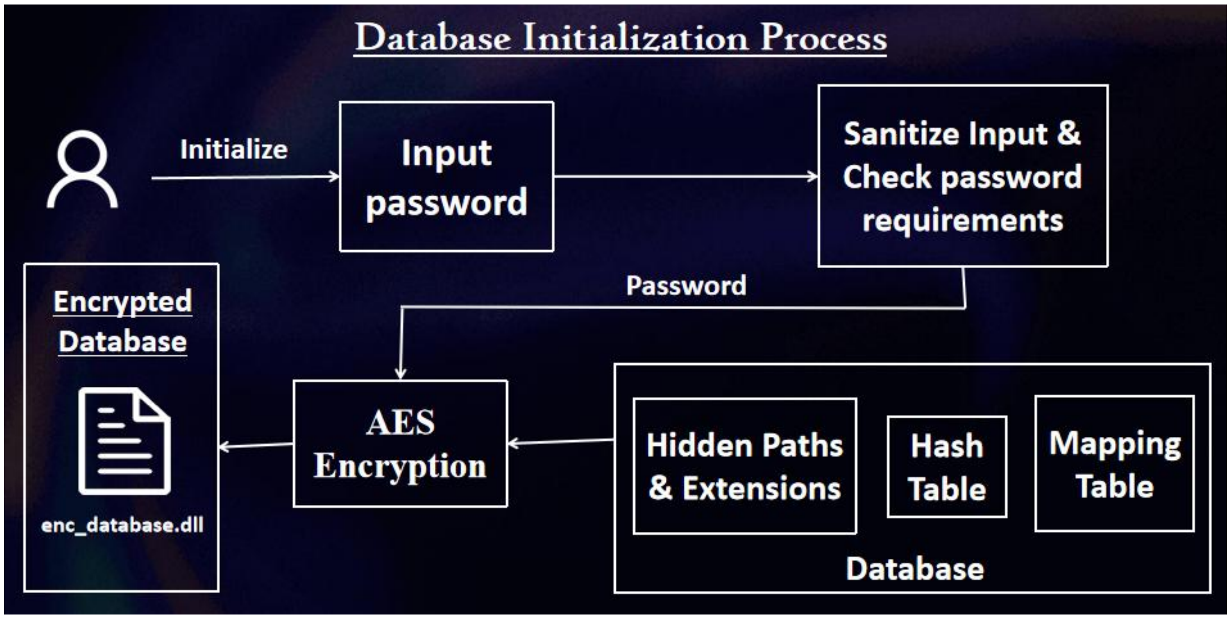


Figure 17: Working of Initialization Module

When ShadowCrypt is launched for the first time, it checks for the existence of the encrypted configuration files:

- **enc_mapping.dll:** Stores the encrypted file mapping and hash database.
- **enc_app_path.dll:** Stores application associations for launching files based on extension.

The init_db.py script is executed as part of the installation or first-run configuration. It asks the user to define a secure password and generates the encrypted versions of the application path file and mapping database. This ensures that all future operations like hiding, linking, or recovery work on encrypted datasets and maintain system integrity.

Steps:

1. If not found, default structures (mapping_table, hash_table, etc.) are created.
2. The Python dictionaries are serialized to JSON.
3. They are encrypted using AESCipher and saved to disk under .dll names to reduce visibility.
4. The initialization ensures compatibility with system paths and validates access privileges.

5.1.2 Hiding Module

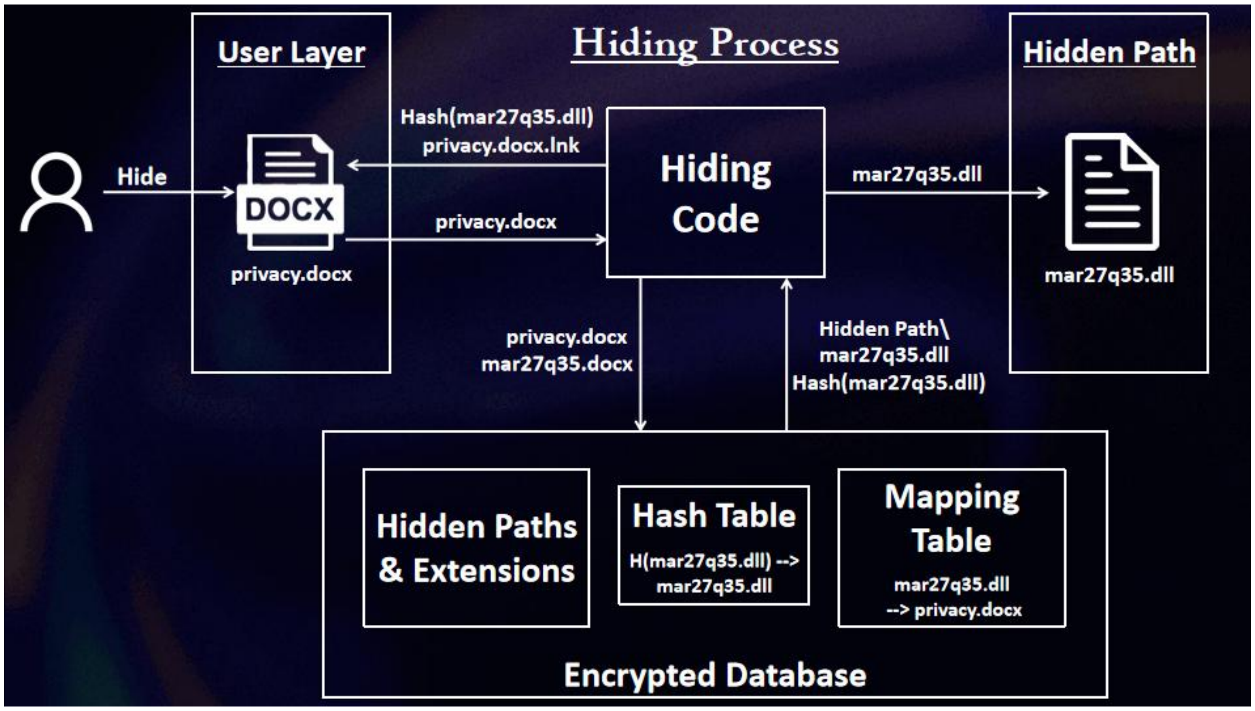


Figure 18: Working of Hiding Module

This is the core feature where selected user files are “hidden” from ransomware access.

Steps:

1. User triggers a hide operation via command-line, executable, or right-click menu.
2. The file path and extension are extracted.
3. A safe system directory is selected from hidden_dir in the database.
4. The file is renamed with an inconspicuous extension (e.g., .dll).
5. A unique SHA256 hash is generated based on the original file path and salt.

6. A .lnk shortcut is created, pointing to linker.py with the hash as argument.
7. The shortcut is placed at the original file location.
8. The mapping and hash data are encrypted and saved.

Key Features:

- Ensures file accessibility via shortcut while completely hiding the real data.
- Keeps the mapping data secure and non-traceable.

5.1.3 Shortcut Linking Module

Shortcuts created by the hiding module point to linker.py or linker.exe, and their role is to open the hidden file securely without exposing its location.

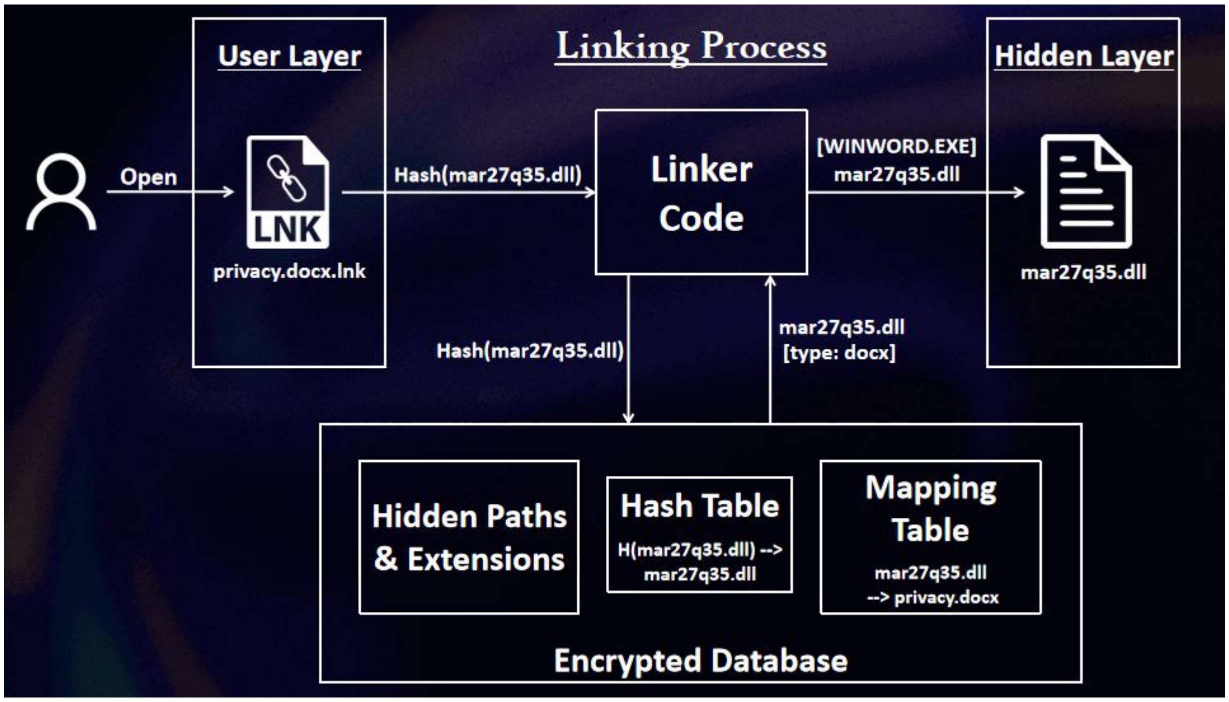


Figure 19: Working of Linking Module

Steps:

1. User opens the .lnk file.
2. It runs linker.py with the hash as an argument
3. The hash is used to:
   a) Look up the original file path in the encrypted mapping.
   b) Determine the hidden file path and the original extension.
4. The appropriate application (e.g., Notepad, VLC) is selected from app_paths.json.
5. The hidden file is launched with the correct program using subprocess.

5.1.4 Recovery Module

The recovery module restores the original file by reversing the concealment operation.

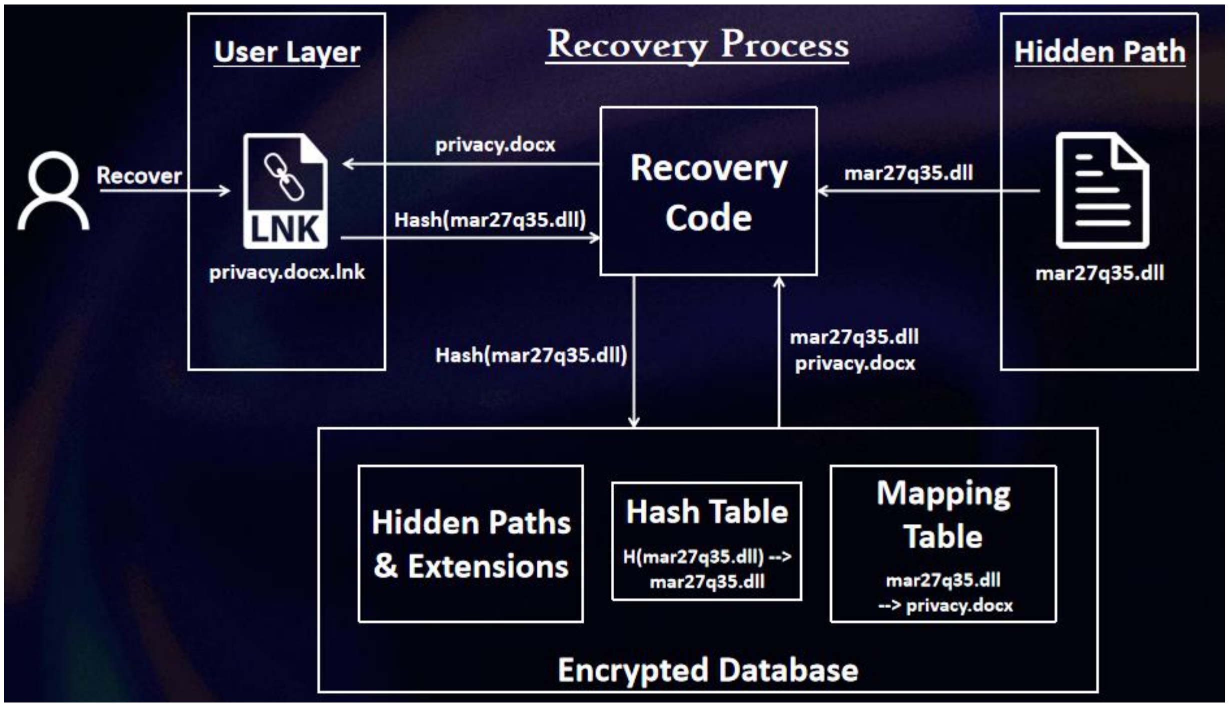


Figure 20: Working of Recovey Module

Steps:

1. User initiates recovery via command-line or right-click.
2. The user selects the .lnk file or provides the hash.
3. recovery.py decrypts the database and retrieves:
    a) Original path
    b) Hidden path
4. The hidden file is moved back to its original location.
5. The shortcut (.lnk) is deleted.
6. Mapping and hash entries are removed from the encrypted database.

Highlights:

- Recovery can also batch-process multiple files.
- Ensures complete restoration with no hidden trace left behind.

5.1.5 Security, Encryption, and Setup Deployment

Security mechanisms are woven into every part of the system:

AES Encryption:

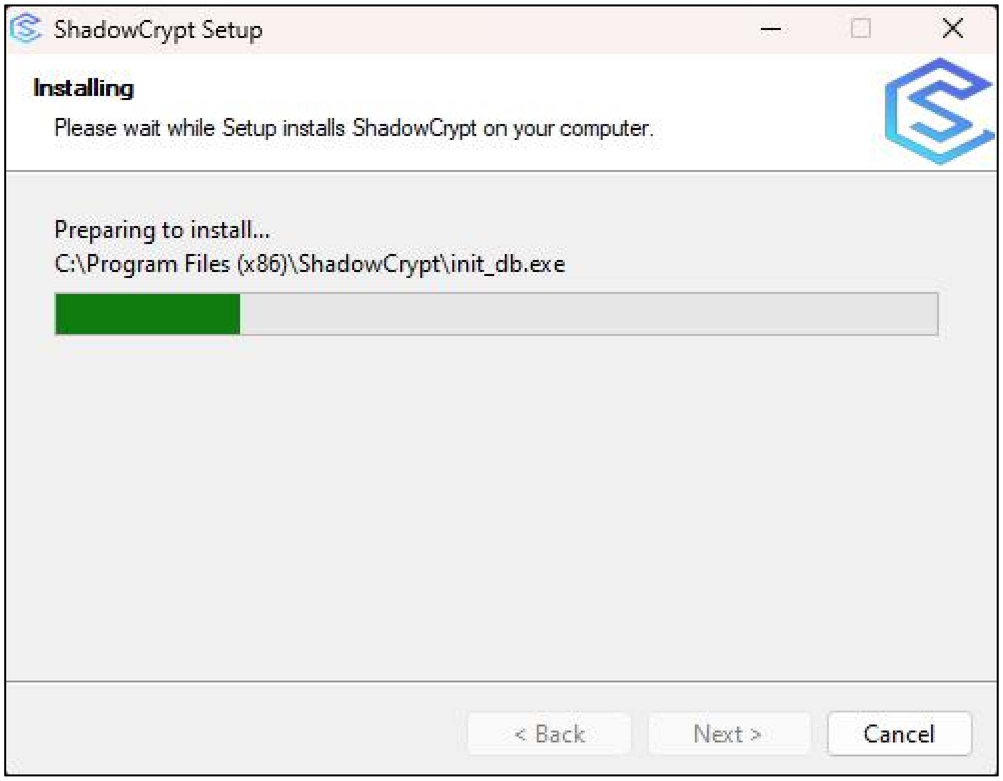
- Used to encrypt both mapping and application configuration.
- Padding and CBC mode ensure strong confidentiality.
- Keys are hashed with SHA256 internally before being used.

Hash-Based Access:

Ensures that even if a shortcut is exposed, it cannot be used to trace or brute-force the hidden file.

Deployment:

- ShadowCrypt can be compiled into a .exe using PyInstaller.
- A complete setup is provided on the GitHub Release Page. (github.com/Raqeeb27/ShadowCrypt)
- Registry scripts allow users to integrate right-click menu options for quick hiding/recovery.



Please wait while Setup installs ShadowCrypt on your computer.

Preparing to install...
C:\Program Files (x86)\ShadowCrypt\init_db.exe

Figure 21: Installation Process of ShadowCrypt

5.1.6 Right-Click Menu Integration

To enhance usability and streamline access, ShadowCrypt includes functionality to **integrate directly with the Windows right-click context menu.** This allows users to perform file hiding and recovery operations without launching the application manually or using command-line options.

This integration is achieved through two batch scripts:

- **Set-RightClick.bat** – Adds all necessary right-click menu entries and creates shortcuts under the “Send To” menu.
- **Remove-RightClick.bat** – Removes these entries and shortcuts, serving as a cleanup or uninstallation utility.

Features Added to the Right-Click Menu

The integration introduces the following contextual options for quick operations:

1. **Hide File (Create lnk File)**

   – Appears when right-clicking on any file

   – **Executes:**

   ShadowCrypt.exe hide --files "<file_path>"

   – Shows with ShadowCrypt icon and supports single selection

2. **Recover File (Extract File)**

   – Available on .lnk (shortcut) files only

   – **Executes:**

ShadowCrypt.exe recover --link_files "<shortcut_path>"

3. **Recover All (Extract All Files)**

   – Appears when right-clicking on an empty folder background

   – Recovers all files mapped in the database

   – **Executes:**

   ShadowCrypt.exe recover --all

4. **Recover Files in This Folder**

   – Recovers only those files that were hidden from the currently selected folder

   – **Executes:**

   ShadowCrypt.exe recover --dir "<current_folder>"

5. **Recover Files Recursively (Include Subfolders)**

   – Recovers hidden files from the selected folder and all its subdirectories

   – **Executes:**

   ShadowCrypt.exe recover --recursive --dir "<current_folder>"

SendTo Menu Shortcuts

The script also adds two .lnk shortcuts under the user's SendTo menu:

1. **Hide Selected Files**

   – Launches ShadowCrypt with the hide --files argument

2. **Recover Selected Files**

   – Launches ShadowCrypt with the recover --link_files argument

These shortcuts are created dynamically via a temporary VBScript that leverages the WScript.Shell interface.

5.1.7 Technical Implementation (Registry Edits)

Using reg add, the script modifies the Windows Registry at the following paths:

HKEY_CLASSES_ROOT\shell\Hide File*

HKEY_CLASSES_ROOT\Lnkfile\shell\Recover File

HKEY_CLASSES_ROOT\Directory\Background\shell\Recover All

HKEY_CLASSES_ROOT\Directory\Background\shell\Recover Files in This Folder

HKEY_CLASSES_ROOT\Directory\Background\shell\Recover Files Recursively

Each entry includes:

- A user-friendly label

- The command to invoke the executable with appropriate arguments
- An optional icon reference for visual clarity

The executable path is dynamically resolved using the %~dp0dist\ShadowCrypt.exe reference to ensure correct linking from any location.

5.1.8 Uninstallation and Cleanup

The Remove-RightClick.bat script safely removes all the above registry entries and deletes the .lnk shortcuts created in the SendTo menu. It provides a clean rollback for system environments where the integration is no longer needed.

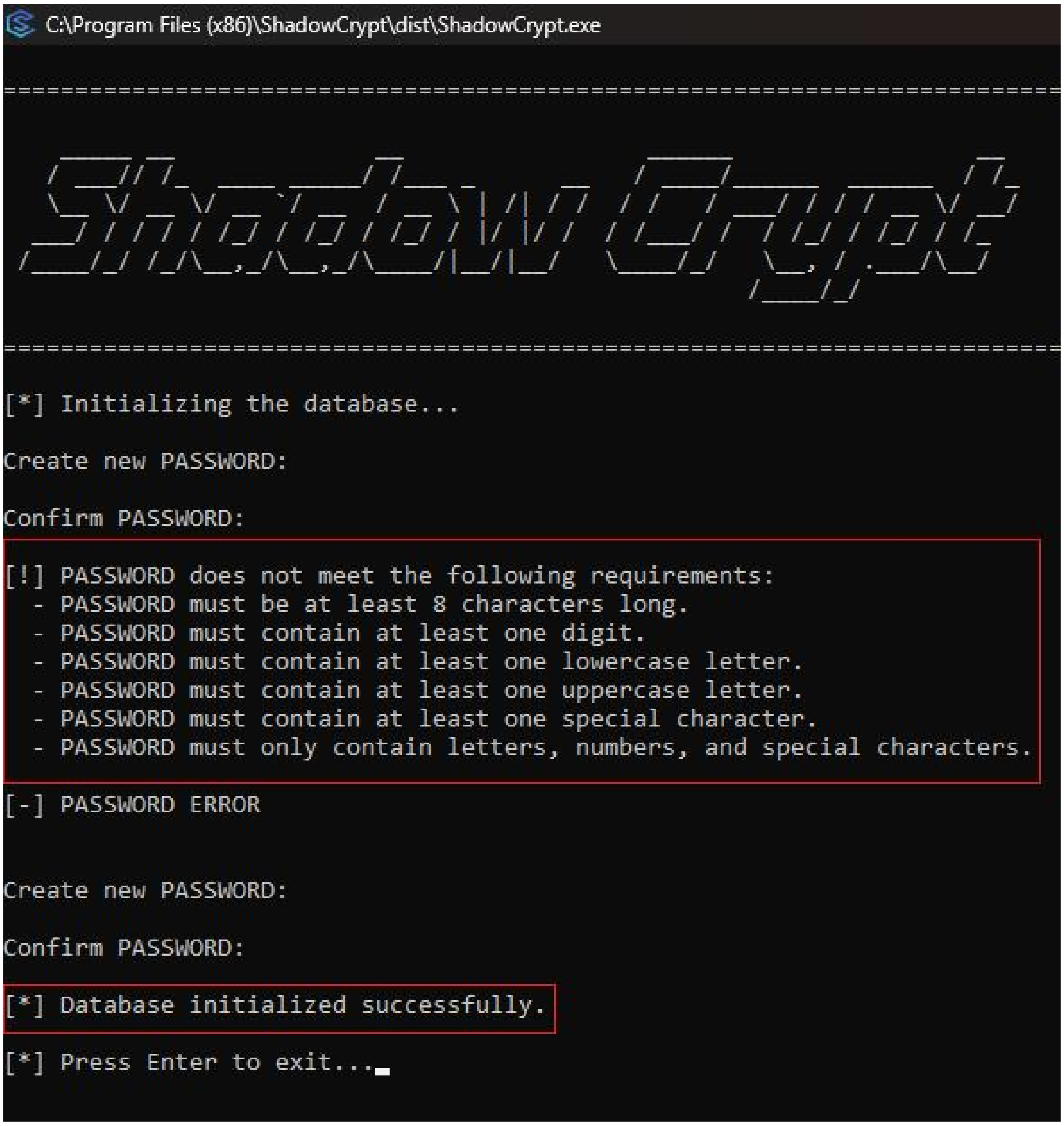
System Requirements and Execution Notes

- These .bat scripts must be run with administrator privileges, as they modify the Windows Registry.
- Integration is currently supported only on Windows OS.
- Users must have the ShadowCrypt executable (ShadowCrypt.exe) located under a dist folder relative to the script path.

5.2 Screenshots

This section provides illustrative snapshots demonstrating the system in action.

5.2.1 Database Initialization



```
C:\Program Files (x86)\ShadowCrypt\dist\ShadowCrypt.exe

==========================================================================

==========================================================================

[*] Initializing the database...

Create new PASSWORD:

Confirm PASSWORD:

[!] PASSWORD does not meet the following requirements:
  - PASSWORD must be at least 8 characters long.
  - PASSWORD must contain at least one digit.
  - PASSWORD must contain at least one lowercase letter.
  - PASSWORD must contain at least one uppercase letter.
  - PASSWORD must contain at least one special character.
  - PASSWORD must only contain letters, numbers, and special characters.
[-] PASSWORD ERROR


Create new PASSWORD:

Confirm PASSWORD:

[*] Database initialized successfully.

[*] Press Enter to exit...
```

Figure 22: Screenshot-1: Initialization Process

- Prompts for password and validates against the minimum password requirements.
- Creates the encrypted database files.

5.2.2 File Hiding

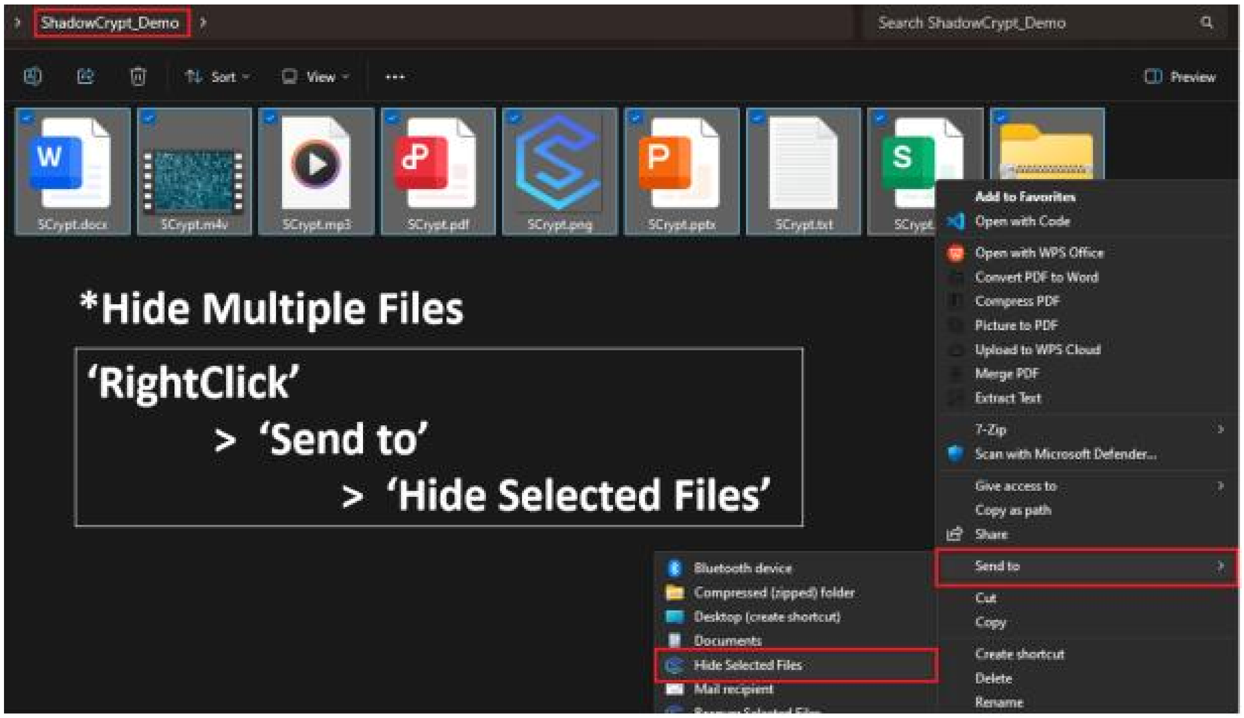
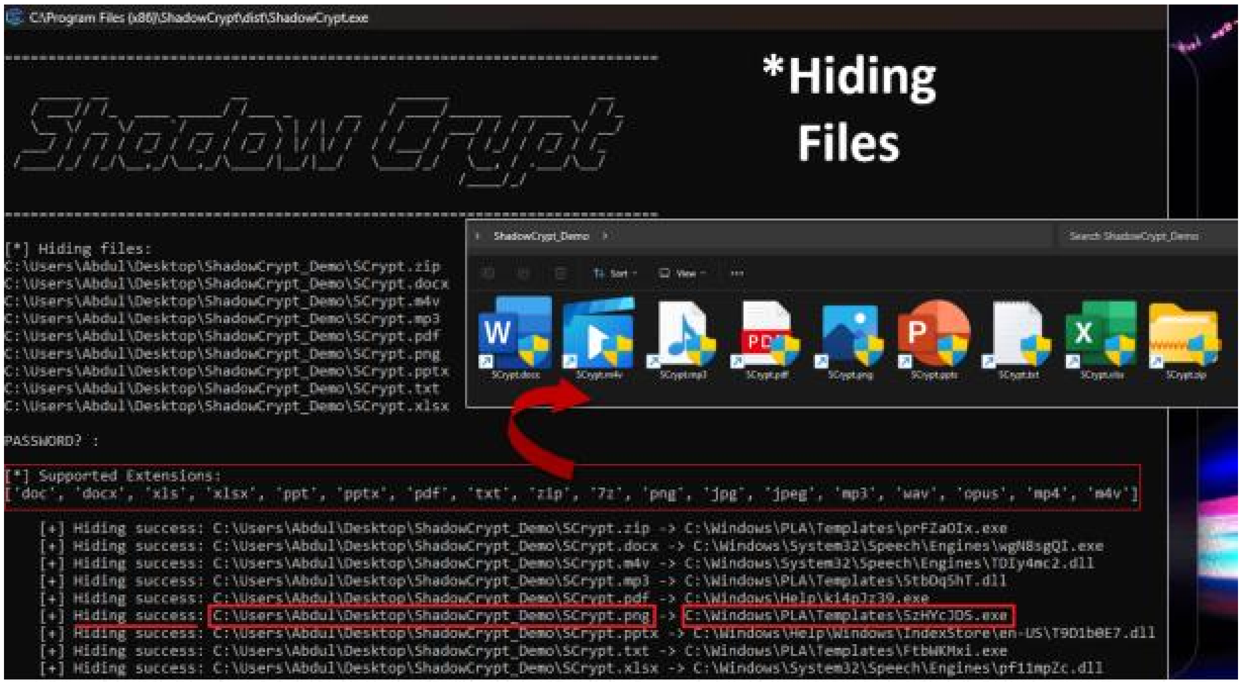


Figure 23: Screenshot-2: Selecting Files for Hiding

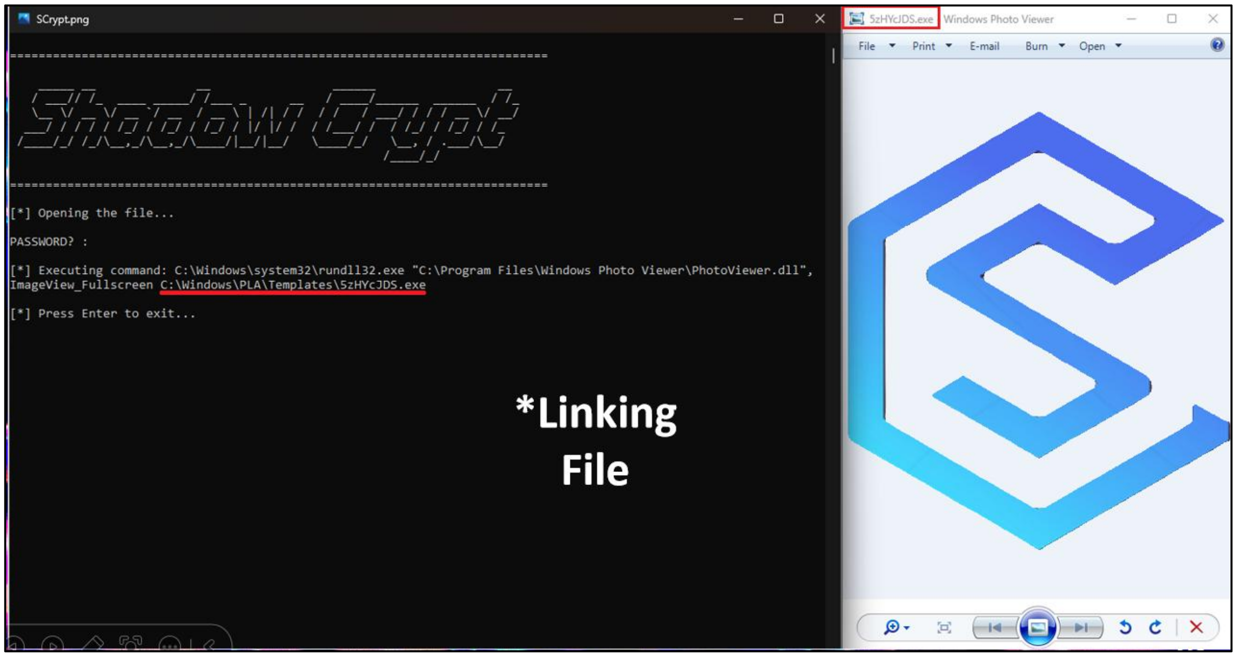


```
C:\Program Files (x86)\ShadowCrypt\dist\ShadowCrypt.exe

[*] Hiding files:
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.zip
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.docx
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.m4v
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.mp3
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.pdf
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.png
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.pptx
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.txt
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.xlsx

PASSWORD? :

[*] Supported Extensions:
['doc', 'docx', 'xls', 'xlsx', 'ppt', 'pptx', 'pdf', 'txt', 'zip', '7z', 'png', 'jpg', 'jpeg', 'mp3', 'wav', 'opus', 'mp4', 'm4v']

    [+] Hiding success: C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.zip -> C:\Windows\PLA\Templates\prFZaOIx.exe
    [+] Hiding success: C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.docx -> C:\Windows\System32\Speech\Engines\wgN8sgQI.exe
    [+] Hiding success: C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.m4v -> C:\Windows\System32\Speech\Engines\TDIy4mc2.dll
    [+] Hiding success: C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.mp3 -> C:\Windows\PLA\Templates\StbOq5hT.dll
    [+] Hiding success: C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.pdf -> C:\Windows\Help\k14pJz39.exe
    [+] Hiding success: C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.png -> C:\Windows\PLA\Templates\SzHYcJDS.exe
    [+] Hiding success: C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.pptx -> C:\Windows\Help\Windows\IndexStore\en-US\T9D1b0E7.dll
    [+] Hiding success: C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.txt -> C:\Windows\PLA\Templates\FtbWKMxi.exe
    [+] Hiding success: C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.xlsx -> C:\Windows\System32\Speech\Engines\pf11mpZc.dll
```

Figure 24: Screenshot-3: Hiding Seleted Files

- Displays file selection followed by the file being moved to a system directory.
- Shortcut replaces the original file with a .lnk and matching icon.

5.2.3 Shortcut Usage



```
[*] Opening the file...

PASSWORD? :

[*] Executing command: C:\Windows\system32\rundll32.exe "C:\Program Files\Windows Photo Viewer\PhotoViewer.dll",
ImageView_Fullscreen C:\Windows\PLA\Templates\5zHYcJDS.exe

[*] Press Enter to exit...
```

Figure 25: Screenshot-4: Linking/Opening a Hidden File

- User double-clicks the .lnk file.
- The correct application opens the hidden file seamlessly, as if the original file never moved.

5.2.4 File Recovery

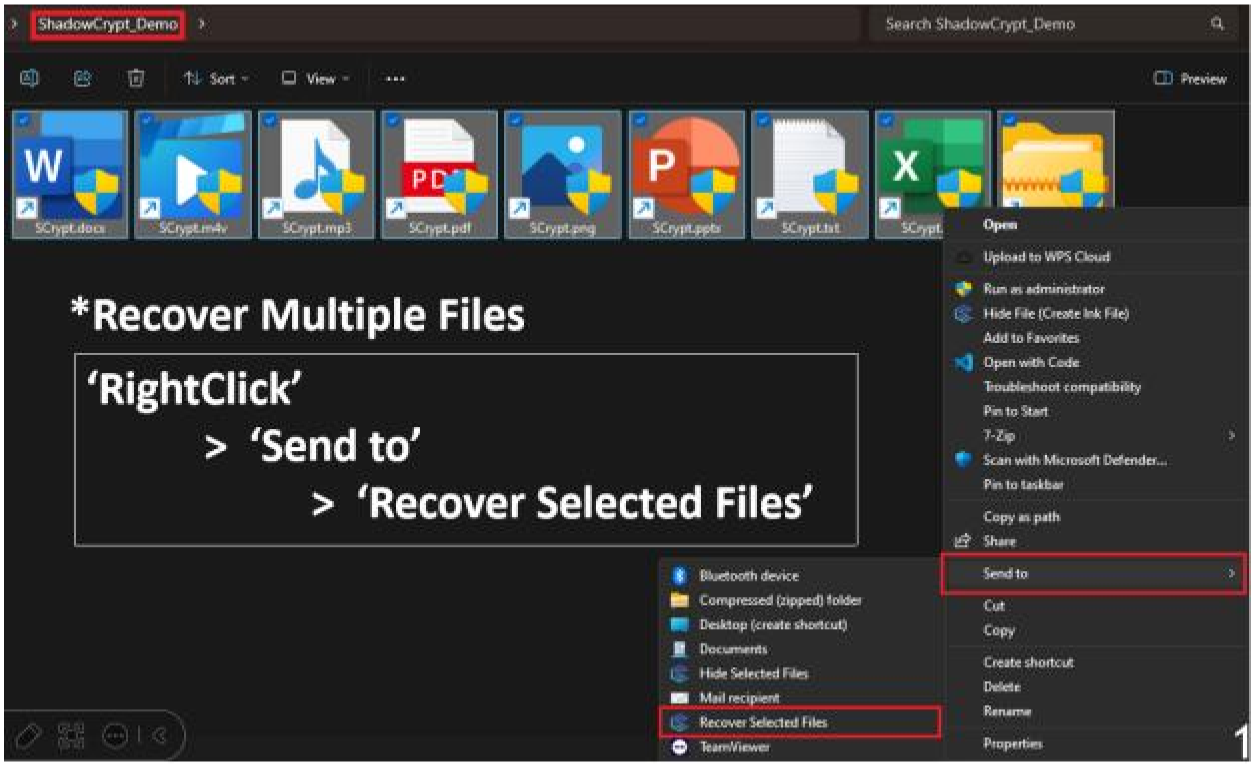
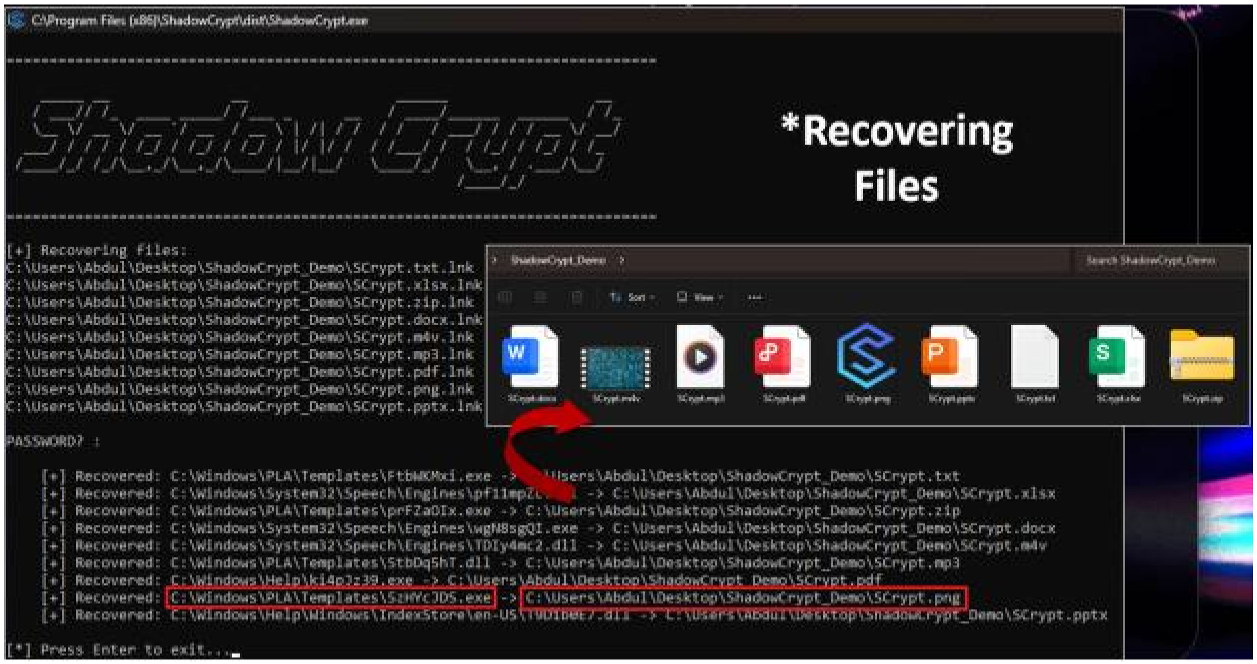


Figure 26: Screenshot-6: Selecting Files for Recovery



```
[+] Recovering files:
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.txt.lnk
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.xlsx.lnk
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.zip.lnk
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.docx.lnk
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.m4v.lnk
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.mp3.lnk
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.pdf.lnk
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.png.lnk
C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.pptx.lnk

PASSWORD? :

    [+] Recovered: C:\Windows\PLA\Templates\FtbWKMxi.exe -> [illegible]Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.txt
    [+] Recovered: C:\Windows\System32\Speech\Engines\pf11mpZ[illegible] -> C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.xlsx
    [+] Recovered: C:\Windows\PLA\Templates\prFZaOIx.exe -> C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.zip
    [+] Recovered: C:\Windows\System32\Speech\Engines\wgN8sgQI.exe -> C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.docx
    [+] Recovered: C:\Windows\System32\Speech\Engines\TDIy4mc2.dll -> C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.m4v
    [+] Recovered: C:\Windows\PLA\Templates\StbDq5hT.dll -> C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.mp3
    [+] Recovered: C:\Windows\Help\ki4pJz39.exe -> C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.pdf
    [+] Recovered: C:\Windows\PLA\Templates\SzHYcJDS.exe -> C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.png
    [+] Recovered: C:\Windows\Help\Windows\IndexStore\en-US\[illegible].dll -> C:\Users\Abdul\Desktop\ShadowCrypt_Demo\SCrypt.pptx

[*] Press Enter to exit...
```

Figure 27: Screenshot-7: Recovering Selected Files

- Demonstrates a recovered file appearing back at its original location.
- Confirms deletion of the .lnk and cleanup of the database entry.

Chapter 6: Conclusion

6.1 Summary of the Project

ShadowCrypt offers a practical, lightweight, and proactive solution to a growing cybersecurity concern — ransomware. Instead of relying on post-infection detection and recovery, ShadowCrypt takes a **preventive concealment approach** by relocating user files to obscure system directories not commonly targeted by ransomware. It ensures continued accessibility via dynamically generated shortcut files that securely launch hidden files through an AES-encrypted mapping mechanism.

The system includes:

- Encrypted mapping databases,
- Hash-based shortcut generation,
- Intelligent file renaming and relocation,
- Seamless integration with system applications and user interface (right-click menus and .exe).

This unique architectural blend of **security through obscurity, data encryption, and user transparency** provides a strong foundation for real-world usage, especially for users with minimal technical expertise.

6.2 Applications

ShadowCrypt can be employed in several scenarios where preemptive data security is paramount:

- **Personal Data Protection:** Users can hide sensitive documents, images, and media to protect them from ransomware, shared device misuse, or accidental modification.

- **Enterprise Workstations:** In corporate environments, ShadowCrypt can serve as a lightweight complement to antivirus software, shielding critical config or license files on employee systems.
- **Research & Development Files:** Developers or researchers working on confidential projects can use ShadowCrypt to protect project files without needing encrypted virtual volumes.
- **Ransomware Simulation Testing:** The application can be used in cybersecurity labs to simulate how concealment affects real ransomware behaviors like directory scanning and heuristic analysis.

6.3 Limitations

While ShadowCrypt is highly effective against many ransomware types and threat vectors, it is important to acknowledge certain limitations:

1. Vulnerability to Full-Disk Encryption Ransomware

Advanced ransomware variants (e.g., NotPetya or enterprise-targeted strains) may encrypt the entire disk or partition — in such cases, even hiding files in protected system directories becomes ineffective.

2. Shortcut-Based Access Limitations

If the .lnk file is moved, renamed, or deleted without knowledge of the hash or mapping, the file may become temporarily inaccessible unless the mapping is manually retrieved.

3. Manual File Copying

The current implementation does not support automatic copying of hidden files when a .lnk file is duplicated. Users must:

- Recover the file,
- Copy it manually,
- Re-hide both the original and copied versions.

This is not intuitive and can lead to unintentional exposure.

4. Windows-Specific Implementation

The design is tightly coupled with the Windows OS, relying on .lnk files, registry edits, and system directory structures. Porting the system to Linux or macOS would require substantial architectural changes.

5. Static Configuration Paths

app_paths.json uses hardcoded paths to external programs like LibreOffice, VLC, etc., which can break if installed elsewhere or uninstalled.

6.4 Future Scope

To enhance ShadowCrypt and extend its utility, the following improvements and features are proposed:

1. Intelligent Shortcut Copy Functionality

Implement a mechanism where copying a .lnk file also replicates the corresponding hidden file in a new mapped location. This may include:

- Monitoring file copy events,
- Using a custom .lnk handler,
- Auto-generating a new mapping entry and encrypted hash for the copied file.

This would greatly improve usability while maintaining concealment.

2. Full Integration with Encrypted Virtual Environments

As ransomware becomes more sophisticated, a future version of ShadowCrypt could include optional encrypted volumes or shadow drives to store hidden files — adding another layer of encryption over the existing one.

3. GUI with Drag-and-Drop Support

Introduce a user-friendly interface with features like:

- Drag-and-drop hiding
- Visual mapping list
- Password and log management

This would make the tool accessible to non-technical users.

4. Real-time File Monitoring

Enhance the tool to monitor protected directories and notify users if unauthorized attempts are made to access, modify, or delete hidden files — combining proactive and reactive defense strategies.

5. Cloud Sync Awareness

Incorporate support for cloud platforms (e.g., Google Drive, OneDrive) to avoid syncing shortcut files or hidden directories inadvertently, which may compromise file security or create conflicts.

REFERENCES

1) S. Lee, S. Lee, J. Park, K. Kim, and K. Lee, "Hiding in the Crowd: Ransomware Protection by Adopting Camouflage and Hiding Strategy With the Link File," IEEE Access, vol. 11, pp. 92693–92704, 2023.
   doi: 10.1109/ACCESS.2023.3309879

2) National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES), FIPS PUB 197, 2001.
   https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

3) Microsoft Docs. Windows Shortcut (.lnk) File Format.
   https://learn.microsoft.com/en-us/windows/win32/shell/links

4) Microsoft Docs. Windows System Directory Structure.
   https://learn.microsoft.com/en-us/windows/win32/sysinfo/what-s-new-for-windows-system-information

5) Symantec Threat Intelligence. Ransomware: Past, Present, and Future Threats.
   https://symantec-enterprise-blogs.security.com/