Volume: 13 Issue: VII Month of publication: July 2025

DOI: https://doi.org/10.22214/ijraset.2025.73455

ShadowCrypt: Bridging the Usability Gap in Link - Based Ransomware Defense through Seamless Copying of Protected Files

Mohammed Abdul Raqeeb[1], Mohammed Fasiuddin Arsalaan[2], Abdul Samad[3], Mohammed Asjad[4]
[1, 2, 3, 4]*Department of Computer Science, Osmania University Hyderabad*

Abstract: Ransomware continues to pose a significant threat to user data, often rendering files irrecoverable even after a ransomware infection has been identified. Existing solutions primarily focus on detection or recovery through backups, which are reactive and often insufficient against advanced threats. This paper presents ShadowCrypt, a proactive defense system that protects sensitive files by rendering them invisible to ransomware. Rather than relying on signature-based detection or behavioral heuristics, ShadowCrypt utilizes a novel combination of file redirection, extension obfuscation, metadata encryption, and shortcut-based access to make files undiscoverable and unmodifiable. Once protected, files are relocated to system-reserved, non-indexed directories, renamed with benign extensions, and replaced with hashed shortcuts that enable seamless user access without revealing the actual location or identity of the data. The system maintains integrity and usability through a secure encrypted database and offers a resilient recovery framework—even in the aftermath of ransomware attacks that delete access points. To address usability limitations inherent in shortcut-based architectures, ShadowCrypt introduces a clipboard monitoring mechanism that allows users to securely copy and paste protected files, enabling true duplication without compromising security. Experimental results demonstrate negligible performance overhead and high reliability across a range of attack scenarios. ShadowCrypt effectively bridges the gap between robust protection and practical usability, positioning itself as a forward-thinking model for future ransomware-resilient systems.
Keywords: Ransomware Defense, File Concealment, Pre-emptive Protection, Clipboard Monitoring, Camouflage-based Security, Shortcut-based Access, Secure File Recovery

I. INTRODUCTION

The escalating prevalence of ransomware has made it one of the most formidable cybersecurity threats in both personal and enterprise environments. Unlike traditional malware, ransomware deliberately targets user data, encrypts it, and demands a ransom for restoration—often leaving victims with limited options for recovery. Modern ransomware variants employ sophisticated evasion techniques, fileless execution, and delayed activation to bypass conventional security layers such as antivirus software, behavior-based detectors, and access control mechanisms. These attacks are not only increasing in frequency but are also growing in technical complexity, rendering many existing defenses either ineffective or insufficient in high-risk scenarios.

Conventional ransomware defense mechanisms fall largely into reactive categories. These include behavioral anomaly detection systems, static or dynamic malware analysis, backup-based recovery frameworks, and network-level domain blocking. While valuable, such approaches depend on post-infection mitigation and often require prior knowledge of attack signatures, or rely on the assumption that backup systems remain uncompromised. Moreover, many solutions demand significant configuration effort, system-level monitoring, or impose restrictions that compromise user experience and usability. As a result, there exists a growing need for ransomware protection strategies that are not only effective and preemptive but also minimally intrusive and operationally seamless.

This paper introduces a novel, proactive defense mechanism, ShadowCrypt, that focuses on file invisibility rather than detection or recovery after compromise. By hiding sensitive user files in obscure, system-reserved directories and disguising them with system-native extensions typically ignored by ransomware (e.g., .dll, .exe), our proposed system effectively removes these files from the ransomware's field of view. The system maintains a secure mapping of these concealed files via encrypted metadata and grants user access through hash-based, launcher-driven shortcuts. This strategy ensures that files remain functional and accessible to the user while being practically undiscoverable and unreachable to malware, even if the ransomware has already penetrated the system.

To ensure practicality, the proposed architecture emphasizes usability through familiar interaction paradigms such as right-click context menu access, seamless file opening through shortcut redirection, and most notably, clipboard-based duplication support. This latter enhancement enables users to copy and paste protected files using traditional keyboard shortcuts (Ctrl+C, Ctrl+V), where a background service intercepts and securely duplicates both the hidden file and its associated launcher. The system also provides robust recovery capabilities, allowing protected files to be restored selectively or in bulk—even when ransomware has damaged the file system or deleted access points. Additionally, intelligent recovery heuristics detect signs of ransomware compromise (e.g., files with suspicious extensions) and adapt recovery behavior accordingly by redirecting output to a secure backup folder.
In summary, the proposed system, ShadowCrypt, addresses key limitations in current ransomware defenses by offering a preemptive, concealment-based protection model that integrates cryptographic security with user-centric design. It aims to strike a balance between strong file protection and practical usability, ensuring that sensitive data remains both invisible to attackers and effortlessly accessible to users.

II. LITERATURE SURVEY

Lee et al. (2023) propose a camouflage-based, link-file strategy for ransomware protection, where original files are hidden and replaced with launcher-based .lnk shortcuts [1]. Their system achieves effective concealment by redirecting access through controlled launcher execution, ensuring that sensitive data is not directly exposed. However, it still leaves usability gaps—specifically around file duplication and shortcut lifecycle management—that can hinder seamless interaction. ShadowCrypt: Bridging the Usability Gap builds on this foundation by adding a clipboard-integrated, secure copy mechanism to resolve these limitations. Urooj et al. (2022) survey dynamic-analysis and machine learning techniques for ransomware detection across diverse file and behavior traits [2]. Their work highlights the promise of ML-based systems in identifying malicious activity by analyzing runtime features and system calls. However, such methods remain probabilistic, sensitive to evasion, and often generate false positives or require expensive feature training. ShadowCrypt diverges by eliminating visibility of target files entirely, reducing reliance on behavior modeling.
Reidys et al. (2022) introduce a hardware-isolated, network-storage based defense mechanism (RSSD) that isolates storage and allows post-attack analysis for ransomware recovery [3]. Their approach ensures that even after compromise, system integrity is preserved through hardware-enforced isolation and secure recovery codes. While powerful for enterprise environments, it requires specialized infrastructure and integration beyond standard operating systems. Von der Assen et al. (2023) unveil MTFS (Moving Target Defense–Enabled File System), which dynamically relocates, renames, and obfuscates files to confuse ransomware scanning heuristics [4]. Their approach systematically disrupts static attacker assumptions by continuously altering file placement and metadata. While innovative, it places emphasis on dynamic filesystem behavior rather than persistent usability features. ShadowCrypt aligns with the MTD principle—redirecting and obfuscating files—but enhances user accessibility with hash-based shortcuts and GUI-driven recovery.
Çalışkan et al. (2024) provide a broad survey on contemporary ransomware detection trends and behavioral-analysis techniques, emphasizing vulnerabilities in static signature-based defenses [5]. They document how rapidly mutating ransomware strains evade conventional detection by modifying behavior patterns. Although these methods are useful for attack prediction, they are inherently reactive and may fail against unseen variants. Oz et al. (2022) present an authoritative taxonomy of ransomware evolution, attack vectors, and defense strategies, spanning privilege escalation, filesystem access, and static extension targeting [6]. Their work underscores the inadequacies of purely signature-based or forensic approaches in the face of rapidly evolving threats. ShadowCrypt's architecture-level concealment mitigates these risks by making targeted files structurally inaccessible to ransomware.
Khan et al. (2022) propose a moving target defense framework that dynamically modifies filesystem layout to prevent malware from identifying sensitive files [7]. Their approach increases uncertainty for attackers by unpredictably renaming and relocating files. Sudheer (2024) offers a systematic review of ransomware incidents and evolving attack mechanisms, identifying trends such as fileless attacks, hybrid threats, and multi-layer encryption strategies [8]. This review reinforces the need for defense solutions that operate independent of ransomware signature or behavior. ShadowCrypt's invisibility-first strategy addresses these evolving threats by eliminating the attack surface, rather than reacting post-infection.
Singh et al. (2022) explore ransomware detection through process memory analysis, leveraging in-memory patterns to identify malicious behavior before disk encryption occurs [9]. Their approach demonstrates fast detection potential, but it remains probabilistic and dependent on transient memory behavior. In contrast, ShadowCrypt avoids prediction altogether by structurally concealing files and resolving them only when required through encrypted mappings. Sheen et al. (2022) introduce R-Sentry, a deception-based detection framework that deploys honey files to detect ransomware via file access traps [10].

The system activates defensive responses when decoy files are targeted. While effective in early detection, it remains inherently reactive and engages with the attacker rather than avoiding attack paths. ShadowCrypt eliminates the need for engagement by keeping actual sensitive files hidden—never visible for attackers to interact with.

Huertas et al. (2022) propose a reinforcement learning–driven framework for dynamically selecting moving target defense strategies in IoT environments [11]. Their approach exploits unpredictability to thwart zero-day ransomware by dynamically changing system structure. Von der Assen et al. (2024) extend MTFS in an IEEE LCN paper, exploring file access delay tactics and metadata decoupling to mitigate ransomware without impacting normal workflows [12]. This refined MTD strategy reinforces the principle that proactive structural disruption enhances security. ShadowCrypt aligns with this architecture and further contributes an integrated user experience, combining obfuscated concealment with seamless shortcut access. Commey et al. (2024) present EGAN, a GAN-based adversarial model that mutates ransomware behavior to bypass traditional detectors [13]. Their work underscores the increasing sophistication of ransomware and the futility of relying on signature-based or behavioral defenses. ShadowCrypt's concealment-centered design works regardless of attack signature or behavior, maintaining protection even in adversarial conditions.

III. SHADOWCRYPT METHODOLOGY AND ARCHITECTURE

All ShadowCrypt is a proactive ransomware defense system designed to conceal sensitive user files in a manner that prevents them from being discovered, accessed, or encrypted by ransomware—even after such malware has breached the system. Rather than relying on post-attack mitigation strategies like backups or restoration points, ShadowCrypt operates on the principle of pre-emptive file invisibility, utilizing an intelligent architecture of file redirection, extension obfuscation, metadata encryption, and shortcut-based access. It assumes the presence of active ransomware and instead focuses on ensuring that critical files simply cannot be found or targeted, thereby minimizing attack surfaces and eliminating exposure at its root.

A. Initialization Phase and Secure Configuration

The system begins with an initialization phase, typically triggered during the first execution or installation of the software. At this stage, the user is prompted to provide a secure password. This password is hashed using the SHA-256 algorithm and is used to derive a symmetric encryption key for the system. This key is then employed to encrypt ShadowCrypt's internal configuration files using AES-256 encryption in CBC mode, ensuring strong confidentiality and integrity of all sensitive metadata. Among these encrypted files, the most critical is the mapping.db, which is conceptually divided into two main components: the mapping table and the hash table. The mapping table stores a one-to-one mapping from the absolute hidden file path to its original file path, enabling reliable restoration and reverse resolution during recovery operations. In parallel, the hash table stores a mapping from a secure hash of the hidden file path to the hidden path itself, which allows shortcut files to indirectly reference concealed files without exposing their true locations. Alongside this, the system also encrypts the app_paths.json configuration, which maintains a mapping between file extensions (e.g., .docx, .mp4, .pdf) and their associated applications—specifically including the executable path, supported extensions, and icon metadata. This structure ensures that when a hidden file is accessed via a shortcut, it is launched with the correct application in a seamless and transparent manner, preserving the original user experience without compromising security.

1) Psuedocode: Initialize ShadowCrypt:

```
Prompt user for password
Derive AES-256 encryption key using SHA-256
Encrypt mapping.db and app_paths.json
```

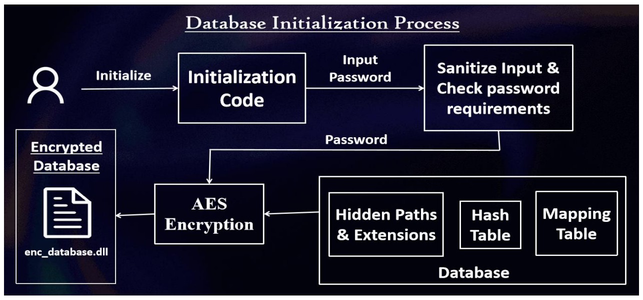

Figure 1: Initialization and Database Setup Process. This diagram outlines ShadowCrypt's initial setup. Upon receiving a secure user password, the system creates an AES-256 encrypted configuration containing hash tables, mapping records, and app associations, establishing the foundation for future hiding and recovery operations.

B. File Concealment Logic and Directory Management

Once initialized, ShadowCrypt enters its active usage phase, where users can begin protecting their files. When a user initiates a file-hiding operation—either via a command-line interface, graphical .exe tool, or right-click context menu—the system performs a series of transformations. First, it selects a target directory for concealment from a predefined list of system-safe locations that are not indexed, are typically ignored by ransomware, and are persistent across Windows updates. Examples include obscure folders under C:\Windows\Help\mui or C:\Windows\System32\IME\SHARED. Then, the file is renamed with a randomly generated identifier (often hash-based) and is assigned a harmless, system-reserved extension such as .dll or .exe. These extensions are specifically chosen because ransomware commonly avoids interacting with such file types to prevent breaking the operating system. The file, now camouflaged, is moved into one of the preselected system directories intended for concealment. At this point, two critical entries are created in the encrypted internal database: first, in the mapping table, the system stores a direct mapping from the absolute path of the hidden (moved) file to its original file path—this ensures accurate restoration and traceability. Second, in the hash table, it computes a secure hash of the hidden file path and maps that hash value to the actual hidden path. This indirection allows the system to later resolve shortcut requests (which only contain the hash) without ever exposing the true file location, maintaining both data integrity and concealment security throughout the system's operation.

1) *Psuedocode: HideFile(original_path):*

```
hidden_path ← SelectSystemSafeDirectory() + RandomFileName() + RandomExt()
Move original_path → hidden_path
hash ← SHA256(hidden_path)
Store (hidden_path → original_path) in mapping table
Store (hash → hidden_path) in hash table
CreateShortcut(original_path, target=LinkerScript, args=hash, icon=ext_icon)
```

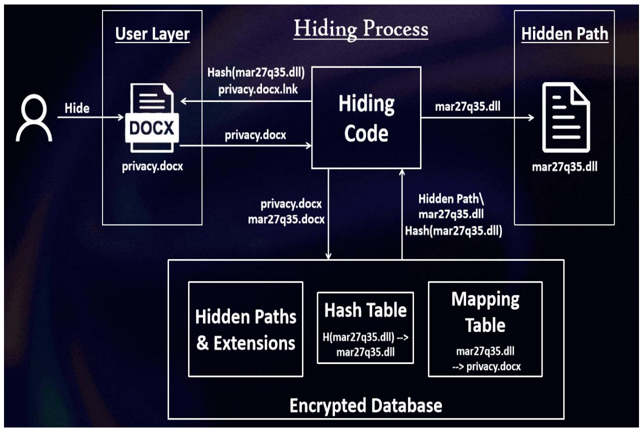

Figure 2: File Hiding and Obfuscation Procedure. The hiding process begins with the user selecting a file for protection. ShadowCrypt renames and moves the file to a non-indexed, system-reserved directory, maps it via encrypted tables, and replaces it with a launcher-based shortcut at its original location.

C. Shortcut-Based Seamless Access Mechanism

However, merely hiding the file is insufficient for usability. Users must still be able to access these protected files without needing to manually decrypt or recover them. To enable this, ShadowCrypt creates a custom .lnk (shortcut) file and places it at the original file's location.

The shortcut doesn't reveal the true path of the file. Instead, it points to a launcher script (or its .exe equivalent), passing a secure hash as a command-line argument. This hash is mapped to the hidden file's path inside the encrypted database. Thus, when a user opens the shortcut, the launcher decrypts the mapping, resolves the hash to the concealed file, determines the original file type, looks up the correct application to open it with, and finally launches the hidden file via subprocess, providing a completely transparent experience to the user. The original file appears to open normally, yet its physical location and metadata remain invisible to ransomware or external software.

1) *Psuedocode: OpenFile(shortcut_path):*

```
hash ← ExtractArgumentFromShortcut(shortcut_path)
Decrypt mapping.db
hidden_path ← GetFromHashTable(hash)
app ← GetAppFromAppPaths(hidden_path)
LaunchFile(hidden_path, with=app)
```

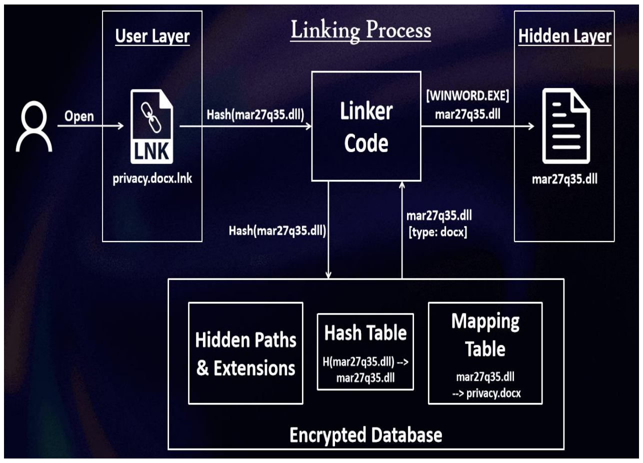

Figure 3: Shortcut Linking and File Opening Workflow. This flow depicts how a user accesses a concealed file via its shortcut. The linker code resolves the hash to the hidden file, identifies the appropriate application from the encrypted app configuration, and launches the file transparently while preserving invisibility to ransomware.

D. Robust and Heuristic-Driven File Recovery

ShadowCrypt incorporates a highly flexible and robust recovery mechanism designed to restore protected files safely and intelligently, regardless of the user's situation—whether it be routine access or during the aftermath of a ransomware attack. Users can recover either a single file or multiple selected files, and this operation can be triggered through manual invocation, context menu options, or command-line flags. The recovery system supports targeted recovery—for example, restoring only the files in the current working directory, or performing a recursive scan and recovery within a directory, including all its subfolders. For larger-scale operations, the system offers a full-drive recovery mode, scanning all accessible drives to identify and restore all hidden files managed by ShadowCrypt.

1) *Psuedocode: RecoverFile(shortcut_path):*

```
hash ← ExtractArgumentFromShortcut(shortcut_path)
Decrypt mapping.db
hidden_path ← GetFromHashTable(hash)
original_path ← GetFromMappingTable(hidden_path)
Move hidden_path → original_path
Delete shortcut_path
Remove entries from both tables
```

When recovering from link files, the system reads the hash stored in each .lnk file’s arguments, decrypts the database, and locates the corresponding hidden path. The hidden file is then safely moved back to its original location with its original filename and extension, after which the shortcut is deleted and the mapping and hash entries are removed from the database. However, ShadowCrypt is also designed to handle cases where the link files are no longer available—such as after a ransomware attack that may have encrypted or deleted .lnk files. In such cases, the system uses the encrypted mapping database alone to recover the files, placing them into a secure backup directory located at C:\Windows\ShadowCrypt_{Username}_Backup. This ensures that even if the original access points are compromised or corrupted, the user’s data is not lost.

2) Psuedocode: RecoverAll():

```
For each hidden_path in mapping table:
    If shortcut exists:
        Perform RecoverFile(shortcut_path)
    Else If shortcut does not exist:
        Move hidden_path → BackupDirectory
    Else If suspicious files (.lockbit, .enc, etc.) found in user folders:
        Recover all hidden files to BackupDirectory
```

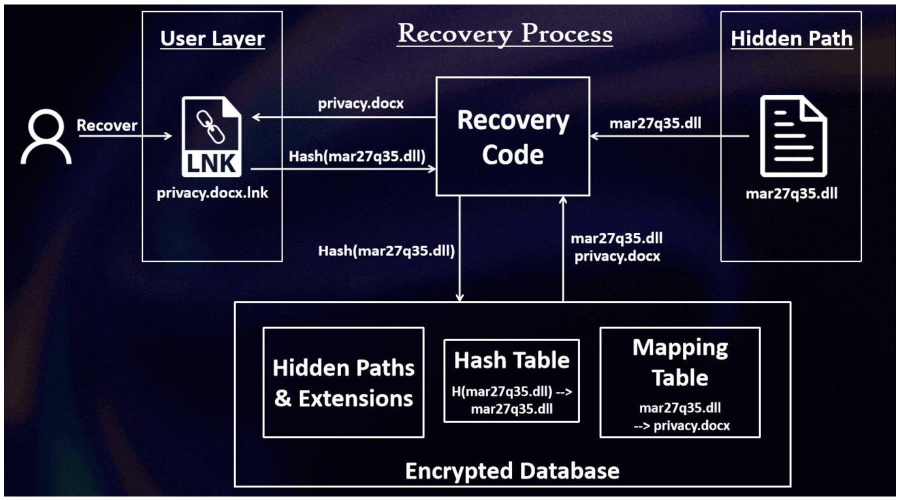

Figure 4: Recovery Process of Hidden Files. The process demonstrates how ShadowCrypt retrieves a hidden file using only the hash embedded within a .lnk file. The recovery module consults the encrypted database's hash and mapping tables to accurately restore the original file, even in post-attack scenarios.

Importantly, the recovery mechanism includes heuristic checks for suspicious file patterns (e.g., extensions like .lockbit, .enc, .locked, or other ransomware indicators). If such files are detected in the user directories, ShadowCrypt automatically recovers hidden files to the backup folder rather than restoring them to potentially compromised directories. This conservative strategy helps preserve the integrity and usability of user data in the presence of active or residual ransomware threats. Overall, the recovery process is designed to be resilient, adaptive, and secure, offering users peace of mind that their data can be retrieved reliably even under adverse conditions.

E. Secure Copy-Paste Enhancement for Usability

Despite this secure structure, an important usability limitation can be noted: users were unable to duplicate protected files easily. Copying a shortcut (via Ctrl+C) and pasting it (via Ctrl+V) simply resulted in multiple shortcuts pointing to the same hidden file, not a true copy. To create a duplicate, users had to recover the file, manually copy it, and then hide both copies—a workflow that was inefficient and discouraged practical use. To address this, ShadowCrypt is significantly enhanced with a module that integrates a background clipboard monitoring service along with a scripted secure-copy mechanism, thereby introducing seamless copy-paste support for protected files.

1) Psuedocode: MonitorClipboard():

```
While service is running:
    If Ctrl+C on shortcut:
        If hash in shortcut argument:
            Save shortcut metadata to clipboard
            Start expiry timer
```

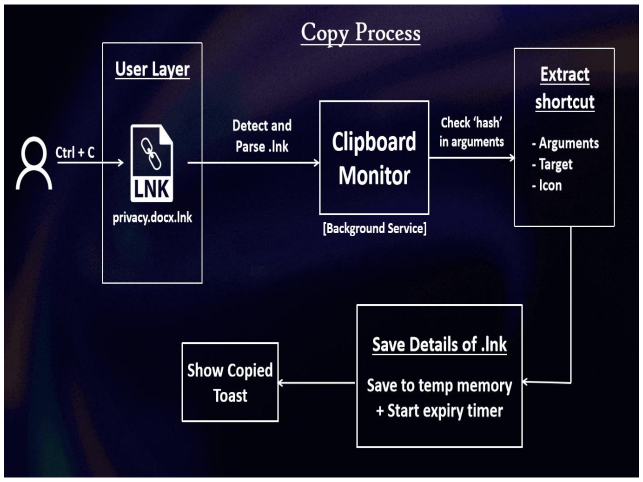

Figure 5: Copy Process in ShadowCrypt. This diagram represents the background operations triggered during a copy (Ctrl+C) event on a ShadowCrypt-generated shortcut. The clipboard monitor validates the shortcut's hash argument and securely stores its metadata along with an expiry timer, enabling safe and controlled duplication during the paste phase.

This enhancement begins with a background process that continually monitors clipboard events for user actions involving Ctrl+C and Ctrl+V. When a Ctrl+C operation is detected, the service inspects the file under selection and checks whether it is a ShadowCrypt-generated shortcut by examining the presence and format of the hash in its target argument. If validated, the system intercepts the operation and captures the shortcut's metadata—its absolute path, command-line arguments (hash), icon reference, and executable target. This metadata is stored temporarily, and an expiry timer is started to prevent stale or unintended copy-paste actions. A Windows toast notification is displayed to inform the user that a secure copy of the link has been prepared.

F. File Duplication and Paste Handler Execution

When the user performs a Ctrl+V operation, the clipboard service checks whether an eligible paste context exists. If the metadata is valid and within expiry, it decrypts the internal mapping database and resolves the original hidden path of the source file. Instead of duplicating the shortcut, the system performs a secure duplication of the hidden file itself. The original file is first copied to a user-owned temporary directory, ensuring that the file retains appropriate user-level permissions and does not inherit restrictive administrative flags from system directories. Then, a new hash is generated for the copied file, which is moved into a randomly selected hidden directory as per the original concealment logic. A new .lnk file is generated and placed at the currently opened path in the active Windows Explorer window, using the updated hash as an argument. The icon and launcher target remain the same as in the original shortcut, preserving the seamless user experience. Simultaneously, the mapping database is updated with two new entries: the mapping table is extended to include a record linking the absolute path of the newly duplicated hidden file to its original shortcut path in the destination folder, while the hash table is updated to map the newly generated hash value to the duplicated hidden file's path. This ensures that the newly created shortcut functions independently and securely, just like the original, while maintaining the integrity and traceability of the protected file system.

1) Psuedocode: PasteHandler():

```
If Ctrl+V and clipboard contains ShadowCrypt metadata:
    hidden_path ← Decrypt from hash
    temp_copy ← CopyFile(hidden_path → UserTempDir)
    new_hidden_path ← Move(temp_copy → NewSystemSafeDirectory)
    new_hash ← SHA256(new_hidden_path)
    current_folder ← GetActiveExplorerFolder()
    CreateShortcut(current_folder, target=LinkerScript, args=new_hash, icon=ext_icon)
    Update mapping and hash tables
```

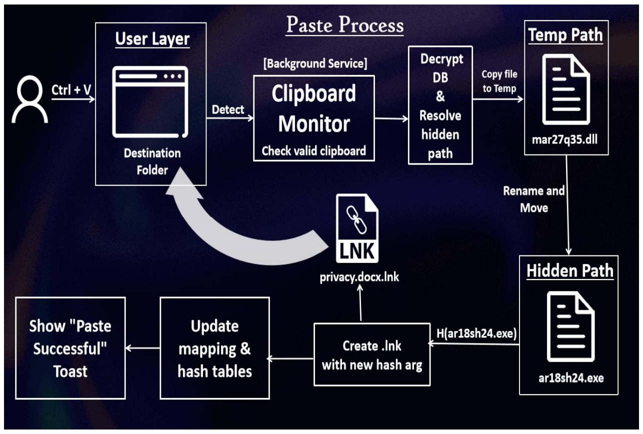

Figure 6: Paste Process in ShadowCrypt. The figure illustrates the internal workflow when a user performs a paste (Ctrl+V) operation on a ShadowCrypt-protected shortcut. The clipboard monitor resolves the hidden file via a secure hash, duplicates it to a temporary location, re-conceals it with a new hash, and generates an independent shortcut in the destination folder.

G. *User Feedback and Resource Performance*

To further reinforce usability and user confidence, toast notifications are presented at each stage: when a shortcut is successfully copied, when the paste is completed, or when the clipboard copy expires. This feedback loop provides clear visibility into what would otherwise be a silent, invisible process. The architecture also ensures that only ShadowCrypt-generated shortcuts are eligible for this behavior, preventing misuse or interception of standard file operations. Moreover, performance metrics collected from live usage indicate that the clipboard monitoring and paste handler consume minimal resources, operating within milliseconds and having negligible impact on system performance or responsiveness.

Therefore, ShadowCrypt represents a comprehensive and mature solution for proactive ransomware defense, encompassing every stage of protection—from secure initialization and intelligent file concealment to shortcut-based seamless access, resilient file recovery, and intuitive and secure duplication of protected files via familiar copy-paste operations. The system fuses robust cryptographic principles—such as AES-256 encrypted metadata and hash-based referencing—with usability-centric features including context menu actions, Explorer-aware paste handling, clipboard-driven duplication, and real-time user notifications. Its architecture is designed not only to prevent data compromise but also to remain functional and recoverable even in the presence of active ransomware infections, employing backup-aware recovery strategies and ransomware-pattern detection to safeguard user data in the worst-case scenarios. By uniting camouflage, encryption, behavioral intelligence, and user-centric design, ShadowCrypt goes beyond traditional defenses and offers a practical, resilient, and transparent security layer, positioning itself as a model for systems that protect both data and the user experience in tandem.

IV. PERFORMANCE CHECK

To evaluate the efficiency and responsiveness of the clipboard-integrated copy-paste mechanism introduced in our proposed system, we conducted real-time performance analysis on its two principal runtime modules:

1) The Clipboard Monitoring Service, responsible for detecting and parsing shortcut-based copy operations initiated via Ctrl+C.
2) The Paste Handler Module, which performs secure duplication of the underlying hidden files when Ctrl+V is triggered by the user.

These evaluations were designed to measure execution latencies across various operational phases and determine the practical feasibility of the copy-paste enhancement without degrading system interactivity.

A. *Experimental Setup*

All measurements were performed on a mid-range system running Windows 11 Pro (64-bit) with 16 GB RAM and an Intel Core i7 processor. The system was configured with minimal background activity to closely simulate standard user environments. Logging mechanisms were embedded into the clipboard service and paste handler routines, capturing timestamp-based execution durations during active user interaction over a 2-minute observation window.

TABLE 1: TECHNOLOGY STACK

Technology	Purpose
Python 3.12+	Core backend logic, CLI interface, encryption workflows
Windows PowerShell	Installation/setup automation, right-click context menu integration
Batch scripting (.bat)	Menu registration/unregistration (Add/remove right-click options)
uv Python package	Dependency for internal packaging and CLI support
Shortcut files (.lnk)	Camouflaged access to hidden files—mimicking system file types
Local DB (JSON)	Encrypted metadata storage (password-safe mapping of hidden files)

The following metrics constitute to the performance check:
1) Detection latency for shortcut-based Ctrl+C events.
2) File duplication to user-owned temporary directories.
3) Shortcut creation and database update durations.

B. *Results and Analysis*

The captured performance trends are illustrated in Figure 7, which plots the execution durations of both clipboard monitoring and paste operations over time. Key observations are as follows:

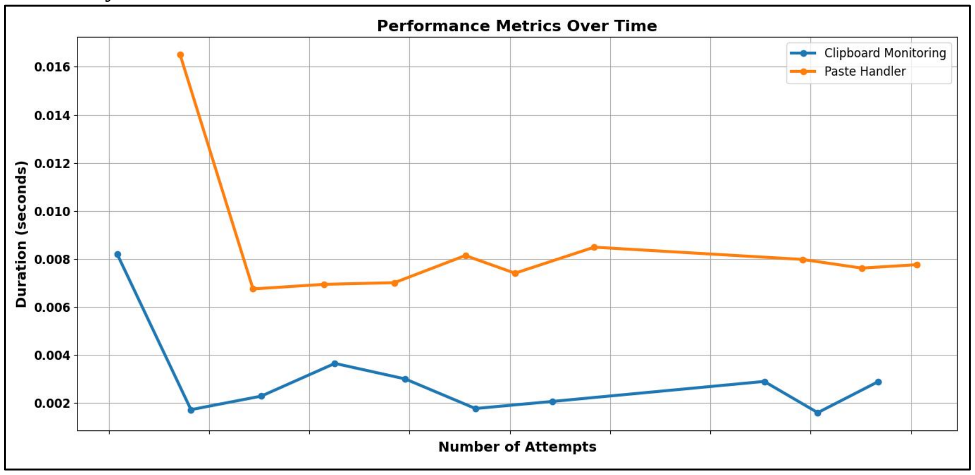

Figure 7: Performance metrics over multiple attempts. This graph shows the time taken (in seconds) by two core components—Clipboard Monitoring and Paste Handler—over successive executions.

The Clipboard Monitoring Service demonstrated an average execution time of approximately 2.5 milliseconds, with an initial spike of ~8.2 ms during the first file access due to cache warm-up. For subsequent detections, latencies remained consistently low. The detection logic, which involves parsing .lnk file metadata and validating the presence of a secure hash, proved to be lightweight and non-intrusive.

The Paste Handler Module exhibited a slightly higher but stable average duration of around 7.8 milliseconds, with a single observed peak of ~16.4 ms during the initial file duplication and database access. This component involves decrypting the internal database, resolving the hash to the concealed path, securely duplicating the file to a temporary user-owned directory, generating a new shortcut with updated hash values, and updating the database mappings. Despite this multifaceted logic, latencies remained within sub-20ms bounds.

These results validate the efficiency of the clipboard-based duplication mechanism. The operations are practically imperceptible to users and maintain system responsiveness even under repeated and rapid file interactions. Furthermore, resource consumption remains minimal, with no noticeable CPU or memory spikes during operation.

C. Implications

The performance profile of our proposed system highlights a critical design strength: the combination of real-time responsiveness with cryptographic and concealment security mechanisms. By maintaining low-latency operations for background services and active file duplication workflows, the system aligns with user expectations for instantaneous actions like copy and paste—without compromising on protection or confidentiality.

This reinforces the suitability of our approach for real-world deployment, where usability and speed must coexist with robust security. The copy-paste extension not only bridges a key usability gap but also adheres to stringent performance expectations essential for wide-scale adoption.

V. CONCLUSIONS

In this work, we proposed a preemptive and usability-aware solution to defend against ransomware by removing sensitive files from the attack surface altogether. Instead of relying on traditional detection, backup, or access control mechanisms, our system camouflages user files within system-reserved directories, renames them with benign extensions, and securely maps them using encrypted metadata. Access is restored seamlessly through dynamically generated shortcut files that reference hidden paths using hash-based redirection. By encrypting internal configurations and maintaining the integrity of file mappings, the system ensures both confidentiality and resilience even in the event of active ransomware presence or link file corruption.

To enhance practicality, ShadowCrypt incorporates a clipboard-driven duplication feature, allowing users to securely copy and paste protected files via familiar interactions such as Ctrl+C and Ctrl+V. A background monitoring service intercepts and validates link-based copy operations, securely duplicates the hidden file, and generates a new functional shortcut without compromising metadata or access controls. Performance metrics demonstrate that this functionality introduces negligible overhead, preserving system responsiveness. Overall, the proposed approach demonstrates that strong ransomware defense need not come at the cost of usability—security can be embedded invisibly into everyday workflows while maintaining proactive resilience and full data recoverability.

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," in IEEE Access, vol. 11, pp. 92693-92704, 2023, doi: https://doi.org/10.1109/ACCESS.2023.3309879
[2] Urooj, Umara, Bander Ali Saleh Al-rimy, Anazida Zainal, Fuad A. Ghaleb, and Murad A. Rassam. 2022. "Ransomware Detection Using the Dynamic Analysis and Machine Learning: A Survey and Research Directions" Applied Sciences 12, no. 1: 172. https://doi.org/10.3390/app12010172
[3] Reidys, Benjamin & Liu, Peng & Huang, Jian. (2022). RSSD: Defend against Ransomware with Hardware-Isolated Network-Storage Codesign and Post-Attack Analysis. https://doi.org/10.48550/arXiv.2206.05821
[4] Von der Assen, Jan & Huertas, Alberto & Sefa, Rinor & Bovet, Gérôme & Stiller, Burkhard. (2023). MTFS: a Moving Target Defense-Enabled File System for Malware Mitigation. https://doi.org/10.48550/arXiv.2306.15566
[5] Çalışkan, Büşra & Gülataş, Ibrahim & Kilinc, Hakan & Zaim, A.. (2024). The Recent Trends in Ransomware Detection and Behaviour Analysis. 1-8. https://doi.org/10.1109/SIN63213.2024.10871663
[6] Harun Oz, Ahmet Aris, Albert Levi, and A. Selcuk Uluagac. 2022. A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions. ACM Comput. Surv. 54, 11s, Article 238 (January 2022), 37 pages. http://dx.doi.org/10.1145/3514229
[7] Khan MM, Hyder MF, Khan SM, Arshad J, Khan MM. Ransomware prevention using moving target defense based approach. Concurrency and Computation: Practice and Experience. 2022 Dec 27. doi: https://doi.org/10.1002/cpe.7592
[8] Sudheer, Sooraj. (2024). Ransomware Attacks and Their Evolving Strategies: A Systematic Review of Recent Incidents. Journal of Technology and Systems. 6. 32-59. https://doi.org/10.47941/jts.2399
[9] Singh, Avinash & Adeyemi, Ikuesan & Venter, Hein. (2022). Ransomware Detection using Process Memory. https://doi.org/10.48550/arXiv.2203.16871
[10] Shina Sheen, K A Asmitha, Sridhar Venkatesan,R-Sentry: Deception based ransomware detection using file access patterns,Computers and Electrical Engineering, Volume 103, 2022, 108346, ISSN 0045-7906, https://doi.org/10.1016/j.compeleceng.2022.108346
[11] Huertas, Alberto & Sánchez, Pedro Miguel & von der Assen, Jan & Schenk, Timo & Bovet, Gérôme & Martinez Perez, Gregorio & Stiller, Burkhard. (2022). RL and Fingerprinting to Select Moving Target Defense Mechanisms for Zero-day Attacks in IoT. https://doi.org/10.48550/arXiv.2212.14647
[12] J. von der Assen, A. H. Celdran, R. Sefa, B. Stiller and G. Bovet, "MTFS: a Moving Target Defense-Enabled File System for Malware Mitigation," in 2024 IEEE 49th Conference on Local Computer Networks (LCN), Normandy, France, 2024, pp. 1-6, doi: https://doi.org/10.1109/LCN60385.2024.10639803
[13] Commey, Daniel & Appiah, Benjamin & Frimpong, Bill & Osei, Isaac & Hammond, Ebenezer & Crosby, Garth. (2024). EGAN: Evolutional GAN for Ransomware Evasion. https://doi.org/10.48550/arXiv.2405.12266