IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

Portable Encryptor

Manoj Shinde[1], Viraj Mahajan[2], Divyanshu Kumar[3], Nihal Thorat[4], Mahesh Kumavat[5]
Assistant Professor, Department of Computer Science, MIT ADT University, Pune, India[1]
Undergraduate Students, Department of Computer Science, MIT ADT University, Pune, India[2-5]

Abstract: *The increasing reliance on cloud storage platforms has intensified concerns regarding unauthorized access and data confidentiality. Traditional encryption solutions often require platform-specific software installations, limiting their usability across heterogeneous computing environments. This paper introduces Portable Encryptor, a hybrid encryption framework that integrates a command-line encryption utility with a browser-based decryption interface. The system enables users to encrypt data locally prior to cloud upload and decrypt it securely on any device without requiring software installation. The proposed approach leverages AES-256 in Galois/Counter Mode (GCM) for authenticated encryption and PBKDF2 with a high iteration count for secure key derivation from user-supplied passwords. The design emphasizes portability, usability, and strong cryptographic guarantees while maintaining minimal computational overhead. Additionally, the system protects sensitive metadataincluding file names, directory structures, and file sizesby bundling all data into a single encrypted archive. Experimental observations demonstrate that the system achieves efficient performance with near-linear time complexity relative to input size while maintaining high security standards across a variety of real-world usage scenarios.* .

Keywords: Cloud Security, AES-GCM, PBKDF2, Client-Side Encryption, Data Privacy, Web Crypto API, Portable Cryptography

I. INTRODUCTION

Cloud computing has fundamentally transformed data storage and accessibility, enabling users to access their files from any device with an internet connection. According to recent industry reports, over 60% of corporate data is now stored in the cloud, and individual users increasingly rely on services such as Google Drive, Dropbox, and OneDrive for personal file management. However, outsourcing data to third-party providers introduces potential vulnerabilities, including data leakage, unauthorized access, insider threats, and government surveillance risks.

While cloud providers typically implement server-side encryption, this approach has inherent limitations. The provider retains control over encryption keys, meaning that data may be accessible to the provider itself, compromised employees, or third parties with legal authority. End-to-end encryption, where only the user holds the decryption key, provides a stronger security guarantee but traditionally requires dedicated software to be installed on every device.

Existing encryption mechanisms such as VeraCrypt, GnuPG, and 7-Zip require installation across multiple devices, limiting their practicality in dynamic usage scenarios such as accessing files from a public computer, a colleague's workstation, or a mobile device. Furthermore, many existing tools fail to protect metadata—file names, directory structures, and timestamps—which can reveal sensitive information even when file contents are encrypted.

To address these limitations, Portable Encryptor adopts a hybrid model that combines local encryption via a lightweight command-line interface (CLI) tool with browser-based decryption using the Web Crypto API. This ensures that sensitive data remains encrypted during storage and transit while remaining accessible to authorized users across platforms without any software installation requirements. The CLI tool is compiled into a single portable binary, further reducing deployment complexity. The decryption module is embedded within the encrypted output itself as an HTML file, enabling decryption on any device with a modern web browser.

The key contributions of this work are as follows:

Copyright to IJARSCT
www.ijarsct.co.in
DOI: 10.48175/IJARSCT-34028

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 14, April 2026 Impact Factor: 8.2

- A hybrid encryption architecture combining CLI-based encryption with zero-installation browser-based decryption.
- Complete metadata protection through encrypted archival of directory structures and file names.
- A self-contained decryption package that requires only a web browser, ensuring universal accessibility.
- Experimental validation demonstrating efficient performance and strong security properties.

The remainder of this paper is organized as follows. Section II defines the problem statement. Section III reviews related work. Section IV describes the system architecture. Section V details the cryptographic framework. Section VI presents the algorithm design. Section VII provides an implementation overview. Section VIII evaluates performance. Section IX analyzes security properties. Section X compares the system with existing tools. Section XI discusses applications, and Section XII outlines future work before the conclusion in Section XIII.

II. PROBLEM STATEMENT

Despite the availability of numerous encryption tools, several challenges persist that hinder their adoption for cloud-based data protection. First, existing encryption tools such as VeraCrypt and GnuPG require platform-specific installations, making them impractical for users who access data from multiple devices or operating systems. A user encrypting files on a Linux workstation may find it difficult to decrypt them on a Windows tablet or Chromebook without installing additional software. Second, most encryption utilities encrypt file contents but leave metadata—such as file names, directory hierarchies, timestamps, and file sizes—in plaintext. This metadata can reveal sensitive information about the nature and organization of encrypted data, even without access to the contents [8]. Third, decrypting files typically requires the same software used for encryption, creating a dependency that limits flexibility and adds complexity. In environments where software installation is restricted (e.g., corporate machines, library computers, or shared workstations), this dependency becomes a significant barrier. Fourth, many encryption tools have steep learning curves and require command-line expertise, discouraging adoption by non-technical users. The gap between security capability and usability remains a persistent challenge in applied cryptography. Finally, systems relying on asymmetric key pairs or key files introduce additional complexity in key distribution, storage, and backup. Password-based systems must balance the ease of password-based access with robust key derivation to resist offline attacks. The proposed system aims to overcome these limitations while preserving robust security properties, providing a solution that is simultaneously secure, portable, and easy to use.

III. RELATED WORK

Several approaches have been proposed and implemented to address cloud data security. This section reviews the most relevant work and identifies the gaps that Portable Encryptor addresses.

A. Traditional Encryption Tools

VeraCrypt [9] provides full-disk and container-based encryption with strong cryptographic primitives. However, it requires installation and administrative privileges, making it unsuitable for portable or cross-device usage. GnuPG offers file-level encryption using public-key cryptography but requires key management infrastructure and is primarily command-line oriented.

B. Archive-Based Encryption

7-Zip supports AES-256 encryption of archive contents. While widely available, 7-Zip does not protect file names by default (only the 7z format supports this), and it still requires installation on the decryption endpoint. WinRAR provides similar functionality but is limited to Windows platforms.

C. Cloud Provider Encryption

Major cloud providers implement server-side encryption by default. Google Drive uses AES-128 or AES-256, while AWS S3 supports both SSE-S3 and SSE-KMS encryption schemes [10]. However, in all these cases, the cloud provider manages the encryption keys, leaving data potentially accessible to the provider or to adversaries who compromise the provider's infrastructure.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

D. Browser-Based Cryptography

The Web Crypto API [4] provides a standardized interface for performing cryptographic operations within the browser environment. Projects such as hat.sh and Crypt.ee leverage this API for in-browser encryption. However, these solutions typically require an active internet connection to access the web application and may expose data to the web server during the process.

E. Research Contributions

Academic research has explored various aspects of cloud data security. Kamara and Lauter [11] proposed a cryptographic cloud storage architecture that supports different security levels. However, their work focuses on the theoretical framework rather than practical implementation with portability constraints. Portable Encryptor differentiates itself by combining offline CLI encryption with an embedded, self-contained browser-based decryption module that operates without any server communication.

IV. SYSTEM ARCHITECTURE

Portable Encryptor is structured into two primary components that work in tandem to provide a complete encryption-decryption workflow.

A. CLI-Based Encryption Module

The encryption module is implemented as a command-line tool compiled into a standalone binary executable. It performs the following operations: (1) Input Processing — accepts a target directory path and a user-supplied password as input, validates the directory structure, and enumerates all files recursively. (2) Key Material Generation — generates a cryptographically random 16-byte salt and derives a 256-bit encryption key using PBKDF2-HMAC-SHA256 with 600,000 iterations. (3) File Traversal and Encryption — iterates through all files in the target directory, reads each file's contents, generates a unique 12-byte Initialization Vector (IV), and encrypts the data using AES-256-GCM. The encrypted content, IV, and GCM authentication tag are stored together. (4) Archive Construction — bundles all encrypted file data along with encrypted metadata (file names, relative paths, and directory structure) into a single output file. (5) Decryptor Embedding — appends a self-contained HTML/JavaScript decryption module to the output, producing a single file that includes both the encrypted data and the decryption interface.

B. Browser-Based Decryption Module

The decryption component is an HTML file containing embedded JavaScript that utilizes the Web Crypto API. It operates entirely within the browser's execution environment and performs the following steps: (1) Password Input — presents a minimal user interface for password entry. (2) Key Derivation — derives the decryption key from the entered password using the same PBKDF2 parameters stored within the encrypted file. (3) Data Extraction — parses the encrypted archive to extract individual file entries, including their encrypted contents, IVs, and authentication tags. (4) Decryption and Verification — decrypts each file using AES-256-GCM, which simultaneously verifies the authentication tag; if tag verification fails, the operation is aborted with an error message. (5) Output Delivery — reconstructs the original directory structure and provides the decrypted files for download as a ZIP archive. No data is transmitted to any external server during the decryption process.

C. Data Flow

The complete data flow of the system is as follows: The user selects a folder containing sensitive files and provides a password to the CLI tool. The CLI tool derives a cryptographic key, encrypts all files with per-file unique IVs, encrypts metadata, and produces a single output file containing the encrypted data and an embedded decryption module. The user uploads the encrypted output file to any cloud storage service. When decryption is needed, the user downloads the file and opens it in any modern web browser. The browser-based decryption module prompts for the password, derives the key, decrypts all files, and provides them for download.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 14, April 2026 Impact Factor: 8.2

V. CRYPTOGRAPHIC FRAMEWORK

The security of Portable Encryptor rests on two well-established cryptographic primitives: AES-256-GCM for authenticated encryption and PBKDF2-HMAC-SHA256 for key derivation.

A. Authenticated Encryption with AES-GCM

The system employs AES-256 in Galois/Counter Mode (GCM) [3], which provides both confidentiality and integrity in a single pass over the data. The encryption operation is expressed as:

(C, T) = AES-GCM-Encrypt(K, IV, P, A)

where K is the 256-bit encryption key, IV is a 96-bit initialization vector, P is the plaintext, A is optional additional authenticated data, C is the resulting ciphertext, and T is the 128-bit authentication tag. GCM mode was selected for its authenticated encryption with built-in integrity verification, parallelizability for efficient processing on modern multi-core processors, standardization by NIST [1] and wide support across cryptographic libraries including OpenSSL and the Web Crypto API, and high throughput with low computational overhead. Each file is encrypted with a unique, randomly generated 96-bit IV to ensure that even identical plaintext files produce different ciphertexts.

B. Key Derivation with PBKDF2

User-supplied passwords are typically low-entropy and vulnerable to brute-force and dictionary attacks. To mitigate this, PBKDF2 (Password-Based Key Derivation Function 2) [2] is employed to derive a high-entropy encryption key:

K = PBKDF2(P, S, c, dkLen)

where P is the user's password, S is a 128-bit cryptographically random salt, c = 600,000 is the iteration count, and dkLen = 256 bits is the derived key length. The high iteration count of 600,000 is chosen based on current OWASP recommendations [12] to ensure that each key derivation attempt requires approximately 250 milliseconds on modern hardware, making large-scale brute-force attacks computationally prohibitive. The random salt ensures that identical passwords produce different derived keys, preventing precomputation attacks using rainbow tables.

C. Initialization Vector Management

Proper IV management is critical for GCM security. The system generates a fresh 12-byte (96-bit) IV for each file encryption using a cryptographically secure random number generator (os.urandom() in Python, crypto.getRandomValues() in JavaScript). This approach ensures IVs are unique with overwhelmingly high probability, avoiding the catastrophic security failure that would result from IV reuse under the same key [5].

VI. ALGORITHM DESIGN

A. Encryption Process

The encryption algorithm processes an entire directory tree and produces a single encrypted output file.

Input: FolderPath, Password **Output:** EncryptedFile

Salt ← GenerateRandomBytes(16)
Key ← PBKDF2(Password, Salt, 600000, 256)
FileList ← RecursiveEnumerate(FolderPath)
Manifest ← {}
For each file in FileList:
IV ← GenerateRandomBytes(12)
Plaintext ← ReadFile(file)
(Ciphertext, Tag) ← AES-GCM-Encrypt(Key, IV, Plaintext)
Store(IV, Ciphertext, Tag)
Manifest.add(EncryptedFileName, RelativePath, Size)
EncryptManifest(Manifest, Key)

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 14, April 2026 Impact Factor: 8.2

EmbedDecryptorHTML()
WriteOutputFile()

B. Decryption Process

The decryption algorithm runs entirely within the browser environment and processes the encrypted archive.

Input: EncryptedFile, Password **Output:** DecryptedFiles
Salt ← ExtractSalt(EncryptedFile)
Key ← PBKDF2(Password, Salt, 600000, 256)
Manifest ← DecryptManifest(EncryptedFile, Key)
For each entry in Manifest:
IV ← ExtractIV(entry)
Ciphertext ← ExtractCiphertext(entry)
Tag ← ExtractTag(entry)
Plaintext ← AES-GCM-Decrypt(Key, IV, Ciphertext, Tag)
If Tag verification fails → Abort with error
ReconstructFile(Plaintext, entry.path)
PackageAsZIP(DecryptedFiles)
OfferDownload()

VII. IMPLEMENTATION OVERVIEW

A. CLI Encryption Module

The encryption module is implemented in Python 3 and utilizes the following libraries: the cryptography library, which provides the AES-GCM encryption implementation and PBKDF2 key derivation and is a well-audited, widely-used Python cryptographic toolkit; os and pathlib, which handle file system traversal, path manipulation, and directory structure enumeration; struct, which manages binary data packing for the encrypted archive format; and base64, which encodes binary encrypted data for embedding within the HTML decryptor file. The CLI tool is packaged as a standalone binary using PyInstaller, producing a single executable that runs on Windows, macOS, and Linux without requiring a Python installation.

B. Browser Decryption Module

The browser-based decryption module is implemented using vanilla JavaScript and the Web Crypto API [4]. The SubtleCrypto interface provides access to AES-GCM decryption and PBKDF2 key derivation, ensuring all cryptographic operations are performed by the browser's native, hardware-accelerated cryptographic engine. JSZip is used to reconstruct the original directory structure and package decrypted files into a downloadable ZIP archive. A minimal, responsive HTML/CSS interface provides password input, progress indication, and download functionality. The Blob API is used to create downloadable file objects from decrypted data without requiring server interaction.

C. Encrypted Archive Format

The output file follows a custom binary format. Table I summarizes the structure.

TABLE IENCRYPTED ARCHIVE FORMAT STRUCTURE

Field	Size	Description
Magic Number	4 bytes	Format identifier
Version	2 bytes	Format version number
Salt	16 bytes	PBKDF2 salt
Manifest IV	12 bytes	IV for manifest encryption
Manifest Length	4 bytes	Encrypted manifest size

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

Encrypted Manifest	Variable	File metadata (encrypted)
File Data Blocks	Variable	Encrypted file contents
Decryptor HTML	Variable	Embedded decryption interface

VIII. PERFORMANCE EVALUATION

The system's performance was evaluated by measuring encryption and decryption times across varying file sizes and directory structures. All tests were conducted on a system with an Intel Core i5-12400 processor (2.5 GHz, 6 cores), 16 GB RAM, and an NVMe SSD, running Ubuntu 22.04 LTS.

A. Encryption Performance

Results indicate a linear increase in processing time relative to file size for both encryption and decryption, which aligns with the O(n) time complexity of AES-GCM. Table II presents the observed processing times across file sizes.

TABLE II ENCRYPTION AND DECRYPTION TIME VS. FILE SIZE

File Size (MB)	Encryption Time (ms)	Decryption Time (ms)
1	12	15
5	50	58
10	95	110
25	230	265
50	410	480
100	815	940
200	1620	1870

Decryption times are approximately 10–15% higher than encryption times due to the overhead of browser-based JavaScript execution compared to native Python with C-optimized cryptographic libraries.

B. Key Derivation Overhead

The PBKDF2 key derivation step introduces a fixed overhead of approximately 250 ms per operation, independent of the data size. This overhead is intentional and desirable, as it provides resistance against brute-force password attacks. The key derivation is performed only once per session (not per file), so its impact on overall throughput is minimal for multi-file operations.

C. Memory Usage

The browser-based decryption module processes files individually to minimize memory consumption. Peak memory usage during decryption of a 200 MB archive was observed to be approximately 450 MB, which is well within the capabilities of modern web browsers. Table III summarizes the memory usage across different input sizes.

TABLE III MEMORY USAGE DURING DECRYPTION

Archive Size (MB)	Peak Memory (MB)
10	45
50	120
100	240
200	450

IX. SECURITY ANALYSIS

A. Threat Model

The system considers the following threat scenarios. A passive cloud adversary who gains read access to the cloud storage can observe the encrypted files but should not be able to recover plaintext or metadata. An active cloud adversary who can modify stored data should be detected through GCM authentication tag verification, preventing

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

undetected data tampering. An offline password attacker who obtains the encrypted file can attempt offline password guessing; PBKDF2 with 600,000 iterations provides resistance by making each guess computationally expensive.

B. Confidentiality

AES-256 provides 256 bits of security against key search attacks. Under current understanding, the best known attack against AES-256 is exhaustive key search, which requires 2^{256} operations—far beyond the capabilities of any foreseeable computing technology, including quantum computers (which would reduce this to 2^{128} using Grover's algorithm, still considered secure) [1].

C. Integrity and Authenticity

GCM's authentication tag provides 128 bits of integrity protection. Any modification to the ciphertext, IV, or additional authenticated data will cause tag verification to fail with overwhelming probability ($1 - 2^{-128}$), alerting the user to potential tampering [3].

D. Password Security

The PBKDF2 configuration with 600,000 iterations and a 128-bit random salt provides brute-force resistance, rainbow table resistance, and replay resistance. At 250 ms per derivation, an attacker using a single modern CPU can test approximately 4 passwords per second. A GPU-accelerated attack might achieve 10^4 guesses per second, requiring approximately 317 years to exhaust a keyspace of 10^{11} (equivalent to a random 8-character alphanumeric password). The 128-bit random salt creates 2^{128} possible salt values, making precomputation attacks infeasible. Each encryption operation generates a fresh salt, ensuring that re-encrypting the same data with the same password produces different output.

E. Metadata Protection

Unlike many existing tools, Portable Encryptor encrypts file metadata (names, paths, sizes) within the encrypted manifest. An adversary observing the encrypted archive can determine only the total archive size, not the number, names, or sizes of individual files.

X. COMPARISON WITH EXISTING TOOLS

Table IV provides a detailed comparison of Portable Encryptor with widely-used encryption tools across multiple security and usability dimensions.

TABLE IV COMPARISON OF ENCRYPTION TOOLS

Feature	PE	VeraCrypt	7-Zip	GnuPG
Portability	High	Low	Medium	Low
Browser Decryption	Yes	No	No	No
No Install Required	Yes	No	No	No
Metadata Protection	Full	Full	Partial	No
Auth. Encryption	Yes	Yes	No	Yes
PBKDF2 Key Deriv.	Yes	Yes	Yes	No
Cross-Platform	Yes	Yes	Yes	Yes
Self-Contained Output	Yes	No	No	No

Portable Encryptor is the only solution that combines self-contained output with browser-based decryption, eliminating the need for any software installation on the decryption endpoint. While VeraCrypt provides stronger full-disk encryption capabilities, it is not designed for the file-level, cross-device use case that Portable Encryptor targets.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

XI. APPLICATIONS

The versatility of Portable Encryptor makes it suitable for a wide range of practical applications. For secure academic file sharing, researchers and students can encrypt sensitive data (e.g., unpublished manuscripts, datasets, examination materials) before uploading to shared cloud repositories, ensuring that only authorized recipients with the password can access the content. For enterprise data protection, organizations can use the system to encrypt confidential documents before cloud backup, providing an additional layer of protection beyond server-side encryption; the portability feature is particularly valuable for employees who access data from multiple devices. For personal cloud security, individual users can protect personal files (financial records, medical documents, personal photographs) stored on cloud platforms, maintaining privacy even if the cloud account is compromised. For secure file transfer, the self-contained encrypted output can be shared via email, messaging platforms, or file transfer services, with the recipient needing only a web browser and the password to decrypt the files. For compliance and regulatory use, organizations subject to data protection regulations (e.g., GDPR, HIPAA) can use client-side encryption to demonstrate due diligence in protecting sensitive data, as the cloud provider never has access to plaintext data.

XII. LIMITATIONS AND FUTURE SCOPE

A. Current Limitations

Browser-based decryption is constrained by available memory; files exceeding 500 MB may cause performance degradation on devices with limited RAM. The security of the system relies entirely on password strength, and weak passwords remain vulnerable despite PBKDF2 hardening. If the password is lost, encrypted data cannot be recovered, as there is no key escrow mechanism.

B. Future Enhancements

Future work includes integrating time-based one-time passwords (TOTP) or hardware security keys (e.g., YubiKey) as an additional authentication factor. Leveraging platform-specific secure enclaves (e.g., TPM, Apple Secure Enclave) for key material storage when available is also planned. Developing native mobile applications for iOS and Android will improve the encryption experience on mobile devices. Implementing chunk-based processing will reduce memory requirements for large files during browser-based decryption. Finally, evaluating the integration of post-quantum key encapsulation mechanisms (e.g., CRYSTALS-Kyber) will provide long-term security against quantum computing threats [13].

XIII. CONCLUSION

Portable Encryptor demonstrates an effective and practical approach to secure cloud storage by combining strong cryptographic techniques with cross-platform accessibility. The system addresses key limitations of existing encryption tools by providing full metadata protection, zero-installation decryption through standard web browsers, and a self-contained output format that bundles encrypted data with the decryption interface. The use of AES-256-GCM ensures authenticated encryption with high throughput, while PBKDF2 with 600,000 iterations provides robust resistance against offline password attacks. Experimental evaluation confirms that the system maintains near-linear performance scaling and operates within acceptable memory constraints for typical use cases. The hybrid CLI-browser architecture successfully balances usability and security, making Portable Encryptor suitable for practical deployment across academic, enterprise, and personal data protection scenarios.

REFERENCES

[1]. National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," FIPS PUB 197, 2001.

[2]. B. Kaliski, "PKCS #5: Password-Based Cryptography Specification Version 2.0," RFC 2898, 2000.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

[3]. D. McGrew and J. Viega, "The Galois/Counter Mode of Operation (GCM)," NIST Special Publication 800-38D, 2007.M. Wegmuller, J. P. von der Weid, P. Oberson, and N. Gisin, "High resolution fiber distributed measurements with coherent OFDR," in Proc. ECOC'00, 2000, paper 11.3.4, p. 109.

[4]. W3C, "Web Cryptography API," W3C Recommendation, 2017. [Online]. Available: https://www.w3.org/TR/WebCryptoAPI/

[5]. Menezes, P. van Oorschot, and S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.

[6]. PortableEncryptor GitHub Repository. [Online]. Available: https://github.com/Viraj2313/portable-encryptor

[7]. Flexera, "2023 State of the Cloud Report," Flexera, 2023.

[8]. J. Greschbach, T. Pulls, L. M. Roberts, and P. Winter, "The Effect of DNS on Tor's Anonymity," in Proc. Network and Distributed System Security Symposium (NDSS), 2017.

[9]. J. Greschbach, T. Pulls, L. M. Roberts, and P. Winter, "The Effect of DNS on Tor's Anonymity," in Proc. Network and Distributed System Security Symposium (NDSS), 2017.

[10]. Amazon Web Services, "Protecting Data Using Encryption," AWS Documentation, 2023.

[11]. S. Kamara and K. Lauter, "Cryptographic Cloud Storage," in Proc. Financial Cryptography and Data Security (FC), Springer, 2010, pp. 136–149.

[12]. OWASP, "Password Storage Cheat Sheet," 2023.

[13]. R. Avanzi et al., "CRYSTALS-Kyber Algorithm Specifications and Supporting Documentation," NIST PQC Round 3 Submission, 2021.