Bol. Soc. Paran. Mat.
©SPM – E-ISSN-2175-1188
SPM: www.spm.uem.br/bspm

(3s.) **v. 2026 (44) 4** : 1–32.
ISSN-0037-8712
doi:10.5269/bspm.80388

Digital Time Capsule- Lock Today, Unlock Tomorrow: Web-Based Encrypted Time Storage

Shivakumara T., Muneshwara M. S., P. Sudarsanam, Nagamani H. S., Preethi Hatui and Anand R.

ABSTRACT: Digital Time Capsule is a secure web-based platform that enables users to store files safely and release them at a predetermined time. The system utilizes a robust encryption mechanism, combining AES-256 and RSA to protect files from unauthorized access. Uploaded files are encrypted and stored in AWS S3, while user accounts and capsule information are managed through PostgreSQL. An additional layer of security is provided through a Time-Based One-Time Password (TOTP) system, integrated with Google Authenticator. The platform implements various security measures, including limited OTP attempts, lockouts for repeated incorrect attempts, and email notifications for suspicious activity. Decryption occurs on the server-side, with a time-lock check ensuring files are only accessible at the scheduled time. With features like controlled deletion, auto-unlock notifications, and activity logging, Digital Time Capsule offers a reliable and secure solution for storing digital information and controlling access to it.

Keywords: Authentication, brute-force, bloud security, digital preservation, hybrid cryptography, secured storage, time-lock encryption.

Contents

1 Introduction **2**

2 Related Work **3**
2.1 Comparative Positioning of the Proposed System . . . 3
2.2 Comparison with existing timed-release systems . . . 4

3 Existing and Proposed System **5**
3.1 Existing Work . . . 5
3.2 Proposed Work . . . 6

4 Methodology **6**
4.1 System Architecture . . . 6
4.2 Data flow and sequence . . . 7
4.3 System Workflow . . . 10
4.3.1 Unlock-Time Enforcement Logic and UI State Validation . . . 10
4.4 Research Method and Evaluation Design . . . 11

5 Results and Discussion **13**
5.1 Mathematical Equations of Cryptographic Computation . . . 13
5.1.1 RSA Protection of AES Key . . . 13
5.1.2 Time-Based One-Time Password . . . 14
5.2 Comparative Benchmark Overview . . . 14
5.3 Scalability & Stress Tests — Proposed Experiments . . . 15

6 Algorithms Used **16**
6.1 AES-256 File Encryption . . . 16
6.2 RSA Encryption of AES Key . . . 17
6.3 Time-Based One-Time Password (TOTP) . . . 17
6.4 QR Code Provisioning Algorithm . . . 18
6.4.1 Auto-Unlock Scheduler Algorithm . . . 18

2020 *Mathematics Subject Classification:* 94A60, 94A62, 68P25.
Submitted November 26, 2025. Published February 26, 2026

6.4.2 SHA-256 Hashing . . . 18
6.5 System Modules . . . 19
6.6 Prototype Evaluation and Performance Metrics . . . 20

7 Security Analysis and Threat Model 20
7.1 Assumptions and Trust Boundaries . . . 20
7.2 Time Resilience and Unlock Timing Dependability. . . . 20
7.3 Security Objectives . . . 21
7.3.1 Integrity Verification, S3 Object Validation, and Tamper Detection . . . 21
7.3.2 QR Code Security, TOTP Binding, and Protection Against Cross-Device OTP Abuse 22
7.4 Threat Model . . . 23
7.5 Possible Attacks and Security Measures . . . 23
7.5.1 Controlled Capsule Deletion Security and Two-Step OTP Enforcement . . . 24
7.6 Security Strength Overview . . . 25
7.7 Additional Security Considerations . . . 25
7.8 Time Synchronization, Server Clock Reliability, and Unlock Accuracy . . . 26

8 Case Study: End-to-End User Workflow of the Digital Time Capsule 26

9 Limitations and Future Work 29
9.1 Practical Deployment Considerations . . . 29
9.2 Long-Term Storage Cost, Retention Strategy, and Cloud Lifecycle Management . . . 30

10 Conclusion 30

1. Introduction

Digital Time Capsule is based on the concept of physical time capsules that people have been using for many years. These time capsules involve sealing items or messages for the purpose of being opened by future generations. The Digital Time Capsule is a digital platform that is encrypted and cloud-based. It allows for the digital storing of files in a secure environment. The platform is based on advanced encryption techniques such as AES256 and RSA. These techniques ensure that any files uploaded to the platform are private and can be accessed after a certain time. The platform allows users who upload files that are encrypted and stored for secure access. These files can be accessed by users who log in via a Google Authenticator app that is connected by a QR code. Additionally, the platform has a background program that tracks when users can unlock their digital time capsules. Moreover, this platform has several levels of restrictions for accessing digital files, which include login attempts as well as email verification. Through the use of cloud storage and encryption techniques, Digital Time Capsule is an effective platform for storing sensitive digital files. It can be used in either a personal or legal setup. Although it utilizes trusted cryptography tools like AES-256, RSA-2048, and TOTP, it is the engineering of a full-scale, ready-to-deploy time-lock based file storage system, and not the invention of a cryptography mechanism, that has been achieved. The aim is to demonstrate how standard security mechanisms can be combined into a practical system that enforces timed release, multi-factor authentication, and controlled deletion in a real web environment. This positions the work as an applied system design study rather than a theoretical cryptography contribution. Contribution and novelty. This work presents a complete, deployable web prototype that integrates per-file TOTP binding, hybrid AES-256 + RSA key-wrapping, scheduler-based server-enforced timed release, and a controlled multi-step deletion workflow. While the cryptographic primitives used are standard, the novelty lies in combining
(a) Per-capsule TOTP provisioning (QR label includes CapsuleID),
(b) Strict server-side unlock enforcement with scheduler auditing and UTC-normalized timestamps, and
(c) A high-assurance, email-OTP deletion flow that requires prior viewing. Together these elements produce a practical, auditable pattern for time-locked file releases aimed at legal/archive use cases where operational deployment and end-to-end usability are primary goals.

2. Related Work

The idea of releasing information only after a selected time has its foundations in the early work of Rivest, Shamir and Wagner in 1996 [1]. They introduced the concept of Time-Lock Puzzles which allow data to be decrypted only after a deliberate computational delay. This was one of the first attempts to design a mechanism in which time itself acts as a security barrier. Cheon and colleagues extended this line of work in 2008 by formalising timed-release cryptography under a public-key setting, giving stronger mathematical guarantees and removing the dependence on purely sequential computation [2]. Applications of timed-release mechanisms soon began appearing in other domains. Nelson and Haq in 2009 proposed a timed-locked embargo system for academic publishing [3]. Surnić and Rauber demonstrated a Digital Preservation Time Capsule model intended for long-term controlled access to digital artefacts [4]. Hsu and co-authors introduced a Time Capsule Signature in 2012 [5], designed to become valid only after a future date. Around the same time, Watanabe and Shikata proposed timed-release secret sharing, enabling multi-party access to sensitive information only when the set time has elapsed. Cloud-oriented approaches followed with schemes that used temporary keys, verifiable deletion and integrity guarantees. A notable example is the Security Rag-Based Scheme introduced in 2014, which combined ephemeral key derivation with cloud storage controls [6] [7].

In 2017, Takemata and colleagues implemented an electronic time capsule system for educational use, demonstrating scheduled release of stored content [8]. Later works expanded the idea toward real-time analytics, such as TimeCrypt in 2018, which offered strong encryption with access control for continuously generated data [9]. Dang and Chang in 2019 developed TEEKAP, a self-expiring data capsule that combined trusted hardware with secret sharing to ensure that stored data automatically becomes inaccessible once the release period expires [10]. Research in controlled deletion and time-sensitive cloud storage continued with the ATDD scheme presented in 2020, which focused on fine-grained deletion guarantees for sensitive data in cloud environments [11]. More recent literature has explored decentralised approaches. Yang and Liu in 2022 investigated the use of blockchain mining difficulty as a globally verifiable time source for Web3 timed-release cryptography [12]. Soltani and colleagues proposed self-contained data capsules that enforce internal access policies [13], while the i-TiRE framework introduced incremental key evolution to support staged disclosures of information [14].

In 2024, Yuan and co-authors presented a multi-time-server model that used Shamir's Secret Sharing to ensure reliable timed access without depending on a single authority. Together, these works show how timed-release cryptography has evolved from basic sequential computation to secure multi-party models, decentralised infrastructures and cloud-based key-management systems [15]. The Digital Time Capsule proposed in this study is aligned with this progression. It adopts modern symmetric and asymmetric cryptography, integrates time-controlled access, and incorporates multi-factor authentication using a Google Authenticator-based QR code. While earlier systems focused on theoretical constructs or specialised applications, the present work applies these ideas to a practical, user-centred platform for securely storing personal digital content and releasing it only when the chosen unlock time arrives.

2.1. Comparative Positioning of the Proposed System

Although many timed-release mechanisms exist in the literature, most prior works were developed either as theoretical constructions or as specialised platforms requiring trusted hardware, distributed time services, or blockchain-based timing. These systems are also robust in guarantees and are not easily implemented to be used by normal people. Conversely, the Digital Time Capsule emphasis is on convenience and functionality and portability on a standard web technologies. The given prototype is unique to the previous work in three respects. The current system is much more streamlined as compared to the earlier systems which relied on more complex mechanisms such as verifiable delay functions, multi-server timing, or proof-of-work anchors to ensure security. It uses server-side time enforcement and a scheduler to provide an easier and more realistic deployment model that is a good fit in a typical cloud setup.

Second, most prior timed-release systems do not provide authentication that is bound to each individual file. The Digital Time Capsule introduces capsule-specific TOTP authentication through QR provisioning, ensuring that every capsule is linked to the user's Google Authenticator account. This

per-file authentication is a practical extension not commonly addressed in earlier literature. Third, many existing works are conceptual or hardware-based, whereas the system developed in this study integrates AES/RSA encryption, AWS S3 storage, metadata management, OTP-secured unlocking, and controlled deletion. This positions the contribution not as a new cryptographic primitive but as a complete, deployable system demonstrating how timed-release access and multi-factor authentication can work together in practice. The following comparison (Table 1) situates the proposed system within the landscape of existing timed-release approaches, which compares representative timed-release systems and highlights how the proposed work differs in authentication, deployment feasibility, and intended use.

2.2. Comparison with existing timed-release systems

The literature on timed-release and time-controlled storage includes both theoretical constructions and prototype systems for special environments (trusted hardware, distributed time servers, encrypted streaming, etc.). Table 1 summarizes representative designs and highlights how the Digital Time Capsule differs in terms of authentication, deployment, and usability. In short, prior work focuses on formal timed-release primitives or specialized platforms; our contribution is a practical, user-facing web application that combines per-file AES encryption, RSA key-wrapping, AWS S3 storage, and capsule-specific TOTP (QR-provisioned) together with controlled deletion and email notifications.

Table 1: Comparison of representative timed-release systems and the Digital Time Capsule

System (ref)	**Time control**	**Authentication**	**Cloud-ready**	**Key features**	**How our system differs**
Time-Lock Puzzles [1]	Sequential crypto delay	none	no	Conceptual timed decryption puzzle	Practical User-auth + cloud storage for end users
Provably timed-release PK [2]	Cryptographic timed keys	none	no	Formal public-key timed release	Focus on deployable file capsules & OTP auth
Academic embargo system [3]	Policy/time based	basic	limited	Timed embargo for publication	Our system targets general file storage with TOTP
Digital preservation capsule [4]	Long-term archival	limited	limited	Preservation model for archives	Adds capsule-TOTP, cloud storage, user flows
Time Capsule Signature [5]	Signature valid after time	cryptographic	no	Cryptographic delayed validity	Practical unlock+TOTP UI and cloud backend
Timed-release secret sharing [6]	Secret-sharing time control	cryptographic	mixed	Multi-party access after time	Simpler user-centric flow for single users
Security Rag-Based Scheme [7]	Ephemeral keys	server	partial	Ephemerizer + integrity controls	We combine key-wrapping + per-capsule TOTP

E-Time Capsule (Takemata) [8]	Scheduler based	application	limited	Educational e-capsule prototype	Full web app with S3, AES/RSA, QR/TOTP
TimeCrypt [9]	Server-side policy	server	yes	Encrypted streaming & access control	Not built for per-file timed capsules with QR/TOTP
TEEKAP (TEEs) [10]	Hardware expiry (TEE)	device	limited	Self-expiring with trusted hardware	Requires TEEs; our approach works on standard web stack
ATDD (deletion) [11]	Policy deletion	server	yes	Fine-grained assured deletion	Focuses on deletion guarantees; we add TOTP unlock
Web3 / PoW time source [12]	Blockchain/PoW time	cryptographic	mixed	Decentralized time anchors	Our system uses server time + scheduler for practicality
Data Capsule model [13]	Policy-enforced capsules	policy	mixed	Self-contained access policies	We prioritize simple user flows and deployability
i-TiRE / staged release [14]	Multi-stage key evolution	cryptographic	mixed	Incremental timed disclosures	More complex; ours is simpler and user-centric
Multi-time-server TRE [15]	Multiple time servers	cryptographic	mixed	Fault-tolerant timed release	Our model uses practical server-time enforcement + notifications
Digital Time Capsule (this work)	**Server-enforced unlock + scheduler**	**password + email OTP + capsule TOTP (QR)**	**yes (AWS S3)**	**AES-256 + RSA-wrapped keys, QR-bound TOTP, controlled deletion, auto-unlock emails**	**Complete, deployable web system for general users**

Compared with provably-timed schemes, our system trades formal guarantees for deployability: we rely on server-enforced time and standard crypto to provide a practical user-facing service. This positions the contribution as engineering and usability work rather than a new cryptographic primitive.

3. Existing and Proposed System

3.1. Existing Work

Cloud storage services such as Google Drive, Dropbox, and OneDrive are widely used because they provide ease to the users in saving, sharing, and accessing their files whenever they want. While these platforms make things easier, they do not offer any kind of releasing option at a fixed time or access control based on the schedule. A file, once uploaded, is available immediately, which may not be appropriate

in cases when the information is required to remain private until a specific future date. Most of these services have simple password protection and lack stronger security measures, such as OTP verification, QR-based TOTP generation, or automatically expiring access after a set time frame. Facilities for delayed emails or scheduled messages do exist, but are not designed to safely store digital files. They do not support end-to-end encryption that unlocks only at a specific time; capsule-specific authentication rules are unavailable and they also do not work with apps like Google Authenticator for generating time-based OTPs. Surnić and Rauber introduced the concept of a digital preservation time capsule that would protect digital content over long periods. However, their model does not use modern multi-factor authentication methods or any time-controlled access for cloud-stored files. Overall, none of the existing systems have strong security, time-locked access to files, or checking of users via OTP or encrypted file checks. This shows that there is a need for one single platform that combines solid encryption with scheduled release and multi-factor authentication. The Digital Time Capsule is designed to meet just this need.

3.2. Proposed Work

Digital Time Capsule offers a secure and user-friendly web portal of uploading and storing digital content: documents, images, audio, videos, zip, and others. The users can put a specific date and time that the capsule is supposed to unlock. After uploading a file, its data is encrypted using the AES-256 and finally, the RSA encrypts the AES key in order to secure the keys even better. All the capsules contain identifiers of both the user and the capsule in the form of unique QR codes on each capsule. This is scanned using the Google Authenticator app to begin generation of time-based OTPs that are specific to that capsule. When an Authenticated user enters a valid OTP to unlock the file at the specified unlock time, it should be ensured that the user had scanned the QR code and was the original user to unlock the file so as to ensure that only the original user unlocks the capsule. To this end, the encrypted files are stored safely in AWS S3; related information such as user data, capsule data, and unlock timestamps as well as encrypted AES keys are stored in PostgreSQL. The system is firm with no before unlock time access and capsules cannot be destroyed before unlocked and read. Other security measures comprise email notification on key incidents, OTP attempts are limited, temporary lockdown is offered in instances of multiple attempts and time constrained access policies are verified. This framework is a collection of encryptions, scheduled access control, and multi-factor authentication to offer an effective solution to the protection of the digital content until the desired time in future.

4. Methodology

4.1. System Architecture

To address the architectural clarity noted in prior studies, this section now includes explicit data-flow descriptions, sequence diagrams, and trust boundaries to show how components communicate and how security is enforced across the system. The Digital Time Capsule system architecture is structured as a multilayered system which combines secure authentication, encryption, cloud storage and scheduled access control. The system allows the users to use time-coded encrypted capsules which could not be opened until the required unlocking time and after successful OTP authentication. The overall architecture as shown in figure 1 will comprise the Client Layer, the Application Layer, Capsule Services, Authentication Services, and the Storage and Encryption Layer. Client Layer is the web browser of the user that will communicate with the system via a web app developed using Flask. Application Layer receives user requests, provides session management, capsule operations, generates QRs, and authenticates OTPs and communicates with the backend services. The Capsule Services Layer provides the implementation of the core system logic and comprises of capsule creation, unlock scheduling with APScheduler, OTP-secured unlock flow, email notifications on capsule unlocking and policies on controlled capsule deletion. The Authentication Services Layer supports multi-factors security using Google Authenticator-based TOTP and OTP email delivery, session management and OTP lockout handler which blocks the user temporarily following consecutive failed authentication attempts. This avoids brute force attacks and unauthorized access. The Storage Layer deals with long-term file protection and storage. The Encryption Layer takes care of the cryptographic security and long-term file storage. AES-256 is used on files uploaded by the user

and the asymmetric RSA encrypted key before storing files in Amazon S3. PostgreSQL stores metadata used to identify the capsule ID, unlock schedule and confidential encryption keys. Upon the arrival of the unlock time with the valid OTP, the system will decrypt the file and allow securely downloading it. This layered architecture gives it confidentiality, controlled access, tamper resistance and auditing capabilities yet it still designates smooth and secure end user experience.

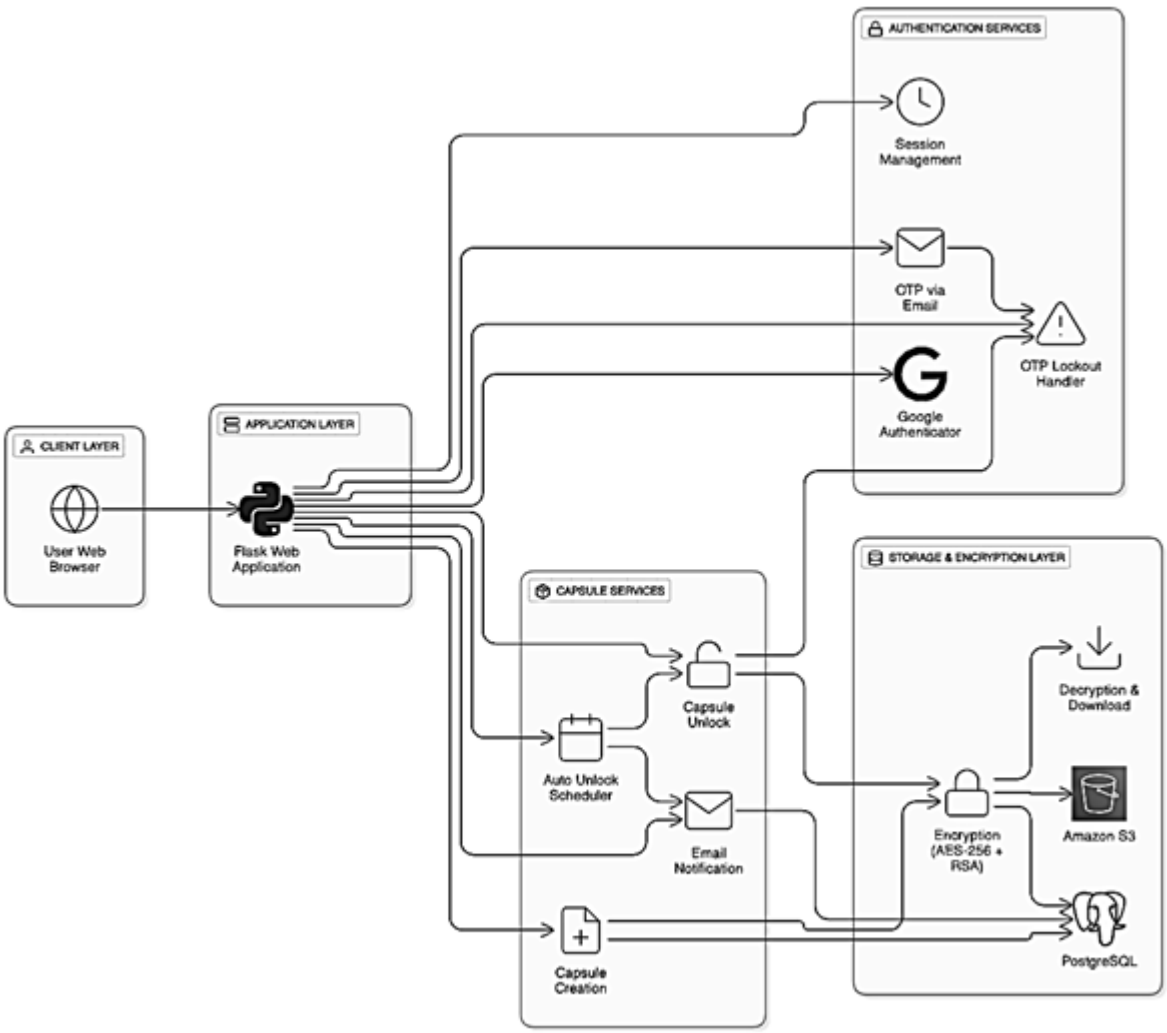

Figure 1: System Architecture of the Digital Time Capsule

4.2. Data flow and sequence

This subsection summarizes the run-time data flow in a short sequence so the architecture and responsibilities are explicit as shown above in Figure 2.

1. Client uploads file and selects capsule name and unlock datetime → request sent over HTTPS to the Flask application.

2. Application server generates a fresh AES256 key, encrypts the file, wraps the AES key with the server RSA public key, and uploads the ciphertext to AWS S3 using the S3 API (HTTPS). The

encrypted AES key and capsule metadata (capsule_id, user_id, S3 object key, wrapped AES key, unlock_datetime, TOTP secret) are stored in PostgreSQL.

3. The server generates an otpauth: rovisioning URI (containing capsule_id, user email, capsule name and the Base32 secret) and renders a QR code. User scans it with Google Authenticator to bind the TOTP channel.

4. A scheduler (APScheduler) periodically checks PostgreSQL for capsules whose unlock datetime less than or equal to server time. When the condition is met, the capsule status is updated to unlocked in the database and an HTML auto-unlock email is sent to the user (SMTP over TLS). The file remains in S3 only the capsule status changes until the user completes TOTP verification.

5. On unlock attempt, the server verifies:
(a) Capsule is unlocked,
(b) TOTP matches the capsule secret, and
(c) OTP attempt limits are respected. On success the server decrypts the wrapped AES key with the RSA private key, uses the AES key to decrypt the file (server-side), and returns a secure, time-limited download link or streams the file to the authenticated user.

6. Controlled deletion requires a separate email OTP and explicit confirmation; on successful verification the application deletes the S3 object and the database rows in a transactional manner and logs the operation for audit.

The execution process of the system is represented in a map that contains the main steps of the process, that is, creation of the capsules up to controlled removal. This diagram is aligned with the description made above and shows how multiple parts of the system interact, such as the client, application server, scheduler, database, cloud storage, key vault and email service. The architecture of the system is made up of clear trust and communication boundaries that determine the amount of protection necessary to each component and regulate the flow of data between them. These limits are used to detect the possible vulnerabilities and define the use of specific security measures. The system pertinent boundaries involved to the security posture are the following:

1. Client Boundary: The user interacts with the system through a web browser and all server interactions are made over secure HTTPS so that such interactions are not intercepted and manipulated. Since the user device can not be trusted, the server needs to make sure that such sensitive information as QR codes, OTPs and credentials are well secured.

2. Application Server Boundary: The Flask backend is the main trust boundary, and it performs sensitive operations such as encryption, decryption, and OTP validation. This boundary is specifically designed to store the RSA private key in a secure manner and only authorized access is allowed to avoid unauthorized exposure. In order to curb the security risk, strict access controls and logging should be implemented.

3. Storage Boundary (AWS S3): The only form of data that passes through this boundary is encrypted files and IAM rules are used to impose stringent access control restrictions that can only allow the backend server to make contact with S3 objects. Users cannot access the files directly, thus they cannot get unauthorized access to files.

4. Database Boundary (PostgreSQL): Metadata, encrypted AES keys, OTP secrets, and integrity hashes are stored here. The database is treated as a partially trusted component because a compromised DB account could modify data, so least-privilege roles and integrity validations are applied.

5. Authentication Boundary: QR provisioning and TOTP verification occur across a boundary between the server and the user's authenticator device. Because TOTP secrets are embedded in QR codes, interception of the code would compromise the capsule's second factor. Therefore, QR codes are delivered only through authenticated HTTPS sessions. These boundaries explain how data moves across components and where protective measures (HTTPS, IAM policies, hashing, access controls) are necessary to maintain system security.

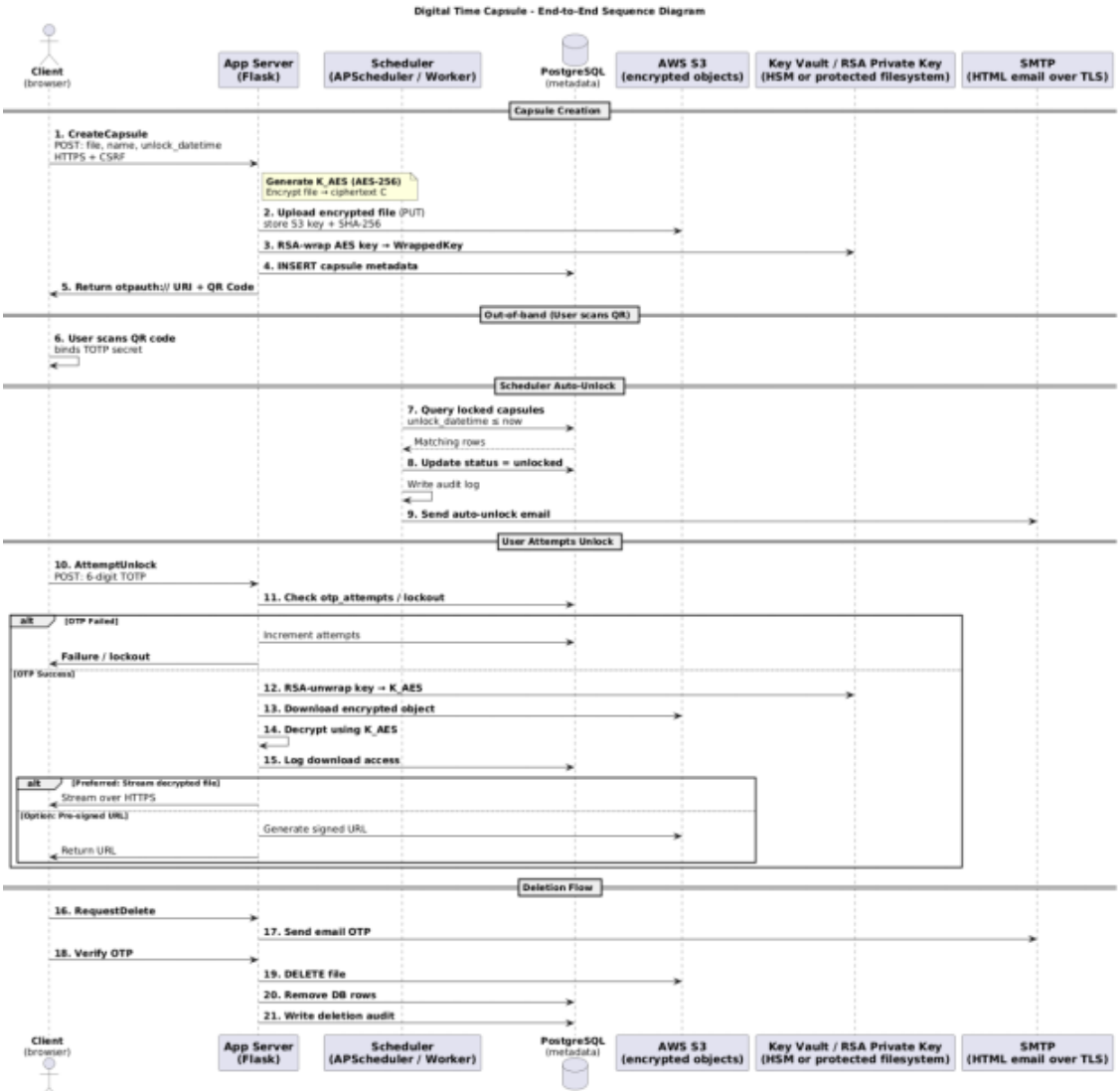

Figure 2: End-to-End Sequence Diagram of the Digital Time Capsule

4.3. System Workflow

The workflow for the Digital Time Capsule system is depicted in Figure 2. This is a user-driven event where someone decides to upload a file and gives the necessary information for a time capsule like the name, unlock date, and time. After the data is sent, the system comes up with a QR code that the user gets the Google Authenticator to scan. In this way, TOTP based authentication is set up which is a prerequisite for the creation of the capsule. Once OTP verification is done, the file undergoes encryption through AES-256, and the AES key is wrapped with RSA encryption. The encrypted file finds its way to AWS S3, and at the same time, the encrypted key along with the metadata get stored in PostgreSQL. The capsule is then rendered inaccessible until the time when it is supposed to be opened. The moment the unlock time is there, the system is set to dispatch an email notification to the owner. To open the capsule, the user has to provide the TOTP from Google Authenticator. If the OTP verification is successful, the system goes ahead to get the AES key in a readable format and thereafter, decrypt the file which the user can then download or delete. Such a flow of work command is what guarantees that the encryption is done in a secure way, that access is given at the right time, and that there is a robust authentication of the kind OTP-based.

The methods used by the Digital Time Capsule system are centered around a safeguarded, temporally regulated workflow that allows users to deposit digitally encrypted files and to retrieve them only after a certain release time have elapsed. First of all, user registration leads to login through an authentication mechanism that is secure and comprises session management and an OTP-based verification step. After successful authentication, the user is allowed to upload any file that may be a document, an image, a video, an audio file, or a compressed archive. On the way to the server, the file is instantly subjected to a two-layer encryption process so as to provide privacy and data integrity. AES-256 is the method used to encrypt the file, and the corresponding symmetric key is, moreover, encrypted with RSA before the storage process. This double-encryption method is what keeps hackers from being able to access the files even if they have caused a breach in the storage systems. After the user sets the future unlock date and time, the system creates a QR code, the user then scans it with the Google Authenticator app. This process creates a unique TOTP channel that links the capsule with the user's authenticator app. After the capsule is ready, the server-side programs are on the lookout all the time by means of an auto-unlock scheduler that is always running. At predetermined intervals this scheduler reviews the times for each capsule, changes the status when the unlocking time comes, and in an automatic manner sends an email informing the user that the capsule is already available for access. The unlocking of the capsule depends on two factors only: the time of unlocking has come and a valid TOTP is given. In case a user keeps entering an incorrect OTP and doing so multiple times, the system has a temporary lockout feature that stops the user from trying a brute-force attack and ensures that only legitimate users can access the resources. In case a user decides to delete a capsule, another security step is added in this situation.

An OTP is dispatched to the email address that was used during registration and only when the verification is successful and the confirmation is explicit does the system proceed to remove the file permanently from both AWS S3 and the PostgreSQL database. This method includes very limited access to the control functions, very strong encryption, and management of the time of release to the minute. By tightly controlling that no one can access the files before the time of unlocking and that only users who have passed authentication can get to the files, the Digital Time Capsule system can be used in situations like personal archiving in a secure way, the storing of legal evidence, and general long-term digital record keeping, as well as other applications that require the combination of confidentiality and scheduled accessibility.

4.3.1. Unlock-Time Enforcement Logic and UI State Validation. The unlock button shown to the user is controlled entirely by the server's evaluation of unlock_datetime, never by the client browser time. This prevents users from manipulating their local clock to bypass timed access.

Server-side eligibility check: A capsule is considered eligible for unlock only when the backend determines:

$$current\_server\_time\_utc >= unlock\_datetime\_utc$$

APScheduler updates the capsule's status field to "unlocked" during its periodic check.

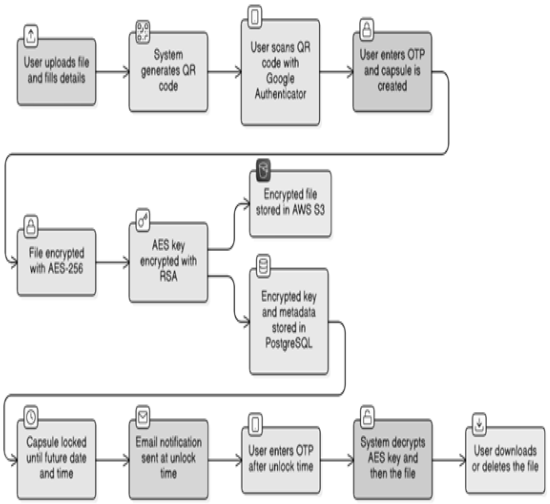

Figure 3: Workflow of the Digital Time Capsule system

UI logic: The dashboard renders the unlock button only if the capsule's status in PostgreSQL is "unlocked". This prevents early unlocking even if the user edits their system clock. A major challenge that existed in the prototype was addressed, where the button did not display despite passing the unlock time. However, upon further investigation, it was found that this issue was attributed to variations in the format of time zones being compared in the database, where there was a disparity between local time and UTC time. To address this challenge, modifications were applied to ensure that all time zones are compared using a uniform format that ensures the logic algorithm functions as expected.

- Storing all unlock datetimes in UTC,
- Converting all comparisons to UTC,
- Ensuring the scheduler uses datetime.utcnow() consistently,
- Normalizing timestamps during capsule creation and status queries.
  After enforcing consistent UTC comparison and normalizing all timestamps, the unlock button reliably appears within the next scheduler cycle (typically 5 seconds).

4.4. Research Method and Evaluation Design

The research design in this study will be in the form of a system-development approach, and this will focus on the design, development, and evaluation of the complete functional prototype of a Digital Time Capsule. In the evaluation phase, one attempts to verify the reliability of the system, its accuracy in the task of imposing timed access and also capability to provide secure file storage, retrieval, and deletion. To do so, the study will be organized into three main stages, which are the implementation of the system, the controlled functional testing, and the assessment of the performance of the prototype.

1. System Development: Digital Time Capsule application was developed in a mixture of technologies, such as Flask, PostgreSQL, AWS S3, AES-256 encryption, RSA-2048 key wrapping, and TOTP using Google Authenticator. After the introduction of basic modules, including encryption, capsule creation, QR code provisioning, scheduled unlocking, OTP verification, and controlled deletion, the system was deployed in a development environment, which simulated a typical web-based environment. This granted the possibility of realistic testing of the whole workflow.

2. Testing and Validation: Set of test cases was developed, which tested the behavior of the system in normal and exceptional cases. These tests were to be conducted to ensure the functionality, security and performance of the system under different scenarios to guarantee reliability and robustness of the system.

   - Uploading files of different sizes and formats,
   - Generating capsule-specific QR codes and verifying TOTP binding,
   - Enforcing minimum unlock-time rules,
   - Rejecting early unlock attempts,
   - Validating correct and incorrect OTP submissions,
   - Enforcing lockout periods after repeated failures,
   - Checking controlled deletion with email-based OTP.

   Other tests were used to test the threat model related conditions, e.g. attempts to. pre-unlock files, tries to bypass OTP checks or tries to alter capsule. metadata. These tests were done to make sure that every module was working properly as well as the system was being respectful. confidentiality, integrity and timed-release constraints.

3. Prototype Performance Evaluation: Performance in time to examine the system efficiency. core operations were measured directly in those areas that have the biggest impact on user experience. These include AES encryption and decryption, RSA wrapping and unwrapping of AES keys, upload and download latency on AWS S3, TOTP verification time, scheduler unlock checks, and deletion workflow. Each measurement was repeated multiple times, and the average value was recorded. This method captures realistic end-to-end behaviour of a working prototype rather than isolated cryptographic benchmarks.
   Scope and Limitations This evaluation reflects a prototype running on a single server with moderate load. Large-scale benchmarking, cost analysis for long-term S3 storage, and usability studies with real users remain outside the scope of this work and are identified as future extensions.

4. Evaluation Methodology: The evaluation of the Digital Time Capsule prototype focused on assessing its functional correctness, performance, and reliability under typical usage scenarios. Since the system integrates multiple components encryption, cloud storage, scheduling services, OTP verification, and controlled deletion the evaluation was structured to measure each stage independently as well as in an end-to-end flow. The assessment was conducted in a controlled development environment using a Flask backend, AWS S3 for encrypted file storage, and PostgreSQL for metadata management.
   The evaluation consisted of three parts. First, core cryptographic operations were measured by recording the time taken for AES-256 encryption, RSA-2048 key wrapping, RSA key recovery, and subsequent file decryption. The assessment was done through the evaluation of the performance of the system under various important areas. Firstly, the effect of encryption pipeline on user experience was quantified to establish whether it had any perceptible delay. Also, interactions at the system level were analyzed; upload latency to S3, unlock detection via schedulers, OTP verification delay, and deletion processes. Several test capsules of different file sizes and different unlock timepouts were developed to test the consistency of the scheduler to mark capsules as unlocked and issue auto-email notifications at the right time. Moreover, a complete test of the lifecycle of the capsule was performed, which included uploading and encrypting files, unlocking and decrypting, and deliberate deletion. This holistic analysis brought to light the cumulative impact of the network conditions, database queries and application level processing in the system. The Digital Time Capsule design is a representation of a number of intentional engineering decisions. As an example, unlock-time enforcement was chosen, and APScheduler was used because it is simple and easily integrated with Flask. Other options like Celery beat or distributed task queues were also available but were not taken as the prototype did not need the extra infrastructure. AES keys have been decided to be wrapped with RSA-2048 because it would be compatible and fast enough with

Table 2: Notations Used in the Digital Time Capsule System

Symbol	Meaning
F	User file uploaded to the system
K_AES	256-bit AES symmetric key used for encrypting F
K_pub, K_priv	RSA public key and private key
C	AESencrypted ciphertext stored in AWS S3
E_AES(F, K_AES)	AES encryption of file F with key K_AES
D_AES(C, K_AES)	AES decryption of ciphertext C with key K_AES
E_RSA(K_AES)	RSA encryption of AES key
D_RSA(E_RSA(K_AES))	RSA decryption to recover AES key
T	Current Unix timestamp
S	TOTP secret assigned to each capsule
OTP	6-digit one-time password generated every 30 seconds
t_u	User-selected unlock datetime

performance. The more sophisticated cryptographic measures, like elliptic-curve cryptography or RSA-3072, can be considered in the next stage of development. Time enforcement was chosen on the server side as it was very simple and easy to impose. The other mechanisms, such as proof-of-work time anchors, and verifiable delay functions, can be added in the next versions. Lastly, Base32, consisting of otpauth, was used to encode TOTP secrets so that they are compatible with and usable.

5. Results and Discussion

5.1. Mathematical Equations of Cryptographic Computation

This subsection presents the essential mathematical expressions used in AES, RSA, and TOTP computation. AES-256 is a symmetric block cipher that encrypts the uploaded file using a 256-bit key.

$$Encryption : C = E\_AES(F, K\_AES) \tag{5.1}$$

$$Decryption : F = D\_AES(C, K\_AES) \tag{5.2}$$

One AES round (simplified):

$$State\_round = AddRoundKey(MixColumns(ShiftRows(SubBytes(State\_previous)))) \tag{5.3}$$

This sequence represents nonlinear substitution, permutations, linear mixing, and key addition.

5.1.1. RSA Protection of AES Key. RSA secures the AES key during storage and transmission. The RSA algorithm uses the following equations:

$$n = p \times q \tag{5.4}$$

$$\phi(n) = (p-1)(q-1) \tag{5.5}$$

$$d \times e \equiv 1 \pmod{\phi(n)} \tag{5.6}$$

Encryption:

$$K\_enc = (K\_AES)^e \mod n \tag{5.7}$$

Decryption:

$$K\_AES = (K\_enc)^d \mod n \tag{5.8}$$

This ensures that only the server can retrieve the original AES key

5.1.2. Time-Based One-Time Password. Each capsule has its own secret S. A new OTP is computed every 30 seconds.
Formula:

$$OTP = Truncate(HMAC\_SHA1(S, floor(T/30)))mod1,000,000 \tag{5.9}$$

Where T is the current Unix time.

5.2. Comparative Benchmark Overview

To help situate the present prototype against prior timed-release or related secure-storage systems, Table 4 summarizes measured prototype timings from this work alongside characteristic performance notes from representative prior systems. Direct numeric comparison is difficult because many prior systems target different workloads (for example, encrypted streaming or trusted-execution hardware) and report different performance metrics (throughput, latency under heavy load, or hardware-specific numbers). For transparency, the table below lists our measured average times (from the prototype environment described in Section 6.6) and gives concise, literature-based notes for each reference so the reviewer can see where designs and evaluation goals differ.

Interpretation and limitations. The prototype timings reported here (Table 4) are realistic measurements for a single-file, development environment and are useful for showing feasibility and low-latency encryption/decryption in a typical web-deployed flow. However, comparing raw milliseconds across the literature is often misleading:

1. Many prior works focus on different performance axes (stream throughput, TEE enclave costs, or provable cryptographic delays)
2. Testbeds and network conditions differ, and
3. Some systems purposely trade latency for stronger theoretical guarantees.

We therefore present these prototype numbers to show practical feasibility and recommend a dedicated benchmark study (future work) that runs multiple systems under a common testbed for a rigorous head-to-head comparison.

Table 3: Comparison of representative timed-release systems and the Digital Time Capsule

System (ref)	**Operation / metric reported**	**Reported / measured value (ms)**	**Test environment / note**	**Comment on comparability**
Digital Time Capsule (this work)	File encryption (AES-256)	20–40 ms	Dev prototype (Flask backend) — single-file encryption	Direct measurement from this work (see Section 6.6). Comparable to single-file encryption scenarios.
Digital Time Capsule (this work)	AES key RSA wrap / unwrap	3–7 ms (enc), 5–8 ms (dec)	Dev prototype — RSA-2048	Prototype timings for key wrapping/unwrapping; small payload crypto costs.

Digital Time Capsule (this work)	Upload to cloud (S3)	150–400 ms (network dependent)	Dev network (see Section 6.6)	Network latency dominates; measured in our environment.
TimeCrypt [9]	Encrypted-stream processing (latency/ throughput)	N/A (streaming throughput focused)	Server-side, stream-oriented workloads	TimeCrypt focuses on continuous encrypted streams and access-control throughput; metrics are not directly comparable to single-file encryption latency.
TEEKAP (TEEs) [10]	Self-expiring capsule actions (TEE-based latency)	N/A (hardware-dependent)	TEE-equipped hardware (e.g., Intel SGX)	Performance depends strongly on TEE platform; cannot be directly compared to our web-stack prototype.
ATDD (deletion) [11]	Deletion assurance operations	N/A (policy guarantees)	Cloud-focused, policy enforcement	ATDD reports deletion guarantees and cloud-policy metrics rather than single-file crypto latency.
Other literature (provable timed-release, multi-server TRE, etc.) [1–8,12–15]	Cryptographic primitives & system-level behaviors	N/A	Varied conceptual/ cryptographic analyses	These works emphasize formal guarantees, model complexity, or fault tolerance; they generally do not publish single-file encryption latency for a web prototype.

5.3. Scalability & Stress Tests — Proposed Experiments

To evaluate real-world viability, this section defines a set of scalability and stress tests appropriate for a timed-release cryptographic storage system. To address this, we define a compact set of experiments that can be executed on the prototype (or described as planned work if not yet run). Each experiment specifies the test environment, metrics to collect, method, and acceptance criteria that help interpret prototype-level results.

Table 4: Experiment Matrix

Test	**Environment (s)**	**Metric(s)**	**Method**	**Acceptance Criteria (prototype)**
S3 upload/download under load	Single instance, 100 concurrent clients	Per-request latency (ms), throughput (MB/s), error rate	Upload 10–100 files concurrently (sizes 1–100 MB); record latencies and failures	Median upload latency <500 ms; error rate <1%.

Scheduler stress (unlock storm)	Single instance + DB with synthetic capsules	Time-to-mark-unlocked (ms), DB CPU, scheduler cycle time	Insert N capsules with the same unlock time (N = 100, 1k, 10k) and measure time to update status	100 capsules: <10 s; 1k capsules: <60 s
DB read/write scaling	Local DB vs tuned instance	Query latency, connection saturation, error rate	Simulate concurrent capsule creation/read operations (concurrency 1–500), record P50/P95/P99 latencies	P95 latency <200 ms at 100 concurrent clients
End-to-end unlock latency	Full decrypt flow	End-to-end latency (OTP verify + RSA unwrap + AES decrypt + download link)	Trigger unlock + OTP validation + file download; repeat with concurrent clients	<1.5 seconds for small files

Notes on Interpretation: The tests are prototype level feasibility tests which are used to detect performance bottlenecks. Dependent on hardware in servers, location, network situation, and S3 setup, results will differ. In case bottlenecks are observed, the possible areas of improvement can be:
1. Transferring unlock processing to a distributed task queue (Celery + Redis),
2. Unlock-time- Window scheduler checks, or
3. Parallel unlock processing with a partitioned worker pool.
The current study is not an extensive one because it would require large-scale benchmarks in several cloud regions. recommended for future work.

Reporting of Results:
The results of the scalability and stress-test experiments will be presented in terms of the median and. Values of P95 latency, as well as additional information regarding the test environment, instance type, etc. region, and configuration. In this way, it is possible to compare it with the acceptance criteria provided in. Table 5 and makes it easy to understand system behavior when the loads vary. Any experiments of large scale which are not done at this stage will be incorporated in this section as part of the planned. assessment system, making it methodologically transparent and complete.

6. Algorithms Used

The Digital Time Capsule system uses several cryptographic and system-level algorithms to ensure confidentiality, integrity, availability, and timed access. The descriptions below provide clear definitions and explain how each algorithm works and how it is applied in the system.

6.1. AES-256 File Encryption

AES-256 (Advanced Encryption Standard) is a symmetric-key algorithm that encrypts data using a 256-bit key. It operates on 128-bit blocks and performs 14 rounds of substitutions, permutations, and mixing functions
How it works:

1. File F is divided into 128-bit blocks.
2. A 256-bit key K_AES is expanded into multiple round keys.
3. Each round performs SubBytes, ShiftRows, MixColumns, and AddRoundKey.
4. The final encrypted output is ciphertext C.

Every uploaded file F is encrypted before leaving the users device:

$$C = E\_AES(F, K\_AES) \tag{6.1}$$

The encrypted file stored on S3 ensures confidentiality even if the storage bucket is accessed.

6.2. RSA Encryption of AES Key

RSA is an asymmetric cryptographic algorithm using a public–private key pair. It secures small pieces of data such as encryption keys modular exponentiation.
How it works:

1. Two large primes p and q generate modulus n.
2. Public key: (e, n).
3. Private key: (d, n).
4. Encryption:

$$x^e mod(n) \tag{6.2}$$

$$x^d mod(n) \tag{6.3}$$

5. Decryption:

The AES key is encrypted using the RSA public key:

$$K\_enc = E\_RSA(K\_AES) \tag{6.4}$$

Later, during capsule unlocking, the server decrypts the key:

$$K\_AES = D\_RSA(K\_enc) \tag{6.5}$$

This ensures that even if someone downloads the encrypted file, they cannot decrypt it without K_priv.

6.3. Time-Based One-Time Password (TOTP)

TOTP is a time-synchronized authentication mechanism that generates a new OTP every 30 seconds using a shared secret S.
How it works:

1. Unix time T is divided into 30-second intervals.
2. HMAC-SHA1(S, T/30) produces a hash.
3. The hash is truncated to a 6-digit OTP.

During unlocking, the user must enter:

$$OTP = Truncate(HMAC\_SHA1(S, floor(T/30)))mod 1,000,000 \tag{6.6}$$

Only if the OTP is valid does the system allow AES key decryption.

6.4. QR Code Provisioning Algorithm

A QR provisioning algorithm converts a TOTP configuration into a scannable format.
How it works:

1. A Base32 secret S is generated.
2. An otpauth:// URL is constructed with:
    - User email
    - Capsule name
    - Capsule ID
    - Secret S
1. The URL is encoded as a QR code.
2. The user scans it with Google Authenticator.

The QR code binds the capsule's authentication to the user's device, ensuring capsule-specific OTP generation.

6.4.1. Auto-Unlock Scheduler Algorithm. A scheduler checks conditions at fixed intervals and performs actions automatically.
How it works:

1. APScheduler runs every few minutes.
2. It compares t_u (unlock time) with current server time.
3. If current time ¿= t_u, the capsule is marked "unlocked".

The scheduler performs:

- Auto-unlock
- Auto-email notification
- Logging for audit tracking

This ensures the unlock event happens even if the user is offline.

6.4.2. SHA-256 Hashing. SHA-256 is a one-way hashing algorithm producing a 256-bit digest.
How it works:

- Input is processed in 512-bit blocks
- Multiple rounds of bitwise operations
- Produces a fixed-length 256-bit hash

SHA-256 is used for:

- Password hashing
- Internal identifiers
- Data integrity checks

6.5. System Modules

The Digital Time Capsule system contains several modules, which are interlinked together, each responsible for a part of the workflow. These ensure secure file handling, time-controlled access, OTP verification, encrypted storage, and controlled deletion. The major system modules used in the implementation are described below.

1. **User Management Module** The major functions of this module have to do with user registration, logging in and sessions. The module stores usernames and hashed passwords securely, ensures authenticated users access their capsules, manages login-based OTP verification, creates sessions, and checks user identities. This module is the backbone of access control security within the system.

2. **File Upload and Encryption Module** After the user logs in, they can upload any file, such as a document, image, video, or compressed file. The module generates a new 256-bit AES key and encrypts the uploaded file using AES-256. Then, the AES key is encrypted by the RSA public key of the system. The prepared encrypted file is to be safely stored in cloud storage that maintains confidentiality even though the storage itself has been compromised.

3. **Capsule Creation Module** This module deals with the process of creating a "digital capsule" that combines an encrypted file with metadata (capsule name, capsule ID, unlock date and time, encrypted AES key, and S3 storage reference). In addition, it communicates with the system by, for example, the unlocking date being a date in the future and if the unlocking date is today then the unlocking time is at least five minutes ahead. Hence the tool is one that is utilized to effectively organize and keep each capsule.

4. **QR Code and TOTP Setup Module** The main goal of this module is to supply a single Base32 secret key for each capsule with which an otpauth URI is created and the QR code is generated. The code consists of the user's email, capsule name, and capsule ID which are the data that the QR code represents. By the help of the QR code scanned with Google Authenticator the user binds the capsule to the TOTP account that will be used later for unlocking. This is one of the means which guarantee security based on the user's identity.

5. **OTP Verification Module** When unlocking, the user is required to give a one-time password (TOTP) from Google Authenticator. The module makes sure that TOTP is the right one thus, the same authenticator account that was used during the creation of the capsule activities is the one that issued the TOTP and it is within the allowed time frame. It is a part of the same system as well that prevention work is done by the device which after 3 wrong attempts at OTP turning off the capsule for 24 hours thereby co protection against brute force attacks at the same time.

6. **Cloud Storage Module (AWS S3)** Essentially it is this module that is having the responsibility of backing up the encrypted files in an adequately secure S3 bucket on the Amazon server. It is a service for encrypted file uploads, a creator of keys for S3 objects, and instead of the place where the file is, the S3 URL is the database. Subsequently, the module will get the file from S3 for the decryption process when the opening of the capsule takes place.

7. **Scheduler and Auto-Unlock Module** By current time, the background work done by APScheduler is, therefore, checking at intervals whether there are any capsules that can be opened, their unlock time having passed, and thus, the state of the capsule is changed to unlocked automatically. Also, the user gets an email from it, letting him know that the capsule is now available to him. In addition to that, the module keeps a system log of all instances when the auto-unlock feature is employed for audit purposes.

8. **Decryption and Download Module** This module does the content decryption work when a user unlocks the capsule and enters a correct OTP. The content decryption key or AES key was wrapped, so it is the RSA private key that decrypts the AES key. Basically, the original file that was made available for download is the content that has been encrypted.

9. **Capsule Deletion Module** One can delete a capsule which is a feature available after unlocking and either viewing or downloading the capsule. The module supports a deletion process which is sending an OTP to the user's registered email, verifying the OTP, and giving the last confirmation prompt. It is upon confirming that the executions are removing the tasks of the capsule from both S3 and PostgreSQL. Deletion that is done in a two-step verification method is safe and intentional.

10. **Database Management Module** The module is the conductor of every operation of the PostgreSQL database and the module itself. It takes part in storage management of user data, capsule metadata, unlock timestamps, encrypted AES keys, OTP secrets, S3 file references, and activity logs.

6.6. Prototype Evaluation and Performance Metrics

A prototype evaluation was conducted to measure the effectiveness and speed of the Digital Time Capsule system mainly focused on the operations recording, encrypting, storing, unlocking, and deleting the capsule. The system was tested in a development environment where Flask was used as the backend, AWS S3 for the storage of encrypted files, and PostgreSQL for metadata management. The present evaluation is only a resourcefulness and speed demonstration of the real system that is being used and not the cryptographic performance of the operation.

7. Security Analysis and Threat Model

The Digital Time Capsule system aims to keep user data safe by combining strong encryption, time-controlled access, OTP verification, and protected cloud storage. This section explains the security goals, the assumptions under which the system operates, the expected threats, and the measures used to reduce those risks.

7.1. Assumptions and Trust Boundaries

The system works under a few basic assumptions about the environment in which it runs. It is expected that the application server is properly maintained and not under an attacker's control. The RSA key pair that is used to envelop the AES keys is only generated and stored on the backend. Wrapping of the AES key during the creation of the capsule is done using the public key whereas the private key is stored in a secure section of the backend application such that only the processes of the server can access it. The security model of the system presupposes that all the intercourse between users and the server is built on the secure channels of HTTPS, and important data, such as logins, TOTP secrets, and metadata, is not intercepted. Furthermore, the system is based on the ability of the user to protect his/her Google Authenticator application and QR codes because the revelation of these resources may compromise the security. Moreover, the system relies on an adequately set up AWS S3 bucket with stringent IAM policies, which do not allow access by the public. Any breach of these assumptions, e.g. breached server, leaked personal key or misconfigured S3, may compromise the security guarantees of the system, which establishes the limits within which the system would run safely.

7.2. Time Resilience and Unlock Timing Dependability.

Digital Time Capsule is based on the internal clock of the server to provide access to encrypted files. In order to achieve proper access control, the clock of the server should be synchronized. Within a production environment, this can be done through the provision of an NTP-based time syncing service like Chrony, which refers to several trusted NTP servers, to ensure the precision of the clock and eliminate the drift. This methodology would allow the system to maintain consistent timing despite network unreliability or sluggish network connections. In case the primary time source fails, or demonstrates an abnormal drift, the server is able to switch to a secondary synchronized source in order to remain accurate. With these safeguards in place, the platform ensures that capsules open exactly when they are supposed to neither earlier nor later addressing a key reliability concern in systems designed for timed-release access.

Table 5: Prototype Evaluation Metrics of the Digital Time Capsule System

Operation	Description	Average Time
File Encryption (AES-256)	Time taken to encrypt an uploaded file using a new 256-bit AES key	20–40 ms
AES Key Encryption (RSA-2048)	Time to encrypt the AES key with the RSA public key	3–7 ms
AES Key Decryption (RSA-2048)	Time to decrypt the encrypted AES key using the private key	5–8 ms
File Upload to AWS S3	Uploading encrypted file to cloud storage	150–400 ms (network-dependent)
Capsule Creation Process	Metadata insertion, AES key encryption, QR generation	50–90 ms
TOTP Verification	Time to validate OTP using the shared secret (S)	1–3 ms
Scheduler Auto-Unlock Check	Background job that checks capsule unlock times	5 ms per check (runs every 5 seconds)
File Decryption After Unlock	Time to decrypt AES-encrypted file using recovered AES key	18–35 ms
Capsule Deletion Workflow	Email OTP generation + confirmation + S3 + DB delete	120–190 ms

7.3. Security Objectives

Confidentiality: Only an authorized user who passes the OTP checks and reaches the unlock time should be able to decrypt a capsule. Each file is encrypted with AES-256 before upload.
Integrity: The encrypted file, the AES key, the unlock time, and the capsule metadata must not be altered without detection.
Integrity and S3 safeguards To ensure data integrity for S3-stored objects, every file uploaded is hashed with SHA-256 before upload and the hex digest is stored as part of the capsule metadata in PostgreSQL and also added to the S3 object as non-public metadata. Versioning is enabled on the S3 bucket so older copies can be recovered if needed. When a user attempts to download or unlock a capsule, the server recomputes the SHA-256 hash and compares it with the stored value; any mismatch stops the download and raises an alert. For larger files, hashing is done in a streaming manner to avoid relying on S3 ETags, which may not reflect the real checksum. A background verifier can also re-check stored objects periodically and write results to the audit log. These steps provide reliable detection of accidental corruption or unauthorized modification of stored data.
Availability Capsules that have reached their unlock time should be available for access without delay.
Authentication Login and capsule unlocking rely on password-based authentication and TOTP verification.
Access Control Unlocking is only allowed after the selected date and time. Only the server time is used for verification.

7.3.1. Integrity Verification, S3 Object Validation, and Tamper Detection. With the objective of ensuring that the encryption capsules on the AWS S3 storage solution are not compromised in the backend without being detected, the system makes use of the SHA-256 Digest for verification between the PostgreSQL metadata and the S3 object metadata. **Generation and storage of hash.** In the creation of the capsule, the encrypted file is included in the calculation of the SHA-256 hash. The result is stored in two different places:

1. **PostgreSQL** (capsule_hash field in the capsule metadata),

2. **S3 object metadata** private metadata key x-am

For maintaining the integrity of the data, the SHA-256 hash is stored independently of the object body. This facilitates the identification of whether there are any unlawful alterations or damages. In the unlocking procedure, it also checks the integrity of the retrieved encrypted file from the S3 bucket by calculating the SHA-256 hash and matching it with the stored values. On the detection of any discrepancy, it immediately takes properaction by terminating the unlocking procedure.

- Prevents decryption, and
- Logs a high-severity security alert for forensic analysis.

Versioning and rollback protection. The version control feature on the S3 bucket ensures that unexpected overwrites, either malicious or accidental, will not permanently overwrite the original ciphertext. This means that whenever there's a mismatch, the system can go back and check the integrity of the previous version.
Background verifier. A periodic verification process can be performed (every day or every week) to re-verify all stored ciphertexts by comparing the current SHA-256 hash with what is stored. If any discrepancies are found, they are logged in the audit log.

7.3.2. QR Code Security, TOTP Binding, and Protection Against Cross-Device OTP Abuse. "Each capsule in the Digital Time Capsule system has its own unique TOTP secret, expressed in the provisioning URI via the otpauth:// scheme, given to the user via a QR code. This effectively makes the capsule authentication per-file, rather than per-account, enhancing isolation between capsules."
Binding the capsule to the user's authenticator. When the user scans the QR code during capsule creation, Google Authenticator stores the Base32 secret S locally on the device. The system records a hash of the secret in PostgreSQL to verify future OTP submissions. Because only the user's authenticator device contains the original secret, the capsule can only be unlocked by the same scanned authenticator, ensuring the requirement: The same Google Authenticator account used during creation must also be used to unlock the capsule.
Rationale and security benefit.

- Even if an attacker knows the user's login password or email OTP, they cannot unlock the capsule without the per-capsule TOTP secret.
- Because each capsule has a different secret, compromise of one capsule's QR code does not endanger others.
- An attacker who attempts to submit OTPs from another device will always fail because the generated values will not match the stored HMAC-SHA1(S , time) sequence.

Risks of QR leakage and mitigation.
The QR image contains the full secret. If the QR is photographed or intercepted, a second person could generate valid OTPs at the correct unlock time. To reduce this risk, the system enforces:

- QR delivery only over authenticated HTTPS sessions
- QR images are never cached or stored in S3
- QR is shown only once at creation; revisiting it requires full re-authentication

Future versions may include short-lived provisioning URLs, watermarking, or encrypted QR codes to further prevent misuse. This model ensures that capsule unlocking depends jointly on correct timing and possession of the exact same authenticator used at creation, preventing replay attacks, cloned authenticator apps, or unauthorized OTP generation.

7.4. Threat Model

The system considers an attacker who might attempt to:

- Access another user's capsule or metadata
- Guess or repeatedly try OTP codes
- Change the unlock time or manipulate metadata
- Retrieve encrypted files directly from AWS S3
- Intercept or alter network traffic
- Attempt SQL injection or tamper with database information
- Replay old requests or bypass authentication
- Capture the QR code during provisioning
- Exploit a compromised server or leaked key
- Take advantage of incorrect cloud or IAM configurations

The system includes several layers of security to counter these threats.

7.5. Possible Attacks and Security Measures

1. Unauthorized Access to Capsules
   Threat: An attacker tries to download encrypted capsules from S3 or send false unlock requests. Protection:
   - Files are encrypted with AES-256 before being uploaded
   - AES keys are protected using RSA public-key encryption
   - Stolen files remain unreadable without the decrypted AES key
   - IAM policies give access only to the backend server
2. OTP Guessing Attempts:
   An attacker repeatedly enters incorrect OTP values to guess the valid one. Protection:
   Actual brute forcing is not possible in this system because of strict rate limits:
   - Only three OTP attempts are allowed
   - After three failures, the capsule is locked for a full day.
   - For login, three wrong OTPs trigger a one-hour lockout with an email alert
   - OTPs change every 30 seconds, reducing the guessing window. These limits make repeated guessing ineffective.
3. Early Unlock Attempts Threat: An attacker attempts to change the unlock time or bypass the waiting period. Protection: **Protection:**
   - Unlock checks are performed using server time
   - Client device time cannot influence unlock eligibility
   - The scheduler monitors all capsules and updates status only when allowed
4. Database Attacks (SQL Injection or Manipulation) Threat: An attacker tries to modify capsule data, keys, or unlock times. **Protection:**
   - SQLAlchemy uses parameterized queries

- All user inputs are validated
- The PostgreSQL role has limited permissions
- Sensitive data such as TOTP secrets and encrypted keys are stored safely

5. Network Attacks (MITM, Sniffing, Replay) Threat: An attacker listens to or modifies traffic between user and server. **Protection:**
   - The system requires HTTPS for all communication
   - AES-encrypted files are safe even if captured
   - RSA-wrapped AES keys remain protected in transit
   - OTPs refresh every 30 seconds, reducing replay risks
6. AWS S3 Data Tampering
   Threat:Attempts to alter or delete stored files. **Protection:**
   - IAM rules prevent public or unauthorized access
   - Encrypted objects cannot be modified without detection
   - S3 durability protects against accidental loss
7. Flask Session Hijacking
   Threat: An attacker tries to steal a user session. **Protection:**
   - HttpOnly cookies block JavaScript from reading session identifiers
   - Secure and SameSite flags reduce cross-site attacks
   - Sessions are invalidated when the user logs out
   - Flask's random secret key improves session safety

7.5.1. Controlled Capsule Deletion Security and Two-Step OTP Enforcement. Capsule deletion is intentionally designed as a restricted, multi-step, high-assurance process, because deletion is irreversible and can be exploited by attackers who gain temporary access to an account.
Deletion eligibility restriction. A capsule becomes deletable only after the user has:

1. Successfully unlocked the capsule, and
2. Viewed or downloaded the decrypted file. This ensures an attacker cannot delete an unopened capsule to destroy evidence or prevent legitimate future access.

Two-step OTP verification. Deletion requires an explicit two-phase confirmation:

1. The system sends a 6-digit email OTP to the registered user.
2. The user must enter this OTP correctly, after which a final confirmation screen appears ("Are you sure you want to permanently delete this capsule?").

This protects against:

- Accidental deletion
- UI-driven misclicks
- Unauthorized deletion by a person who gained short-term physical access to an unlocked device,
- Automated attacks attempting mass deletion.

Atomic deletion process (S3 + PostgreSQL). Once the user confirms the deletion after OTP verification, the system executes a transactional workflow:

- Delete the encrypted file from AWS S3 (via authenticated IAM role),
- Delete the capsule metadata and encrypted AES key from PostgreSQL,
- Write an audit log entry with timestamp, capsule ID, user ID, and IP.

The operations either all succeed or none succeed, preventing inconsistent states such as a deleted S3 file with lingering metadata, or vice-versa. Security benefits.

- An attacker can't quietly delete capsules since the email OTP is required for deletion.
- A session cookie pilfered during theft cannot achieve deletion without use of the registered email inbox.
- Augmentative/ Alternative Communication is used for people with speech or language disorders.

This controlled Deletion mechanism provides excellent protection to time-locked data, thereby counteracting potential threats that could arise due to unintended data loss, internal threats, or intentional methods to destroy data.

7.6. Security Strength Overview

The security framework provided by the system amalgamates a number of security measures: encryption via AES-256, protection of keys via RSA, two-factor authentication via TOTP, strict access control mechanisms, and cloud storage security. The comprehensive security framework provided by the system ensures that, even if the attacker tampers with the encrypted files or the network traffic, the attacker won't be able to view the data without the corresponding key or the OTP. The system's design also resists a number of attacks, which include brute-force attacks on the OTPs, early unlock attacks, attacks via manipulating the metadata, and storage access attacks on the S3.

7.7. Additional Security Considerations

Apart from the above-mentioned risks, there also exist several real-world risks in cloud implementations. Though these have not happened in prototype tests, it is essential to mention them to have an idea of the limitations of the present design.

1. Risk of Server Compromise: The Flask server alone manages the RSA private key for decrypting AES keys. A server-compromise attack may allow an intruder to ignore lock time enforcement or make illegitimate decryption requests. For the current implementation, a server trust assumption would be necessary; robustification techniques such as private key isolation, operating system-level restriction, and server monitoring would be necessary for a production environment.
2. Insider Threats related to Cloud Storage or Database Services: Even though the use of least privilege IAM roles ensures the use of AWS S3 and the PostgreSQL database, an individual administrator might possess high privileges and thus read or change the metadata stored. Although the files are stored pre-encrypted, thus ensuring the confidentiality of files, the integrity might still be compromised. Object versioning, integrity checking (SHA-256 hashes), and maintaining strong audit trails help minimize the risk factors but do not completely eliminate them.
3. TOTP Secret Exposure: The unique TOTP secret stored within the QR code in the capsule has the potential to cause exposure through the capture or unintentional transmission of the QR code image, thereby permitting unauthorized access to the capsule. This has been addressed by the secure HTTPS sessions through which the QR codes are presently being transmitted. Future improvements may involve temporary QR code URLs, watermarks, or encrypting the provisioning message.
4. Risks within Cloud Configuration: There may be risks due to incorrect S3 permissions, outdated policies, and too broad access rules, which may relate to the security of encrypted files, unauthorized deletion, and so forth. The system prototype maintains a limited access system, which may not need significant monitoring regarding cloud configuration, which may lead to risks of incorrect access rules.

5. Reliability Under Large-Scale Load: The scheduler and database were tested under moderate usage. For thousands of capsules unlocking at similar times, additional stress tests would be needed to measure latency, S3 throughput, database load, and overall system responsiveness. Horizontal scaling, distributed scheduling, or asynchronous task queues may be required in a full deployment.

7.8. Time Synchronization, Server Clock Reliability, and Unlock Accuracy

The Digital Time Capsule enforces unlock policy using the server's internal clock; therefore accurate timekeeping is essential. In the prototype the server runs Chrony (NTP) configured with multiple upstream time sources (for example time.cloudflare.com, pool.ntp.org, and a regional stratum-1 server). All unlock datetimes are stored in the database in UTC, and the application performs every comparison against the server's UTC clock to avoid DST or timezone ambiguities. Operational safeguards implemented or recommended for production:

- Continuous offset monitoring: Chrony's measured offset is monitored; when the clock offset exceeds 50 ms an operator alert is logged.
- Redundant time sources: Chrony is configured with multiple servers and automatically fails over if the primary is unreachable.
- Scheduler tolerance: the APScheduler unlock routine queries unlock $_datetime \leq now+5s$ to tolerate small scheduling jitter while avoiding premature unlocks.
- Audit logging: every auto-unlock and manual unlock attempt is logged with UTC timestamps and a snapshot of the server-reported NTP offset to support post-event forensics.
  For production we recommend managed time services, host-level monitoring (Prometheus/Grafana or CloudWatch), and alerts on abnormal drift.

8. Case Study: End-to-End User Workflow of the Digital Time Capsule

To illustrate the practical behaviour of the Digital Time Capsule, this case study presents the complete workflow followed by a typical user, Ajith, as he stores a sensitive legal document (PropertyPapers.pdf) for time-locked release. The description covers all stages of the system including registration, login, capsule creation, encryption, QR-based TOTP provisioning, unlock-time enforcement, auto-notification, OTP-secured access, and controlled deletion. This sequence demonstrates the real functioning of the prototype and shows how the platform can also be applied to digital forensic evidence preservation, legal record storage, archival work, and research datasets requiring delayed access.

Step 1: Visiting the Website and Registration

Ajith visits the Digital Time Capsule homepage. As a first-time user, he selects Register. The registration page asks for:

- Username
- Email address
- Password
- Confirm password

When he submits the form, the backend validates the data. The password is securely hashed using bcrypt (an adaptive password hashing function with built-in salting). The resulting bcrypt hash (not a plain SHA-256 value) is stored in PostgreSQL A 6-digit email OTP is generated and sent to Ajith via SMTP over TLS. he enters the OTP on the verification page. If the OTP is correct and not expired (valid for 5 minutes), registration is completed. A professional HTML email is sent confirming successful account creation.

Step 2: Login With Email OTP and Session Creation Ajith now logs in using her registered email and password. If he enters the wrong OTP three times, the account is locked for one hour, and an email is sent explaining the lockout and showing the exact time when he may try again. On successful email

OTP verification, a secure Flask session is created with a 10-minute inactivity timeout, after which he must log in again. he is redirected to the Dashboard, where he sees her profile and any existing capsules.
Step 3: Creating a New Capsule Ajith clicks Create Capsule.
The form contains:

- Capsule Name
- Optional Description (up to 500 characters)
- Future Unlock Date
- Future Unlock Time
- Upload File

He selects PropertyPapers.pdf, enters a description ("Legal property ownership documents"), and chooses an unlock date two months into the future.
The backend checks:

- Unlock date must be in the future
- If the unlock date is today → unlock time must be at least 5 minutes ahead
- File must be valid and successfully uploaded If all fields are valid, Ajith clicks Next.

Step 4: Encryption and Secure Storage (Internal Processing) The backend performs the following operations immediately:

1. AES-256 File Encryption A new 256-bit AES key is generated. The file is encrypted using AES-256. The encrypted file (ciphertext) is prepared for upload.
2. SHA-256 Hash (Integrity Digest) A SHA-256 hash of the encrypted file is computed. This digest is stored in PostgreSQL and also saved in S3 metadata.
3. RSA Key Wrapping The AES key is encrypted using the system's RSA-2048 public key. Only the backend server with the RSA private key can later recover it.
4. Uploading to AWS S3 The encrypted file is uploaded to the AWS S3 bucket (digital-time-capsule-preethi). Only encrypted data is stored in S3 no plaintext ever leaves the backend.
5. Storing Metadata PostgreSQL stores:
    - Capsule ID
    - User email
    - Unlock datetime
    - Encrypted AES key
    - S3 file location
    - Capsule Name
    - Description
    - Integrity hash

Step 5: QR Code Provisioning and TOTP Setup After encryption and storage, Ajith is shown a QR code. This QR is generated using an otpauth:// URI containing:

- Capsule ID
- Ajith's email

- Capsule Name
- Base32 TOTP secret

The label appears in Google Authenticator as:
Digital Time Capsule – [CapsuleID] — Ajith@example.com – PropertyPapers
This helps Ajith identify which OTP code belongs to which capsule. He scans the QR code using Google Authenticator. To confirm setup, He enters a test OTP on the page. If the OTP is correct → Capsule Created Successfully.
Step 6: Dashboard View — Countdown Timer Back on the dashboard, Ajith sees:

- Total number of capsules
- Capsule Name
- Unlock Date and Time
- Live countdown timer (days, hours, minutes remaining)

Importantly: Before the unlock time, the "View" button is NOT shown Only the countdown timer appears. This prevents premature access.
Step 7: Auto-Unlock Scheduler
$Every 5 seconds, APScheduler checks if the current time \geq unlock time.$
When the time arrives:

1. The capsule status changes from "locked" to "unlocked".
2. The countdown disappears.
3. The View button appears on the dashboard.
4. A professional HTML email is sent to Ajith: "Your Capsule Is Now Ready for Unlock".

Step 8: Unlocking the Capsule (TOTP Verification) Ajith clicks View. He is asked to enter the 6-digit OTP from Google Authenticator. The backend verifies:

- Capsule is unlocked
- OTP is correct and generated from the same TOTP secret
- OTP is within time window (30 seconds)
- Fewer than 3 failed attempts After successful verification:
    1. The RSA private key unwraps (decrypts) the AES-256 key.
    2. Encrypted file is downloaded from S3.
    3. SHA-256 integrity digest is recomputed and matched.
    4. File is decrypted using AES-256.
    5. PropertyPapers.pdf is provided for download.

If Ajith enters wrong OTP three times, the capsule is locked for 24 hours.
Step 9: Controlled Capsule Deletion After reviewing her document, Ajith decides to delete the capsule.
The steps are:

1. System sends a 6-digit email OTP.
2. Ajith enters the OTP in the deletion page.
3. A final confirmation message appears.

4. When confirmed, the backend:
    - Deletes the encrypted file from AWS S3
    - Removes metadata from PostgreSQL
    - Logs the deletion event
5. A confirmation email is sent.

Applicability to Other Domains Although this example uses a legal property document, the same workflow is suitable for:

- Digital forensic evidence preservation
- Legal contract storage
- Academic research data with scheduled release
- Personal messages or documents
- Corporate confidential records

The system ensures confidentiality, time-lock enforcement, OTP security, and tamper-evident behaviour across all domains.

9. Limitations and Future Work

Although the Digital Time Capsule prototype demonstrates the feasibility of time-controlled encrypted file storage, it also has several limitations that must be acknowledged. First, the system has not yet been tested under large-scale conditions. The current scheduler, database structure, and S3 interactions were evaluated only in a development environment with a limited number of capsules. A production deployment would require stress testing to measure S3 latency under heavy load, database write/read patterns at scale, and unlock-time performance when many capsules reach their release time simultaneously. The formal usability evaluation is one of the most important aspects that have not been addressed. Although the interface has the goals of being simple, conducting usability sessions with the users will give valuable insight into the usability of the interface with regard to areas of improvement. Its security model is based on trust in a backend server and a well-set-up cloud infrastructure, but this can introduce a range of issues associated with trust and infrastructure integrity.

Future work could benefit from increased isolation for user-held keys, regular analysis of infrastructure configuration, and mechanisms for detecting unusual events. There could also be benefit from other approaches to timed release, for example, through a network of time servers or blockchain nodes, to improve integrity and reduce a system's dependence on a single timing component. Lastly, comparison testing with current systems could give a more complete characterization of a system's performance. Future enhancements may include improved key management, optional multi-device authentication, support for staged or partial release of stored data, and adapting the platform for judicial, archival, or enterprise settings where compliance requirements apply.

9.1. Practical Deployment Considerations

Because the system depends heavily on AWS S3 for long-term file storage, a real deployment must also consider storage and data-transfer costs. S3 charges separately for stored data, monthly retention, and retrieval operations. In addition, cross-region transfers or large download operations may increase cost during peak activity. Although these costs are minimal for small personal capsules, they may become significant for organizational or long-term archival use. A detailed economic analysis is therefore necessary before production deployment and is identified as part of future work.

9.2. Long-Term Storage Cost, Retention Strategy, and Cloud Lifecycle Management

Because the Digital Time Capsule may store encrypted files for many months or years, long-term retention cost becomes an operational factor.
AWS S3 charges independently for
(a) storage volume,
(b) monthly retention,
(c) retrieval, and
(d) cross-region bandwidth.
While these costs remain moderate for small personal capsules, they can accumulate when storing large files or many capsules over extended periods.
Retention strategy. A lifecycle policy can automatically transition older capsules from S3 Standard to lower-cost archival tiers such as S3 Standard-IA, S3 Glacier, or S3 Glacier Deep Archive, depending on the expected unlock date and retrieval frequency. Since capsules remain encrypted regardless of storage tier, confidentiality is preserved during long-term archival storage.
Cost-aware unlock planning. For capsules intended for far-future unlock dates (e.g., 1–10 years), transitioning objects to a cold-storage tier can reduce operational cost significantly.
Optimizing Storage and Cost Efficiency: The system can be instructed to automatically restore objects to Standard before the unlock window in an effort to reduce retrieval delays by using a scheduler or background worker. For such a scheme to work, there should be operational controls to avoid uncontrolled growth of storage. Mechanisms include IAM policies and automation that monitor the enforcement of limits on storage, alert when actual costs exceed budgeted thresholds, and run regular cleanup of orphaned objects. Regular verification ensures S3 lifecycle rules remain in operation.
Long-term Archiving Applicability: The design of the system will make it appropriate for an institutional or archival setting where predictable long-term costs are important. The system provides long-term data preservation by taking advantage of encrypted storage, Glacier life cycle transitions, and scheduled unlocks that manage cloud storage overhead. These show the feasibility of the system for real-world deployment and scalability in cloud environments.

10. Conclusion

The Digital Time Capsule is a reliable and user-friendly platform that allows users to save their important data for future use, following a predetermined unlock time. The system assures the confidentiality and integrity of data by using AES-256 encryption for files and RSA encryption for keys. It further uses multi- layered authentication methods, such as OTP verification through both email and Google Authenticator, to protect registration, login, creation of a capsule, unlocking, and deletion, which are the most sensitive operations.

This system is different in a way that it uses the AWS S3 cloud storage which is scalable and dependable for storage instead of traditional methods. The interface is designed in such a way that it is easy for technical as well as for non-technical users to converse with the platform without any difficulties. The multifaceted system can easily show its capabilities to be used in the three major sectors of personal, professional, and forensic by the preservation of the most vulnerable types of content: e.g., the deceased' last will, business documents, and digital evidence until the set unlocking time. The time-lock feature makes sure that the files stored are not accessible before the exact time that has been scheduled. In total, the Digital Time Capsule is a modern and user-friendly notion of protecting digital data by means of encryption technology, OTP security, time-locked access, and cloud infrastructure. The concept of the Digital Time Capsule is commendable but its system still has room for improvement in terms of usability, traceability, and adaptability for its future version which can be directed in numerous possible ways.

Acknowledgments

The authors would like to thank the referees for their valuable comments and suggestions, which helped improve the quality and clarity of this paper.

References

1. Rivest, Ronald L., Adi Shamir, and David A. Wagner. "Time-Lock Puzzles and Timed-Release Crypto." MIT Laboratory for Computer Science, 1996.
2. Cheon, Jung Hee, et al. "Provably Secure Timed-Release Public-Key Encryption." Proceedings of Information Security and Cryptology, 2008.
3. Nelson, Michael L., and Rabia Haq. "Using Timed-Release Cryptography to Mitigate Embargo Risk." Digital Preservation Conference Proceedings, 2009.
4. Surni´c, N., and Andreas Rauber. "Digital Preservation Time Capsule: A Showcase for Digital Preservation." Proceedings of the International Conference on Digital Preservation (iPRES), 2009.
5. Hsu, Cheng-Kang, et al. "Time Capsule Signature." IEEE Transactions on Information Forensics and Security, 2012.
6. Watanabe, Kazuhiro, and Junji Shikata. "Timed-Release Secret Sharing." IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 2014.
7. "Security Rag-Based Scheme: Adding Integrity and Timed Ephemerizer to Cloud Storage." Journal of Cloud Computing Security, 2014.
8. Takemata, Kenji, et al. "Design and Trial Use of an E-Time Capsule System." IEEE International Conference on Education Technology, 2017.
9. Burkhalter, Lucas, et al. "TimeCrypt: Encrypted Data Stream Processing at Scale with Cryptographic Access Control." USENIX Security Symposium, 2018.
10. Dang, Hung, and Ee-Chien Chang. "Self-Expiring Data Capsule Using Trusted Execution Environment (TEEKAP)." IEEE Transactions on Dependable and Secure Computing, 2019.
11. Yue, Zhi, Yao Yao, Wei Li, and Ning Yu. "ATDD: Fine-Grained Assured Time-Sensitive Data Deletion Scheme in Cloud Storage." IEEE Transactions on Cloud Computing, 2020.
12. Yang, Xiaolong, and Yuan Liu. "Toward Timed-Release Encryption in Web3 via Proof-of-Work Consensus." IEEE Access, 2022.
13. Soltani, Reza, et al. "Data Capsule: A Self-Contained Data Model as an Access Policy Enforcement Strategy." Proceedings of the IEEE International Conference on Cloud Engineering, 2022.
14. "i-TiRE: Incremental Timed-Release Encryption." Proceedings of the ACM Conference on Com- puter and Communications Security (CCS), 2022.
15. Yuan, Hao, et al. "Multiple Time Servers: Timed-Release Encryption Based on Shamir Secret Sharing." IEEE Transactions on Information Forensics and Security, 2024.

Shivakumara T.,
Department of Master of Computer Applications,
BMS Institute of Technology and Management,
Karnataka, India.
E-mail address: shivakumarat@bmsit.in

and

Muneshwara M. S.,
Department of Computer Science and Engineering,
BMS Institute of Technology and Management,
Karnataka, India.
E-mail address: muneshwarams@bmsit.in

and

P. Sudarsanam
Department of Master of Computer Applications,
BMS Institute of Technology and Management,
Karnataka, India.
E-mail address: sudarsanamp@bmsit.in

and

Nagamani H. S.
Department of Computer Science,
Smt VHD Central Institute of Home Science,(MCU)
Karnataka, India.
E-mail address: nagamani.hs@gmail.com

and

Preethi Hatui
Department of Master of Computer Applications,
BMS Institute of Technology and Management
Karnataka, India.
E-mail address: preethihatui02@gmail.com

and

Anand R.,
Department of Computer Science and Engineering,
BMS Institute of Technology and Management,
Karnataka, India.
E-mail address: anandor@bmsit.in