Loom: Weaving PCS-Preserving and Searchable Cloud Backups for Secure Messaging

TIANCHENG ZHU, Nanjing University of Science and Technology, China
JIABEI WANG*, Nanjing University of Science and Technology, China
YONGBIN ZHOU*, Nanjing University of Science and Technology, China

Post-compromise security (PCS) ensures that exposure of the current secret state does not permanently compromise future security. Modern secure messaging protocols such as IETF Messaging Layer Security provide mature realizations of PCS through continuous key updates, ensuring that compromise of the current secret state does not endanger future communications. However, extending this protection to cloud backups often breaks PCS if data are encrypted under static key. While Dodis et al.'s Compact Key Storage (CKS) preserves PCS for encrypted backups, it supports only all-or-nothing retrieval, rendering keyword-based selective retrieval impractical.

We present Loom, the first searchable encrypted cloud backup system that preserves PCS while supporting efficient multi-keyword conjunctive search. Loom embeds data keys into the secure searchable index, eliminating the need for separate key storage and ensuring that "*searchable-implies-decryptable*". For group scenarios, we introduce LoomRelay, which avoids redundant uploads and ensures consistent access control under dynamic group membership. Implemented on *mlspp*, an open-source IETF MLS reference, Loom and LoomRelay outperform prior schemes. Single-message backup completes in 3-15 ms across variable cipher suites, while searching 10^4 encrypted messages takes 9.06-28.97 s (RTT 10-100 ms, bandwidth 10-50 Mbps). Compared to CKS, which requires downloading the entire backup, Loom achieves 25×-87× faster keyword-based retrieval, with substantially lower communication and storage overhead.

CCS Concepts: • **Security and privacy → Management and querying of encrypted data**.

Additional Key Words and Phrases: Dynamic Searchable Symmetric Encryption; Secure Messaging Protocol; Post-Compromise Security

ACM Reference Format:
Tiancheng Zhu, Jiabei Wang, and Yongbin Zhou. 2026. Loom: Weaving PCS-Preserving and Searchable Cloud Backups for Secure Messaging. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 205 (June 2026), 26 pages. https://doi.org/10.1145/3802082

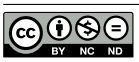
1 Introduction

1.1 Motivation

With the widespread adoption of cloud computing, users increasingly rely on third-party cloud services to store sensitive data. To mitigate the risk of server-side data breaches, a common practice is to encrypt data on the client side before uploading it to the cloud. However, protecting only the cloud-stored data is insufficient: if a user's device is compromised, locally stored encryption

*corresponding author

Authors' Contact Information: Tiancheng Zhu, ztc_cscode@njust.edu.cn, School of Cyber Science and Engineering, Nanjing University of Science and Technology, Nanjing, China; Jiabei Wang, wangjiabei@njust.edu.cn, School of Cyber Science and Engineering, Nanjing University of Science and Technology, Nanjing, China; Yongbin Zhou, zhouyongbin@njust.edu.cn, School of Cyber Science and Engineering, Nanjing University of Science and Technology, Nanjing, China.



ACM 2836-6573/2026/6-ART205
https://doi.org/10.1145/3802082

keys may be exfiltrated, thereby compromising the confidentiality of all future encrypted data. This threat motivates the notion of post-compromise security (PCS), which aims to ensure that exposure of the current secret state does not permanently compromise future security. Modern end-to-end secure messaging (SM) protocols [2, 26, 28], such as the IETF Messaging Layer Security (MLS) [4, 36, 37] and Signal [3], provide the most mature realizations of PCS through continuous key update (as advocated in Data Security Standard of Payment Card Industry [19] and NIST Special Publication 800-57 [5]): even if the current secret key is exposed, an adversary cannot decrypt future communications. This is accomplished by updating the session key after each message and securely discarding prior keys.

Directly applying this paradigm to data management systems, however, introduces a fundamental tension. In messaging, keys are used transiently for real-time encryption and need not be retained long-term. In contrast, data management systems must preserve decryption capability indefinitely to support future retrieval of historical messages. Storing a distinct key for every message leads to unbounded client-side key storage overhead, while discarding old keys renders historical data permanently unrecoverable.

Compact Key Storage (CKS) [24] addresses this problem by leveraging the deriving secrets in SM protocols to store only a compressed local state, thereby preserving PCS for backups without retaining all historical keys. While CKS preserves PCS in cloud backups, it is inherently *all-or-nothing*: acquiring even a small portion of records for some special keywords requires downloading the entire database.

Recall that replacing a device due to loss or theft, which would require a full download of all backup data, is rare. In urgent situations, immediate access to critical information is essential. In such scenarios, downloading and decrypting the entire backup, or recalling the timing of specific records, is not ideal. A more efficient approach is to retrieve and download only the data corresponding to the keyword, minimizing both cost and overhead. In enterprise environments, compliance audits demand fast keyword-based search over logs without exposing unnecessary data. Additional considerations include saving network bandwidth in low-connectivity conditions, enabling efficient cross-device synchronization, supporting incremental backup updates, and allowing multi-user collaboration with privacy-preserving selective access.

Achieving selective access under PCS is non-trivial. In an encrypted data storage system, new data are continuously generated and must be archived to the cloud. A natural starting point for enabling selective retrieval in this dynamic setting is Dynamic Searchable Symmetric Encryption (DSSE) [10, 13, 14], which supports efficient encrypted search over continuously updated data. However, integrating DSSE into PCS-enabled secure *storage systems* introduces a series of challenges that existing designs fail to address.

(C1) *Preserving PCS beyond the index layer.* In a PCS-enabled backup system, it is undesirable for newly added indexes to weaken the PCS of the entire system. While recent DSSE schemes, such as Bamboo [16], achieve PCS for the index layer, they remain largely *structure-only*, leaving message ciphertexts protected by static or long-lived keys. As a result, once the key is compromised, data confidentiality is immediately lost, even if the index remains secure.

One might attempt to extend PCS to encrypted data by periodically updating encryption keys. However, encrypting each message under an independent key requires retaining all historical keys for future decryption and search, which directly contradicts the compact key design principle of PCS-enabled systems. Conversely, discarding old keys breaks access to previously backed-up data, rendering selective retrieval infeasible.

Moreover, keys of the encrypted database and secure index may evolve independently. If updates are not aligned, such desynchronization can create correlations between ciphertexts and index tokens, leading to unintended leakage (as detailed in Sec.4.1).

(C2) *Expressive query support.* Even with strong security guarantees, practical deployments require efficient query processing and state coordination. For conjunctive keyword search, most existing OXT-like structures [11, 32] are difficult to adapt to dynamic key-update settings while preserving matchable cross-tags.

(C3) *Supporting dynamic group scenario.* In dynamic group scenario, such as group messaging, how to share constantly changing states among multiple users to achieve consistent key synchronization and index updates between members becomes an important challenge, one that directly affects the system's scalability. At the same time, the system must also ensure that, once a member is removed, their ability to search and decrypt historical data is promptly and thoroughly revoked to maintain access control boundaries.

This gap motivates the following main question:

How can we construct a practical and secure multi-keyword searchable cloud backup and query system that robustly guarantees the confidentiality and integrity of both data and search indexes, even under key compromise?

1.2 Our Contributions

To address the challenges outlined above, we present Loom, the first PCS-preserving, multi-keyword searchable encrypted backup architecture. The name Loom captures our core design philosophy of interweaving the updating key, encrypted messages, and searchable indexes into a coherent and efficient fabric of secure cloud backup. We formalize the system and security model, in which an adversary may be an honest-but-curious server or a key thief capable of compromising secret keys or internal states. We further define the notion of a Loom-compatible application and prove that composing any such application with Loom preserves security under our security model. Our main contributions are summarized as follows:

PCS-preserving searchable backup with synchronized key update. We identify a new post-compromise threat that arises when key updates of ciphertexts and indexes are desynchronized but inherently intertwined, potentially exposing sensitive information. To address this, we introduce an *index key chaining* mechanism that ensures message keys and index tokens evolve in lockstep, enabling Loom to achieve *forward privacy* and *backward privacy*, as required by standard dynamic SSE, while preserving *system-wide PCS*.

Compact message keys into secure indices. We design the system to upload ciphertexts generated in the SM protocol as backup ciphertexts, while compacting message keys that should have been additionally stored into secure indices such that once "*messages become searchable, they are decryptable*". This design leverages the concept of convergent encryption, effectively prevents the loss of message PCS caused by re-encryption and eliminates the need to locally store additional keys, thereby resolving the inherent trade-off between security and storage efficiency.

Prototype implementation with significant performance improvements. We implement our system Loom on top of *mlspp*, an open-source MLS reference developed by Cisco and regularly participating in IETF MLS interoperability testing [27], and evaluate it against CKS and Bamboo. Loom achieves an average backup efficiency of 5-15 ms per message and a search efficiency of 9.06-28.97 s for 10^4 messages, evaluated under simulated network environment with RTT of 5-100 ms and bandwidths of 10 Mbps and 50 Mbps. Compared to CKS, which requires downloading followed by local search, our system achieves 25×-87× faster access to messages matching specific keywords. Moreover, it matches Bamboo in security while supporting sublinear multi-keyword search and significantly reducing key storage overhead.

Compatibility Beyond Messaging. Although our exposition and evaluation instantiate Loom in the context of secure messaging, a demanding workload that stresses frequent key updates and dynamic group membership, its core design directly targets system-level requirements common across encrypted data management workloads, including append-heavy updates, evolving access rights, and selective retrieval without full data scans. As a result, Loom naturally extends to database-oriented applications beyond messaging, such as encrypted audit logs, collaborative document archives, and IoT telemetry stores. As a concrete example, consider an encrypted audit log system where each log entry is labeled with multiple descriptive keywords. Using Loom, an auditor can issue conjunctive queries (e.g., `admin` $\wedge$ `delete-operation` $\wedge$ `after-hours`) to retrieve only matching records without exposing unrelated entries. Meanwhile, PCS ensures that a compromised search key cannot be used to query or decrypt log entries generated and uploaded afterward, preserving long-term confidentiality.

1.3 Related Work

Secure Messaging Protocol. Secure messaging protocols provide end-to-end confidentiality and integrity for instant communication, even in the presence of compromised servers. To mitigate the risk that long-term key compromise endangers message security, Off-the-Record (OTR) [7] introduced short-term keys to achieve forward secrecy, but it was limited to synchronous two-party communication. The double ratchet [34] paradigm was later proposed to support asynchronous message transmission in end-to-end encryption, providing formal guarantees of forward and post-compromise security, and was subsequently adopted in the Signal protocol [3]. Recently, the IETF standardized the MLS protocol [4, 36, 37], extending end-to-end encryption to group messaging.

Beyond secure messaging, research attention has shifted toward secure message storage. Dodis et al. [23, 24] observed that conventional backup systems can compromise the post-compromise security guarantees of secure messaging protocols, and proposed Compact Key Storage (CKS) for key storage efficient backups.

Dynamic Searchable Symmetric Encryption. The database community has long investigated how to support efficient query processing over encrypted data while mitigating information leakage, leading to a rich body of work on encrypted data management. Representative efforts span flexible privacy–performance trade-offs for key–value queries [40], oblivious and enclave-assisted index structures for spatial-textual search [20, 38], compression-aware searchable encryption [22], and ORAM-based techniques for complex relational operators such as joins [15]. While these systems significantly advance the efficiency and expressiveness of encrypted queries, they typically assume static or long-lived secret keys and do not explicitly model key compromise as part of the threat model. As a result, providing PCS for encrypted indexing and query processing remains largely unexplored in database-oriented settings.

Searchable encryption (SE) schemes aim to enable efficient search while preserving the privacy of both data (documents and keywords) and user queries. Since the dynamic SSE (DSSE) schemes proposed by Kamara et al. [29], research has progressed in two main directions: enhancing functionality and strengthening security.

Following the basic DSSE frameworks, Cash et al. [11] proposed the OXT index, enabling conjunctive keyword queries by locating the document set for a primary keyword and intersecting it with those of other keywords. As a representative construction, OXT achieves sublinear search efficiency while maintaining query expressiveness. Patranabis et al. [33] extended this design to dynamic settings with the ODXT structure, supporting data update. From the security perspective, Zhang et al. [41] revealed that DSSE schemes are susceptible to file-injection attacks, where adversaries insert crafted files containing specific keywords and observe search patterns to infer

sensitive information. To mitigate such leakages, Bost et al. [8, 9] introduced forward and backward privacy, which have since become standard security properties for DSSE.

Moving beyond query-leakage resistance, Chen et al. [16] identified that conventional DSSE schemes lose confidentiality once the index key is compromised, posing a severe threat to database security, and proposed Bamboo to achieve PCS. However, Bamboo fails to address the critical interference between document encryption and encrypted indexing.

2 Preliminaries

Privacy of DSSE. Following the formal definitions by Bost et al. [8, 9], a DSSE scheme $\mathsf{DSSE} = (\mathsf{Setup}, \mathsf{Update}, \mathsf{Search})$ with leakage function $\mathcal{L} = (\mathcal{L}^{\mathsf{Stp}}, \mathcal{L}^{\mathsf{Upd}}, \mathcal{L}^{\mathsf{Srch}})$ is adaptively *forward private* if there exists a stateless function $\mathcal{L}'$ such that, for any operation, document identifier, and keyword tuple $(\mathsf{op} \in \mathit{add}, \mathit{del}, \mathsf{id}, w)$, we have $\mathcal{L}^{\mathsf{Upd}}(\mathsf{op}, (\mathsf{id}, w)) = \mathcal{L}'(\mathsf{op}, \mathsf{id})$, ensuring that updates do not leak information about previously added keywords.

DSSE is adaptively Type-II *backward private* if there exists stateless functions $\mathcal{L}''$ and $\mathcal{L}'''$ such that, for any $(\mathsf{op}, \mathsf{id}, w)$, we have $\mathcal{L}^{\mathsf{Upd}}(\mathsf{op}, (\mathsf{id}, w)) = \mathcal{L}''(\mathsf{op}, \mathsf{id})$ and $\mathcal{L}^{\mathsf{Srch}}(w) = \mathcal{L}'''(\mathsf{TimeDB}(w), \mathsf{Upd}(w))$, where $\mathsf{TimeDB}(w)$ returns the list of all file identifiers containing w that have not yet been deleted, along with their respective timestamps of insertion, and $\mathsf{Upd}(w)$ returns the timestamps of all update operations on w.

Invertible Pseudorandom Function. An invertible pseudorandom function (IPF) [6] is an efficiently-computable injective function $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ equipped with an efficient inversion algorithm $\mathsf{F}^{-1} : \mathcal{K} \times \mathcal{Y} \to \mathcal{X} \cup \{\perp\}$. The inversion algorithm is required to satisfy the following two properties for all $k \in \mathcal{K}$: (i) $\mathsf{F}^{-1}(k, \mathsf{F}(k, x)) = x$ for all $x \in \mathcal{X}$. and (ii) $\mathsf{F}^{-1}(k, y) = \perp$ if y is not in $\mathsf{F}(k, x) \mid x \in \mathcal{X}$.

Decisional Diffie-Hellman Assumption. Suppose $\mathbb{G}$ is a cyclic multiplicative group of prime order p with a generator g. The decisional Diffie-Hellman (DDH) assumption is that for all probabilistic polynomial-time (PPT) algorithms $\mathcal{A}$, $|\Pr[\mathcal{A}(g, g^\alpha, g^\beta, g^{\alpha\cdot\beta}) = 1] - \Pr[\mathcal{A}(g, g^\alpha, g^\beta, g^\gamma) = 1]| \leqslant \mathsf{negl}(\lambda)$, where $\alpha, \beta, \gamma \xleftarrow{\$} \mathbb{Z}_p^*$.

3 System Overview

3.1 System Architecture

As shown in Figure 1, Loom is designed as an encrypted backup and secure query system. It adopts a three-tier architecture, where the client-side components (APP and Client) operate within a trusted domain, while the Backup Server is assumed to be untrusted. The client simultaneously persists a compact cryptographic state to enable search and rollback. The server maintains two encrypted data structures: EID (Encrypted Index), which enables secure keyword search, and EDB (Encrypted Database), which stores encrypted data objects. This abstraction is agnostic to the underlying storage technology—it can be implemented by a key-value store, a relational table with a primary key, an object store, or even a flat file with an external index. All interactions between the client and the server are mediated through well-defined APIs and protocols, enabling efficient and selective access to encrypted data under post-compromise security protection.

3.2 Loom API and Workflow

Loom exposes a compact API for encrypted backup, keyword-based search, result retrieval, and state rollback, abstracting the core underlying cryptographic operations.

Definition 1. *An encrypted backup and secure query system* Loom *is composed of algorithm* Setup *and protocol* DataUpdate, Search, Request *and* Rollback *defined as:*

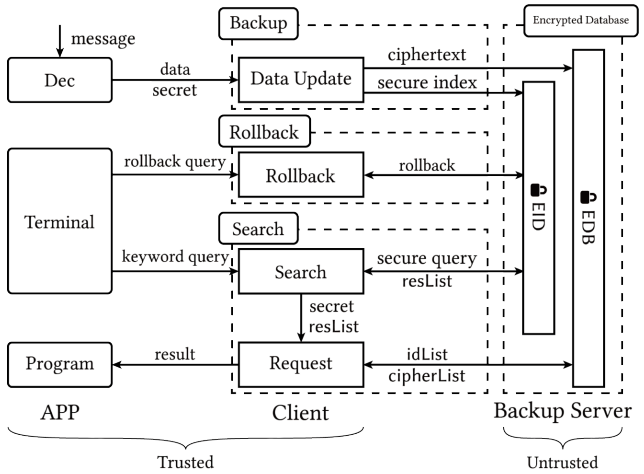


Fig. 1. The System Architecture of Loom

- $\mathsf{Setup}(\lambda)$*: A probabilistic algorithm that takes as input a security parameter* λ *and outputs a public parameter* pp*, a secret key* k_t*, a state* State_c *for the client, and an encrypted database* EDB *together with a secure searchable index* EID *for the server.*
- $\mathsf{DataUpdate}\langle C(\mathsf{State}_c, k_t, \tilde{c_t}, s_t, s_{t+1}, (\mathsf{W}_t, id, op)); S(\mathsf{EDB}, \mathsf{EID})\rangle$*: An interactive protocol between the client and the server. The client's inputs are its state* State_c*, secret key* k_t*, an SM ciphertext* $\tilde{c_t}$*, SM secret (key)* s_t*, the secret for next epoch* s_{t+1}*, the keyword set* W_t *extracted from its plaintext, its identifier* id *and an operator* $op \in \{add, del\}$*. The server's inputs are the encrypted database* EDB *and the secure searchable index* EID*. Finally, the client obtains the updated state* State_c'*, while the server obtains the updated database* EDB′ *and the updated index* EID′*.*
- $\mathsf{Search}\langle C(\mathsf{State}_c, \mathsf{W}_s); S(\mathsf{EID})\rangle$*: An interactive protocol between the client and the server. The client's inputs are its state* State_c *and the keyword set* W_s *intended for conjunctive search, while the server's input is the secure index* EID*. Finally, the client obtains the result set* resList*, while the server obtains nothing.*
- $\mathsf{Request}\langle C(\mathsf{State}_c, \mathsf{resList}, \mathsf{W}_s); S(\mathsf{EDB})\rangle$*: An interactive protocol between the client and the server. The client's inputs are its state* State_c*, the result set* resList *and the corresponding keyword set* W_s*, while the server's input is the encrypted database* EDB*. Finally, the client obtains the original message, while the server obtains nothing.*
- $\mathsf{Rollback}\langle C(\mathsf{State}_c, \mathsf{W}_{rb}); S(\mathsf{EID})\rangle$*: An interactive protocol between the client and the server. The client's inputs are its state* State_c *and the keyword set* W_{rb} *to be rolled back, the server's input is the secure index* EID*. Finally, the client obtains the rolled back client state* State_c'*, while the server obtains nothing.*

Correctness. Loom is *correct* if, for any $\lambda \in \mathbb{N}$, $(pp, k_t, \mathsf{State}_c; pp, \mathsf{EDB}, \mathsf{EID}) \leftarrow \mathsf{Setup}(\lambda)$, and any $poly(\lambda)$ sequence of DataUpdate, Search, Request, and Rollback operations satisfies: executing $\mathsf{Search}\ \langle C(\mathsf{State}_c, \mathsf{W}_s); S(\mathsf{EID})\rangle$ followed by $\mathsf{Request}\langle C(\mathsf{State}_c, \mathsf{resList}, \mathsf{W}_s); S(\mathsf{EDB})\rangle$ with the same W_s always returns exactly those messages whose ciphertexts, keys, and keywords were (i) previously added via $\mathsf{DataUpdate}\langle \cdot, add\rangle$, (ii) not subsequently removed via $\mathsf{DataUpdate}\langle \cdot, del\rangle$, and (iii) not rolled back via $\mathsf{Rollback}\langle \cdot \rangle$.

Workflow. Figure 1 also illustrates the workflow of Loom. Upon data generation at the application layer, the client encrypts the data and invokes protocol DataUpdate to store ciphertexts in EDB and

update the encrypted index EID. For data search, the client translates user-issued keyword queries into a secure Search request evaluated over EID, upon which the server returns an encrypted result list of matching identifiers. The client then invokes Request to fetch the corresponding ciphertexts from EDB and locally decrypts them before returning the results to the application. In case of compromise or recovery, the Rollback protocol allows the client to revert to a previous consistent state while preserving synchronization between EID and EDB.

4 Security Properties of Loom

4.1 Post-Compromise Security

In our design, the PCS of a DSSE system built upon a key-updatable strategy, consistent with the NIST Special Publication 800-57 [5], is divided into two categories: one concerns the message layer ($\mathrm{PCS}_{\mathrm{MSG}}$) and the other the secure index layer ($\mathrm{PCS}_{\mathrm{IDX}}$). We argue that assuming only a single key is compromised after a breach is unrealistic. In practice, user coercion or device compromise is more likely to expose all related keys. Because messages and keywords are inherently correlated, separately updating message and index keys may achieve post-compromise security for each component but remains insufficient for an overall post-compromise secure DSSE system.

When message and index key updates are not synchronized, two inevitable exposure windows arise (Figure 2). The first is the period between a message key update and the subsequent index key update, termed the *message vulnerable period*. The second is the period between an index key update and the subsequent message key update, termed the *query vulnerable period*. If the adversary compromises keys before the message vulnerable slot, they can still use the stale index key to guess keywords, identify those involved in updates, and exploit keyword-message correlations to infer message contents. Conversely, if the adversary learns plaintexts, they can easily deduce the associated keywords, compromising the privacy of subsequent user queries. Thus, a secure backup system should synchronize message and index key updates to eliminate these exposure windows and ensure full post-compromise security.

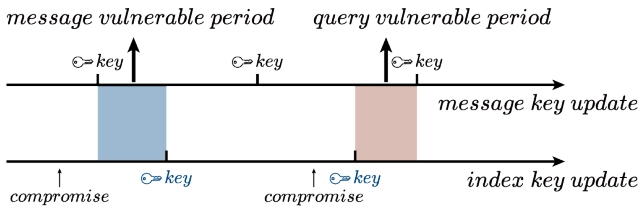


Fig. 2. Vulnerabilities from Unsynchronized Key Updates

4.2 Formal Security Definition

Our security model is defined from two aspects: characterizing the adversary to define the threat model, and specifying the security goals that formalize the requirements the scheme must satisfy.

Threat model. We consider two main types of adversary: the *server* and the *key thief*, assuming they may collude, which means the information they obtain can be shared. Thus, under the condition of equivalent capabilities, we define their potential threats:

- Adversarial Server $\mathcal{A}_{srv}$: The server provides index and ciphertext storage services and can perform secure keyword searches. We assume the server follows the protocol honestly except for potentially forging key ciphertexts.
  - *Indices and query privacy within dynamic data and key compromised setting:*

(i) *Forward and backward privacy of* Loom: Server attempts to learn the underlying keywords from search indices and queries, potentially exploiting access / search patterns leakage.
(ii) *Post-compromise security of* Loom ($\mathsf{PCS}_{\mathsf{IDX}}$): Even after colluding with the key thief to obtain current keys, it remains interested in the indices and queries after compromising, intending to break the post-compromise security of Loom.

- *Integrity:* Since Loom encrypts and uploads secrets for backup, which lack semantic meaning, the server may attempt to forge fake key ciphertexts.

- Key thief $\mathcal{A}_{thf}$: This adversary can compromise the client's keys and eavesdrop on communications between user and server.
  - Post-compromise security of SM protocol ($\mathsf{PCS}_{\mathsf{MSG}}$): After compromising the current key, the key thief attempts to exploit the leakage from Loom to break the post-compromise security of the underlying SM protocol.

Notably, whether the user stores all SM secrets or re-encrypts plaintext with a fresh key before backup, it is unrealistic to assume that only the secrets of a single epoch are leaked upon key compromise. Therefore, we do not consider forward secrecy, i.e., the leakage of a specific epoch's key not affecting the confidentiality of prior messages, as we assume prior secrets are also compromised, which better reflects practical threat scenarios.

Security against $\mathcal{A}_{srv}$. PCS in our scheme differs from that in Bamboo in that our scheme does not involve the *special time slot* described in Bamboo. This difference arises because the keys in Bamboo are updated periodically or at fixed intervals, independent of document updates, resulting in a *special time slot* between a key compromise and the subsequent key update. The PCS described in Bamboo is specifically designed to protect update privacy during this interval and to maintain system security after key leakage.

In Loom, data updates and key updates occur simultaneously, thus eliminating aforementioned *special time slot*. Additionally, in our design, the user's current keys k_t and k_d are independent of subsequent communication keys $s_{t+1}, \ldots$. Even if the adversary compromises the user's current keys, they cannot derive future communication keys or newly generated key materials unrelated to these compromised keys.

We require that there exists a simulator $\mathcal{S}$ to simulate an ideal game. In the ideal game, $\mathcal{A}_{srv}$ can adaptively issue queries of Setup, DataUpdate, Search Request, Rollback. $\mathcal{S}$ can simulate the corresponding response to $\mathcal{A}_{srv}$ with leakage functions $\mathcal{L}_{srv}^{\mathsf{Stp}}$, $\mathcal{L}_{srv}^{\mathsf{Upd}}$, $\mathcal{L}_{srv}^{\mathsf{Srch}}$, $\mathcal{L}_{srv}^{\mathsf{Req}}$ and $\mathcal{L}_{srv}^{\mathsf{Rbk}}$. Additionally, we specify that the adversary can query Compromise to simulate key leakage. When the adversary queries Compromise, $\mathcal{S}$ can simulate the corresponding response to $\mathcal{A}_{srv}$ with the leakage function $\mathcal{L}_{srv}^{\mathsf{Comp}}$.

Definition 2 ($\mathcal{L}_{srv}$-Adaptive Security). *Given leakage functions* $\mathcal{L}_{srv}=(\mathcal{L}_{srv}^{\mathsf{Stp}},\mathcal{L}_{srv}^{\mathsf{Upd}},\mathcal{L}_{srv}^{\mathsf{Srch}},\mathcal{L}_{srv}^{\mathsf{Req}},\mathcal{L}_{srv}^{\mathsf{Rbk}},\mathcal{L}_{srv}^{\mathsf{Comp}})$, Loom *is secure if for any sufficiently large security parameter* $\lambda\in\mathbb{N}$ *and PPT adversary* $\mathcal{A}_{srv}$*, there exists a simulator* $\mathcal{S}=(\mathcal{S}.\mathsf{Setup}, \mathcal{S}.\mathsf{DataUpdate}, \mathcal{S}.\mathsf{Search}, \mathcal{S}.\mathsf{Request}, \mathcal{S}.\mathsf{Rollback}, \mathcal{S}.\mathsf{Compromise})$ *such that*

$$|\Pr[\mathsf{Real}^{\Sigma}_{\mathcal{A}_{srv}}(\lambda)=1]-\Pr[\mathsf{Ideal}^{\Sigma}_{\mathcal{A}_{srv},\mathcal{S},\mathcal{L}_{srv}}(\lambda)=1]|\leqslant negl(\lambda)$$

where $\mathsf{Real}^{\Sigma}_{\mathcal{A}_{srv}}(\lambda)$ *and* $\mathsf{Ideal}^{\Sigma}_{\mathcal{A}_{srv},\mathcal{S},\mathcal{L}_{srv}}(\lambda)$ *are defined as:*

- $\mathsf{Real}^{\Sigma}_{\mathcal{A}_{srv}}(\lambda)$*: In this real game, after initializing* Loom *via* Setup*, the adversary* $\mathcal{A}_{srv}$ *adaptively queries* DataUpdate, Search, Request *and* Rollback *oracles and is allowed one* Compromise *query, finally outputs a bit.*

Game 1: The detailed description of $\mathsf{G}_{\mathsf{Int}}$

main:
  $b \leftarrow 0$;
  $\mathsf{realSecret}[\cdot], \mathsf{Key}[\cdot], \mathsf{St}[\cdot,\cdot] \leftarrow \perp$;
  $\mathcal{A}_{thf}^{O_{\mathsf{stp}}, O_{\mathsf{upd}}, \ldots, O_{\mathsf{rbk}}}(\lambda)$;
  return b
oracle O_{stp}**:**
  $(pp, \mathsf{Key}[0], \mathsf{EDB}, \mathsf{EID}) \leftarrow \mathsf{Setup}(\lambda)$;
oracle $O_{\mathsf{upd}}(s_t, \mathsf{W_t})$**:**
  $\langle \mathsf{St}[w \in \mathsf{W_t}, d], \mathsf{Key}[t]; \mathsf{EID}', \mathsf{EDB}' \rangle \leftarrow \langle \mathsf{DataUpdate}_C(\mathsf{St}, \mathsf{Key}, s_t, \mathsf{W_t}); \mathcal{A}_{srv} \rangle$ where $d = \mathsf{St}[w].size$;
  $\mathsf{realSecret}[t] \leftarrow s_t$;
oracle $O_{\mathsf{src}}(\mathsf{W_s})$**:**
  $\langle \mathsf{resList}; \perp \rangle \leftarrow \langle \mathsf{Search}_C(\mathsf{St}, \mathsf{Key}, \mathsf{W_s}); \mathcal{A}_{srv} \rangle$;
oracle $O_{\mathsf{req}}(\mathsf{resList})$**:**
  $\langle \mathsf{secList}, \perp \rangle \leftarrow \langle \mathsf{Request}_C(\mathsf{St}, \mathsf{resList}, \mathsf{W}_s); \mathcal{A}_{srv} \rangle$;
  if $\mathsf{secList} \neq \mathsf{realSecret}$ **then**
    $b \leftarrow 1$
  end
oracle $O_{\mathsf{rbk}}(\mathsf{W_{rb}}, t)$**:**
  $(\mathsf{St}[w \in \mathsf{W_{rb}}, \cdot]; \perp) \leftarrow \langle \mathsf{Rollback}_C(\mathsf{St}, \mathsf{W}_{rb}); \mathcal{A}_{thf} \rangle$;
  $\mathsf{realSecret}[t] \leftarrow \perp$

- $\mathsf{Ideal}^{\Sigma}_{\mathcal{A}_{srv}, \mathcal{S}, \mathcal{L}_{srv}}(\lambda)$*: In the ideal game, simulator* $\mathcal{S}$ *uses the leakage functions* $\mathcal{L}_{srv}$ *to simulate responses to* $\mathcal{A}_{srv}$*'s queries identically to the real game. Upon receiving these queries,* $\mathcal{S}$ *takes* $\mathcal{L}_{srv}$ *as input and executes algorithms* $\mathcal{S}$.Setup, $\mathcal{S}$.DataUpdate, $\mathcal{S}$.Search, $\mathcal{S}$.Request, $\mathcal{S}$.Rollback *and* $\mathcal{S}$.Compromise *to generate simulated oracle outputs and returns them to* $\mathcal{A}_{srv}$*. Finally, the adversary outputs a bit.*

Although SM protocols typically provide message integrity verification—for example, in MLS, each encrypted message is accompanied by a signed `PublicMessage` from the sender, while the ciphertext itself is protected with AEAD encryption to ensure integrity. However, our scheme requires uploading semantically meaningless encrypted keys to the server. To prevent malicious servers from forging key ciphertexts, we need to define the integrity of Loom:

Definition 3 (Integrity). Loom *satisfies integrity if*

$$\Pr[\mathsf{G}^{\mathcal{A}}_{\mathsf{Int}} \rightarrow 1] \leqslant \mathsf{negl}(\lambda)$$

for Game 1 and any PPT adversary $\mathcal{A}$*.*

Intuitively, $\mathsf{G}^{\mathcal{A}}_{\mathsf{Int}}$ involves normal client-server interaction. If during the search and request phases, the client derives a key sequence different from the previously encrypted sequence based on the server's response without returning $\perp$, the server successfully forges a key sequence.

Post-compromise security against $\mathcal{A}_{thf}$*.* We adopt definitions and proof techniques from CKS [24] to formalize the compatibility of Loom with the SM protocol and to define security against the key thief. Specifically, we identify a class of applications whose security remains unaffected by any additional leakage potentially introduced by Loom. By integrating Loom with such applications, backing up messages and keys preserves the original security guarantees. To model the adversary's

capabilities, we require that Loom be applied only to applications resilient to queries on test and expose oracles without compromising security.

Definition 4. *A security game* $\mathsf{G} = (C, \alpha)$ *is characterized by a challenger* C*, which upon interaction with an adversary* $\mathcal{A}$ *outputs a bit* b*, and the advantage function* $\alpha : \mathbb{N} \to [0, 1]$ *such that the advantage of* $\mathcal{A}$ *winning* G *is characterized by*

$$\mathsf{Adv}_{\mathsf{G}}(\mathcal{A}) = |\Pr[\mathcal{A}(1^\lambda) \leftrightarrow C(1^\lambda) \Rightarrow 1] - \alpha(\lambda)|$$

where the output bit of the interaction is decided by the challenger C*.*

Definition 5. *A game* $\mathsf{G} = (C, \alpha)$ *of an application is said to be* Loom*-compatible if* C *has the access to the oracle* O_{key} *and* $\mathcal{A}$ *has the access to oracle* O_{test} *and* O_{expose} *described as follows:*

- O_{key}*: the key oracle takes an epoch* e *as input. If* e *is queried for the first time, it outputs a* $k \xleftarrow{\$} \{0,1\}^\lambda$*; otherwise, returns the same result as the previous identical query.*
- O_{test}*: the test oracle takes an epoch* e *and a key* k' *as input and output whether* $O_{\mathsf{key}}(e) = k'$*.*
- O_{expose}*: the expose oracle takes an epoch* e *as input and output* $k \leftarrow O_{\mathsf{key}}(e)$*.*

With the above definitions, we demonstrate the security definition of our scheme against $\mathcal{A}_{thf}$.

Definition 6. *For a* Loom*-compatible game* $\mathsf{G} = (C, \alpha)$ *and* Loom*, we define the enhanced* Loom *game* $\mathsf{G}' = (C', \alpha)$*. The* C' *interact with* $\mathcal{A}'$ *as follows:*

- *In the setup phase, the adversary* $\mathcal{A}'$ *can get all the public information from* C'*.* C' *gets these information by forward queries and responses between* $\mathcal{A}'$ *and* C*.*
- C' *accepts queries from* $\mathcal{A}'$ *simulating the executions of* Loom*.*
- $\mathcal{A}'$ *can query the random oracle which* Loom *depends.*
- $\mathcal{A}'$ *can query the compromise oracle at once. Upon receive the query,* C' *forward the query to the* O_{expose} *of* C *to get information to simulate* Loom*.*

Definition 7 (Loom-security against $\mathcal{A}_{thf}$). Loom *is secure if for any* Loom*-compatible game* G *and any admissible PPT adversary* $\mathcal{A}$*, there exists a PPT adversary* $\mathcal{A}'$*, and a negligible function* $\mathsf{negl} : \mathbb{N} \to [0, 1]$ *such that:*

$$\mathsf{Adv}_{\mathsf{G}'}(\mathcal{A}') \leqslant \mathsf{Adv}_{\mathsf{G}}(\mathcal{A}) + \mathsf{negl}(\lambda).$$

5 Details of Loom

This section presents Loom, a searchable cloud backup system achieving forward privacy, backward privacy, and post-compromise security with sublinear search efficiency.

5.1 Design Principles and Core Mechanisms

In SM protocol, each message is encrypted with a fresh secret. Consequently, each backup operation requires a new message key to prevent interference between message ciphertexts and the search index. If we naively adopt a DSSE scheme with a single key for indexing, desynchronization occurs: key updates cannot keep pace with keyword, causing not all keyword head nodes to be updated simultaneously. As illustrated in Figure 3, not all messages (generated from different end-to-end encrypted transmissions) contain all keywords. For example, in Figure 3, msg_1 and msg_2 include keywords w_1, w_2, w_3, whereas msg_3 contains only w_1 and w_3. A newly generated key key_3 can access $index_{1,3}$ and $index_{3,3}$, but fails to retrieve $index_{2,2}$, resulting in incomplete or erroneous query results.

Per-Keyword Key Updates. To resolve this, we follow the common practice in DSSE by maintaining a distinct key for each keyword, which coincides with the design of most forward/backward-private

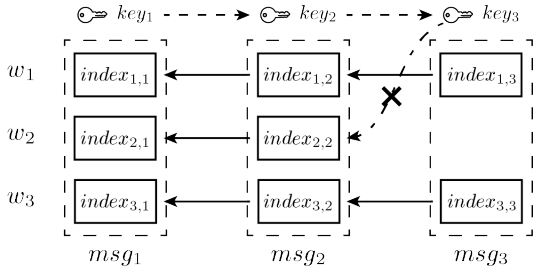


Fig. 3. The Synchronization between Message and Key

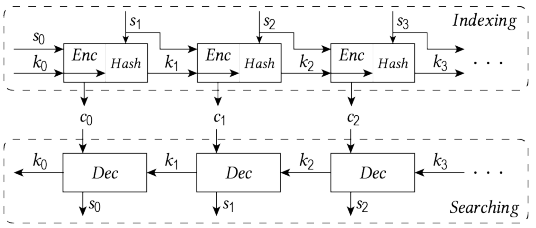


Fig. 4. Illustration of Index Key Chaining in Loom

DSSE schemes [8, 9, 16]. In our approach, each update uses a fresh key to ensure that new updates cannot be linked to previous ones, while each deletion is treated as an independent update, thereby achieving both forward and backward privacy.

Index Key Chaining. Despite per-keyword keys, storing all message keys locally is inefficient. To address this, we propose an *index key chaining*: as shown in Figure 4, the ciphertexts produced by the SM protocol are directly used as database ciphertexts to avoid redundant re-encryption, and new index key for each update encrypts the previous index key and the corresponding message key, embedding it within the index. During search, the current index key iteratively decrypts both the message key and previous index keys, recovering all historical keys. This method significantly reduces local storage overhead and plies the three types of PCS into the PCS of the index.

Progressive Token Conversion for Conjunctive Keyword Search. To enable multi-keyword queries, Loom employs an OXT-like index structure (TSet + XSet). Each update generates keyword indices and $xtags$ inserted into TSet and XSet, respectively. For query processing, TSet is queried to locate entries matching the main keyword, whereas XSet, which stores the associated $xtag$s, is used to verify the presence of secondary keywords. However, with index key updates, a newly generated $xtoken$ using the latest key cannot match prior $xtags$. To overcome this limitation, we construct a *conversion term* during index construction that depends on both the current and the new index keys, allowing the latest $xtoken$ to be gradually transformed into previous $xtokens$. As a result, the $xtoken$ size becomes independent of $|upd(w_1)|$ (the number of updates for the main keyword), reducing communication overhead and preventing the search token from directly revealing the update frequency of w_1. This approach, however, introduces a new risk: since the conversion term can transform both $xtokens$ and $xtags$, new updates may become linkable to previous ones, thereby compromising forward privacy. To mitigate this, we incorporate a blinding factor into the conversion term, which is activated only during search.

Efficient Cross-Group and In-Group Backup. In Loom, secrets and states are decoupled from specific groups, allowing a member to consolidate messages from multiple groups into a single searchable index. Within a group, Loom achieves efficient in-group state synchronization through shared

secrets, enabling one member to back up messages that other members can subsequently search. Details of the group messaging design are provided in Section 5.

5.2 Data Indexing and Query Processing

Algorithm 2 and Protocol 3-6 present the details of Loom. To initialize the scheme, algorithm Setup takes a security parameter λ as input, initializes group parameter, three cryptographic hash functions and an IPF family. It then samples a key k_t, creates an empty client state $\mathsf{State_c}$, and initializes three empty lists EDB, TSet, and XSet. Finally, the client uploads EDB, TSet, and XSet to the server.

Algorithm 2: Loom.Setup(λ)

1 Generate group parameter $\mathsf{GP} = (\mathbb{G}, q, g)$;
2 Initialize five cryptographic hash functions $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$, $\mathsf{H}_1 : \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{H}_2 : \{0,1\}^* \to \mathbb{Z}_q^*$, $\mathsf{H}_m : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$, $\mathsf{H}_x : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \mathbb{Z}_q^*$;
3 Select an IPF family $\mathsf{F}' : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$, F'^{-1} denotes its inverse function and a PRF family $\mathsf{F} : \{0,1\}^* \times \{0,1\}^\lambda \to \{0,1\}^\lambda$;
4 Initialize empty lists $\mathsf{State_c}$, EDB, TSet, XSet $\leftarrow \phi$;
5 Sample a key $k_t \leftarrow_\$ \{0,1\}^\lambda$;
6 Send EDB, EID = (TSet, XSet) to the server;

When the SM secret is updated, client extracts keyword set W_t from the message and executes DataUpdate. For each keyword in W_t, it generates an index tuple $(addr, val = (B, C, D, E))$ and a cross-term $xtag$, which are uploaded along with the SM ciphertext. $addr$ serves as the address of the node, B acts as a pointer to the next node in the linked list, C stores identifier and secret, D contains the value for cross-term verification and E converts $xtoken$ to the previous one.

When performing Search, the client finds the keyword w_1 with the fewest updates among the conjunctive search keyword set by checking their counters cnt in $\mathsf{State_c}$. The client then computes a search token stk, which contains a main keyword token $mtk = (addr_s, k_s)$ and cross tokens. The component $addr_s$ locates the latest update node for the keyword w_1. The cross tokens then check for the number of cross-terms and store them with C (Line 16–27). Components k_s compute using the pointer to obtain the next address, iterating to find the complete update list for w_1.

After Search, the client may execute Request immediately or delay it to reduce potential leakage. Executing Request allows the client to recover identifiers and secrets from the search results (Line 7–13), check if the document contains all searched keywords (Line 14–19), upload the identifiers to request corresponding ciphertexts (Line 22–26), and decrypt the messages associated with the searched keywords (Line 28–31).

If a client mistakenly backs up sensitive information, Rollback can revert to the previous state by repeatedly executing the search operation on the prior update, recovering the previous secret, and restoring the client's state to the prior one. Notably, this operation differs from the del operation in DataUpdate. Once Rollback is executed, the client's subsequent state no longer includes the rolled back secret information, meaning the uploaded secret ciphertext and message ciphertext can no longer be searched or decrypted.

An Example of Loom. Figure 5 and Figure 6 illustrates the operation of Loom using a concrete example, where we focus on keyword w_1 and its associated operations for clarity. We first execute two consecutive updates, DataUpdate(w_1, w_2) (depicted by dashed lines) and DataUpdate(w_1, w_2, w_3) (solid lines). For each update, the server inserts the corresponding entries into TSet and XSet.

Protocol 3: Loom.DataUpdate$\langle C(\mathsf{State_c}, k_t, \tilde{c_t}, s_t, s_{t+1}, (\mathsf{W}_t, id, op)); S(\mathsf{EDB}, \mathsf{EID})\rangle$

Client:
Set id_t as the identifier of $\tilde{c_t}$; Compute $k_{t+1} \leftarrow \mathsf{H}(s_{t+1}||k_t)$; Initialize two sets uSet and xuSet;
for $w \in \mathsf{W}_t$ **do**
  if $\mathsf{State_c}[w] = \perp$ **then**
    $\mathsf{State_c}[w] \leftarrow (k_{w,d} \leftarrow \{0,1\}^\lambda, k_{w,s} \leftarrow \{0,1\}^\lambda, cnt_w \leftarrow 0, T_{w,d} \leftarrow \{0,1\}^\lambda)$
  end
  Retrieve $(k_{w,d}, k_{w,s}, cnt_w, T_{w,d})$ from $\mathsf{State_c}[w]$;
  Compute $tk_w \leftarrow \mathsf{F}_{k_{w,d}}(w)$, $tk'_w \leftarrow \mathsf{F}_{k_{w,d+1}}(w)$, $k_{w,d+1} \leftarrow \mathsf{H}(s_{t+1}||k_{w,d})$, $k_{s+1} \leftarrow \mathsf{F'}_{\mathsf{H}_1(tk'_w)}(k_s)$,
  $B \leftarrow \mathsf{H}_1(tk_w) \oplus \mathsf{H}_m(k_{s+1}, (\mathsf{H}_1(tk'_w)), C \leftarrow \mathsf{Enc}_{k_{w,d+1}}(op||id||s_t||k_{w,d})||T_{w,d}, T_{w,d+1} \leftarrow \mathsf{H}(C)$,
  $D \leftarrow \mathsf{H}_2(\mathsf{F}_{k_t}(op||id)) \cdot \mathsf{H}_2(\mathsf{F}_{k_{w,d+1}}(w||(cnt_w+1)))^{-1}$,
  $E \leftarrow \mathsf{H}_2(\mathsf{F}_{k_{w,d}}(w||cnt_w)) \cdot \mathsf{H}_2(\mathsf{F}_{k_{w,d+1}}(w||(cnt_w+1)))^{-1} \cdot \mathsf{H}_x(k_{s+1}, (\mathsf{H}_1(tk'_w))$
  Insert $(addr = \mathsf{H}_1(tk'_w), val = (B, C, D, E))$ into uSet;
  Compute $xtag_w \leftarrow g^{\mathsf{H}_2(\mathsf{F}_{k_t}(op||id)) \cdot \mathsf{H}_2(w)}$ and insert $xtag_w$ into xuSet;
  Update $\mathsf{State_c}[w] \leftarrow (k_{w,d+1}, k_{w,s+1}, cnt_w + 1, T_{w,d+1})$;
end
Update $k_t \leftarrow k_{t+1}$, delete s_t and send $utk = ((id_t, \tilde{c_t}), \mathsf{uSet}, \mathsf{xuSet})$ to the server;
Server:
Store $\mathsf{EDB}[id_t] \leftarrow \tilde{c_t}$;
for *each* $(addr, val)$ *in* uSet **do**
  Store $\mathsf{TSet}[addr] \leftarrow val$;
end
for *each* $xtag$ *in* xuSet **do**
  Set $\mathsf{XSet}[xtag] \leftarrow 1$;
end

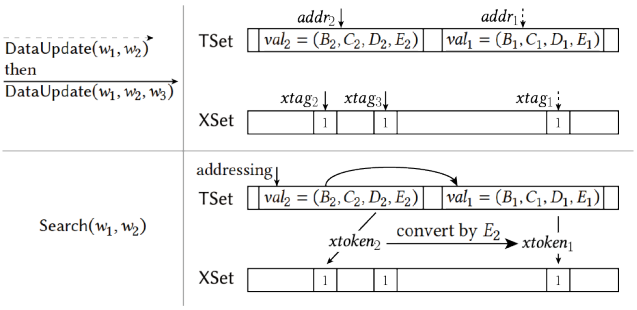


Fig. 5. An Update Example of Loom (Perspective of w_1)

When a conjunctive search $\mathsf{Search}(w_1, w_2)$ is issued, the server starts from the most recent address $addr_2$ associated with w_1. It computes $xtoken_2$ using D_2 to test the presence of $xtag_2$ in XSet. Upon a successful cross-term check, the server follows the backward pointer B_2 to $addr_1$, while simultaneously applying E_2 to transform $xtoken_2$ into $xtoken_1$. This enables iterative cross-term verification against earlier updates without revealing intermediate plaintexts.

When the second update needs to be rolled back (Figure 6), the client invokes the Search protocol to decrypted prior secrets, thereby rolling back its locally maintained state (as illustrated by the transition from $\mathsf{State_c}[w]_2$ to $\mathsf{State_c}[w]_1$) while the server-side index and data remains unchanged.

Protocol 4: Loom.Search$\langle C(\mathsf{State_c}, \mathsf{W}_s); S(\mathsf{EID})\rangle$

Client:
Parse $\mathsf{State_c}[w_1].cnt$ is minimum in W_s; \\Select the keyword with the fewest updates;
Retrieve $(k_{w_1,d}, k_{w_1,s}, cnt_{w_1}, T_{w_1,d})$ from $\mathsf{State_c}[w_1]$;
Compute $tk_w \leftarrow \mathsf{F}_{k_{w,d}}(w)$, $addr_s \leftarrow \mathsf{H}_1(tk_w)$, set $mtk = (addr_s, k_{w_1,s})$;
Initialize an empty list xtokenList;
for $i = 2$ *to* $\mathsf{W}_s.size$ **do**
  Set $xtoken_i \leftarrow g^{\mathsf{H}_2(w_i)\cdot \mathsf{H}_2(\mathsf{F}_{k_{w_1,d}}(w_1||cnt_{w_1}))}$, insert $xtoken$ into xtokenList;
end
Send $stk = (mtk, \text{xtokenList})$ to the server;
Server:
Parse $stk = ((addr_s, k_s), \text{xtokenList})$;
Initialize a result list resList;
while $\mathsf{TSet}[addr_s] \neq$ NULL **do**
  Retrieve $(B, C, D, E) \leftarrow \mathsf{TSet}[addr_s]$; Set $xcnt = 1$;
  for $i = 0$ *to* xtokenList.$size - 1$ **do**
    Set $xtoken_i \leftarrow$ xtokenList$[i]$; Compute $xtag_i \leftarrow xtoken_i^D$;
    if $\mathsf{XSet}[xtag_i] = 1$ **then**
      $xcnt \leftarrow xcnt + 1$ \\Check the cross-term to confirm the co-presence of w_1 and w_i;
    end
    xtokenList$[i] \leftarrow xtoken_i^{E/\mathsf{H}_x(k_s, addr_s)}$; \\Conversion;
  end
  Compute $htk \leftarrow B \oplus \mathsf{H}_m(k_s, addr_s)$, $k_s \leftarrow \mathsf{F}'^{-1}_{addr_s}(k_s)$, $addr_s \leftarrow htk$; Insert $(C, xcnt)$ to resList;
end

As a result, the first addressing during the Search(w_1, w_2) after an update (Search(w_1, w_2) in Figure 6) now resolves to $addr_3$, and the iterative traversal skips the rolled-back node entirely, proceeding directly from $addr_3$ to $addr_1$. The second update of w_1 becomes an orphaned branch in the search chain: it is neither searchable nor decryptable, effectively achieving the rollback effect.

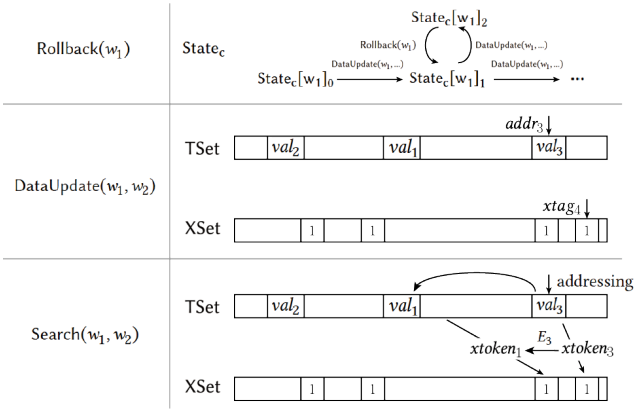


Fig. 6. A Rollback Example of Loom (Perspective of w_1)

Protocol 5: Loom.Request$\langle C(\mathsf{State_c}, \mathsf{resList}, \mathsf{W}_s); S(\mathsf{EDB})\rangle$

Client:
Initialize two empty lists idList, secList; Parse $\mathsf{State_c}[w_1].cnt$ is minimum in W_s;
Retrieve $(k_{w_1,d}, k_{w_1,s}, cnt_{w_1}, T_{w_1,d})$ from $\mathsf{State_c}[w_1]$; Set $k_d = k_{w_1,d}, T_d = T_{w_1,d}$
for $i = 0$ *to* $i = cnt_{w_1} - 1$ **do**
  Retrieve $(C_i, xcnt_i)$ from resList$[i]$; Parse $\mathsf{CT}||T_i \leftarrow C_i$;
  if $\mathsf{H}(\mathsf{CT}||T_i) \neq T_d$ **then**
    return $\perp$;
  end
  Compute $(op_i||id_i||s_i||k_i) \leftarrow \mathsf{Dec}_{k_d}(\mathsf{CT})$; Set $k_d \leftarrow k_i, T_d \leftarrow T_i$, insert s_i into secList;
  if op_i *is add and* $xcnt_i = \mathsf{W}_s.size$ **then**
    Insert id_i to idList; \\Contains all searched keywords;
  end
  if op_i *is del* **then**
    Delete id_i from idList;
  end
end
Server:
Initialize an empty list cipherList;
for *each* $id_i \in$ idList **do**
  Insert $\tilde{c_i} = \mathsf{EDB}[id]$ to cipherList;
end
Return the result set cipherList to client;
Client:
for *each* $\tilde{c_i} \in$ cipherList **do**
  Retrieve s_i in secList; Decrypt and get the message $m_i \leftarrow \mathsf{MLS.Dec}_{s_i}(\tilde{c_i})$;
end

Protocol 6: Loom.Rollback$\langle C(\mathsf{State_c}, \mathsf{W}_{rb}); S(\mathsf{EID})\rangle$

Client:
Initialze an empty list rbkList;
for w *in* W_{rb} **do**
  Execute resList $\leftarrow$Loom.Search(w); Add $(w, \mathsf{resList}[0])$ into rbkList;
end
for $i = 0$ *to* rbkList.$size - 1$ **do**
  Retrieve $(w_i, (C_i, xcnt_i))$ from $\mathsf{rbkList}_{rb}[i]$; Parse $\mathsf{CT}||T_i \leftarrow C_i$;
  Retrieve $(k_{w_i,d}, k_{w_i,s}, cnt_{w_i}, T_{w_i,d})$ from $\mathsf{State_c}[w_i]$;
  if $\mathsf{H}(\mathsf{CT}||T_i) \neq T_{w_i,d}$ **then**
    return $\perp$;
  end
  Compute $(op_e||id_e||s_e||k_e) \leftarrow \mathsf{Dec}_{k_{w_i,d}}(\mathsf{CT})$, $tk \leftarrow \mathsf{F}_{k_{w_i,d}}(w)$, $k_{w_i,s-1} \leftarrow \mathsf{F'}^{-1}_{\mathsf{H}_1(tk)}(k_s)$;
  Rollback $\mathsf{State_c}[w_i] \leftarrow (k_e, k_{w_i,s-1}, cnt_w - 1, T_e)$; Set $k_{w,d} \leftarrow k_{w,d-1}$;
end

Correctness of Loom. When the Search protocol is executed with the keyword set W (assuming w_1 is the least frequently updated keyword in W), the tuple $addr, val$ can be located by $addr_s$ in stk.

The value D in val can locate $xtoken$ to the storage address of the cross-term, while the remaining values in val can correctly transform stk to locate the tuple from the most recent update. This allows iteratively identifying all C items of w_1 and the occurrence frequencies of other keywords in the conjunctive query, stored in resList. The client can decrypt id, op, and message key s using the locally stored state $k_{w,d}$ while occurrence frequencies of the joint keywords can determine whether an item contains all keywords simultaneously. The client then requests the message corresponding to the id from the server and decrypts it using the corresponding message key to obtain the plaintext. Rollback protocol reverts the client state, preventing it from searching the most recently updated index and decrypting the message key contained therein, thus rendering the message ciphertext unlocatable and undecryptable.

5.3 Security Analysis

5.3.1 The Security Against $\mathcal{A}_{srv}$. We will demonstrate that Loom achieves the forward and backward privacy mentioned earlier, beginning with a formal definition of these leakage functions. Let Q be a list with the following types of entries: (i) $(e, op, (id, w))$: update query $op \in \{add, del\}$ on identity-keyword pair (id, w) at epoch e; (ii) (e, rbk, w): rollback query on keyword w at epoch e; (iii) (t, w): search query on keyword w at timestamp t.

Output Leakages. Since the client must execut Request after performing Search to retrieve the corresponding message ciphertext from the server, the server can be assumed to associate these message identifiers with Search. We further assume that the user performs a rollback before Search in each time period. For any keyword w, we define $\mathsf{TimeDB}(w)$ to be the function that returns the list of all message identifiers containing w that have not yet been deleted and rolled back along with their respective timestamps of insertion.

$$\mathsf{TimeDB}(w) = \{(e, id) | (e, add, (id, w)) \in Q \wedge \forall e' : (e', del, (id, w)) \notin Q \ \wedge \forall e'' : (e'', rbk, w) \notin Q\}$$

For conjunctive keyword search, we overload the notation to define $\mathsf{TimeDB}(q)$ for any conjunctive query $q = (w_1 \wedge \cdots \wedge w_n)$ as:

$$\mathsf{TimeDB}(q) = \{(e_{i\in[n]}, id) | (e, add, (id, w_i)) \in Q \wedge \forall e' : (e', del, (id, w_i)) \notin Q \wedge \forall e'' : (e'', rbk, w_i) \notin Q\}$$

Index Leakages. During a search, as the server links the latest update with previous updates, all update epochs in this chain are leaked to the server. For any keyword w, keyword pair (w_1, w_2) and query $q = (w_1 \wedge \cdots \wedge w_n)$ where w_1 is the s-term, we define $\mathsf{Upd}(w)$, $\mathsf{Upd}(w_1 \wedge w_2)$ and $\mathsf{Upd}(q)$ as follows:

$$\mathsf{Upd}(w) = \{t \mid \exists (op, id) : (t, op, (id, w)) \in Q\},$$

$$\mathsf{Upd}(w_1, w_2) = \{(t_1, t_2) \mid \exists (op, id) : (t_1, op, (id, w_1)) \in Q \wedge (t_2, op, (id, w_2)) \in Q\},$$

$$\mathsf{Upd}(q) = \mathsf{Upd}(w_1) \ \cup \ \left(\bigcup_{i=2}^{n} \mathsf{Upd}(w_1, w_i) \right).$$

Compromise Leakages. When a compromise occurs, the adversary can obtain the user's current key and, based on our assumptions, the server can learn the value of previous keys. However, according to our construction, even after obtaining previous keys, the server cannot discern the keyword content in the indexes of TSet and XSet, only the information about the document identifiers these indexes contain. In addition, when new updates encrypted with new keys arrive, the server cannot discern their content or associate them with previous updates before a search is performed, which is the security property satisfied by our defined PCS. As the function that returns

a list of all operations and message identifiers updated before t, which have not been rolled back, along with their corresponding epochs, all the previous keys and states.

$$\mathsf{Comp}(t) = \{\{k_i\}_{i=1}^{\lfloor t \rfloor}, \{\mathsf{State}_{c,i}\}_{i=1}^{\lfloor t \rfloor}, \{s_i\}_{i=1}^{\lfloor t \rfloor}\} \cup \{(e, op, id) | (e, op, (id, w)) \in Q \wedge \forall e' : (e', rbk, w) \notin Q\}$$

Loom *Leakage Profile.* We now formally define the leakage profile for Loom as:

$$\mathcal{L} = (\mathcal{L}_{srv}^{\mathsf{Stp}}, \mathcal{L}_{srv}^{\mathsf{Upd}}, \mathcal{L}_{srv}^{\mathsf{Srch}}, \mathcal{L}_{srv}^{\mathsf{Req}}, \mathcal{L}_{srv}^{\mathsf{Rbk}}, \mathcal{L}_{srv}^{\mathsf{Comp}})$$

where

$$\mathcal{L}_{srv}^{\mathsf{Stp}} = \perp, \mathcal{L}_{srv}^{\mathsf{Upd}}(op, id, \mathsf{W}) = \perp, \mathcal{L}_{srv}^{\mathsf{Srch}}(q) = \mathsf{Upd}(q), \mathcal{L}_{srv}^{\mathsf{Req}}(\mathsf{resList}, q) = \mathsf{TimeDB}(q),$$

$$\mathcal{L}_{srv}^{\mathsf{Rbk}}(q) = \bigcup_{i=2}^{n} \mathsf{Upd}(w_i), \mathcal{L}_{srv}^{\mathsf{Comp}}(t) = \mathsf{Comp}(t).$$

Finally, we state the following theorem for the security of Loom, along with a sketch of its proof.

Theorem 1 (Post-compromise security against $\mathcal{A}_{srv}$ of Loom). *Assuming that* F *and* F′ *are secure PRFs, the decisional Diffie-Hellman assumption holds over the group* $\mathbb{G}$ *and hash functions are modeled as random oracle, the* Loom *scheme is post-compromise secure against* $\mathcal{A}_{srv}$ *with respect to a leakage function* $\mathcal{L}_{srv}$.

Proof. The proof's core is to ensure that an adversary interacting with simulator $\mathcal{S}$ cannot distinguish it from the real game. $\mathcal{S}$ emulates the components of DataUpdate using random values as follows:

- Main keyword index: Random values are used to simulate B and C during index construction. During search, simulator programs the blinding values through a programmable random oracle so that the updated chain's address matches the previous one.
- Cross-term index: D and $extag$ are simulated as random values, while $xtoken$ is simulated as $g^{\frac{extag}{D}}$. The linking element E, which connects two $xtoken$, is generated as a random value due to its blinding component. During search, the simulator programs the blinding component via a programmable random oracle to unblind as $\frac{extag_{t-1} \cdot D_t}{D_{t-1} \cdot extag_t}$.

The $\mathcal{S}$ can simulate data backup without keyword information, effectively using $\mathcal{L}_{srv}^{\mathsf{Upd}}$ for the backup phase. During search, it only needs to know which updates are related (generated by a keyword), without requiring any keyword-related information, matching the output of $\mathcal{L}_{srv}^{\mathsf{Srch}}$.

If any keys are leaked, $\mathcal{S}$ invokes $\mathcal{L}_{srv}^{\mathsf{Comp}}$ to obtain the prior keys. It then uses a programmable random oracle and non-committing encryption to link these prior keys with the uploaded ciphertexts, thereby making the simulation indistinguishable from a real game. Thus, $\mathcal{S}$ can use $\mathcal{L}$ to simulate a game indistinguishable from the real game to the adversary. □

Based on the definitions of forward privacy and type-II backward privacy provided aforementioned, it is evident that Loom scheme satisfies both forward and backward security. In other words, the following is two natural corollary:

Corollary 1 (Forward Privacy of Loom). *Assuming that* F *and* F′ *are secure PRFs and the decisional Diffie-Hellman assumption holds over the group* $\mathbb{G}$, *the* Loom *scheme is adaptively forward private.*

Corollary 2 (Backward Privacy of Loom). *Assuming that* F *and* F′ *are secure PRFs and the decisional Diffie-Hellman assumption holds over the group* $\mathbb{G}$, *the* Loom *scheme is adaptively Type-II backward private.*

We then state the following theorem for the integrity of Loom.

Theorem 2. *Assuming that hash functions* H_1 *and* H_2 *are collision resistant and SE scheme is correct,* Loom *scheme satisfies integrity, i.e., for any PPT adversary* $\mathcal{A}$ $\Pr[\mathsf{G}^{\mathcal{A}}_{\mathsf{Int}} \rightarrow 1] \leqslant \mathsf{negl}(\lambda)$.

Proof. Intuitively, to mislead the client to compute the key of previous messages without distinguishing the ciphertext as fake and rejecting the computation during the process, the server must find a collision pair $\mathsf{H}(\mathsf{CT}||T) = \mathsf{H}(\mathsf{CT}_f||T_f)$ and send CT_f, T_f as ciphertext to the client. However, the cryptographic hash is collision-resistant, meaning that any PPT adversary $\mathcal{A}$ can only find a collision for a specific pair (CT, T) with negligible probability. □

5.3.2 The Security Against $\mathcal{A}_{thf}$. Next, we address the security against the key thief, ensuring that after a key compromise, our scheme does not compromise the post-compromise security of the SM protocol $\mathsf{PCS}_{\mathsf{MSG}}$. We first demonstrate the Loom-composability of MLS, observing that our definition of Loom-compatible aligns with the CKS-compatible definition by Dodis et al[24]. They established Signal protocol and similar secure messaging protocols are CKS-compatible, so we omit the proof of Loom-compatibility of SM protocol. For the post-compromise security against $\mathcal{A}_{thf}$, we require the convergent encryption to be non-committing. We present the following theorem:

Theorem 3. *Assuming SE scheme is correct, non-committing and one-time IND-CPA secure and hash functions are modeled as a random oracle,* Loom *is secure, i.e., for any* Loom*-compatible game* G *and any admissible PPT* $\mathcal{A}$ *there exists a PPT adversary* $\mathcal{A}'$ *such that* $\mathsf{Adv}_{\mathsf{G}'}(\mathcal{A}') \leqslant \mathsf{Adv}_{\mathsf{G}}(\mathcal{A}) + \mathsf{negl}(\lambda)$.

Proof. The proof constructs $\mathcal{A}$ to interact with $\mathcal{A}'$, forwarding the answer of $\mathcal{A}'$ to C. If $\mathcal{A}'$ breaks the simulated game (constructed by $\mathcal{A}$) with non-negligible probability, $\mathcal{A}$ wins the game interacting with C with at least the same probability, establishing the relation: $\mathsf{Adv}_{\mathsf{G}'}(\mathcal{A}') \leqslant \mathsf{Adv}_{\mathsf{G}}(\mathcal{A}) + \mathsf{negl}(\lambda)$. The core of proof, as in theorem-1, is that $\mathcal{A}$ ensure $\mathcal{A}'$ cannot distinguish whether it is interacting with $\mathcal{A}$ or with a real game. In detail, we assume that the symmetric encryption scheme used to encrypt message keys is *one-time IND-CPA*, , which guarantees that ciphertexts are indistinguishable to $\mathcal{A}'$. Hence, $\mathcal{A}$ can simulate ciphertexts with random values. If the index key is compromised, $\mathcal{A}$ queries the expose oracle to obtain the real message key. Since the underlying symmetric encryption is *non-committing*, $\mathcal{A}$ can open the index keys and pass them to $\mathcal{A}'$ as the answer of the compromise oracle. The adversary then programs the random oracle so that its output on the message key and the previous index key matches the index key which *non-committing encryption* opens. Thus, $\mathcal{A}$ successfully simulates the game. □

6 Supporting Dynamic Group Scenario

In Loom, secrets and states are not tied to group-specific identifiers. This design naturally enables a member involved in multiple groups to back up all chat histories, consolidating messages from vary groups into a single searchable index. However, if each group member independently backs up every received message, three inefficiencies arise: (i) redundant computation across members, (ii) duplicated ciphertext storage on the server, and (iii) difficulty maintaining synchronization for offline members.

Inherently, Loom employs the secret of each new message or session for state evolution, rather than generating a fresh random state each time. This facilitates efficient state synchronization among group members without requiring explicit coordination. In a group-messaging scenario, this mechanism allows a single member, typically the online message sender, to compute and upload a secure index, while other members simply update their state using the secret associated with the new message or session. When performing a search, members can directly use the latest state. This approach reduces both client-side computation and server storage overhead.

Protocol 7: LoomRelay Construction

This construction unifies the client-server operations for state updates, searches, and member removals.
DataUpdate$\langle C(\mathsf{State_c}, r, \tilde{c_t}, s_t, (\mathsf{W_t}, id, op)); S(\mathsf{EDB}, \mathsf{EID})\rangle$
Client: ... compute $B \leftarrow tk_w \oplus \mathsf{F}_{k_{s+1}}(tk'_w) \oplus r$ ...
Server: ... process update on EDB, EID ...
Search$\langle C(\mathsf{State_c}, w); S(\mathsf{EID})\rangle$
Client: ... set $mtk \leftarrow (addr_s, val_s, r, k_s)$ ...
Server: ... compute $tk \leftarrow B \oplus val_s \oplus r$ ...
UserRemove$\langle C(r, s_t); S(\mathsf{EID})\rangle$
Client: $r' \leftarrow \mathsf{H}_1(r, s_t)$; $\Delta \leftarrow r' \oplus r$; send Δ
Server: for *every val in* EID **do**
  $val.B \leftarrow val.B \oplus \Delta$
end

By contrast, if the index key were randomly generated, as in schemes such as Bamboo, members would be unable to update independently. In that case, the uploader must distribute the *new state* to all members after each message or session. Drawing on convergent encryption and CKS, Loom avoids frequent state sharing by having messages specify only the keywords that *need to be updated*. Members can then derive the latest state using new secrets.

To build a backup system for group scenario, it is important to integrate with applications that perform dynamic group operations, such as adding or removing members. This allows our system to enforce access control based on the group membership maintained by the application, ensuring that individuals removed by the application cannot access the backup system.

Adding Members. When a new member joins the group, the MLS protocol issues a `welcome` message to the newcomer. In Loom, the inviter can further delegate the current state to the new member along with this message. This process allows the new member to query previously backed-up messages immediately upon joining, while ensuring that all group members remain synchronized with the latest state.

Erasing Members. When a member is removed, the remaining participants receive fresh entropy to update their keys. The removed member, lacking access to this entropy, cannot derive new cryptographic material (such as the secret keys). In our backup system, this mechanism also revokes the removed member's search capability. Each member maintains a local state for index construction and query execution; upon member removal, this state is refreshed using the newly shared entropy. Since the removed member does not receive the updated state, they lose the ability to generate valid indexes or initiate search requests.

To support these operations, we present an enhanced version, LoomRelay, as described in Protocol 7 (here we only outline the modifications, omitting parts identical to the original version). As shown in Figure 7, in LoomRelay, the group member state r, which is shared among members within the group, is embedded within the links of the index's linked list structure. Member removal triggers a newly shared application secret. In algorithm UserRemove, the remaining members can compute a new state r' and an incremental update Δ to the database with the new secret, ensuring that the server's stored index reflects the revised membership. Notably, only the "links" of the list are updated in UserRemove, the removed member can only access the latest backup created before their removal. Since the index addresses remain unchanged, reinserting all indexes is unnecessary. This design achieves a practical balance between security and efficiency.

In the LoomRelay model, group dynamic operations are initiated and executed by the application integrated with it. LoomRelay maintains its internal state to stay synchronized with the evolving

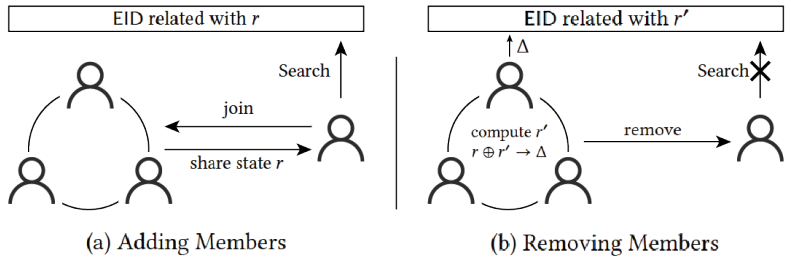


Fig. 7. LoomRelay for Group Scenario

group membership and enforces access control over search and decryption capabilities. The backup/ search/ update efficiency of Loom is therefore largely unaffected by the group state. Consequently, our experimental evaluation focuses exclusively on the operations directly involving LoomRelay.

7 Implementations and Evaluations

To evaluate the practicality and system efficiency of Loom, we implemented a prototype on top of the existing MLS protocol and conducted comprehensive experiments. Our evaluation focuses on three aspects: (i) overall system performance under realistic settings, (ii) search efficiency and client-side workload compared to CKS, and (iii) improvements over Bamboo in client key storage and conjunctive keyword search efficiency.

7.1 Implementation

Implementation Detail. For the cryptographic component instantiation, we used AES-CTR for the pseudorandom and invertible pseudorandom functions (PRF/IPF), and the SHA family from OpenSSL 3.0.2 [35] for hashing functions. Non-committing encryption followed the design of CKS. Among the multiple cipher suites supported by *mlspp*, three representative suites were selected to evaluate performance under different cryptographic configurations. In our prototype, we instantiate the encrypted database (EDB) and encrypted index (EID) using *PostgreSQL* tables for persistently storage. Specifically:

- TSet (the main index structure of EID) is mapped to a table with schema `CREATE TABLE indexList (addr TEXT PRIMARY KEY, val TEXT);`.
- XSet (the cross-tag set) is mapped to a table with schema `CREATE TABLE xtagList (value TEXT PRIMARY KEY);` with a `HASH` index on `value` to accelerate membership testing.

All interactions are mediated via the *pqxx* library. During the data upload process, the content $(addr, val)$ is inserted into the TSet using the SQL statement `INSERT INTO indexList (addr, val) VALUES ($1, $2)` where the content of *val* is serialized and stored in the val field. The xtag is inserted into the XSet using the SQL statement `INSERT INTO xtagList (value) VALUES ($1) ON CONFLICT (value) DO NOTHING`. During the search process, addressing in the TSet is performed using the SQL query `SELECT val FROM indexList WHERE addr = $1`, while the cross-tag checking is done using `SELECT EXISTS (SELECT 1 FROM xtagList WHERE value = $1)`. Critically, no cryptographic guarantee or functional property of Loom depends on *PostgreSQL* or any relational features.

Experimental Setup. Loom, LoomRelay, CKS, and Bamboo were implemented in C++ and built upon Cisco's open-source MLS protocol implementation, *mlspp* [17]. All schemes were evaluated under identical conditions to ensure a fair comparison. The experimental environment emulated a secure cloud backup service with one client and one server connected through a local area

Table 1. Hardware and OS Configurations.

Configuration	Client	Server
CPU	Intel Core i7-12700	Intel Xeon Gold 5220R
Memory	35GB	503GB
OS	Ubuntu WSL2 20.04 x64	Ubuntu 20.04 x64

Scheme	CS	Backup Latency (ms)	Search Latency (ms) (Search time + Decrypt time)							
			BW = 10 Mbps				BW = 50 Mbps			
			RTT = 10	RTT = 20	RTT = 50	RTT = 100	RTT = 10	RTT = 20	RTT = 50	RTT = 100
Loom	S_1	50714	8883 + 179	11245 + 152	16999 + 258	27535 + 526	9381 + 115	12552 + 187	17388 + 259	27059 + 483
	S_2	144863	10327 + 187	12130 + 149	17582 + 265	28157 + 513	9784 + 121	13257 + 228	19254 + 324	27735 + 473
	S_3	85472	10076 + 120	12056 + 142	17325 + 283	27296 + 496	11512 + 165	12792 + 159	18294 + 288	27347 + 473
LoomRelay	S_1	57980	9216 + 121	10852 + 154	16759 + 268	27171 + 466	9627 + 132	11662 + 161	17429 + 327	27281 + 473
	S_2	144207	10084 + 130	11761 + 154	17601 + 273	27405 + 478	9898 + 130	12497 + 189	18315 + 283	28470 + 499
	S_3	85350	9696 + 124	11744 + 176	17075 + 265	27399 + 473	10483 + 128	13102 + 155	17833 + 304	27519 + 481

Table 2. Backup and search performance of Loom and LoomRelay in different cipher suites and network environments, with 10^4 messages backed up, each containing 10 keywords, and a keyword space size of 10^3. (CS–Cipher Suite, S_1–P256-AES128-SHA256, S_2–P384-AES256-SHA384, S_3–P521-AES256-SHA512, BW–Bandwidth, RTT–Round-Trip Time)

network (LAN), using C++ TCP sockets for communication. The hardware and OS configurations are summarized in Table 1. Network latency and bandwidth were simulated using the `tc` command.

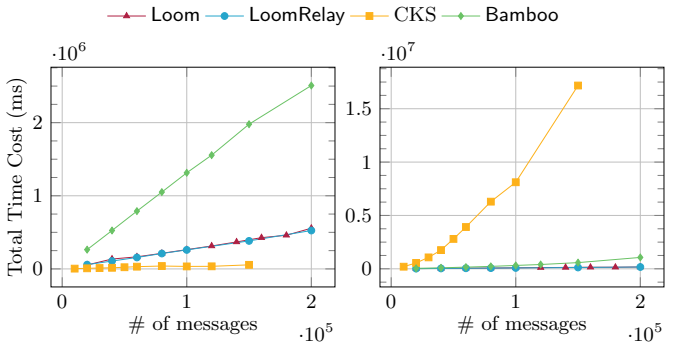
Dataset. We used the Enron email dataset [25], where each email naturally serves as a protocol message whose size and structure closely resemble those in real communication systems. Each email was associated with a set of keywords selected from a predefined keyword space, encrypted under the MLS protocol, and backed up by Loom to emulate a realistic cloud backup workload.

7.2 Performance Evaluation

Overall System Performance. We first measured the backup and search performance of Loom and LoomRelay across different cipher suites and network environments. Table 2 shows the total backup time for 10^4 messages, each uploaded individually. The average per-message latency is 5-15ms, comparable to typical IM message latency (tens of milliseconds). The search performance was evaluated on this set of message (size 10^4), with bandwidth varied from 10Mbps to 50Mbps and RTT from 10ms to 100ms to reflect different network conditions. In these settings, the search latency ranges from 9.06 to 28.97 s. We found that bandwidth has little impact on the scheme's search efficiency, as the process is mainly computation-bound: both the search token and the retrieved files are small, making transmission time nearly independent of bandwidth variations. Conversely, RTT has a pronounced effect, since decryption requires an additional query-response round with the server to fetch the corresponding document, thereby amplifying the impact of network latency. Without loss of generality, subsequent comparative experiments use the P256-AES128-SHA256 suite as the default configuration.

Comparison with Baselines. To demonstrate the benefits of integrating searchability while retaining compact key management, we compared Loom and LoomRelay with CKS (supporting compact keys for MLS) and the DSSE scheme Bamboo with $\mathsf{PCS}_{\mathsf{IDX}}$. Figure 8 presents the backup and search efficiency of all schemes. To eliminate disk I/O bottlenecks, multi-threaded database insertion was used. Compared with CKS, which directly uploads keys after derivation, searchable schemes (Loom, LoomRelay and Bamboo) incur slightly higher backup latency due to per-message indexing. However, they drastically reduce search overhead: CKS must download and decrypt the entire backup to locate keyword-related messages, whereas searchable schemes allow direct retrieval.

Consequently, all searchable schemes outperform CKS by 25× - 87× in search efficiency. Among the searchable schemes, Loom and LoomRelay achieve comparable security while outperforming Bamboo in both backup and search performance, primarily due to Bamboo's heavier reliance on hash and modular exponentiation operations during index construction and query processing.



(a) Backup Efficiency Comparison (b) Search Efficiency Comparison

Fig. 8. Backup and Search Time Comparison

Performance Improvement from Search Capability. To gain deeper insight into the advantages introduced by search functionality, we further analyzed the performance improvements of Loom compared with CKS. As shown in Table 3, we measured the communication overhead of retrieving messages associated with specific keywords across datasets of varying size. The results demonstrate that search capability effectively eliminates the excessive communication cost incurred by downloading large volumes of irrelevant data. Moreover, in the client-server model, search functionality offers an additional advantage by offloading a portion of computation to servers equipped with sufficient computing and storage resources.

To validate this observation, we further examined the client-side workload in keyword retrieval, quantified by the decryption time required after downloading the corresponding messages. We adopted single-threaded, sequential key derivation and decryption to more accurately capture client-side pressure. The results in Table 3 show that the search capability significantly reduces both computational and storage overhead on the client. Overall, for clients with limited bandwidth, processing power, or storage, Loom alleviates the client-side burden compared to CKS over MLS.

Table 3. Overhead Comparison Between CKS and Loom

Metric	Scheme	Number of Messages		
		2×10^4	4×10^4	6×10^4
Communication Overhead	CKS	131.52 MB	245.18 MB	385.80 MB
	Loom	7 KB	13 KB	49 KB
Computational Overhead	CKS	552.77 s	1761.20 s	2957.67 s
	Loom	176 ms	248 ms	359 ms

Performance Improvement from Compact Index and Conjunctive Search. As shown in Figure 9 (a), Loom compacts all message keys into the index, effectively reducing client-side key storage. In this design, the client only needs to maintain a single index key, instead of keeping a separate key

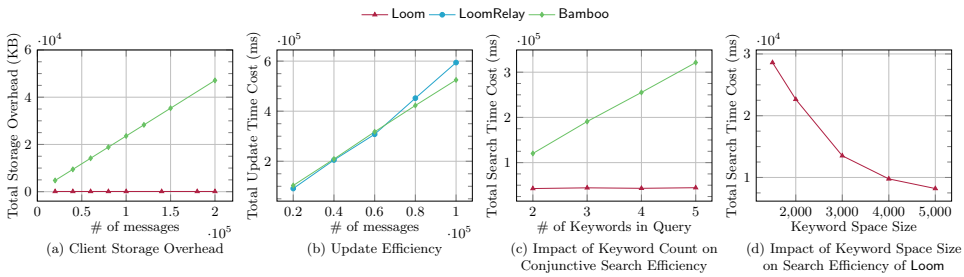
for each message. In contrast, Bamboo must retain all historical message keys locally to decrypt message contents, which introduces substantial key storage overhead that can easily exceed typical client capacity. This compact index design significantly lightens client-side state management, an essential property for lightweight or mobile clients.

For schemes supporting ciphertext updates, such as LoomRelay and Bamboo (triggered under different conditions), we measured their update efficiency, as shown in Figure 9 (b). Interestingly, LoomRelay performs comparably to Bamboo, which deviates from our earlier analysis. Interestingly, LoomRelay performs comparably to Bamboo, which deviates from our earlier analysis. This reduced performance gap can be attributed to database I/O emerging as the dominant bottleneck, rather than the cryptographic update operations themselves.

Regarding search functionality, Loom supports conjunctive keyword queries and achieves higher efficiency when retrieving messages matching multiple keywords. To provide a fair comparison, we implemented the same functionality in Bamboo by repeatedly invoking Bamboo.*search* and intersecting the results of multiple searches, as illustrated in Figure 9 (c). Even without accounting for the extra communication and computation caused by repeated searches, Loom demonstrates markedly better efficiency.

This improvement arises because each search in Bamboo requires sequential retrieval from the database, whereas Loom performs a single primary keyword retrieval and verifies cross-keywords through lightweight membership tests using a hash-like structure. Intuitively, for a conjunctive query involving x keywords, Loom performs one retrieval and $x - 1$ membership tests, while Bamboo performs x database retrievals. This pattern is reflected in the results, where Bamboo's overhead grows linearly with the number of queried keywords.

Finally, to investigate the factors influencing Loom's multi-keyword performance, we examined how search efficiency varies with the update frequency of the primary keyword (Figure 9 (d)). The results confirm that the search efficiency of Loom improves as the primary keyword is updated more frequently.



(a) Client Storage Overhead (b) Update Efficiency (c) Impact of Keyword Count on Conjunctive Search Efficiency (d) Impact of Keyword Space Size on Search Efficiency of Loom

Fig. 9. Performance Comparison with Bamboo

8 Discussion

Malicious Server. While our threat model assumes an honest-but-curious server, in practice the server may deviate from the protocol by omitting, modifying, or injecting search results. To mitigate such active behaviors, Loom can be augmented with mechanisms such as incremental Message Authentication Codes (MACs) [18, 39, 42] or homomorphic MACs [1], allowing clients to verify that returned results were uploaded by authorized users and are complete. The incremental and homomorphic properties enable authentication states to be updated efficiently without storing all prior data or recomputing tags from scratch, making these mechanisms compatible with Loom's append-heavy workload.

Search Latency. As commonly observed in encrypted data management systems, Loom incurs higher search latency than plaintext databases, reflecting the well-known efficiency–privacy trade-off. This overhead arises from the need to protect both keyword and query privacy, which requires additional server-side disk addressings to decouple protocol execution from observable access patterns. Prior work [8, 9, 12] has shown that query privacy can leak through protocol operations, making such extra addressings unavoidable. We note that once a search occurs, the linkage among the searched indices is already exposed and no longer requires protection; the remaining sensitive information lies in the linkage between subsequent updates and historical indices. Accordingly, searched items can be *cached* at their latest addresses after a query, avoiding redundant addressings in future searches without compromising security.

Search efficiency can be further improved through two technical routes: (1) *Local SSE* [12, 21, 30, 31] addresses the trade-off between security and the number of addressings required for each search, and we could consider combining Loom with *local SSE*. (2) By increasing the locally stored keys (which can be viewed as addressing handles), the chain-like computation-addressing process (as shown in Figure 4) can be transformed into a tree structure with multiple entry points. This tree structure allows a single computation to produce multiple addresses of the tree node at the next depth, and multi-threading can then be utilized to perform parallel addressing on the disk, thereby reducing search latency.

Failure Mode Tolerance. In practical network environments, transient failures such as packet loss or connection interruptions may cause partial or incomplete uploads. Loom tolerates such failure modes through a built-in rollback mechanism, which allows the client to revert its local state, ensuring that incomplete updates do not affect index consistency or search correctness. Robustness can be further strengthened by incorporating lightweight integrity mechanisms, such as modular MAC-based checks or error-correcting codes, to detect and recover from transient failures.

Acknowledgments

This work was supported by the Yunnan Provincial Major Science and Technology Special Plan Projects (202502AD080008), National Natural Science Foundation of China (No.62302226, No.U2336205, No.62302224, No.62502207) and Yunnan Provincial New R&D Institution Cultivation Project (202404BQ040148).

References

[1] Shweta Agrawal and Dan Boneh. 2009. Homomorphic MACs: MAC-Based Integrity for Network Coding. In *Applied Cryptography and Network Security, 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5536)*. 292–305.

[2] Omer Akgul, Wei Bai, Shruti Das, and Michelle L. Mazurek. 2021. Evaluating In-Workflow Messages for Improving Mental Models of End-to-End Encryption. In *30th USENIX Security Symposium, USENIX Security 2021*. USENIX Association, 447–464.

[3] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. 2019. The double ratchet: security notions, proofs, and modularization for the signal protocol. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 129–158.

[4] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. 2020. Security analysis and improvements for the IETF MLS standard for group messaging. In *Annual International Cryptology Conference*. Springer, 248–277.

[5] Elaine Barker. 2020. *Recommendation for Key Management: Part 1 – General*. Technical Report. National Institute of Standards and Technology (NIST). https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf

[6] Dan Boneh, Sam Kim, and David J. Wu. 2017. Constrained Keys for Invertible Pseudorandom Functions. In *Theory of Cryptography - 15th International Conference, TCC 2017, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10677)*. Springer, 237–263.

[7] Nikita Borisov, Ian Goldberg, and Eric Brewer. 2004. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*. 77–84.

[8] Raphael Bost. 2016. ∑οφος: Forward Secure Searchable Encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1143–1154.

[9] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. 2017. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 1465–1482.

[10] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. [n. d.]. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014*. The Internet Society.

[11] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. 2013. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Annual cryptology conference*. Springer, 353–373.

[12] David Cash and Stefano Tessaro. 2014. The Locality of Searchable Symmetric Encryption. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8441)*. Springer, 351–368.

[13] Javad Ghareh Chamani, Dimitrios Papadopoulos, Mohammadamin Karbasforushan, and Ioannis Demertzis. 2022. Dynamic Searchable Encryption with Optimal Search in the Presence of Deletions. In *31st USENIX Security Symposium, USENIX Security 2022*. USENIX Association, 2425–2442.

[14] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. 2018. New Constructions for Forward and Backward Private Symmetric Searchable Encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*. ACM, 1038–1055.

[15] Zhao Chang, Dong Xie, Sheng Wang, and Feifei Li. 2022. Towards Practical Oblivious Join. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 803–817.

[16] Tianyang Chen, Peng Xu, Stjepan Picek, Bo Luo, Willy Susilo, Hai Jin, and Kaitai Liang. 2023. The Power of Bamboo: On the Post-Compromise Security for Searchable Symmetric Encryption. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023*. The Internet Society.

[17] Cisco Systems, Inc. 2025. mlspp: Implementation of Messaging Layer Security. https://github.com/cisco/mlspp. Accessed: 2025-08-08.

[18] Dwaine E. Clarke, Srinivas Devadas, Marten van Dijk, Blaise Gassend, and G. Edward Suh. 2003. Incremental Multiset Hash Functions and Their Application to Memory Integrity Checking. In *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings (Lecture Notes in Computer Science, Vol. 2894)*. Springer, 188–207.

[19] PCI Security Standards Council. 2024. *Payment Card Industry Data Security Standard: Requirements and Testing Procedures, v4.0.1*. Technical Report. PCI Security Standards Council. https://docs-prv.pcisecuritystandards.org/PCI%20DSS/Standard/PCI-DSS-v4_0_1.pdf

[20] Ningning Cui, Jianxin Li, Xiaochun Yang, Bin Wang, Mark Reynolds, and Yong Xiang. 2019. When Geo-Text Meets Security: Privacy-Preserving Boolean Spatial Keyword Queries. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*. IEEE, 1046–1057.

[21] Ioannis Demertzis, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2018. Searchable Encryption with Optimal Locality: Achieving Sublogarithmic Read Efficiency. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10991)*. Springer, 371–406.

[22] Ioannis Demertzis, Charalampos Papamanthou, and Rajdeep Talapatra. 2018. Efficient Searchable Encryption Through Compression. *Proc. VLDB Endow.* 11, 11 (2018), 1729–1741.

[23] Yevgeniy Dodis and Daniel Jost. 2024. Compact Key Storage in the Standard Model. In *Theory of Cryptography Conference*. Springer, 444–475.

[24] Yevgeniy Dodis, Daniel Jost, and Antonio Marcedone. 2024. Compact Key Storage - A Modern Approach to Key Backup and Delegation. In *Advances in Cryptology - CRYPTO 2024, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 14921)*. Springer, 75–109.

[25] Enron Corporation. 2004. Enron Email Dataset. https://www.cs.cmu.edu/~enron/. Accessed: 2025-10-07.

[26] Liang Guo, Jinwei Zhu, Jiayang Liu, and Kun Cheng. 2021. Full Encryption: An end to end encryption mechanism in GaussDB. *Proc. VLDB Endow.* 14, 12 (2021), 2811–2814.

[27] IETF MLS Working Group. 2024. MLS Interoperability Test Results. https://github.com/mlswg/mls-implementations. [Online; accessed 2025-10-18].

[28] Fabian Ising, Damian Poddebniak, Tobias Kappert, Christoph Saatjohann, and Sebastian Schinzel. 2023. Content-Type: multipart/oracle - Tapping into Format Oracles in Email End-to-End Encryption. In *32nd USENIX Security Symposium, USENIX Security 2023*. USENIX Association, 4175–4192.

[29] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. 2012. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 965–976.
[30] Brice Minaud and Michael Reichle. 2022. Dynamic Local Searchable Symmetric Encryption. In *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV (Lecture Notes in Computer Science, Vol. 13510)*. Springer, 91–120.
[31] Brice Minaud and Michael Reichle. 2023. Hermes: I/O-Efficient Forward-Secure Searchable Symmetric Encryption. In *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part VI (Lecture Notes in Computer Science, Vol. 14443)*. Springer, 263–294.
[32] Sikhar Patranabis and Debdeep Mukhopadhyay. 2020. Forward and Backward Private Conjunctive Searchable Symmetric Encryption. *IACR Cryptol. ePrint Arch.* (2020), 1342.
[33] Sikhar Patranabis and Debdeep Mukhopadhyay. 2021. Forward and Backward Private Conjunctive Searchable Symmetric Encryption. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021*. The Internet Society.
[34] Trevor Perrin and Moxie Marlinspike. 2016. The double ratchet algorithm. *GitHub wiki* 112, 4 (2016).
[35] The OpenSSL Project. 2025. OpenSSL 3.0.2: Cryptography and SSL/TLS Toolkit. https://github.com/openssl/openssl. Version 3.0.2, Accessed: 2025-08-08.
[36] Paul Wouters. 2023. *The Messaging Layer Security (MLS) Protocol*. Technical Report RFC 9420. IETF. Proposed Standard; specifies asynchronous group key establishment with forward secrecy and post-compromise security.
[37] Paul Wouters. 2025. *The Messaging Layer Security (MLS) Architecture*. Technical Report RFC 9750. IETF. Informational; describes architecture, deployment considerations, and security goals for MLS.
[38] Zikai Ye, Xiangyu Wang, Zesen Liu, Dan Zhu, and Jianfeng Ma. 2025. OBIR-tree: An Efficient Oblivious Index for Spatial Keyword Queries on Secure Enclaves. *Proc. ACM Manag. Data* 3, 1 (2025), 58:1–58:24.
[39] Dandan Yuan, Shujie Cui, and Giovanni Russello. 2022. We Can Make Mistakes: Fault-tolerant Forward Private Verifiable Dynamic Searchable Symmetric Encryption. In *7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022*. IEEE, 587–605.
[40] Jiaoyi Zhang, Liqiang Peng, Mo Sha, Weiran Liu, Xiang Li, Sheng Wang, Feifei Li, Mingyu Gao, and Huanchen Zhang. 2025. Femur: A Flexible Framework for Fast and Secure Querying from Public Key-Value Store. *Proc. ACM Manag. Data* 3, 3 (2025), 162:1–162:29.
[41] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2016. All your queries are belong to us: the power of {File-Injection} attacks on searchable encryption. In *25th USENIX Security Symposium (USENIX Security 16)*. 707–720.
[42] Zhongjun Zhang, Jianfeng Wang, Yunling Wang, Yaping Su, and Xiaofeng Chen. 2019. Towards Efficient Verifiable Forward Secure Searchable Symmetric Encryption. In *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11736)*. Springer, 304–321.

Received October 2025; revised January 2026; accepted February 2026