Threat-Driven Extensions to the Signal Protocol: Private Contact Discovery, Forward-Secure Media, and Ephemeral Key Management

Suraj Pandey
Zynclave Tech
suraj@zynclave.com

April 8, 2026
Version v0.0.1

This is a preprint. We welcome review and critique from the security community.

Abstract

We treat time as a first-class security boundary and enforce it across all cryptographic state, not just message keys. The Signal Protocol provides strong forward secrecy for message content via the Double Ratchet, but production deployments accumulate auxiliary state—contact discovery metadata, media encryption keys, and auxiliary cryptographic material—that persists outside the ratchet lifecycle with unbounded exposure windows. We argue that forward secrecy must extend to all auxiliary state. We operationalize this principle through three defenses unified by a single enforcement layer: (1) an OPRF-PSI contact discovery protocol on Ristretto255 that bounds discovery exposure to 24 hours, (2) ratchet-derived media keys with per-step erasure that bind media forward secrecy to the Double Ratchet chain, and (3) an Ephemeral Key Framework (EKF) that enforces temporal guarantees across all state classes. Our design enforces a single invariant: *no cryptographic material is both cross-domain and long-lived.* We implement all three extensions in a production messaging application and report +2.2% text message latency, +8.5% media overhead, and 92.3% auxiliary state reduction. The complete system adds approximately 7,400 lines of code across Rust and Swift.

1 Introduction

End-to-end encrypted (E2EE) messaging has achieved mainstream adoption through the Signal Protocol, which underpins Signal, WhatsApp, Google Messages, and other services reaching over two billion users [1, 2]. The protocol's Double Ratchet provides strong forward secrecy and post-compromise recovery for message content: each message is encrypted under a unique key, and chain keys are erased after advancement.

However, forward secrecy stops at message content. Production deployments accumulate *auxiliary cryptographic state* outside the ratchet's lifecycle: contact graphs, media encryption keys, presence metadata, and pre-key bundles. This auxiliary state persists with no formal lifecycle management. A compromise at time t exposes not only the current ratchet step but the entire history of auxiliary state since deployment.

Core thesis. We argue that forward secrecy must extend beyond message content to all auxiliary cryptographic state. We operationalize time as a security boundary through explicit expiration semantics enforced at the system level, not just message keys.

Consider three concrete threats that this gap enables:

T1: Contact discovery via hash upload. The standard approach—uploading SHA-256 hashes of phone numbers—is catastrophically weak. The E.164 phone number space contains approximately 10^{10} valid numbers; at SHA-256's throughput of ∼50M hashes/second on a consumer GPU, the entire space is enumerable in ∼200 seconds. A precomputed rainbow table costs ∼320 GB of storage. Server compromise yields the complete social graph of every user. Signal mitigates this via Intel SGX enclaves [17], which have been broken by multiple side-channel attacks [9, 10].

T2: Media keys outside the ratchet. Each media attachment is encrypted with a random AES-256 key stored

alongside message metadata. The Double Ratchet erases chain keys after advancement, but media keys are independent artifacts that persist on the device indefinitely. Device compromise at time t exposes *all historical media*—the ratchet's forward secrecy does not extend to attachments.

T3: Auxiliary state accumulation. Pre-key bundles, presence indicators, discovery sessions, and cached contact hashes accumulate without lifecycle management. At approximately 200 bytes/user/day, one million users produce $\sim$73 GB of unmanaged cryptographic material per year.

We present three defenses unified by a single enforcement layer:

1. **Defense 1 (D1):** An OPRF-PSI contact discovery protocol on Ristretto255 with a hybrid cached-hash negative filter, bounding discovery exposure to 24 hours (§3).
2. **Defense 2 (D2):** Ratchet-derived media keys via a parallel HKDF chain with device-local access key caching, bounding post-compromise media exposure to 30 days (§4).
3. **Defense 3 (D3):** The Ephemeral Key Framework (EKF), a temporal enforcement layer that makes D1 and D2's security guarantees hold by enforcing TTL-based lifecycle management across all auxiliary state classes (§5).

All three defenses are implemented in a production messaging application (Rust backend, native iOS client) and evaluated against real workloads.

The remainder of this paper is organized as follows. Section 2 defines our system and threat model. Sections 3–5 present the three defenses. Section 6 analyzes composed security properties. Section 7 reports implementation and performance results. Section 8 compares with existing systems. Section 9 discusses limitations and concludes.

2 System Model & Threat Model

2.1 Actors

Our system consists of five principals: *client devices* running the messaging application, a *messaging server* that routes encrypted messages, a *contact discovery service* that resolves phone numbers to registered users, a *media storage service* that stores encrypted attachments, and an *adversary*.

2.2 Trust Assumptions

- **Server:** Honest-but-curious. The server follows the protocol faithfully but may inspect any data it stores or processes.
- **Client devices:** Trusted until compromised, following the standard post-compromise security model.
- **Network:** Adversary-controlled (standard Dolev-Yao model).

2.3 Compromise Scenarios

We define three compromise scenarios with an explicit temporal model:

Scenario	Description
S1	**Server compromise at time t.** Read access to all server-side storage: encrypted auxiliary state, discovery metadata, media ciphertext references.
S2	**Device compromise at time t.** Full device state including secure enclave: message keys, AccessK entries, device-local secret, cached contacts.
S3	**Staggered cross-domain compromise.** Server at time t, device at time $t+\delta$. The adversary combines recovered state across trust domains and across time.

Table 1: Compromise scenarios.

Scenario S3—the *staggered cross-domain compromise*—is the critical case where traditional designs fail completely. Without our extensions, server breach yields MediaK (persists indefinitely), making subsequent device breach irrelevant—all media is already recoverable. With our extensions, server breach yields nothing device-local, and device breach yields only the unexpired AccessK window.

2.4 Threat Definitions

ID	Description	Scenario	Capability
T1	Contact enumeration	S1	Discovery service storage
T2	Media decryption	S2, S3	Device state at time t
T3	State accumulation	S1	Server auxiliary storage

Table 2: Threat definitions.

2.5 Security Goals

Goal	Description	Defense
G1	Contact discovery reveals no phone numbers to server	D1
G2	Device compromise exposes $\leq T_{\text{media}}$ days of media	D2
G3	Server compromise exposes $\leq$ max_ttl of auxiliary state	D3

Table 3: Security goals.

2.6 Out of Scope

Message content confidentiality (handled by the Signal Double Ratchet), key distribution (X3DH), group key management (Sender Keys), traffic analysis, and device security below the OS level are out of scope.

3 Defense 1: Private Contact Discovery

3.1 Attack Analysis

SHA-256 hash upload is the standard contact discovery mechanism [17]. The E.164 phone number space contains $\sim 10^{10}$ valid numbers. At SHA-256's throughput of $\sim$50M hashes/second on a consumer GPU, the entire space is enumerable in $\sim$200 seconds, and a precomputed rainbow table costs $\sim$320 GB. Server compromise under scenario S1 yields the complete social graph of every user.

Signal's mitigation relies on Intel SGX enclaves for Private Contact Discovery, but SGX has been broken by Foreshadow [9], Plundervolt [10], and AEPIC Leak, rendering hardware-enclave-based approaches unreliable as a sole defense.

3.2 Protocol: Hybrid Cached Hash + OPRF-PSI

We employ a two-layer design that combines a cached hash negative filter with a full oblivious pseudorandom function (OPRF) private set intersection (PSI) protocol.

3.2.1 Layer 1: Cached Hash (Negative Filter)

The first layer is a Bloom filter of $\text{SHA-256}(e164 \,\|\, daily\_salt)$, used *exclusively* for negative filtering: contacts confirmed as non-users are excluded from the OPRF path. The filter leaks only negative membership information within a 24-hour window, while all positive matches are resolved through the full OPRF path.

The daily salt is rotated via EKF's discovery class (24-hour TTL, §5). The privacy model is explicit: an observer learns that certain numbers are *not* on the platform within a 24-hour window. This is a strictly smaller information leak than the positive-membership leak of hash-based discovery. The asymmetry is intentional—negative information (knowing someone is *not* a user) has lower privacy impact than positive information (knowing they *are*).

3.2.2 Layer 2: Full OPRF-PSI (Privacy Path)

For all contacts not eliminated by the negative filter, we execute a full OPRF-PSI protocol on Ristretto255:

1. Client generates random blinding scalar r_i per contact.
2. Client computes blinded point: $B_i = r_i \cdot H(e164_i)$ via hash-to-curve and scalar multiplication.
3. Client sends $\{B_1, \ldots, B_n\}$ to server.
4. Server computes $R_i = k \cdot B_i$ using server secret k, returns $\{R_1, \ldots, R_n\}$.
5. Client unblinds: $T_i = r_i^{-1} \cdot R_i = k \cdot H(e164_i)$.
6. Client compares T_i against the server's set of $k \cdot H(registered\_e164)$ values (transmitted as set intersection).

Security. The server sees only random Ristretto255 points (blinded). Under the Decisional Diffie-Hellman (DDH) assumption on Ristretto255, blinded points are computationally indistinguishable from random group elements.

3.2.3 Production Optimizations

- **Batch processing:** Vectorized scalar multiplication for 500+ contacts.
- **Bloom filter pre-filtering:** Skip OPRF for contacts already confirmed as non-users.
- **Background execution:** OPRF runs asynchronously; the UI shows cached results immediately.

3.3 Security Analysis

Theorem 1 (OPRF Obliviousness) *Under the DDH assumption on Ristretto255 in the random oracle model, the OPRF-PSI protocol does not enable the server to distinguish queried non-intersecting elements from random, excluding leakage from protocol metadata (batch cardinality, request timing, intersection size).*

[Reduction sketch] Given a DDH distinguisher D, construct an adversary A that breaks the OPRF's obliviousness. A embeds the DDH challenge into the blinding operation: if (g^a, g^b, g^c) is a DDH tuple, A uses g^a as the blinded point and g^c as the server response. If $c = ab$ (DDH tuple), unblinding produces a valid OPRF output; if c is random, the output is random. A's advantage equals D's advantage, so OPRF obliviousness holds under DDH.

4 Defense 2: Forward-Secure Media Encryption

4.1 Attack Analysis

In Signal's current design, each media attachment is encrypted with a random AES-256 key stored alongside message metadata. The Double Ratchet erases chain keys after advancement, but media keys are independent artifacts that persist on the device indefinitely. Device compromise at time t exposes all historical media—the ratchet's forward secrecy does not extend to attachments.

4.2 Protocol: Ratchet-Derived Media Keys

We bind media encryption keys to a parallel HKDF chain that advances in lockstep with the conversation:

$$\text{MediaK}_n = \text{HKDF}(\text{CK}_n,\ \text{file\_hash},\ \texttt{"media-v1"}) \tag{1}$$

where CK_n is the parallel media chain key at step n (erased after advancement), file_hash = SHA-256(plaintext) provides per-file uniqueness, and `"media-v1"` is the domain separation label.

4.2.1 Why a Parallel Chain

Binding media derivation to the message ratchet directly would couple reliability failures (upload retries, network timeouts) to cryptographic state progression, violating the ratchet's monotonicity assumptions. Media upload is asynchronous and may span multiple ratchet steps; a failed upload should not advance or block the message chain. The parallel chain maintains independence between message delivery and media upload lifecycles while inheriting equivalent forward secrecy properties via HKDF chaining.

4.2.2 Encryption

AES-256-GCM with MediaK_n as the key, chunked at 1 MB blocks. Encrypted ciphertext is uploaded to media storage. MediaK_n is wrapped under the recipient's current ratchet key and transmitted as message metadata.

4.3 The Re-Access Problem

After ratchet advancement, CK_n is erased. The recipient cannot re-derive MediaK_n to re-open previously received media.

4.3.1 Solution: Device-Local Access Key Cache

$$\text{AccessK}_i = \text{HKDF}(\text{MediaK}_n,\ \textit{dls},\ \texttt{"access-v1-<id>"}) \tag{2}$$

where *dls* is a 256-bit device-local secret generated at application installation, stored in the device's secure enclave, and never transmitted. AccessK_i is stored exclusively in the device's secure storage and enables re-decryption of the specific media file without retaining MediaK_n. EKF enforces a 30-day TTL on AccessK_i entries (§5).

4.4 Core Invariant

Definition 1 (Cross-Domain Persistence Invariant) *No cryptographic material is both* cross-domain *(accessible from both server and device trust domains) and* persistent *(surviving beyond a protocol step or its assigned TTL).*

MediaK_n is cross-domain (transits the server inside Signal-encrypted messages) but not persistent (erased at ratchet step). AccessK_i is persistent (up to 30 days) but not cross-domain (never leaves the device). No key violates both constraints. This invariant is what makes the design resilient to staggered cross-domain compromise (S3).

4.5 Trust Domain Separation Analysis

MediaK_n transits the server; AccessK_i never does. This eliminates server-side recoverability of historical media keys—a qualitative shift from the baseline design. We focus on cryptographic key exposure; metadata leakage (access patterns, file sizes, ciphertext correlation) is out of scope.

Server compromise alone (S1): Yields encrypted message ciphertext containing wrapped MediaK_n, but no device-local secret to derive AccessK_i. The server gains nothing beyond what the ratchet already protects.

Device compromise alone (S2): Yields only the unexpired AccessK_i entries (bounded to $\leq$30 days by EKF).

Compound compromise (S3): Without our design, server breach at time t recovers MediaK_n for all media ever sent, making later device breach irrelevant. With our design, server breach yields no device-local key material; device breach at $t + \delta$ yields only AccessK entries that have not TTL-expired. Blast radius reduces from "all media ever" to "device-local + time-bounded subset."

4.6 Security Claims

Claim 1 (Ratchet Binding) *$MediaK_n$ inherits forward secrecy from CK_n. Compromise of CK_{n+1} reveals nothing about $MediaK_n$.*

Claim 2 (Device Isolation) *$AccessK_i$ is derived from dls. Compromise of one device's $AccessK_i$ reveals nothing about other devices' access keys for the same media.*

Claim 3 (Bounded Re-Access) *EKF's 30-day TTL on the media class ensures $AccessK_i$ is erased after 30 days. Post-compromise media exposure is bounded to 30 days.*

Claim 4 (Trust Domain Separation) *Server-side breach alone cannot recover $AccessK_i$ for any media file, regardless of the time window.*

4.7 Comparison with Baseline

Property	Baseline	Ours
Media key lifetime	Indefinite	Erased at ratchet step
Re-access mechanism	Retained random key	Device-local AccessK (30-day TTL)
Post-compromise exposure	All historical media	$\leq$30 days
Per-file uniqueness	Yes (random)	Yes (file_hash binding)
Ratchet binding	None	HKDF from CK_n

Table 4: Media encryption: baseline vs. our design.

5 Ephemeral Key Framework

EKF provides a uniform abstraction for temporal scoping of cryptographic state across heterogeneous subsystems. It is not an independent defense parallel to D1 and D2—it is the enforcement mechanism that makes their temporal security guarantees hold. Without EKF, D1's Bloom filter salt never rotates (the 24-hour bound becomes unbounded) and D2's AccessK never expires (the 30-day bound becomes indefinite).

5.1 Attack Analysis: Auxiliary State Accumulation

Production E2EE deployments accumulate auxiliary cryptographic state outside the ratchet: contact discovery sessions, media access keys, presence metadata, pre-key bundles. Without explicit lifecycle management, this state persists indefinitely. Server compromise at time t exposes all auxiliary state since deployment.

At $\sim$200 bytes/user/day, one million users over one year produce $\sim$73 GB of unmanaged cryptographic material.

5.2 Design: Unified Lifecycle Management

Every piece of auxiliary cryptographic state is assigned a structured lifecycle:

```
EphemeralEntry {
   material: bytes
   class: KeyClass // {discovery, media,
                   // presence, prekey}
   created_at: timestamp
   ttl: duration
   policy: ExpPolicy // {Rotate, Delete,
                   // Overwrite}
}
```

5.2.1 Key Classes and TTLs

Class	Material	TTL	Policy
discovery	OPRF state, hashes	24 h	Rotate
media	AccessK entries	30 d	Delete
presence	Typing/read state	5 min	Overwrite
prekey	One-time pre-keys	7 d	Replenish

Table 5: EKF key classes.

5.2.2 Expiration Policies

- **Rotate:** Replace with freshly derived equivalent; zeroize old value. For continuous-availability material.
- **Delete:** Destroy with no replacement. For cacheable material that can be rebuilt.
- **Overwrite:** Replace with null sentinel (not delete). Prevents absence-of-record from leaking information.

5.3 Server-Side Enforcement

EKF runs as a server-side garbage collection layer with client callbacks:

```
lifecycle_tick(current_time):
  for entry in auxiliary_store:
    if current_time - entry.created_at >
        entry.ttl:
      match entry.policy:
        Rotate: notify_client("
            rotation_needed")
```

```
            schedule_deletion(grace=1h
                )
    Delete: secure_erase(entry)
    Overwrite: entry.material =
        NULL_SENTINEL
            entry.created_at =
                current_time
```

Critical property: The server never decrypts auxiliary material to enforce TTLs. Lifecycle management operates purely on metadata (timestamps, class tags, policy flags).

5.4 EKF as Enforcement Layer for D1 and D2

- **D1 depends on EKF:** The 24-hour bound on Bloom filter exposure (§3) holds only because EKF's discovery class forces daily salt rotation. Without EKF, the salt persists indefinitely and the cached hash layer degrades to the same rainbow table vulnerability as baseline hash upload.
- **D2 depends on EKF:** The 30-day bound on post-compromise media re-access (§4) holds only because EKF's media class enforces TTL expiration on AccessK_i entries. Without EKF, AccessK persists indefinitely and the trust domain separation argument collapses under device compromise.
- **Dependency direction:** D1 and D2 provide cryptographic mechanisms (OPRF blinding, HKDF chain derivation). EKF provides the temporal enforcement that converts those mechanisms into bounded security guarantees.

5.5 Security Claims

Claim 5 (Temporal Bounding) *EKF limits auxiliary state exposure from $O(\text{deployment\_lifetime})$ to $O(\text{max\_ttl})$ per class.*

Claim 6 (Overwrite Indistinguishability—Cryptographic) *After overwrite-policy execution, the stored ciphertext is statistically independent of the original key material, assuming HKDF output is indistinguishable from random. The null-sentinel overwrite ensures the stored bytes carry no information about the prior key.*

Claim 7 (Overwrite Indistinguishability—Systems) *The cryptographic claim (Claim 6) holds assuming complete overwrite at the storage layer. In append-oriented storage engines (e.g., LSM-tree-based stores), the overwrite guarantee depends on compaction completing within the policy window. Operators should configure compaction intervals to align with EKF TTLs. Side channels (timing of overwrite operations, metadata about entry existence) are not covered by the cryptographic claim and represent residual systems-level leakage.*

Claim 8 (Rotation Preservation) *EKF rotation preserves security properties of underlying protocols: fresh blinding factors are unlinkable under DDH; media re-derivation inherits ratchet forward secrecy.*

6 Composed Security Analysis

6.1 Temporal Compromise Analysis

Table 6 shows the exposure under device compromise at time t, with and without our extensions.

For the compound scenario (S3: server at t, device at $t+\delta$), server-side breach alone yields no device-local key material under our design. The adversary must independently compromise the device to obtain AccessK_i entries, and even then recovers only the unexpired subset.

6.2 Unified State/Domain/Exposure Table

Table 7 unifies all cryptographic state across the system, mapping each item to its trust domain, lifetime, and exposure under each compromise scenario. The core invariant (Definition 1) is enforced visually: no row is simultaneously "cross-domain" and "persistent."

6.3 Composition Model

The three defenses share state surfaces through four interfaces:

1. Parallel media chain key (D2 derivation, EKF rotation scheduling).
2. Ristretto255 scalars (D1 OPRF, EKF discovery rotation).
3. HKDF instantiation (D1, D2, EKF—all domain-separated).
4. Server auxiliary store (D1 cached hashes, D2 media references, EKF lifecycle metadata).

6.4 Domain Separation

All HKDF derivations use prefix-free domain separation labels:

$$\text{MediaK}_n = \text{HKDF}(\text{CK}_n, \text{file\_hash}, \texttt{"media-v1"})$$
$$\text{AccessK}_i = \text{HKDF}(\text{MediaK}_n, \textit{dls}, \texttt{"access-v1"})$$
$$\text{CachedHash} = \text{HKDF}(\text{SHA256}(\textit{e164}), \textit{salt}, \texttt{"disc-v1"})$$
$$\text{RotationKey} = \text{HKDF}(\textit{fresh}, \textit{epoch}, \texttt{"rotate-v1"})$$

Data Class	Baseline (compromise at t)	Ours (compromise at t)
Messages after t	Protected (ratchet)	Protected (ratchet)
Messages before t	Exposed (if stored)	Exposed (if stored)
Media keys	All historical MediaK exposed	MediaK erased at ratchet step; $\leq$30 d AccessK window
Contact graph	Full graph via hash inversion	OPRF state: 24 h window only; Bloom: 24 h salt
Presence metadata	All historical presence	$\leq$5 min window (EKF overwrite)
Pre-key bundles	All uploaded pre-keys	$\leq$7 d window (EKF replenish)

Table 6: Temporal compromise analysis: exposure with and without our extensions.

State	Trust Domain	Lifetime	S1 (Server)	S2 (Device)	S3 (Compound)
Message chain keys	Device	Per-step	None	Current step	Current step
MediaK$_n$	Cross-domain	Per-step	Encrypted only	Erased	Erased
AccessK$_i$	Device-only	$\leq$30 d (EKF)	None	$\leq$30 d window	$\leq$30 d window
Bloom filter salt	Server-only	$\leq$24 h (EKF)	$\leq$24 h	N/A	$\leq$24 h
OPRF blinding factors	Device-only	Session	None	Session only	Session only
OPRF server secret	Server-only	Weekly rotation	Current key	N/A	Current key
Presence state	Server-only	$\leq$5 min (EKF)	$\leq$5 min	N/A	$\leq$5 min
Pre-key bundles	Server-only	$\leq$7 d (EKF)	$\leq$7 d	N/A	$\leq$7 d

Table 7: Unified cryptographic state map. No entry is simultaneously cross-domain and persistent, enforcing the core invariant (Definition 1).

Theorem 2 (Domain Separation) *Given distinct info labels, HKDF outputs are computationally independent under the random oracle model [6]. Our info labels are prefix-free, satisfying the distinctness requirement.*

6.5 Composed Forward Secrecy

Data Type	Forward Secrecy Window
Messages	Per-message (immediate)
Media keys	Per-ratchet-step (immediate)
Media re-access	30 days (EKF bounded)
Contact graph	24 hours (EKF bounded)
Presence metadata	5 minutes (EKF bounded)
Pre-keys	7 days (EKF bounded)

Table 8: Forward secrecy windows by data type.

6.6 Threat Coverage Matrix

Table 9 maps each threat to the defenses that address it, including the staggered cross-domain scenario (S3).

Threat	Sc.	D1	D2	EKF	Residual
T1	S1	✓	–	✓	$\leq$24 h Bloom
T2	S2	–	✓	✓	$\leq$30 d AccessK
T2	S3	–	✓	✓	No device-local keys from server
T3	S1	✓	✓	✓	$\leq$30 d max

Table 9: Threat coverage matrix.

6.7 Failure Mode Analysis

No single-component failure cascades below baseline Signal guarantees:

- **OPRF unavailable:** Falls back to cached hash (reduced privacy, still functional).
- **EKF tick delayed:** Auxiliary state persists beyond TTL (security bound extends proportionally; no catastrophic failure).
- **Device secret lost:** AccessK$_i$ irrecoverable (re-request via ratchet channel).

- **Chain key desync:** Direct sender-encrypted media key transmission.

7 Implementation & Performance

7.1 Implementation Overview

Component	Language	LOC
OPRF-PSI service	Rust	∼2,200
Media key derivation	Rust + Swift	∼1,400
EKF lifecycle manager	Rust	∼1,800
Client key management	Swift	∼1,100
Protocol integration	Rust	∼900
Total		**∼7,400**

Table 10: Implementation breakdown.

The backend runs on a managed Kubernetes cluster (3 nodes) with a wide-column persistent store and gRPC microservices. The client is a native iOS application using CryptoKit for cryptographic operations and Keychain for secure storage.

7.2 OPRF-PSI Benchmarks

Benchmarks were conducted on an x86-64 server (2.8 GHz, single core) and a modern iOS device with an A17-class SoC.

Operation	p50	p99
Single OPRF eval	0.12 ms	0.18 ms
Batch 500 contacts	48 ms	62 ms
Bloom filter check	0.002 ms	0.004 ms
Full discovery (cold)	52 ms	71 ms
Full discovery (warm)	6 ms	11 ms

Table 11: Server-side OPRF-PSI latency.

Client-side: blind 500 contacts = 34 ms, unblind 500 = 31 ms, full round-trip = 180 ms. Compared to baseline hash upload, the OPRF path introduces 173× server computation overhead, but the absolute cost is 52 ms—acceptable for a background operation. The warm cache reduces this to 6 ms.

7.3 Media Encryption Benchmarks

Storage overhead: ∼304 KB per 1,000 media files (0.03% of media size).

Operation	Latency
MediaK derivation (server)	0.003 ms
AccessK derivation (client)	0.008 ms
AES-256-GCM 1 MB (server)	0.4 ms
AES-256-GCM 1 MB (client)	0.6 ms
Full 10 MB image e2e	320 ms

Table 12: Media encryption latency. Baseline 10 MB e2e = 290 ms (+10.3%).

7.4 EKF Benchmarks

Benchmarks simulated one million users.

Operation	p50	p99
EKF tick (scan + expire)	12 ms	34 ms
Rotation notification	2 ms	8 ms
Presence overwrite (10K)	18 ms	45 ms

Table 13: EKF lifecycle management latency (1M users).

Auxiliary state reduction over 90 days: 18.2 GB → 1.4 GB (92.3% reduction).

7.5 System-Level Impact

Scenario	Baseline	Ours	Δ
Text message	45 ms	46 ms	+2.2%
5 MB image	410 ms	445 ms	+8.5%
Contact sync (cold)	15 ms	195 ms	+180 ms
Contact sync (warm)	15 ms	21 ms	+40%
Memory (idle)	42 MB	44 MB	+4.8%

Table 14: End-to-end system impact.

8 Comparison with Existing Systems

We compare against Signal (production), WhatsApp, Matrix/Element (Megolm), and SEAL [4].

8.1 Contact Discovery

Signal relies on SGX enclaves broken by Foreshadow [9] and Plundervolt [10]. SEAL provides full OPRF but no messaging integration. Our hybrid design achieves practical warm-path latency (6 ms) while resolving all positive matches through the full OPRF path.

Property	Signal	WA	Matrix	SEAL	Ours
Server learns IDs	Yes	Yes	Yes	No	No
Rainbow resistant	No	No	N/A	Yes	Yes*
Warm sync (ms)	<1	<1	<1	∼60	6

Table 15: Contact discovery comparison. *Partial (Bloom) + full (OPRF).

8.2 Media Forward Secrecy

Property	Signal	WA	Matrix	Ours
Ratchet-bound media	No	No	No	Yes
Post-comp. exposure	∞	∞	Session	≤30 d

Table 16: Media forward secrecy comparison.

No existing system binds media keys to the ratchet lifecycle.

8.3 Auxiliary State Lifecycle

Property	Signal	WA	Matrix	Ours
Formal lifecycle	No	No	Partial	Yes
Null-sentinel indist.	No	No	No	Yes
Max exposure	∞	∞	∞	30 d

Table 17: Auxiliary state lifecycle comparison.

No production system provides formal auxiliary state lifecycle management.

8.4 Cost-Benefit Summary

Combined overhead for text messaging (+2.2%) and warm contact sync (+6 ms) is negligible. Media adds +8.5%. The one-time cold sync adds 180 ms. These costs are well within the acceptable range for production deployment.

9 Limitations, Future Work, and Conclusion

9.1 Limitations

L1 Negative-filter leakage. The cached hash layer leaks negative membership information, bounded by 24-hour rotation via EKF. All positive matches are resolved through the full OPRF path.

L2 Policy-driven TTL. The 30-day AccessK TTL is a policy choice, not a cryptographic bound. It is configurable but not formally optimized.

L3 Single OPRF key. The single server secret is a central trust bottleneck, mitigated by weekly rotation and HSM storage. Threshold OPRF constructions (e.g., Jarecki et al. [3]) could distribute trust across multiple servers, but introduce multi-party coordination, honest-majority assumptions, and multi-round latency overhead that we leave to future work.

L4 No machine-checked proofs. We provide semi-formal reductions only; Tamarin verification is deferred.

L5 Single-platform benchmarks. iOS only; Android performance may differ by 20–40%.

L6 Storage-layer dependence. EKF overwrite indistinguishability depends on storage-layer compaction (Claim 7). LSM-tree engines may retain pre-compaction state.

9.2 Future Work

F1 Threshold OPRF: (t,n)-shared server secret for distributed trust (see L3 for feasibility discussion).

F2 Formal verification: Tamarin prover models, prioritizing EKF temporal claims and OPRF composition.

F3 Group media forward secrecy: Binding to Sender Key ratchet.

F4 Adaptive TTL policies: Threat-level and usage-aware expiration (e.g., shorter TTLs under elevated threat).

F5 Cross-device AccessK synchronization: Extending trust domain separation to multi-device models.

9.3 Conclusion

We extend the Signal Protocol's forward secrecy guarantees from message content to all auxiliary cryptographic state. We operationalize time as a security boundary

through explicit expiration semantics enforced at the system level—not as an aspirational property but as a mechanism with concrete TTLs, policies, and enforcement loops. This principle unifies three otherwise independent defenses: OPRF-PSI bounds contact discovery exposure to 24 hours, ratchet-derived media keys erase media access at each chain step, and the Ephemeral Key Framework enforces these temporal guarantees as a system-wide discipline.

A single invariant underlies the design: *no cryptographic material is simultaneously accessible across trust domains and persistent across time.* Forward secrecy is not a property of a single mechanism, but of how a system constrains the lifetime and scope of its secrets. The Double Ratchet advances time for message keys; our extensions advance time for everything else.

References

[1] M. Marlinspike and T. Perrin. The Double Ratchet algorithm. Signal Foundation, 2016. https://signal.org/docs/specifications/doubleratchet/

[2] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the Signal messaging protocol. *Journal of Cryptology*, 33(4):1914–1983, 2020.

[3] S. Jarecki, A. Kiayias, and H. Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In *ASIACRYPT*, pages 233–264, 2018.

[4] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert. Mobile private contact discovery at scale. In *USENIX Security Symposium*, pages 1447–1464, 2019.

[5] M. Chase and P. Miao. Private set intersection in the Internet setting from lightweight oblivious PRF. In *CRYPTO*, pages 34–63, 2020.

[6] H. Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In *CRYPTO*, pages 631–648, 2010.

[7] D. J. Bernstein. Curve25519: New Diffie-Hellman speed records. In *PKC*, pages 207–228, 2006.

[8] F. Hao. Schnorr non-interactive zero-knowledge proof. RFC 8235, 2017.

[9] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Gras, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *USENIX Security Symposium*, pages 991–1008, 2018.

[10] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens. Plundervolt: Software-based fault injection attacks against Intel SGX. In *IEEE S&P*, pages 1466–1482, 2020.

[11] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith. SoK: Secure messaging. In *IEEE S&P*, pages 232–249, 2015.

[12] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In *ACM CCS*, pages 1773–1788, 2017.

[13] R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara, and K. Cohn-Gordon. The Messaging Layer Security (MLS) protocol. RFC 9420, 2023.

[14] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.

[15] C. G. Günther. An identity-based key-exchange protocol. In *EUROCRYPT*, pages 29–37, 1990.

[16] E. Barker. Recommendation for key management: Part 1 – General. NIST Special Publication 800-57 Part 1, Revision 5, 2020.

[17] M. Marlinspike. Technology preview: Private contact discovery for Signal. Signal Blog, 2017. https://signal.org/blog/private-contact-discovery/

[18] H. de Valence, J. Grigg, G. Tankersley, F. Valsorda, and I. Lovecruft. Ristretto255 and Decaf448. Internet-Draft, IETF CFRG, 2023.

[19] P. Borrello, A. Kogler, M. Schwarzl, M. Lipp, D. Gruss, and M. Schwarz. ÆPIC Leak: Architecturally leaking uninitialized data from the microarchitecture. In *USENIX Security Symposium*, pages 3917–3934, 2022.

[20] Matrix.org Foundation. Megolm encryption. https://spec.matrix.org/latest/client-server-api/#end-to-end-encryption

[21] Signal Foundation. Sender Keys for group messaging. Signal Technical Documentation, 2020.

[22] M. Marlinspike and T. Perrin. The X3DH key agreement protocol. Signal Foundation, 2016. https://signal.org/docs/specifications/x3dh/

[23] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use PGP. In *ACM Workshop on Privacy in the Electronic Society*, pages 77–84, 2004.

[24] T. Perrin and M. Marlinspike. The Noise Protocol Framework. 2016. https://noiseprotocol.org/

[25] S. Meier, B. Schmidt, C. Cremers, and D. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *CAV*, pages 696–701, 2013.