privitty

Privitty Whitepaper

Cryptographic Control in a Zero-Trust Communication World

info@privittytech.com
www.privittytech.com

Table of Contents

About Privitty

Modern high-security environments rely on encrypted communication systems to protect sensitive data. However, traditional end-to-end encryption (E2EE) only secures data in transit, leaving a critical gap once data is delivered. Messages, files, and intelligence can still be copied, forwarded, or persist indefinitely on compromised devices.

Privitty addresses this fundamental limitation by transforming encryption into an enforceable control mechanism. It introduces a cryptographic framework where access policies—such as user identity, device binding, expiry, and revocation—are embedded directly into encrypted data objects.

Unlike conventional systems, Privitty ensures that data remains protected even after delivery. Decryption is only possible when all cryptographic policy conditions are satisfied, eliminating unauthorized access, persistent exposure, and uncontrolled dissemination.

The platform combines decentralized architecture, dynamic key derivation, and optional multi-party access using secret sharing. It supports secure messaging, file sharing, and API-based data exchange across mobile, desktop, and enterprise systems.

By enforcing control at the cryptographic layer rather than relying on application logic, Privitty delivers defense-grade security with verifiable data lifecycle control, making it ideal for tactical, strategic, and inter-agency communication.

Privitty transforms encryption from passive protection into active, enforceable control.

Encryption is Not Enough

Encryption today solves only part of the security problem.

- Once decrypted:
- Data can be copied or forwarded
- Screenshots and external capture are possible
- Files persist indefinitely on endpoints
- Compromised devices expose historical data

This creates a "**last-mile security gap**", where:

Data is secure in transit, but uncontrolled at rest on endpoints.

In defense and critical infrastructure environments, this leads to:

- Intelligence leakage
- Unauthorized data propagation
- Loss of chain-of-custody
- Long-term breach exposure

70 % Data Breach

Human Errors

Slip-ups

Wrong Forward

Intentional

privitty

Weakness of Existing Secure Messaging Systems

Even advanced E2EE platforms suffer from structural limitations:

1. No Post-Delivery Control

- Encryption ends at decryption. No control over:
- Forwarding
- Retention
- Access duration

2. Application-Level Enforcement

Policies (if any) are enforced via UI, not cryptography → easily bypassed.

3. Centralized Identity Dependencies

- Phone numbers / emails required
- Metadata exposure risk
- Not suitable for sovereign or defense deployments

4. Lack of Auditability

No verifiable tracking of:

- Who accessed data
- When it was accessed
- How it propagated

The Privitty Approach

Privitty is a **decentralized, end-to-end encrypted (E2EE) messaging and collaboration platform** with **cryptographic access control** designed to protect sensitive information not only in transit, but also **after delivery** ("last-mile privacy"). The system is **edge-first**: user data and keys stay on user devices; infrastructure is used only as a **transport medium** and does not store user keys or plaintext.

Privitty is available as:

- **Messaging Apps**: Android, iOS, Desktop (Linux, Windows, macOS)
- **SDK**: Integrates into business workflows (secure file sharing, field ops, incident response, internal collaboration portals, document distribution)

Core Principle

Access to data is not granted by possession, but by satisfying cryptographic policy conditions.

Key Capabilities

- Policy-bound encryption
- Time-based and condition-based access
- Cryptographic revocation
- Device-bound decryption
- Controlled forwarding

Outcome

Even if ciphertext is obtained:

- It cannot be decrypted without valid policy compliance

Technical Architecture & Approach

Edge-first, decentralized design

- **Endpoints are the trust anchor**: encryption, decryption, key management, access-control enforcement, and policy state all live on user devices.
- **Transport is untrusted by design**: the network is treated purely as a delivery substrate; it does not hold plaintext, does not escrow keys, and does not participate in authorization decisions.
- **No centralized identity dependency**: the system does **not require phone numbers, email addresses, or any PII** for account creation or messaging.

Key establishment and trust bootstrap (Autocrypt + SecureJoin)

Privitty uses two complementary mechanisms to establish cryptographic trust in a decentralized environment:

Autocrypt (opportunistic key discovery + upgrade path)

- Enables **automatic key dissemination** between peers via normal message exchange, so secure communication can begin with minimal friction.
- Supports a practical path from "first contact" to strong cryptographic protection without requiring a central directory or phone-number-based identity.

SecureJoin (interactive authentication / anti-MITM onboarding)

- Provides a **high-assurance, user-verifiable handshake** (e.g., QR / short authentication strings) to defeat active attackers and establish authenticated trust.
- Enables device pairing and contact verification without exposing PII.

Together:

- Autocrypt makes decentralized secure messaging **deployable at scale** (low friction).
- SecureJoin provides **defense-grade assurance** when threat models require authenticated key verification.

Forward-looking cryptography: Autocrypt 2.0 and post-quantum readiness

- Privitty is architected to adopt **Autocrypt 2.0**, which introduces a modernized cryptographic baseline and an upgrade path toward **post-quantum security**.
- This ensures the system can evolve from "classical" E2EE into **quantum-resilient** secure messaging without centralizing trust or rewriting the entire transport layer.

Core components

- **Client Apps (Android/iOS/Desktop)**: secure UX, local storage, policy indicators, secure viewer integration.
- **Cryptographic core engine** (shared): object encryption/decryption, Autocrypt/SecureJoin handling, policy enforcement, protocol control messages, event emission.
- **Local encrypted database:** stores access-control state machine, cryptographic metadata, and policy constraints.
- **Transport adapters:** pluggable delivery backends (store-and-forward), treated as untrusted.

Innovation - What's Unique

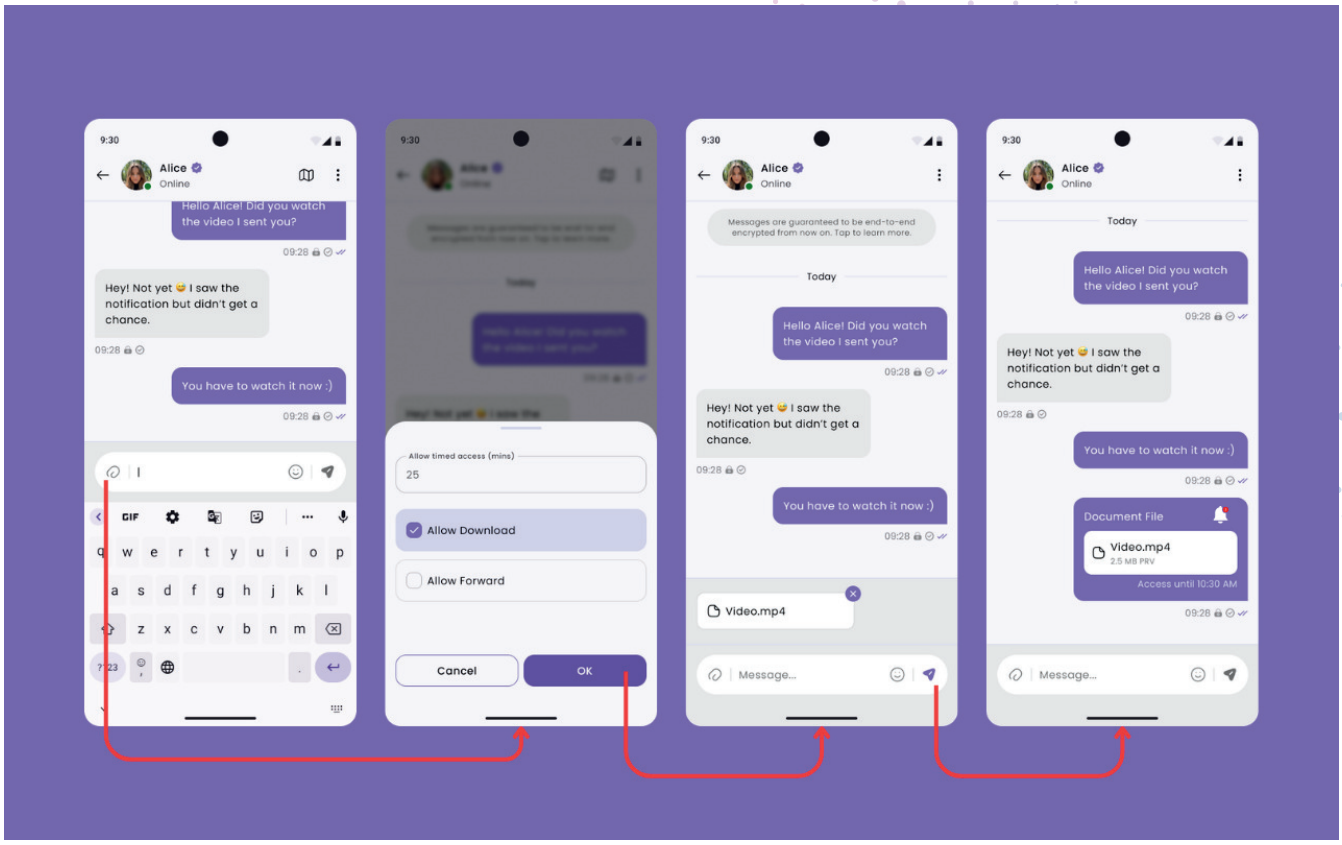
Cryptographic access control for last-mile privacy

Privitty extends beyond "classic E2EE" by adding a **cryptographic authorization layer**:

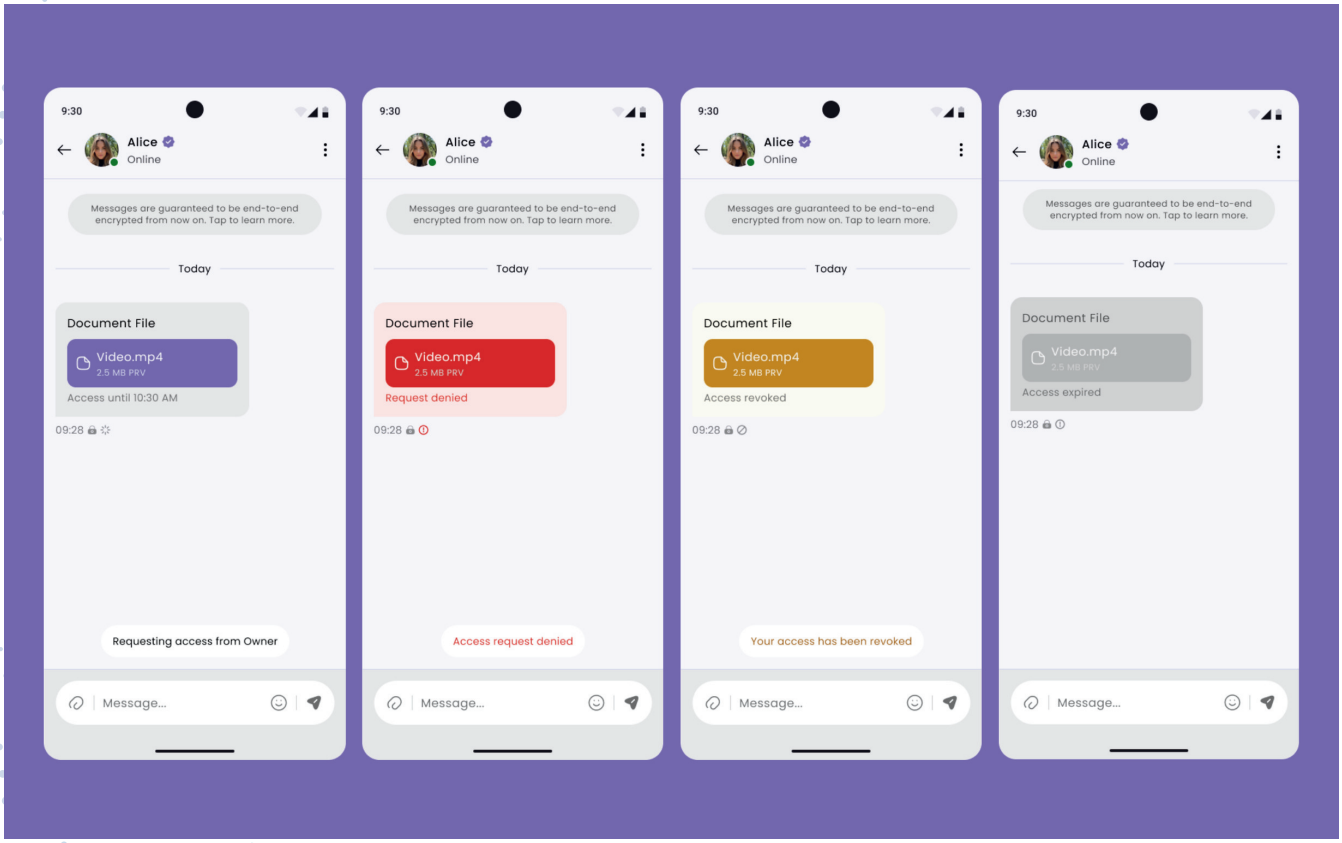
- Recipients may receive ciphertext, but **cannot decrypt** unless the device has the required cryptographic grants and policy conditions are satisfied.
- Access is governed by:

  1. **time windows** (duration-based access)
  2. **absolute expiry**
  3. **download / retention permissions**
  4. **forwarding permissions**

- Policy is enforced at the cryptographic layer, not merely by UI rules.
- Privitty provides cryptographic auditability with full message traceability–who accessed or forwarded data–enabling a verifiable chain of custody and dashboard-level visibility across units.

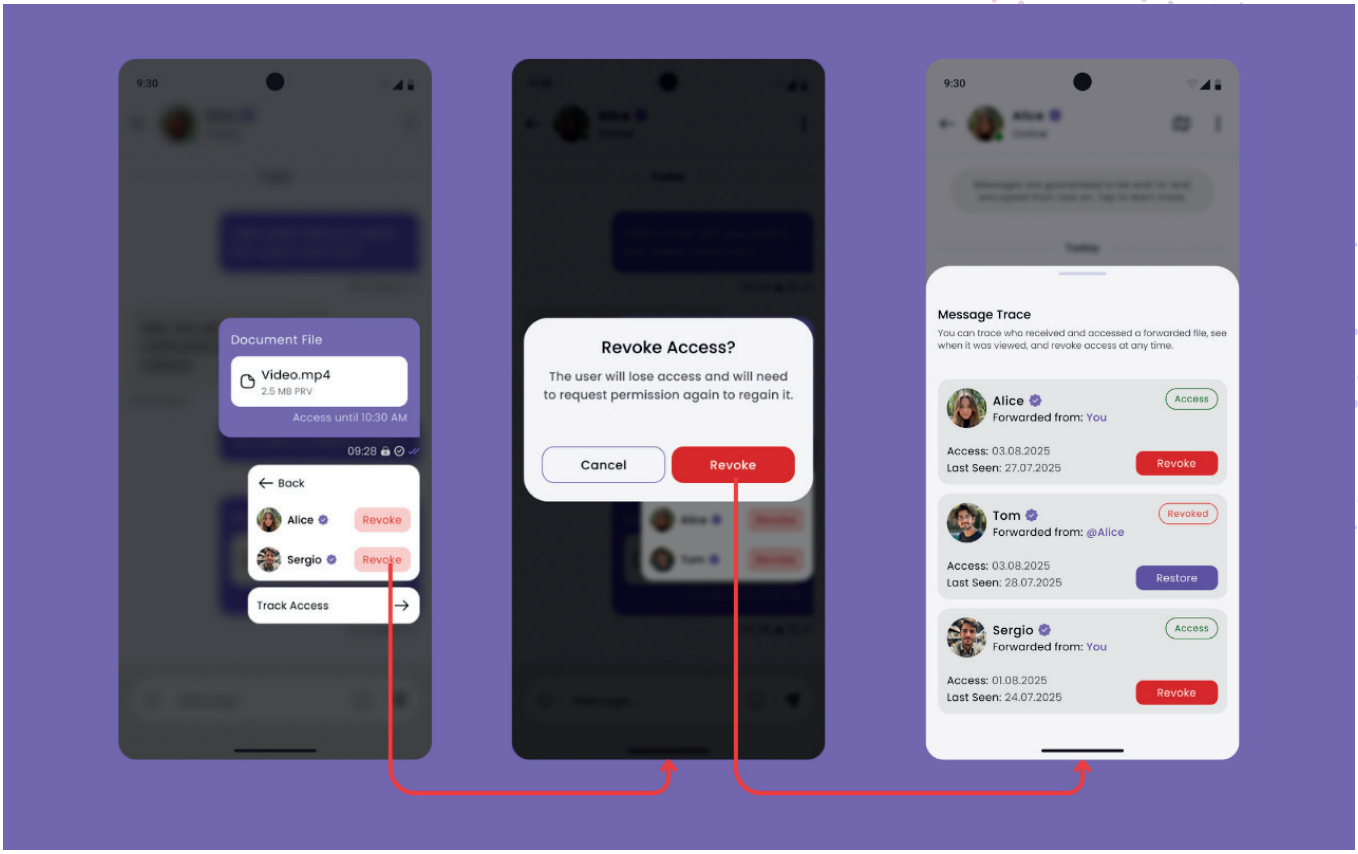


Sharing file with peer with complete control



Access control at various phases

privitty



Revoke and message tracing

privitty

Patented access-control approach

Privitty has **patent coverage** in the access-control encryption space (for controlled access to encrypted content in decentralized messaging/collaboration scenarios). This provides defensible differentiation and signals novelty beyond commodity messaging encryption.

Implementation & Integration

Single security core, deployed across:

- Android, iOS, Desktop apps
- The core SDK is implemented in Rust, leveraging memory-safe guarantees to prevent entire classes of vulnerabilities such as buffer overflows and memory corruption
- SDK targets (mobile/desktop/embedded enterprise apps)

SDK integration enables:

- secure object encryption for sharing inside existing workflows
- controlled grants/revocation policies
- event-driven status updates (granted/expired/revoked/decryptable)

Scalable and resilient because:

- computation is edge-local
- transport is stateless and interchangeable
- metadata and policy state are stored locally per device and synchronized through protocol messages when needed

Use Cases

Financial Systems: Controlled Document Exchange

Sensitive documents (KYC, loan files, claims) are shared via email/portals, leading to:

- Uncontrolled downloads and forwarding
- Persistent exposure of PII
- Lack of auditability

Privitty Enforcement:

- Policy-bound encrypted documents
- View-only access (no download/forward)
- Time-bound expiry + true revocation

Enterprise Workflows: Secure Document Distribution

Email/cloud sharing provides no control after delivery:

- No revocation
- No access tracking
- Uncontrolled duplication

Privitty Enforcement:

- Cryptographic access control embedded in documents
- Full traceability of access events - Perfect compliance

Defense & Tactical Communication

Mission-critical data requires:

- Time-bound access
- Strict “need-to-know” enforcement
- Controlled dissemination

Privitty Enforcement:

- Policy-bound encrypted payloads
- Device + identity-bound decryption
- Automatic expiry aligned with mission windows
- Optional multi-party authorization

Cross-Use Case Insight

Capability	Traditional Systems	Privitty
Post-delivery control	✗	✓
Cryptographic enforcement	✗	✓
Revocation	✗	✓
Auditability	Limited	Strong

Conclusion

The future of secure communication demands more than encryption–it requires control.

Privitty redefines secure messaging by embedding enforcement directly into cryptography, ensuring that sensitive data remains protected throughout its lifecycle.

In an era of increasing cyber threats, device compromise, and data proliferation, Privitty provides a fundamentally new security primitive:

"Data that enforces its own protection "

Thank you!

privitty

info@privittytech.com
www.privittytech.com