TELOSIS-RP-2026-005 · PRIVACY & SECURITY · JULY 2026 · 9 SECTIONS, 7 MIN READ

Privacy-Preserving Architectures for Modern Infrastructure

TELOSIS Research

Privacy is often treated as a compliance requirement - a checklist of controls to satisfy auditors. This paper argues that privacy is an architectural property, not a compliance artifact. We present four patterns for privacy-preserving infrastructure: encryption at rest with customer-managed keys, encryption in transit with forward secrecy, zero-access architectures where the provider cannot access user data, and metadata minimization to reduce the surface area of surveillance. We also address the limits of technical privacy: what architecture cannot protect, and where law, policy, and user behavior must fill the gap.

privacy encryption zero-access architecture metadata minimization data protection

Design patterns for data protection. Encryption at rest and in transit. Zero-access architectures. Metadata minimization. The limits of technical privacy.

1. Introduction

Most software collects data by default and restricts access through policy. The policy says “we will not look at your data.” The architecture says “we can look at your data, but we promise not to.”

This is not privacy. It is a promise. Promises can be broken. They can be changed when the company is acquired. They can be overridden by a court order. They can be violated by a rogue employee or an attacker who compromises the production database.

Privacy-preserving architecture inverts this relationship. The architecture says “we cannot look at your data.” The policy is a description of the architecture, not a substitute for it.

2. Pattern 1: Encryption at Rest with Customer-Managed Keys

2.1 The Standard Model

Data is encrypted at rest using a provider-managed key. The provider holds the key. The provider can decrypt the data at any time.

Trust model: The user trusts the provider not to access their data and not to lose the key.

2.2 Customer-Managed Keys

The user provides the encryption key. The provider never holds the key in plaintext. Data is encrypted before it reaches the provider’s storage. Decryption requires the user’s key.

Trust model: The user trusts the provider to store ciphertext reliably. The user does not need to trust the provider with plaintext access.

2.3 Implementation

- Key generation: Client-side, in the user's browser or application.
- Key storage: User-managed. Could be a hardware security module, a password manager, or a paper backup.
- Encryption: AES-256-GCM. Performed client-side before data transmission.
- Server role: Receives and stores ciphertext. Never sees plaintext. Cannot decrypt.

2.4 Limitations

- Key loss: If the user loses their key, data is permanently unrecoverable. No password reset.
- Search: Server-side search of encrypted data is impossible without specialized techniques (homomorphic encryption, searchable encryption) that are not yet practical for most applications.
- Metadata: Encryption protects content, not metadata. The server still knows when the user connected, how much data they stored, and who they communicated with.

3. Pattern 2: Encryption in Transit with Forward Secrecy

3.1 The Standard Model

TLS encrypts data between client and server. But if the server's private key is compromised, all past sessions can be decrypted if they were recorded.

3.2 Forward Secrecy

Forward secrecy ensures that compromising the server's long-term key does not compromise past sessions. Each session uses an ephemeral key pair. The private key is discarded after the session ends.

Implementation: TLS 1.3 with ephemeral Diffie-Hellman key exchange. Mandatory for all connections.

3.3 Certificate Transparency

Certificate Transparency (CT) logs publicly record every TLS certificate issued. This allows domain owners to detect unauthorized certificates issued for their domains - whether by a compromised certificate authority or a government actor.

Implementation: Monitor CT logs for thetelosis.com, codecx.app, and all subdomains. Alert on any certificate not issued by your team.

4. Pattern 3: Zero-Access Architecture

4.1 Definition

A zero-access architecture means the provider has no technical capability to access user plaintext data. This is stronger than a policy against access. It is a structural guarantee.

4.2 Design Principles

1. **Client-side encryption:** Data is encrypted before it leaves the user's device. The server receives ciphertext.

2. **No key escrow:** The provider never holds user encryption keys. No "key recovery" process. No "we can help if you lose your password."

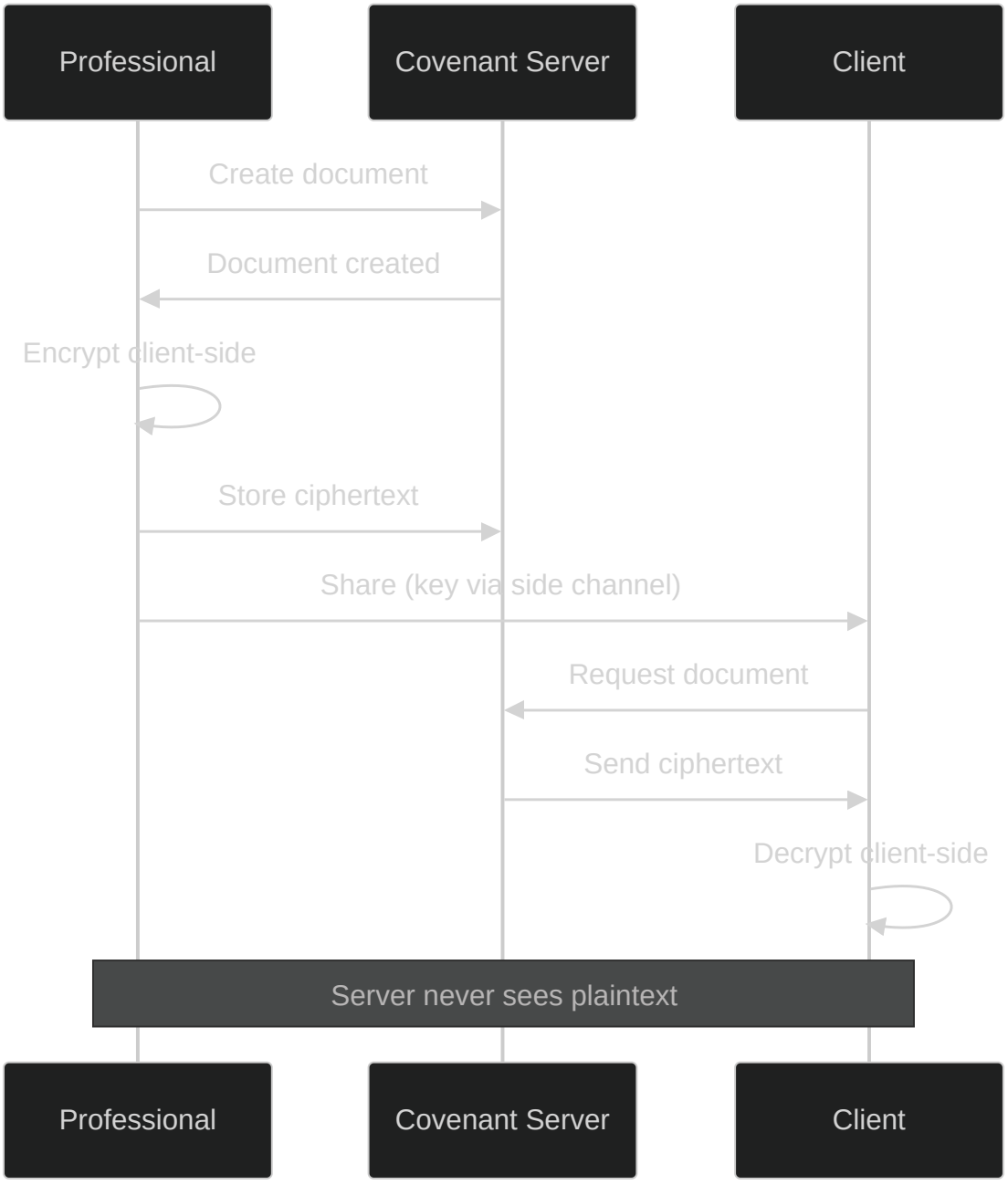
3. **Integrity verification:** Users can verify that the server is serving the correct data. Merkle trees or similar structures allow the client to detect tampering.

4. **Open source:** The client code is open source. Users can verify that encryption happens as claimed.

4.3 Example: Covenant

Covenant, a client experience platform, implements zero-access for document content. Agreement text, signature evidence, and invoice details are encrypted client-side before transmission. The Covenant server stores ciphertext. The server can serve documents, manage sharing, and track status - but it cannot read document contents.

When a professional shares a document with a client, the professional's client software encrypts the document with a key shared between the two parties. The server relays the ciphertext. The server never possesses the decryption key.



5. Pattern 4: Metadata Minimization

5.1 The Problem

Encryption protects content. It does not protect metadata. Metadata is everything other than the content: who communicated with whom, when, for

how long, from what IP address, using what device.

Metadata is often more revealing than content. A list of who someone emails reveals their professional and personal relationships. A log of when they connect reveals their sleep schedule. A record of what documents they view reveals their interests, concerns, and priorities.

5.2 Minimization Strategies

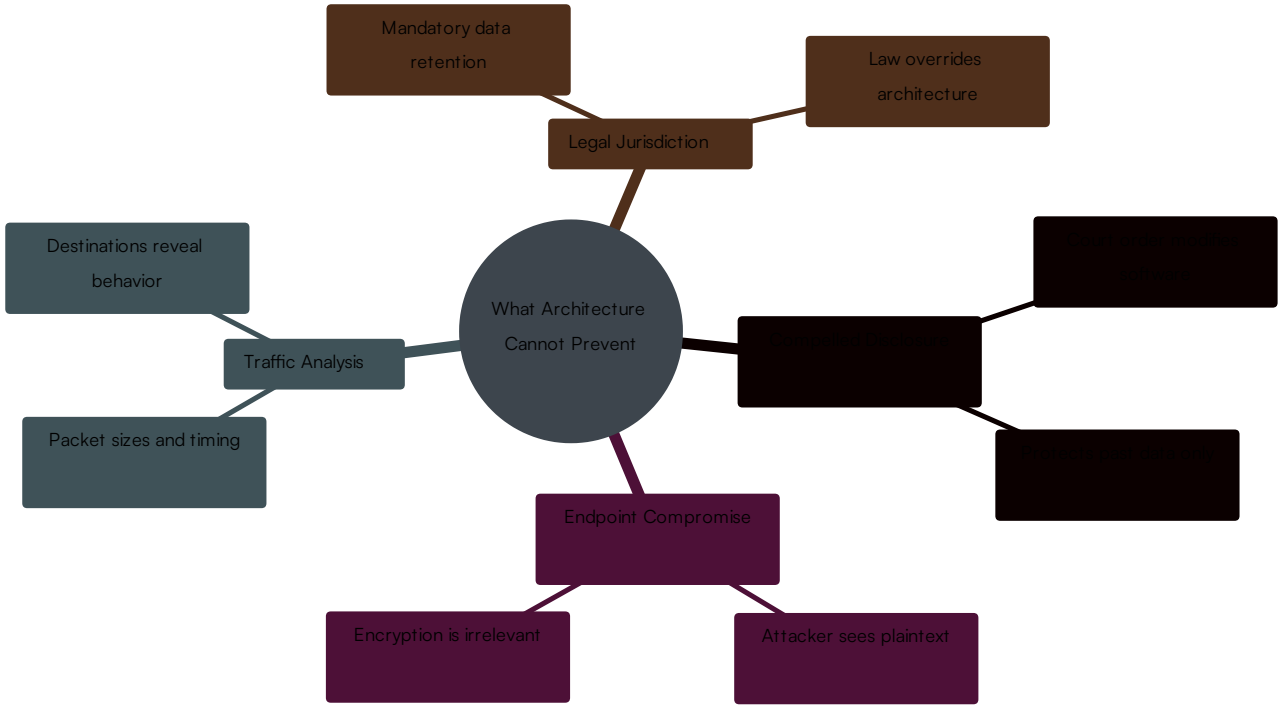
STRATEGY	IMPLEMENTATION
No persistent IP logging	Log IP addresses only in memory for rate limiting. Never write to disk.
Connection aggregation	Batch connections through a single persistent channel. Server sees one connection, not hundreds.
Request batching	Send multiple logical requests as one physical request. Server sees one timestamp, not many.
Data minimization	Do not collect what you do not need. If a feature does not require metadata, do not log it.
Retention limits	Delete metadata after its operational purpose expires. Seven-day log retention, not seven-year.

5.3 What Cannot Be Hidden

Some metadata is inherent to network communication. The server must know the client's IP address to send a response. The server must know when a request arrives to process it. Minimization reduces the surface area. It does not eliminate it.

6. The Limits of Technical Privacy

Architecture can protect against many threats. It cannot protect against all of them.



6.1 What Architecture Cannot Prevent

THREAT	WHY ARCHITECTURE FAILS
Compelled disclosure	A court order can require the provider to modify the software to capture data going forward. Architecture protects past data, not future data.
Endpoint compromise	If the user's device is compromised, encryption is irrelevant. The attacker sees plaintext before encryption.
Traffic analysis	Even with encrypted content and minimized metadata, an observer can analyze packet sizes, timing, and destinations to infer behavior.

THREAT	WHY ARCHITECTURE FAILS
Legal jurisdiction	Data stored in a jurisdiction with mandatory data retention laws cannot be protected by architecture alone. The law overrides the architecture.

6.2 The Role of Policy

Where architecture ends, policy begins. Privacy requires both.

- **Transparency reports:** Publish requests for user data received from governments. Users should know how often their data is demanded.
- **Warrant canaries:** A statement that the provider has not received a secret court order. The statement is removed if an order is received. Its absence signals the order.
- **Jurisdiction selection:** Incorporate and host infrastructure in jurisdictions with strong privacy protections. Avoid jurisdictions with mandatory key disclosure or data retention.

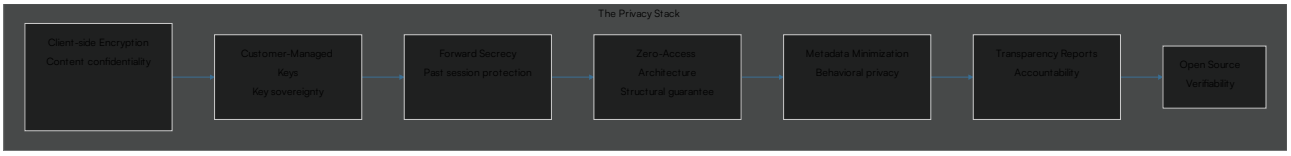
7. The Privacy Stack

Privacy is not a single decision. It is a stack of decisions, each protecting against a different class of threat.

LAYER	PROTECTION	THREAT ADDRESSED
Client-side encryption	Content confidentiality	Provider access, server breach
Customer-managed keys	Key sovereignty	Provider coercion, key theft

LAYER	PROTECTION	THREAT ADDRESSED
Forward secrecy	Past session protection	Future key compromise
Zero-access architecture	Structural guarantee	Policy changes, acquisition
Metadata minimization	Behavioral privacy	Traffic analysis, surveillance
Transparency reports	Accountability	Secret government requests
Open source	Verifiability	Dishonest claims about privacy

No single layer is sufficient. All are necessary.



8. Conclusion

Privacy is not a policy. It is not a compliance checkbox. It is not a promise in a terms of service document.

Privacy is an architectural property. It is enforced by the code, guaranteed by the infrastructure, and verifiable by the user.

The patterns described in this paper - client-side encryption, customer-managed keys, forward secrecy, zero-access architecture, metadata minimization - are not novel. They are standard cryptographic techniques. What is rare is the commitment to implement them as defaults, not as premium features.

The software industry has accepted surveillance as the cost of convenience. It is not. It is a choice. And the alternative is not harder to build. It is harder to monetize. That is the real resistance. And that is precisely why it matters.

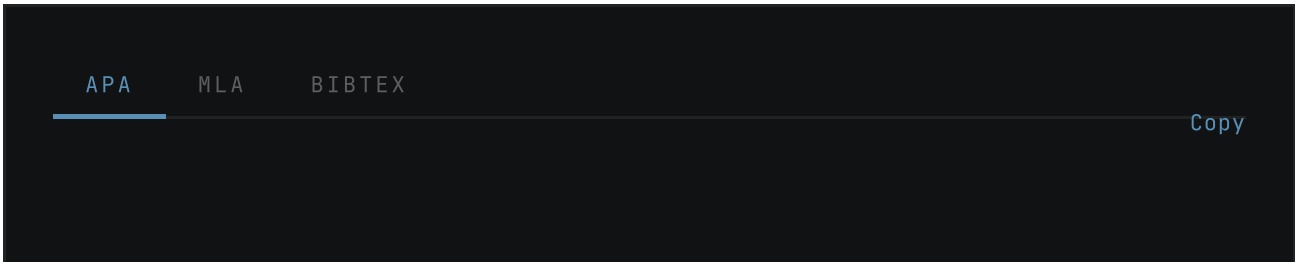
References

1. Rogaway, P. *The Moral Character of Cryptographic Work.* 2015.
2. TELOSIS Research. *The Case Against Multi-Tenant Architecture for Privacy Systems.* TELOSIS-RP-2026-004, 2026.
3. CODECX Engineering. *Self-Hosting Is Not a Feature - It's Infrastructure.* CODECX Journal, 2026.
4. IETF. *RFC 8446: TLS 1.3.* 2018.

RELATED

TELOSIS-RP-2026-004: The Case Against Multi-Tenant Architecture for Privacy Systems

CITATION



TELOSIS Research. (2026). Privacy-Preserving Architectures for Modern Infrastructure. TELOSIS-RP-2026-005.

← RP-2026-007: Exportability as a Structural Property

RP-2026-004: The Case Against Multi-Tenant Architecture for Privacy Systems →