EDGEVAULT

EdgeVault whitepaper

ABSTRACT

Client-Carried, Authenticated-Encrypted Context for Stateless LLM Chat in Anova

Backyard Bandwidth

Abstract

EdgeVault is the context protection and transport mechanism used by Anova to support multi-turn LLM chat without storing conversation history server side for continuity.

Instead of persisting chat logs or chat state in a database, Anova uses a client carried encrypted context capsule. The client stores this capsule locally and submits it with each message. The server decrypts the capsule only for the duration of the request to reconstruct context for the model, then returns an updated capsule for the client to store.

EdgeVault uses AES-GCM (Authenticated Encryption with Associated Data, AEAD) to provide:

- Confidentiality: the capsule contents cannot be read without the encryption key.
- Integrity: modifications to the capsule are detected and rejected.

This document focuses on describing the mechanism precisely: token format, cryptographic operations, request lifecycle, and the resulting stateless server behavior.

1. Goals and non-goals

1.1 Goals

EdgeVault is designed to:

1. Enable stateless chat continuity. Conversation continuity must not rely on server stored chat history. The server should be able to answer a request using only the request payload (prompt + capsule) and the server's configured encryption secret.
2. Protect context contents in application space. Even though transport (HTTPS) already encrypts traffic, EdgeVault ensures the context payload is encrypted as a distinct application layer artifact.
3. Detect tampering. If a client or network attacker modifies a capsule, the system must detect the modification and refuse to use it as valid context.

1.2 Non-goals

EdgeVault does not claim:

- End-to-end encryption against the server. The server must decrypt context to construct model input. The goal is "no server-side stored history," not "server cannot see plaintext during processing."
- Replay prevention by itself. A valid older capsule can be resent and will be valid as context for the request.

2. Key idea: Client-carried encrypted context

Most chat systems keep conversation state server side (database rows, cache entries, conversation IDs). EdgeVault replaces that model:

- The client stores the conversation context as an encrypted token in the browsers localStorage.
- Each time the user sends a message, the client includes this encrypted context token in the request. If there is no capsule the server assumes that there is no context and starts fresh.
- The server decrypts it, generates a response, encrypts the updated context, returns the new token, and it is stored in localStorage.

This creates a closed loop in which the server does not need to persist conversation context to support multi-turn interaction. This provides Anova with chat context and removes the possibility of chat leaks, database leaks, or server attacks. There is nothing stored, so it is not possible to leak what's not there.

3. Token format: EdgeVault capsule

EdgeVault encodes encrypted context into a single string:

EdgeVault_<version>.<nonce_b64url>.<tag_b64url>.<ciphertext_b64url>

3.1 Fields

- Prefix: EdgeVault: Identifies the token as EdgeVault with a format version.
- nonce_b64url: 12-byte random nonce, URL-safe Base64 encoded
  Used as the AES-GCM nonce (also called IV).
- tag_b64url: authentication tag, URL-safe Base64 encoded. Used to verify the ciphertext has not been altered and that the correct key & nonce were used.
- ciphertext_b64url: encrypted bytes of the serialized context, URL-safe Base64 encoded. Contains the encrypted JSON representation of the conversation context.

3.2 Why URL-safe Base64

The token is stored in browser storage and transported in web requests. URL-safe Base64 ensures the token is safe in form posts and storage without encoding conflicts.

4. Cryptography: AES-GCM (Authenticated Encryption)

EdgeVault uses AES in GCM mode:

- AES is a symmetric block cipher.
- GCM combines counter-mode encryption with a polynomial authenticator (GHASH) to produce an integrity tag.

AES-GCM is an AEAD scheme, meaning it provides both:

- Encryption (confidentiality) and
- Authentication (integrity / tamper detection)

4.1 Security properties relied upon

Under correct use, AES-GCM provides:

- Confidentiality: Without the key, ciphertext does not reveal plaintext.
- Integrity: Changes to nonce/tag/ciphertext are detected during verification.
- Forgery resistance: Without the key, an attacker cannot produce a valid (ciphertext, tag) pair for a chosen plaintext.

4.2 The critical rule: nonce uniqueness

AES-GCM requires that the same (key, nonce) pair is never reused. EdgeVault generates a new random 12-byte nonce for every encryption. A 96-bit random nonce has extremely low collision probability at typical system volumes.

5. EdgeVault encryption and decryption flow

What gets protected

EdgeVault protects the conversation context (the structured chat history/state that the model needs to respond coherently). The context is treated as a JSON compatible data structure (commonly a list of message objects or tokens), then converted into a text representation for encryption.

A) Creating an EdgeVault capsule (encryption)

Inputs

- Conversation context: a JSON serializable structure representing the current chat state. If the provided data is not JSON serializable it assumed that nothing exists.

Process

1. Obtain the session encryption secret

- When the service starts, it creates (or loads) a high-entropy secret used as the encryption key.
- This secret is kept on the server (for example in environment/config) and is used to encrypt and decrypt context capsules.

2. Serialize the context

- The context structure is converted into a JSON string.
- That JSON string is then converted into bytes (UTF-8).
  This produces the plaintext that will be encrypted.

3. Generate a unique nonce (12 bytes)

- A fresh, cryptographically secure random nonce (96-bit) is generated for this encryption.
- This nonce is not a secret, but it must be unique per encryption under the same key for AES-GCM security.

4. Encrypt using AES-GCM

- The plaintext bytes are encrypted with AES in GCM mode, using the session secret and the nonce.
- AES-GCM produces two outputs:

    - Ciphertext (the encrypted context)
    - Authentication tag (integrity proof)

5. Package into a transportable token

- The nonce, tag, and ciphertext are encoded using URL-safe Base64.
- They are assembled into a single “capsule” string with a version prefix:

EdgeVault_<version>.<nonce>.<tag>.<ciphertext>

Output

- A single EdgeVault capsule string containing:
    - the nonce (Base64 URL safe)
    - the authentication tag (Base64 URL safe)
    - the ciphertext (Base64 URL safe)

What the authentication tag guarantees

The tag cryptographically binds the ciphertext to:

- the encryption key, and
- the nonce

If any part of the capsule is altered (even one character), the tag verification will fail during decryption, and the plaintext will not be accepted, this will make it so that the server defaults to no context for the response.

B) Opening an EdgeVault capsule (decryption)

Inputs

- EdgeVault capsule: a string in the form EdgeVault_<version>.<nonce>.<tag>.<ciphertext>

Process

1. Obtain the same session encryption secret

- The server loads the same encryption secret used to create the capsule.

2. Parse the capsule

- The capsule is split into its components:

    - version marker
    - nonce
    - tag
    - ciphertext

3. Decode the components

- The nonce, tag, and ciphertext are Base64 URL safe decoded back into raw bytes.

4. Decrypt and verify in one step (AES-GCM)

- Using the decoded nonce and the server secret, AES-GCM attempts to:
    - decrypt the ciphertext, and
    - verify the authentication tag

This verification step is critical:

- If the key is wrong, or the nonce/tag/ciphertext were modified, corrupted, truncated, or malformed, verification fails.
- On verification failure, the capsule is treated as invalid and no context is supplied to the LLM.

5. Deserialize back into structured context

- If verification succeeds, the plaintext bytes are decoded as UTF-8 and parsed from JSON back into the original context structure. This is then passed to the LLM request as the context for that prompt.

6. Apply size limits

- If the reconstructed context exceeds the maximum allowed size, it is truncated to a safe bound (e.g., keeping only the first 2000 entries once a maximum threshold is exceeded).
  This prevents uncontrolled growth and resource exhaustion, while keeping context within a set window.

Output

- The reconstructed context object (typically a list of messages), or
- an empty context if the capsule cannot be verified or decoded safely.

Tamper behavior (what happens if someone edits the capsule)

Any modification to the capsule causes AES-GCM verification to fail. The system then rejects the capsule and proceeds as if there is no prior context, rather than accepting attacker controlled content.

C) How this supports “no server side stored conversations”

EdgeVault enables stateless chat continuity because:

- The client keeps the latest EdgeVault capsule locally (in browser storage).
- Each request sends the capsule back to the server alongside the new user prompt.
- The server reconstructs context only from what the client supplied, then returns a new capsule.

Conversation continuity is maintained without storing chat history in a server side database. The encrypted context is kept on the client and provided with each request, so a server breach does not expose a stored archive of encrypted conversations to decrypt.

6. Client storage and request lifecycle (stateless chat flow)

6.1 Client storage location

The client stores the EdgeVault capsule in localStorage:

- Key: anova_ctx
- Value: the EdgeVault string token

6.2 Request behavior

When a message is sent:

- The client reads anova_ctx.
- The client submits a request to the chat endpoint containing:
    - prompt
    - username
    - authenticated token
    - context (EdgeVault token, possibly empty for new chat)

6.3 Response behavior

The server response includes:

- answer (assistant reply)
- context (a new EdgeVault token containing updated conversation state)

The client replaces its stored anova_ctx with this returned token.

6.4 Why this is stateless server-side

Conversation continuity is achieved without the server needing to store a conversation record, because:

- The entire conversation state required for continuity is provided on each request as context.
- The server reconstructs state from the decrypted token each time.
- If the server process restarts or a different worker handles the next message, chat continuity still works because the client resends the capsule.

In other words: the “memory” is carried in-band by the client, not stored server-side for future retrieval.

7. What “we do not store conversations server side” means

The EdgeVault design supports the following statement:

- Conversation continuity does not depend on server side persistence of conversation history. The server does not need to store chat logs or chat history in a database to continue a conversation, because the client supplies the encrypted context each turn.

To avoid confusion, the following clarifications are also true:

- The server necessarily decrypts context in memory to generate a reply (the model needs plaintext input).
- The plaintext context is therefore visible to the server process transiently during request handling.
- The mechanism described here is about not retaining conversation history server-side as durable state for chat continuity.

8. Integrity, confidentiality, and failure modes

8.1 Integrity and tamper detection

AES-GCM authentication means:

- If any bit of nonce/tag/ciphertext is modified, verification fails.
- Decryption returns an empty list if any exception occurs.
- Modified capsules are not accepted as valid context.

This prevents a client or attacker from altering earlier conversation messages inside the capsule without detection.

8.2 Confidentiality

The capsule contains ciphertext of the JSON context. Without the server's key:

- The ciphertext cannot be decrypted into readable conversation content.

8.3 Failure handling behavior

Decryption returns an empty list on failure. Common causes:

- Token is malformed (wrong number of fields).
- Token is truncated or corrupted in storage/transit.
- Wrong key configured (key mismatch).
- Token has been tampered with.
- Base64 decoding fails.

Effect:

- The system treats the conversation as having no prior context for that request.

9. Context bounding and size control

The decrypt function enforces a maximum length check:

- By default the max context length is 4,000 tokens.
- If the context is greater than the max length, it is truncated to the newest 2,000 tokens.

This is intended to prevent unbounded growth of context state (e.g., extremely long sessions or maliciously large context objects). The bounding logic operates on the length of the deserialized object (e.g., number of items in a list), not the number of LLM tokens.

10. Frequently asked questions

Q1: Is the conversation stored server-side?
Conversation continuity does not rely on a server-side conversation database. The client keeps the encrypted context capsule locally and includes it with each request. Because the server does not maintain a stored archive of chat history for continuity, the risk of conversations being exposed through a database leak or accidental retention is significantly reduced.

Q2: Can the server read the conversation?
Yes—during request processing. The server must decrypt the capsule to reconstruct context and supply it to the model. EdgeVault is intended to prevent persistent storage of conversation history, not to prevent the service from accessing plaintext while generating a response. The decrypted context exists transiently in memory during processing and is not meant to be written to disk as part of the continuity mechanism.

Q3: What prevents a user from forging or editing their context?
EdgeVault uses AES-GCM authenticated encryption. Any change to the nonce, tag, or ciphertext causes authentication to fail during decryption. When authentication fails, the capsule is rejected and the system proceeds with an empty context rather than accepting modified or attacker-controlled context.

Q4: What is the nonce, and why does it exist?
The nonce is a unique, random value used by AES-GCM for each encryption. Using a fresh nonce prevents the same plaintext from producing the same ciphertext and is required for AES-GCM security. EdgeVault generates a new 12-byte nonce for every context update, making nonce collisions extremely unlikely in practice.

Q5: What happens if the token is corrupted?
If the capsule is malformed, truncated, tampered with, or otherwise fails authentication, decryption fails and the system uses an empty context. The assistant then responds without prior conversational state for that request, effectively treating it as a new session context.

Q6: Does this replace HTTPS?
No. HTTPS secures transport between client and server. EdgeVault adds application-layer protection for the context payload and enables stateless continuity by keeping conversation state client-side rather than in a server-side history store.

Q7: Can an old token be resent (replay)?
Yes. With EdgeVault alone, replaying a previously valid capsule replays an older

conversation state. EdgeVault provides confidentiality and integrity, not freshness guarantees. Users can reset state by clearing their local context (for example, using the “Clear” action in the interface).

Q8: Why Base64 URL-safe encoding?
Nonce, tag, and ciphertext are binary data. URL-safe Base64 converts them into a transport- and storage-safe text form suitable for browser storage and web request payloads without issues from reserved or unsafe characters.

11. Summary

EdgeVault enables Anova's stateless chat by packaging the entire conversation state into a client-stored, encrypted context capsule. Instead of keeping chat history in a server-side database, the client holds the latest capsule locally and includes it with every message. Each request therefore arrives with everything needed to continue the conversation: the user's new prompt plus the most recent encrypted context.

On the server, the capsule is decrypted only for the duration of the request to reconstruct the current context in a structured form (the same structure used to build the model's input). The server then generates the next assistant response, updates the context to reflect the new turn, and immediately re-encrypts the updated state into a fresh capsule. That new capsule is returned in the response and replaces the client's previously stored capsule, becoming the single source of continuity for the next request.

The cryptographic layer uses AES-GCM authenticated encryption, which provides two critical guarantees for the capsule: confidentiality (the context cannot be read without the encryption secret) and tamper detection (any modification to the capsule causes authentication to fail, preventing altered context from being accepted). At the architectural level, this design ensures that conversation continuity is maintained without relying on server-side stored conversation history. If a server instance restarts or requests are routed to different workers, the conversation can still continue because the client supplies the encrypted state each time, rather than the server needing to retrieve it from a persistent chat store.