PeerAuth Extension Security & Privacy Analysis (v0.2.5)

This document analyzes the PeerAuth browser extension for generating proofs of bank transactions.

> **Note:** PeerAuth is **closed source** - this analysis was performed on compiled/minified JavaScript bundles (`extension/`), not source code. Some implementation details may be incomplete or inferred from observed behavior.

Table of Contents

Executive Summary

PeerAuth v0.2.5 is a browser extension that generates cryptographic proofs of bank transactions using the Reclaim Protocol. It supports 18+ payment providers and allows users to prove they made a payment without revealing all transaction details.

Critical Privacy/Security Findings:

Finding	Severity
Attestor sees ALL your bank data (balance, transactions, personal info)	CRITICAL
Selective disclosure is trust-based, not cryptographic	CRITICAL
PostHog analytics tracks wallet addresses and proof activity	HIGH
Extension has access to ALL websites (`https://*/*`)	HIGH
Content script injected on every HTTPS page	HIGH
Single centralized attestor (no self-hosting option)	MEDIUM

Motivation

I'm writing this report out of ❤️ , not to complain. I've been waiting for a product like this for a long time.

What zkp2p enables is genuinely exciting. It allows instant on-ramp and off-ramp between fiat and crypto. Here's how it works: a buyer wants USDC, so they send money to a seller via Revolut, Venmo, or any supported payment service. The PeerAuth extension attests that the payment happened, and the USDC is released instantly. No waiting, minimal disputes, reduced friction. The seller just configures their bank "address," deposits USDC, and waits - they don't even need the extension.

This brings cryptoanarchy tools closer to the present. The product experience has been great, and overall - good job to the team.

The possibilities are even more exciting. Imagine a "Pay with Revolut" button for merchants to onboard users who aren't yet in the crypto ecosystem. You could buy anonymous services, cashu tokens, or anything else by paying with Revolut - the merchant receives stablecoins (hopefully privacy-preserving ones in the future) and never has to touch bank fiat directly.

Why write this report then? Because I want this to succeed. The goal is to:

- Improve the ecosystem by highlighting what could be better
- Give users clarity about how the service they're using actually works
- Hope for a better, more private, more trustless version

The technical critiques in this report come from a place of wanting to see this technology reach its full potential - true zero-knowledge, fully open source, and verifiable.

How It Works

Proof Generation Flow

```
┌─────────────────────┐             ┌──────────────────────────────────────┐
│  Your Browser       │             │   Attestor (zkp2p.xyz)               │
│                     │             │                                      │
│ 1. You log into     │             │                                      │
│    your bank        │             │                                      │
│                     │             │                                      │
│ 2. Extension        │             │                                      │
│    captures TLS     │────────────►│ 3. Decrypts using temp keys          │
│    session:         │             │    ATTESTOR SEES EVERYTHING:         │
│    - Encrypted      │             │    ├─ Your balance: $5,420           │
│      traffic        │             │    ├─ Transaction: $100 to @john     │
│    - Temp TLS keys  │             │    ├─ Your name, address             │
│    - Redaction      │             │    └─ All API response fields        │
│      config         │             │                                      │
│                     │             │ 4. Applies redaction config          │
│                     │             │    (TRUST-BASED filtering)           │
│                     │◄────────────│                                      │
│ 5. Receive signed   │             │ 5. Signs proof containing only:      │
│    proof            │             │    ├─ Amount: $100                   │
│                     │             │    └─ Recipient: @john               │
└─────────────────────┘             └──────────────────────────────────────┘
```

The Reclaim Protocol

PeerAuth uses Reclaim Protocol's "key-upgrade mechanism":

1. **Capture**: Extension intercepts TLS session between you and your bank
2. **Send**: Extension sends encrypted traffic + temporary session keys to attestor
3. **Decrypt**: Attestor decrypts and sees your **complete bank API response**
4. **Filter**: Attestor applies redaction config (removes balance, personal info, etc.)
5. **Sign**: Attestor signs proof containing only whitelisted fields
6. **Return**: You receive signed proof to share with third parties

What Gets Revealed in Proofs

For a typical transfer proof:

Field	In Proof?
Transaction amount	Yes
Currency	Yes
Recipient username	Yes
Transaction date	Yes
Transaction ID	Yes
Transaction state	Yes
Your balance	No (redacted)
Your name	No (redacted)
Your address	No (redacted)
Other transactions	No (redacted)

What is the "Proof"?

Unlike traditional zero-knowledge proofs, the "proof" generated by PeerAuth is **not a cryptographic proof you generate locally**. It is a **signed attestation from zkp2p.xyz's server**.

Proof is Generated by the Attestor, Not Locally

```
┌─────────────────────┐                  ┌──────────────────────────────┐
│     Extension       │                  │   Attestor (zkp2p.xyz)       │
│                     │                  │                              │
│  You do NOT        │  Send: traffic,  │                             │
│  generate the      │  temp keys,      │  Attestor GENERATES         │
│  proof locally     │  redaction config│  the proof:                 │
│                    │─────────────────▶│  1. Decrypts your data      │
│                    │                  │  2. Applies redaction       │
│                    │                  │  3. SIGNS the result        │
│                    │◀─────────────────│                             │
│  Receive signed    │  Return: signed  │                             │
│  attestation       │  attestation     │                             │
└─────────────────────┘                  └──────────────────────────────┘
```

What the "Proof" Contains

The proof is essentially a **signed document** from zkp2p.xyz stating "we saw this data":

Component	Description
Revealed fields	Transaction amount, recipient, date, etc.
Attestor signature	zkp2p.xyz's cryptographic signature
Attestor identifier	Public key or ID of the signing attestor
Timestamp	When the attestation was created

What it is NOT:

- Not a zero-knowledge proof (attestor saw everything)
- Not generated locally (attestor generates and signs it)
- Not independently verifiable (you must trust the attestor)

Proof Storage and Export

The extension stores proofs locally and exposes APIs to retrieve them:

```
window.zktls = {
  fetchProofs(),        // Get all stored proofs
  fetchProofById(id),   // Get specific proof
  generateProof(...),   // Request new proof generation
}
```

Proofs can be downloaded/exported, but they are fundamentally **attestor signatures**, not independent cryptographic proofs.

How Verification Works

This is the critical difference from true ZK proofs:

Aspect	True ZK Proof	PeerAuth "Proof"
What you verify	Mathematical correctness	Attestor's signature
Trust required	None (math is trustless)	Must trust zkp2p.xyz
Who can forge?	No one (cryptographically impossible)	zkp2p.xyz can forge any proof
Analogy	Mathematical theorem	Notarized document

To verify a PeerAuth proof, you are asking:

> "Did zkp2p.xyz sign this document?"

You are NOT asking:

> "Is this mathematically proven to be true?"

Trust Implications for Third Parties

If you receive a PeerAuth proof from someone:

1. **You must trust zkp2p.xyz** - They could have signed a false attestation
2. **You must trust zkp2p.xyz wasn't compromised** - An attacker with their keys can forge proofs
3. **You cannot independently verify** - Unlike ZK proofs, you can't check the math yourself
4. **zkp2p.xyz is a single point of trust** - No alternative attestors available

This is fundamentally different from TLSNotary (original extension) where the proof is cryptographically verifiable by anyone without trusting a third party.

Privacy Concerns

1. Attestor Sees ALL Your Data - CRITICAL

The attestor at `wss://attestor.zkp2p.xyz/ws` sees your complete decrypted bank API responses.

This includes:

- Your account balance
- All transaction details

- Your name, address, personal information
- Any data in the API response

Redaction is NOT cryptographic - it relies on the attestor following configuration instructions honestly.

What you must trust zkp2p.xyz to do:

1. Not log or store your full bank API responses
2. Actually apply the redaction config honestly
3. Not extract session data for replay attacks
4. Not correlate your proofs with your identity
5. Secure their infrastructure against breaches

2. Attestor Can Log Everything - CRITICAL

Since the attestor decrypts and processes your complete bank data, **there is nothing preventing them from logging it**.

What the attestor can log:

Data	Risk
Complete bank API responses	Your balance, all transactions, personal details
Your IP address	Geographic location, ISP identification
Timing of requests	Usage patterns, when you use banking
Wallet address (from analytics)	Link to your crypto identity

No way to verify they don't log:

- Attestor server is **closed-source** - you cannot audit it
- No published privacy policy specific to attestor data handling
- No transparency reports about data retention
- No third-party audits of their infrastructure
- Pure "trust us" model

What logged data enables:

Threat	Description
Deanonymization	Link your crypto wallet address to your real bank identity
Financial surveillance	Track your spending, income, transaction patterns
Data breach exposure	If zkp2p.xyz is breached, your bank data could leak
Regulatory/legal access	Law enforcement could compel disclosure of logs
Insider threat	Malicious employee could access your data

Critical comparison:

Aspect	TLSNotary (Original)	Reclaim (PeerAuth)
Can notary/attestor log your data?	**NO** - never has plaintext	**YES** - sees everything
Logging prevention	Cryptographic (MPC-TLS)	Trust-based (policy only)
Breach impact	Minimal (no sensitive data)	Catastrophic (full bank data)

Bottom line: Every time you generate a proof, you are sending your complete bank account data to zkp2p.xyz's servers and trusting they will delete it immediately. There is no technical mechanism preventing them from keeping a complete copy.

3. PostHog Analytics - HIGH

The extension sends telemetry to `https://us.i.posthog.com` :

Event	Data Sent
`user_identified`	User ID, wallet address
`extension_proof_generation`	Proof ID, duration, status, provider
`app_error_captured`	Error details with stack traces
`extension_memory_event`	Memory usage metrics
`screen_view`	Route/page tracking
`app_launched/foregrounded/backgrounded`	Usage patterns

No apparent opt-out mechanism found.

This data could be used to:

- Link users across sessions
- Correlate proof attempts with wallet addresses
- Build user behavior profiles

4. Selective Disclosure is Trust-Based

The extension uses configuration fields to control what appears in proofs:

Config Field	What It Means	Attestor Sees Original?
`secretHeaders`	"Don't include these headers in proof"	**YES**
`responseRedactions`	"Don't include these JSON fields"	**YES**
`proofMetadataSelectors`	"Only include these fields in proof"	**YES**

These are **instructions to the attestor**, not cryptographic enforcement. The attestor sees everything and is trusted to filter correctly.

Security Concerns

1. Overly Broad Permissions - HIGH

```
"host_permissions": [
  "https://*/*",
  "http://localhost:3000/*"
]
```

The extension can access **ANY HTTPS website**. This is far broader than necessary for supporting specific bank providers.

2. Global Content Script Injection - HIGH

The extension's content script is injected on **every HTTPS page you visit**:

```
"content_scripts": [{
  "matches": ["https://*/*", "http://localhost:3000/*"],
  "js": ["contentScript.bundle.js"]
}]
```

3. Global API Exposure - HIGH

The extension exposes `window.zktls` on ALL websites:

```
window.zktls = {
  requestConnection(),
  generateProof(...),
  fetchProofById(...),
  fetchProofs(),
  onMetadataMessage()
}
```

Any website can call these methods. While user approval is required via sidebar UI, this creates social engineering attack vectors.

4. Single Centralized Attestor - MEDIUM

All proofs go through a single attestor:

- `wss://attestor.zkp2p.xyz/ws`

Concerns:

- Single point of failure
- No geographic redundancy
- Cannot self-host
- Must trust zkp2p.xyz infrastructure entirely

Supported Providers

PeerAuth supports 18+ payment providers:

1. Alipay (transfer + register)
2. Bank of America (Zelle)
3. CashApp
4. Chase (Zelle)
5. Chime
6. Citi (Zelle)
7. IDFC Bank
8. Luxon
9. Mercado Pago
10. Monzo

11. N26
12. PayPal
13. Revolut
14. Royal Bank Canada (Interac)
15. US Bank (Zelle)
16. Venmo
17. Wise
18. Mercury (wires)

Provider configurations are fetched dynamically from:

```
https://raw.githubusercontent.com/zkp2p/providers/refs/heads/releases/prod/
```

Technical Details

Extension Structure

```
extension/
├── manifest.json
├── background.bundle.js (322 KB)
├── offscreen.bundle.js (5.2 MB)
├── sidePanel.bundle.js (659 KB)
├── contentScript.bundle.js (18 KB)
├── injectScript.bundle.js
└── browser-rpc/resources/
    └── (SNARKJS circuits: AES-256-CTR, AES-128-CTR, ChaCha20)
```

Key URLs

Purpose	URL
Attestor	`wss://attestor.zkp2p.xyz/ws`
Analytics	`https://us.i.posthog.com`
Provider Config	`https://raw.githubusercontent.com/zkp2p/providers/refs/heads/releases/prod/`
Support	`https://support.zkp2p.xyz`
Developer Docs	`https://developer.zkp2p.xyz`

ZK Circuits

The extension includes SNARKJS circuits for symmetric cryptography:

- AES-256-CTR
- AES-128-CTR
- ChaCha20

These could theoretically enable client-side ZK proofs that hide data from the attestor, but they do not appear to be used for this purpose currently.

Author's Perspective

The following are my personal concerns and suggestions for improving PeerAuth:

1. This is NOT Zero Knowledge

The name "zkp2p" is misleading. What actually happens:

- You temporarily leak a TLS session key to a third party (the attestor)
- The attestor decrypts your data, sees everything, and signs an attestation
- You trust them not to log it

This is **attestation**, not zero-knowledge proof. True ZK would mean the attestor learns nothing beyond the validity of the statement.

2. Telemetry Must Be Opt-In

Currently, PostHog analytics is:

- Not opt-in (active by default)
- Not even opt-out (no apparent mechanism to disable)

Telemetry that tracks wallet addresses and proof activity should require explicit user consent.

3. Extension Must Be Free and Open Source

I will not run a closed-source extension that:

- Has access to all my websites
- Processes my bank account data
- Cannot be audited

The extension needs to be:

- Fully open source
- Auditable by security researchers
- Reproducibly buildable

4. Prover Design Leaks Too Much to Attestor

The current design sends your complete bank data to the attestor. A better approach:

Wrap TLSNotary proof in a ZK proof:

1. Generate TLSNotary proof locally (attestor never sees plaintext)
2. Create ZK circuit that:
    - Verifies TLS certificate signatures up to root CA
    - Verifies regexp matching for public parameters
    - Outputs only: recipient, amount, date, status
3. Everything else remains private - attestor never sees it

This is now technically feasible. See @oskarth's talk on this exact topic.

5. Global Page Injection is Actually Useful

While the security analysis flags `https://*/*` permissions as a concern, I believe this is actually valuable:

- Enables on/off-ramp services to integrate anywhere

- Any website could verify payment proofs
- Creates an open ecosystem for P2P payments

The issue isn't the broad permissions - it's the other concerns (closed source, attestor trust, analytics).

6. Attestation Server Must Be Verifiable

If the attestor must see data (current design), it should be verifiable:

Ideal setup:

- Open source the attestor code
- Run reproducible build in a TEE (Trusted Execution Environment)
- Provide TEE attestation proving the code running matches the source
- End-to-end encryption into the TEE
- This would be a sufficient proof that logging is impossible

This would transform "trust us" into "verify the code and TEE attestation."

Recommendations

For Users

1. **Understand that zkp2p.xyz sees your complete bank data** - Your balance, transactions, and personal information are visible to the attestor before redaction
2. **Redaction relies on trust, not cryptography** - You must trust zkp2p.xyz to not log, store, or misuse your data
3. **Be aware of analytics** - Your wallet address and proof activity are tracked via PostHog
4. **The extension monitors all websites** - Content script is injected everywhere you browse
5. **There is no self-hosting option** - All proofs must go through zkp2p.xyz's single attestor

For Developers

1. **Open source everything, including the extension** - This is a prerequisite for trust; closed-source extensions handling bank data are unacceptable
2. **Make telemetry opt-in** - Currently not even opt-out; wallet address tracking requires explicit consent
3. **Provide multiple attestor options** for redundancy and self-hosting

4. **Open source and verify the attestor** - Run in TEE with attestation, or implement true client-side ZK
5. **Consider client-side ZK** - Wrap TLSNotary in ZK proof so attestor never sees plaintext (see oskarth's talk)
6. **Document data retention policy** clearly
7. **Host permissions are fine** - Global `window.zktls` API enables open on/off-ramp ecosystem (but fix the trust issues first)

Support my work

If you want more work like this, you can support my work here.

And if you want to learn more and stay in the loop, follow me on Nostr: npub1m2mvvpjugwdehtaskrcl7ksvdqnnhnjur9v6g9v266nss504q7mqvlr8p9, X at @jurbed, Listen to the OptionPlus podcast and read my blog.