BTC Medusa

Adversarial Privacy Analysis

Why Every Constraint Exists, What Breaks If You Remove It,
and Why Even a Malicious Server Cannot Deanonymize You

This document walks through every constraint in the BTC Medusa ZK spend circuit (Groth16 on BN254) and every layer of the privacy transport stack from an adversarial perspective, as implemented. For each component, we describe the concrete attack that becomes possible if it is removed and argue that the defense is sound. The analysis assumes the strongest possible adversary: a fully malicious server operator. Designed features that are not yet built are marked with amber PLANNED callouts and excluded from the implemented security claims.

June 2026
Groth16 Spend Circuit (6 Constraints, ~21,000 R1CS), Depth-16 Revocation Tree, Single-Point Protocol,
Payment-Gated Issuance, Activation Epoch — FROST / OHTTP / Attribution Capsules marked Planned

1. Threat Model

Throughout this analysis we assume the strongest adversary: a malicious BTC Medusa server operator. This server:

- Has full control of the server software, database, and network stack.
- Can observe all API requests (timing, payloads, and — absent transport privacy — IP addresses).
- Can modify server responses (return wrong data, inject fake proofs).
- Can collude with exchanges, chain-analysis firms, or law enforcement.
- Wants to learn: which user is querying which UTXO, how many tokens a user has, when tokens expire, when they activate, and the identity behind each wallet.

We also consider external observers: network-level adversaries (ISPs, state actors) who can monitor traffic, and on-chain analysts who can see payment transactions.

The goal is to show that even under this maximal threat model, the adversary learns nothing beyond: (1) a valid spend occurred, (2) the public nullifier (unlinkable to any token or user), (3) the global current_month parameter, and (4) the revocation tree root (which the server itself published).

2. The Spend Circuit at a Glance

The Spend Circuit is a Groth16 R1CS circuit with 6 constraint groups and approximately 21,000 total R1CS constraints. The prover demonstrates, in zero knowledge, that they hold a legitimate, activated, unexpired, non-revoked, server-signed token and know the secret seed that generated it. The circuit enforces a start_month activation check (Constraint 5), uses PK_server as a hardcoded

R1CS constant, and proves non-inclusion in a depth-16 Sparse Merkle revocation tree (16 levels, 65,536 leaf slots).

#	Constraint	~R1CS	What It Prevents
1	Base-Point Derivation	4,500	Arbitrary point injection, DL extraction, pack_id substitution
2	Nullifier Binding	1,200	Nullifier grinding, double-spend evasion, pack_id mismatch
3	DLEQ Signature Verification	6,000	Forged server signatures, token fabrication
4	Expiration Check	254	Expired token reuse
5	Activation Check	254	Premature spending of future-dated packs
6	Revocation Non-Inclusion (depth 16)	8,600	Spending revoked (chargebacked) token packs

Public inputs (3 Fr): nullifier, current_month, revocation_root.
Private witnesses: token_seed, token_index, pack_id, sigma_i(x,y), dleq_c, dleq_s, expiration_month, start_month, smt_siblings[0..15], smt_path_bits[0..15].
Derived in-circuit: P_i = Poseidon(seed, pack_id, idx) · H_TOKEN.
Constants (hardcoded): H_TOKEN, G, PK_server.

NOTE: Design note: PK_server as a hardcoded constant is strictly stronger than as a public input — the prover cannot even attempt key substitution. The depth-16 SMT provides 65,536 revocation slots, and the Activation Check enforces start_month ≤ current_month. Key rotation today requires a new trusted setup ceremony; FROST threshold key management is planned but not implemented (Section 3, Constraint 3 box).

2.1 What the Circuit Deliberately Omits (and Why It's Safe)

Omitted Constraint	What It Would Do	Why It Is Unnecessary
Pedersen Commitment	Prove quota binding to board entry	DLEQ proves the token was server-signed; the server only signs during paid issuance
Board Membership	Prove pack exists on bulletin board	DLEQ proves the server signed the token — phantoms impossible
Range Check	Prevent quota overflow	No quota in circuit — DLEQ limits to signed tokens
Alpha Binding	Bind OPRF query to proof	Single-point protocol — proof is authentication, query is separate

3. Constraint-by-Constraint Adversarial Analysis

Each constraint is analyzed from first principles: what it enforces, what concrete attack becomes possible if it is removed, and why the defense is sound.

Constraint 1: Base-Point Derivation (~4,500 R1CS)

```
scalar = Poseidon(token_seed, pack_id, token_index)
P_i = scalar * H_TOKEN
```

The token's base point P_i is derived deterministically from the seed, pack_id, and index. This is critical because P_i feeds DLEQ verification (Constraint 3), and the pack_id binding prevents substitution attacks against the revocation tree (Constraint 6).

ATTACK — Arbitrary Point Injection. If P_i were a free witness, a malicious prover could choose k at random, set P_i = k·G, then compute sigma_i = k·PK_server. This satisfies the DLEQ relation sigma_i = v·P_i because k·(v·G) = v·(k·G). The prover has forged a valid server signature without the server ever signing — unlimited free tokens.

DEFENSE — Hash-to-Curve Derivation. Forcing P_i = Poseidon(seed, pack_id, idx) · H_TOKEN means the scalar is determined by the hash, and H_TOKEN is a Nothing-Up-My-Sleeve point (SHA-256 try-and-increment hash-to-curve), so nobody knows its discrete log relative to G. Including pack_id also prevents using tokens from a revoked pack under a different pack_id — changing pack_id changes P_i, which invalidates the DLEQ signature.

> **REMARK:** Why this attack is the most subtle: it exploits the algebraic relationship between G and PK_server (= v·G). If the prover can choose P_i freely, they can set up a DLEQ that 'proves' the server signed their chosen point — but the server never saw it. The derived base point is the only defense.

Constraint 2: Nullifier Binding (~1,200 R1CS)

```
nullifier_derived = Poseidon(token_seed, pack_id, token_index)
enforce: nullifier_derived == nullifier
```

ATTACK — Nullifier Grinding. Without this binding, a prover could present a random nullifier on each spend of the same token; the server would treat each as a unique spend — unlimited double-spending from a single token.

DEFENSE — Deterministic Derivation. The nullifier is computed inside the circuit from (token_seed, pack_id, token_index); the same token always produces the same nullifier. Zero degrees of freedom.

Privacy property: the nullifier reveals nothing about (token_seed, pack_id, token_index) because Poseidon is preimage-resistant. The server sees the nullifier but cannot determine which token was spent, which pack it belongs to, or which seed generated it.

Constraint 3: DLEQ Signature Verification (~6,000 R1CS)

```
// PK_server read from hardcoded R1CS constant
R1' = s * G - c * PK_server
R2' = s * P_i - c * sigma_i
c' = Poseidon(PK.x, PK.y, sigma_i.x, sigma_i.y, R1'.x, R1'.y, R2'.x, R2'.y)
enforce: c == c'
```

The most critical constraint in the circuit: it verifies the server's blind signature (sigma_i) on the specific token base point P_i.

ATTACK — Forged Server Signature. Without in-circuit DLEQ verification, sigma_i is just a private witness — the prover sets it to any point and generates unlimited tokens without ever contacting the server.

DEFENSE — In-Circuit Chaum-Pedersen Verification. The circuit recomputes the Fiat-Shamir challenge with Poseidon over the transcript coordinates and enforces c == c'. PK_server is a hardcoded R1CS constant, so the prover cannot substitute a key they control. Forging (c, s) requires breaking discrete log on BabyJubJub (~125-bit security) or finding a Poseidon preimage/collision (128-bit security).

PK_server as Constant — Key Management Today, FROST Planned

PK_server is a hardcoded R1CS constant rather than a public input — strictly stronger: the key is baked into the R1CS at setup time, so substitution is not even expressible. The cost is operational: as implemented, rotating the server key requires a new trusted setup ceremony.

> **PLANNED — NOT YET IMPLEMENTED:** FROST threshold key management — distributing v across multiple holders so the aggregate key V = v·G stays fixed while shares rotate, with no ceremony for share refresh — is the planned mechanism that recovers rotation flexibility without giving up the constant-key security posture. No FROST code exists in the codebase today.

Constraint 4: Expiration Check (~254 R1CS)

```
diff = expiration_month - current_month
bits = bit_decompose(diff, 18)
enforce: reconstruction fits in 18 bits
```

ATTACK — Expired Token Reuse. Without the expiration check, a user who purchased a single month could spend tokens indefinitely.

DEFENSE — Bit Decomposition of Non-Negative Difference. The subtraction is performed in Fr(BN254); a negative (expired) difference wraps to a $\sim 2^{254}$ field element that cannot be represented in 18 bits.

Privacy property: expiration_month is a private witness, not a public input. The server learns only that the token is still valid. current_month is a global parameter — the same value for everyone in a given month, leaking no per-user information.

Constraint 5: Activation Check (~254 R1CS)

```
diff = current_month - start_month
bits = bit_decompose(diff, 18)
enforce: reconstruction fits in 18 bits
```

ATTACK — Premature Pack Spending. Without the activation check, a yearly subscriber (13 packs: a bridge pack plus 12 staggered monthly packs) could burn every pack in the first month — 12 months of service consumed in month one.

DEFENSE — Mirror of the Expiration Check. The constraint proves current_month − start_month is non-negative; a future start_month wraps and fails the 18-bit range check.

Privacy property: start_month is a private witness. Combined with expiration_month also being private, the server cannot determine the subscription tier, duration, or pacing schedule of any user.

ATTACK — Activation-Expiration Inversion. A client sets start_month > expiration_month, creating a pack that can never be spent. DEFENSE — not needed in-circuit: this is a client-side configuration error that only harms the client; no security property is violated.

Constraint 6: Revocation Non-Inclusion (~8,600 R1CS, depth 16)

```
leaf_key = Poseidon(pack_id)        // truncated to 16 bits
current = EMPTY_LEAF                 // zero — must be empty (not revoked)
for level in 0..16:
  sibling = smt_siblings[level]
  if smt_path_bits[level] == 0: current = Poseidon(current, sibling)
  else:                         current = Poseidon(sibling, current)
enforce: current == revocation_root
```

Proves the token's pack_id is not in the Sparse Merkle revocation tree (16 levels, 65,536 slots). Policy: revocations apply only to credit-card (Stripe) payments; Bitcoin and Lightning payments are final and irrevocable.

ATTACK — Spending Revoked Token Packs. Without this constraint, a user who charges back their card keeps their tokens. DEFENSE: the circuit walks the 16-level Merkle path from an empty leaf to the root; a revoked pack's leaf is non-zero, so no valid witness exists.

ATTACK — Pack ID Substitution. A client with a revoked pack_id proves non-inclusion for a different pack_id. DEFENSE: pack_id is woven into base-point derivation, the nullifier, and the SMT proof; substitution changes P_i and invalidates the DLEQ signature.

ATTACK — Stale Revocation Root Replay. The client proves against a root from before their pack was revoked. DEFENSE: the server checks the proof's root against the current published root at spend time (403 on mismatch).

Privacy property: pack_id and the SMT path are private witnesses — the server never learns which pack_id was checked. All non-revoked packs share the same root, so even the proof-generation fetches are uniform across users.

PLANNED — NOT YET IMPLEMENTED: Operational note: revocation is currently performed by an operator through the admin-authenticated /revocation/revoke endpoint. Automatic insertion from Stripe chargeback webhooks is planned.

4. Why the Omitted Constraints Are Unnecessary

Four constraints that might seem natural are deliberately absent. Here is why each is safe to omit.

- 4.1 Pedersen Commitment — OMITTED: C = r·G_PED + q·H_PED. Why safe: the DLEQ (Constraint 3) already proves the token was server-signed, and the server only signs during legitimate paid issuance (payment-gated, Section 8 row 'Issuance payment gate').
- 4.2 Board Membership — OMITTED: Merkle proof from leaf to board_root. Why safe: the DLEQ signature is a stronger guarantee than board membership.
- 4.3 Range Check — OMITTED: bit_decompose(quota, 10). Why safe: quota is not a circuit value; DLEQ limits spending to signed tokens.
- 4.4 Alpha Binding — OMITTED: alpha == rho·P_i. Why safe: single-point protocol — the ZK proof is authentication; the OPRF query is separate.

5. The Single-Point Spend Protocol

The spend protocol is single-point, at POST /oprf/evaluate. The ZK proof is the authentication; the OPRF query point is separate and unauthenticated in-circuit.

```
alpha_query = rho * H2C(outpoint)        // the only OPRF point
```

The client sends the ZK proof (3 public inputs) and alpha_query (the blinded UTXO lookup). The server verifies the proof, checks the revocation root, burns the nullifier (persistent ledger), then evaluates beta = v · alpha_query.

ATTACK — Can the server link alpha_query to a UTXO? No. The server would need rho (private) to recover H2C(outpoint), and even then would need to invert H2C (preimage-resistant).

ATTACK — Can the server correlate alpha_query values across spends? No. Each spend uses a fresh random rho; even the same outpoint queried twice yields independent random group elements.

ATTACK — Payment-free token minting (the gating bug class). The cryptography above only constrains tokens the server signed; if any endpoint blind-signs without payment, the whole economic model collapses. DEFENSE — payment-gated issuance: the only signing paths are /stripe/redeem, /btc/redeem, and /ln/claim, each consuming a one-shot credential atomically before signing; the bare /issue endpoint returns 403 unless an explicit development flag is set. This invariant is pinned by regression tests on both the server (Rust) and the payment scanner (Python): ungated issuance rejected, forged/malformed/replayed credentials rejected.

6. Transport Layer Privacy Analysis

The ZK circuit and OPRF protocol protect the content of queries. The transport layer protects the source.

6.1 Why Transport Privacy Is Non-Negotiable

- Temporal correlation: the same IP issues a pack then spends a token minutes later.
- Geolocation: the IP reveals geographic region; combined with on-chain data it narrows the anonymity set.
- Legal compulsion: a subpoena could force disclosure of IP logs.
- Cross-service correlation: if the IP appears in exchange logs, the server can link activity to a real identity.

6.2 Tor Transport (Default Preference) — Native .onion Hidden Service

Client → local Tor daemon (SOCKS5, default socks5h://127.0.0.1:9050) → 3-hop Tor circuit → the server's .onion service. Traffic never leaves the Tor network; no exit node is involved; the server's own IP is also hidden. Sensitive one-shot calls (token redemption) use an isolated fresh circuit. The wallet's default transport mode is AUTO, which prefers Tor.

NOTE: The wallet relies on an external Tor daemon reached over SOCKS5 — it does not bundle its own Tor client. If no daemon is running, AUTO mode falls through (ultimately to Direct) — users who need guaranteed IP privacy should run Tor and select the TOR mode explicitly. Bundling an embedded client remains an option for a future release.

ATTACK — Traffic Analysis. A global passive adversary could correlate Tor entry/exit timing. DEFENSE: multi-hop latency plus the fact that BTC Medusa queries are infrequent and small; filter fetches are additionally padded with Laplace-distributed decoy blocks (default 10 decoys, ±4,000-block spread), so even a compromised transport layer does not reveal which block heights are real.

6.3 OHTTP Transport (Planned Fallback)

PLANNED — NOT YET IMPLEMENTED: Oblivious HTTP (RFC 9458) for Tor-blocked environments is designed but not implemented: the transport-layer code is a placeholder and the Settings option is disabled ("coming soon"). Until it ships, users in Tor-blocked environments have no in-app IP-privacy fallback — the relevant attack row in Section 8 is scored accordingly. The planned design: relay sees IP but not content; server sees content but only the relay's IP; the relay URL is user-configurable to avoid relay-server collusion.

6.4 Direct Transport (Development Only)

Standard HTTP(S); the server sees the client's IP. Selecting it requires acknowledging an explicit warning in Settings, and the status-bar shield turns orange (green = Tor, yellow = OHTTP when it ships). Exception by design: Stripe checkout-session creation and status polling are clearnet even in Tor mode, because the user immediately opens a clearnet Stripe page in their browser; token redemption still rides Tor (Section 7, Payment Unlinkability).

7. Broader Protocol Privacy Properties

OPRF Obliviousness: The Server Cannot See Your Queries

The VOPRF guarantees the server evaluates f_v(x) = v·H2C(x) without learning x. The DLEQ proof returned with each evaluation proves the server used its real key — a malicious server cannot selectively corrupt evaluations without detection.

Nullifier Unlinkability

Different tokens from the same pack produce independent nullifiers: Poseidon(seed, pack_id, 0) and Poseidon(seed, pack_id, 1) are uncorrelated outputs. The server cannot tell whether two nullifiers came from the same pack.

Timing Correlation Resistance

ATTACK: the server correlates issuance timing with spend timing. DEFENSE: a growing anonymity set plus Tor transport, which eliminates IP-based correlation entirely.

Payment Unlinkability

Bitcoin: subscription payments use BIP-352 silent payments — each payment derives a unique, unlinkable on-chain output; no address reuse. Lightning: invoices are single-use and claimed with a one-shot nonce over Tor. Stripe: the card never touches the BTC Medusa server (hosted Checkout); the redemption that actually mints tokens runs over Tor with an anonymous one-shot code and blinded points, so issued tokens are unlinkable to the card. Caveat: the checkout-creation and status-poll calls are clearnet, so the server does observe the user's IP during payment bookkeeping (not during spends); routing these over OHTTP is planned.

Attribution Board Privacy

As implemented, the board records one plaintext (pack_id, tag, expiration_month) triple per issued pack on an append-only Merkle tree. This reveals aggregate issuance volume and pack expiration months to anyone — but contains no user identity, no payment data, no amounts, and no linkage to spends: the pack_id never appears again in any spend (it is a private witness), so board entries cannot be joined to nullifiers or queries.

> **PLANNED — NOT YET IMPLEMENTED:** The encrypted-capsule board — each leaf an ECDH capsule (R, tag = H('wallet_attr' ‖ ss), encrypted payload) with no cleartext metadata — is the planned design, not the current board. Likewise, dual-layer attestation (wallet-encrypted amount under P_dev cross-checked against the server-claimed revenue) — the defense against server revenue fraud toward wallet makers — is planned and not yet built.

Activation Epoch Privacy

start_month is a private witness — the server never learns when a pack activates. Combined with expiration_month also being private, the server cannot determine whether a user has a monthly or

yearly subscription, which month of a yearly subscription they are in, or how many future packs remain. The server's view is identical regardless of tier: a stream of nullifiers arriving over anonymous Tor circuits.

8. "Remove Any One Component" — Attack Surface Table

The concrete attack that becomes possible if any single implemented component is removed:

If You Remove...	The Attack Is...	Severity
1. Base-Point Derivation	Inject known-DL point, forge signatures, fabricate unlimited tokens	CRITICAL
2. Nullifier Binding	Double-spend the same token unlimited times with random nullifiers	CRITICAL
3. DLEQ Signature	Fabricate tokens without server interaction — no payment needed	CRITICAL
4. Expiration Check	Use expired tokens forever — one payment, lifetime access	HIGH
5. Activation Check	Burn all yearly packs in month one — defeats subscription pacing	HIGH
6. Revocation Check (depth-16 SMT)	CC-chargebacked packs remain usable (BTC/LN packs are never revoked)	HIGH
Revocation Root Freshness Check	Client replays a stale root from before revocation	HIGH
Issuance Payment Gate	Server blind-signs without payment — free token packs for anyone (this regression occurred during development and is now pinned by tests)	CRITICAL
Nullifier Ledger (server)	Double-spend all tokens — unlimited queries from one payment	CRITICAL
Tor Transport	Server logs client IPs, builds behavioral profiles, subpoena-vulnerable	HIGH
Decoy Block Fetching	Filter requests reveal which block heights (and so which txs) interest the wallet	MEDIUM
Attribution Board	No tamper-evident issuance record (capsule-based revenue verification is planned)	MEDIUM

What About the Omitted Constraints?

Omitted Component	The Attack It Would Address	Why It Is Covered
Pedersen Commitment	Hijack another user's board entry	Board not in circuit; DLEQ prevents token forgery
Board Membership	Spend from phantom (never-issued) packs	DLEQ proves the server signed the token — phantoms impossible
Range Check	Overflow quota via field arithmetic	No quota in circuit — DLEQ limits to signed tokens
Alpha Binding	Prove with token A, query unrelated data B	Single-point: proof is authentication, query is separate

9. What the Server Learns

Information	Leaked?	Why
Which UTXO you're querying	NO	OPRF blinding (rho) hides the input
Which block heights you care about	NO	Laplace decoy fetches (default 10, ±4,000 blocks)
Your token_seed	NO	Private witness, never transmitted
Which token_index you spent	NO	Private witness, feeds nullifier derivation
Your token's base point P_i	NO	Derived in-circuit, not an input
When your pack expires	NO	expiration_month is a private witness
When your pack activates	NO	start_month is a private witness
Your subscription tier/duration	NO	Both start_month and expiration_month are private
Which pack_id was checked (revocation)	NO	pack_id and the SMT path are private witnesses
Your IP address (spends, redemptions, filters)	NO	Tor .onion transport (AUTO default prefers Tor)
Your IP address (Stripe checkout bookkeeping)	YES (today)	Checkout creation + status polls are clearnet by design; OHTTP routing planned
Aggregate issuance volume / pack expirations	YES	Board entries are plaintext (pack_id, tag, expiration_month); not linkable to spends
A valid spend occurred	YES	Nullifier is public — by design
The nullifier value	YES	Public input, but unlinkable to any token or user
The global current_month	YES	Public input, same for all users
The revocation_root	YES	Public input, but the server published it — no new info

The server's view of the spend protocol is: a stream of (nullifier, current_month, revocation_root) tuples arriving over anonymous Tor circuits, each with a blinded OPRF query point. It cannot link successive spends to the same user, cannot determine which token was spent, cannot see what UTXO was queried, cannot learn which pack_id was checked against the revocation tree, and cannot determine the subscription tier or activation schedule of any user.

SUMMARY: The public surface is the minimum needed: a nullifier (double-spend prevention), a month (expiration + activation enforcement), and a revocation root (chargeback enforcement) — note the server's own public key is deliberately NOT among the public inputs (it is a circuit constant). The remaining implemented leaks are confined to payment bookkeeping (Stripe clearnet calls) and plaintext board metadata — both with planned mitigations (OHTTP routing; encrypted attribution capsules).

— End of Analysis —