LLM-Tor Whitepaper

Author: Prince Gupta | **Repository:** github.com/prince776

Abstract

Large Language Models (LLMs) often require users to provide payment and identity information, creating a permanent, verifiable link between a user and their prompts. This poses a significant privacy risk, as user data can be logged, analyzed, or leaked. We propose **LLM-Tor**, a privacy-preserving proxy system that decouples user identity from their LLM queries. LLM-Tor achieves two-sided anonymity:

- **From the LLM Provider:** The provider only sees requests from the LLM-Tor proxy, not the end-user.
- **From the LLM-Tor Host:** The proxy itself cannot deterministically link an incoming user (who pays for service) to a specific prompt they submit.

This is achieved by combining a BlindRSA-based authentication protocol for unlinkable authorization with the Tor network for network-level anonymity.

1 The Problem of Verifiable Anonymity

Accessing high-performance LLMs (e.g., GPT-4, Gemini 2.5 Pro) typically requires an API key tied to a payment method. This creates an unavoidable link between a user's identity and their query history.

Existing solutions are insufficient as they fail to solve the trust problem:

- **VPNs / Standard Proxies:** These services simply shift the trust. The proxy provider can still see and log both the user's IP and their unencrypted prompt, creating a single point of failure.
- **Tor Network:** While Tor provides network anonymity, it does not solve the payment problem. A user must still authenticate with the LLM provider, linking their (paid) account to the query, even if the request originates from a Tor exit node.

LLM-Tor is designed to be a *trustless proxy*, where user anonymity is protected cryptographically from the proxy itself, not by a "no-logging" promise.

2 System Architecture and Protocol

LLM-Tor is a proxy service that maintains a pool of its own LLM API keys. It acts as the intermediary, paying the LLM providers on the user's behalf. The core of the system is a three-step protocol that separates payment and authentication from the actual query.

To solve terminology confusion, we define:

- **API Credit:** A purchasable unit of service (e.g., "1 GPT-4 request") sold by LLM-Tor.
- **Auth Token:** A unique, random cryptographic message (m) created by the client to be signed.
- **Nullifier:** A public, one-way hash of the Auth Token, used to prevent double-spending, denoted as $n = \text{hash}(m)$.

2.1 The 3-Step Anonymity Protocol

Step 1: Purchase (via Clearnet)

1. A user connects to the LLM-Tor service over the regular internet (clearnet).
2. They purchase API Credits for specific models (e.g., 10 credits for GPT-4, 20 for Gemini Pro).

At this stage, the LLM-Tor server knows the user's identity (e.g., IP, payment info) and their balance of API Credits.

Step 2: Blind Authentication (via Clearnet)

When a user wishes to spend a credit, their client application:

1. Generates a cryptographically secure random message, the Auth Token (m).
2. Generates a Nullifier ($n = \text{hash}(m)$).
3. Blinds the Auth Token to create $b = \text{blind}(m)$.

The client sends b and n to the LLM-Tor server over the clearnet, authenticating as the user from Step 1.
The LLM-Tor server:

1. Checks that the user has sufficient API Credits for the requested model.
2. Checks its database to ensure the Nullifier n has not been used before.
3. If valid, it signs the blinded token b with its private key for that specific model, creating a blind signature b_{sig}.
4. It decrements the user's API Credit balance and records the Nullifier n as pending.
5. It sends b_{sig} back to the client.

Crucially: Because of the properties of BlindRSA, the server knows it signed something for this user, but it does not know the original Auth Token m or the final unblinded signature m_{sig}.

Step 3: Anonymous Query (via Tor Network)

1. The client receives b_{sig} and unblinds it to get the final signature m_{sig}, which is a valid signature on the original, secret Auth Token m.
2. The client now disconnects from the server and establishes a new connection via the Tor network.
3. From a new anonymous identity, the client sends the LLM-Tor server: the user's prompt (e.g., "What is BlindRSA?"), the Auth Token m, and the unblinded signature m_{sig}.

The LLM-Tor server (receiving this anonymous request):

1. Uses its public key to verify that m_{sig} is a valid signature for m.
2. Calculates $n' = \text{hash}(m)$ and checks its database to ensure this nullifier is in a pending state (i.e., paid for) but not spent.
3. If valid, it marks the nullifier as spent (preventing replay attacks).
4. It proxies the user's prompt to the upstream LLM provider using its own API key.
5. It returns the LLM's response to the user over the Tor connection.

Result: The server cannot link the user from Step 1/2 to the anonymous request in Step 3. It only knows that someone who previously paid for a credit has now spent it.

3 Practical Considerations

3.1 Model-Specific Keys

To manage billing for different models (which have different costs), LLM-Tor hosts a separate public/private key pair for each model it supports (e.g., `gpt-4-key`, `gemini-pro-key`). A user's API Credit purchase and blind signature request are specific to one of these keys.

3.2 Content Moderation

As a proxy service, LLM-Tor is subject to legal requirements and must perform content moderation. Moderation is applied at the proxy level (e.g., via the Azure Content Moderation API) before the prompt is sent to the upstream LLM provider. This is a necessary operational cost.

3.3 Client Application

While the LLM-Tor endpoints are open for any third-party client to use, we provide a default open-source desktop application. A desktop client is preferred over a web application to minimize client-side attack vectors like browser fingerprinting, which could compromise user anonymity.

4 Limitations and Security Analysis

The anonymity of LLM-Tor is not absolute and is subject to three main limitations:

1. **Prompt-Level PII:** If a user includes their own Personally Identifiable Information (PII) in a prompt, the system cannot protect them.
2. **Tor Network Attacks:** The system inherits the limitations of Tor, including vulnerability to global passive adversaries or timing/correlation attacks (e.g., if the host can monitor both the user's internet connection and the Tor exit node).
3. **Anonymity Set Size (Temporal Correlation):** This is the most significant practical limitation. The user's anonymity is only as strong as the "anonymity set"—the pool of other users they are hiding within. The true anonymity set is not "all registered users." It is "all users who performed Step 2 for the same model in a given time window T."

Example Attack

If a malicious LLM-Tor host sees "User A" perform Step 2 for GPT-4 at 10:01 AM, and only one anonymous Step 3 request for GPT-4 arrives via Tor at 10:02 AM, the host can statistically correlate the two, even without a cryptographic link.

Mitigation

This attack becomes ineffective as the number of concurrent users (N) increases, as the probability of linking a query to a user approaches $1/N$. Therefore, the practical anonymity of LLM-Tor is directly proportional to its user base and request volume. The clients can also add unpredictable behavior in when they perform Step 2 and Step 3.

5 Conclusion

LLM-Tor provides a practical and cryptographically-sound architecture for anonymous LLM access. By separating payment from querying using BlindRSA and the Tor network, it creates a trustless environment where users are protected from both the LLM provider and the proxy host. While limited by the statistical risks of temporal correlation, it offers a robust privacy model that scales with user adoption.

6 References

1. Blind RSA Signatures. IETF. RFC 9474.
2. The Tor Project. `torproject.org`