Generating trusted sphinx packets

aurelien.chassagne

April 22, 2025

1 Introduction

The Nym mixnet [2] is a privacy-preserving network designed to routes each packet through a different path. At its core, the Nym mixnet relies on the Sphinx [1] packet format and Coconut [3] anonymous credentials to allow clients to securely and anonymously interact with various applications. While these mechanisms enhance privacy and security, they are vulnerable to misuse by dishonest users who might exploit valid Coconut credentials to deceive the system by altering the hidden destination within the Sphinx header. Such misuse would be detected only at the final node of the mixnet preventing the user from accessing another application. However, prior mixnodes would have already wasted computational resources processing an invalid packet. This vulnerability enables Denial of Service (DoS) attack by exhausting mixnodes computational power with illegitimate packets.

This paper aims to address the critical challenge of ensuring trust in the generation of Sphinx headers, thereby preventing such misuse. Two potential solutions are proposed:

- **Multi-Party Computation (MPC)**: used to decentralize the generation of Sphinx headers among trusted third parties while ensuring they learn nothing about the destination or the path.
- **Zero-Knowledge Proofs (ZKP)**: used to prove that the hidden destination encoded in the Sphinx header actually corresponds to an address authorized by the user's anonymous credentials without revealing the actual destination.

JT: Can we enumerate security objectives or requirements? So that we can come back to them in the eval section?

By exploring these approaches, the paper seeks to enhance the trustworthiness of the Nym mixnet.

2 Mixnet

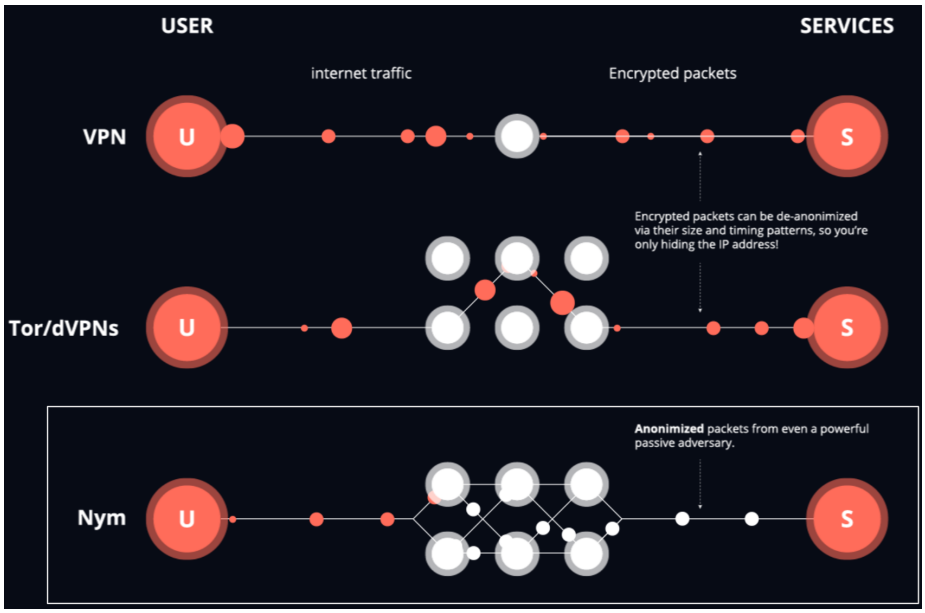
A mixnet (mix network) is a routing protocol designed to provide strong anonymity and resist eavesdropping, even against global adversaries capable of observing

the entire network. It achieves this by relaying messages through a series of intermediary nodes, called mixnodes, which apply cryptographic transformations to the messages before forwarding them.

Messages are recursively encrypted, creating multiple layers that are peeled off by each mixnode along the path. This ensures that each node only knows its immediate predecessor and successor, preventing any single node from accessing both the sender's and the receiver's information. Additionally, each encryption layer includes an integrity tag, which prevents tampering and improves the network's resistance against malicious mixnodes and active adversaries.

Although mixnet shares similarities with onion routing, they differ in key aspects as illustrated by Figure 1. First, mixnet is packet-based, whereas onion routing is circuit-based. This means that each packet follows a different path rather than using the same intermediate nodes during the whole communication. Furthermore, mixnet includes message batching and reordering mechanism to mitigate timing analysis attacks. To further prevent correlation, mixnet relies on fixed-size packet format such as Sphinx packet (section 3), making it difficult for external observers to link incoming and outgoing messages at any given node.

In summary, mixnets provide stronger privacy guarantees than onion routing at the cost of increased latency.



Explain difference between red packets and white packets in caption, Explain difference between red and white nodes in caption, typo: Anonymized, Explanatory text in figure could use bigger font.

Figure 1: Nym traffic compared to TOR and VPN. [source ?]

3 Sphinx

Sphinx packets consist of a header and an encrypted payload. The header itself contains a *cryptographic element* α (e.g. g^x or an elliptic curve point), *encrypted routing information* β, and an *integrity tag* γ, as illustrated in Figure 2.

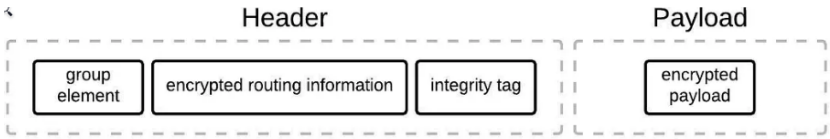


Figure 2: Structure of sphinx packet. [source]

The *encrypted routing information (β)* is constructed in layers, applied in reverse order along the path. First, the final destination is encrypted, and an integrity tag (γ_i) is computed. The IP address of the last mixnode (n_i) is then prepended. As shown by Figure 3, this process repeats iteratively: each new header is encrypted, an integrity tag (γ_{i-1}) is computed, and the IP address of the preceding mixnode (n_{i-1}) is prepended. This layered encryption ensures that each mixnode can only decrypt its own layer, revealing the next forwarding address while preserving end-to-end confidentiality and protecting against tampering.

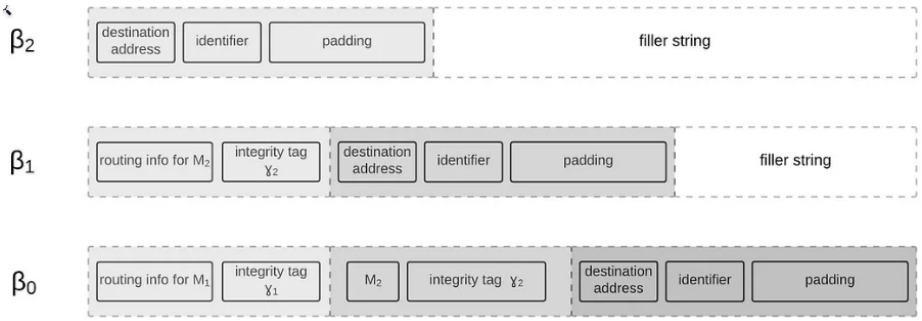


Figure 3: Sphinx encrypted routing information encapsulation. [source]

To encrypt the routing information, the nym client first chooses a nonce x and compute $\alpha = g^x$ as the *cryptographic element* of the header. Since each mixnode i has a private key x_i and a public key $y_i = g^{x_i}$, the user can create a shared secret s_i with mixnode i as followed: $s_i = y_i^x = (g^{x_i})^x$. Then the mixnode i receiving the packet will get α allowing him to compute the shared secret as followed: $s_i = \alpha^{x_i} = (g^x)^{x_i}$.

previously "user" but JT prefers "nym client"

Instead of sending a unique *cryptographic element* α at each node in the path, the sphinx format uses a single *cryptographic element* α, which is progressively modified at each node. Each mixnode updates the cryptographic element using

its shared secret as follows:

$$\alpha_{i+1} = \alpha_i^{\text{hash}(\alpha_i, s_i)}$$

Thus, the user iteratively computes the shared secrets in the path's order as:

$$\begin{array}{lll} \alpha_0 = g^x, & s_0 = y_{n_0}^x, & b_0 = \text{hash}(\alpha_0, s_0) \\ \alpha_1 = g^{xb_0}, & s_1 = y_{n_1}^{xb_0}, & b_1 = \text{hash}(\alpha_1, s_1) \\ \vdots & \vdots & \vdots \\ \alpha_i = g^{xb_0 \cdots b_{i-1}}, & s_i = y_{n_i}^{xb_0 \cdots b_{i-1}}, & b_i = \text{hash}(\alpha_i, s_i) \end{array}$$

This formulation ensures that each mixnode can independently derive the necessary cryptographic elements without requiring the full path's information, preserving privacy and unlinkability.

The *encrypted routing information (β)* is computed, as illustrated in Figure 3, by processing the path in reverse order. This involves XORing the routing information (β_{i-1}) from the previous layer (with the node's address and integrity tag) with a value derived from the shared secret s_i. Then prepending this new encrypted routing information (β_i) with an integrity tag (γ_i) and the previous mixnode address (remember we build it in reverse order). We repeat the same process for each layer (i.e. each mixnode in the path).

~\ref{}

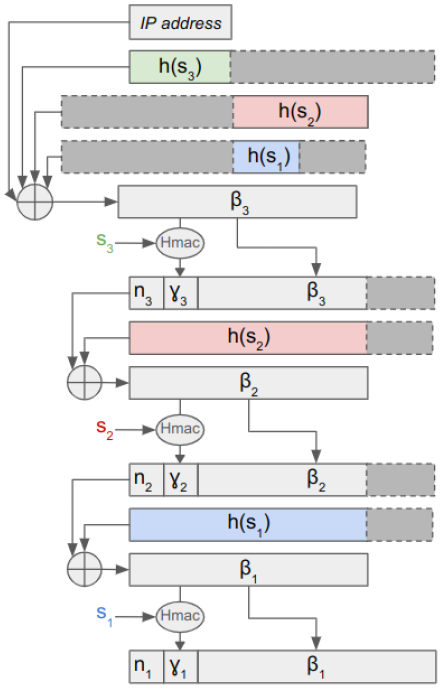


Figure 4: Construction of the Sphinx header (modified from [1]) [TO FIX: $h(s_1)$ at first XOR]

JT: Explain what the modifications are

JT: Might help to put some (1) (2) (3) labels into the figure and refer to these labels in the text.

The first round of XOR operations differs from the others because it requires combining parts of all shared secrets. Specifically, the destination address is XORed with the last node's shared secret, truncated to match the address size. Next, the result is concatenated with the XORed values of the ending parts of the shared secrets from the other nodes in the path. This ensures that when the entire header is XORed with the full shared secret, these appended values cancel out, allowing the header to be processed in reverse order by the mixnodes. This design choice guarantees fixed-size headers, enabling fixed-size packets which is a crucial property in mixnets for maintaining unlinkability.

JT: Now maybe follow up with an example of the attack from the introduction? We should also compare computational overheads from attack vs. the new protocol design.

4 Multi-Party Computation (MPC)

The first approach to ensuring trust in the Sphinx header is to prevent user manipulation by decentralizing the header construction to Trusted Third Parties (TTP) through the use of Multi-Party Computaftion (MPC).

We consider TTPs as *honest-but-curious*. This means that they follow the protocol correctly but may attempt to infer additional information from the data they process. Our design aims to ensure that TTPs must not be able to infer any information about the shared secrets s_i nor the mixnodes involved in the path, even when TTP are colliding (assuming that at least one TTP remains honest).

What if malicous TTP...

JT: In the Nym ecosystem, who are the TTPs, who operates them, what exactly are they trusted for?

To decentralize the construction of the Sphinx header, we first examined how to partition and distribute the computation. Three approaches were considered.

The first and most naive approach involves that each TTP computes a different layer of the header. This approach reveals two consecutive nodes in the path and one of the corresponding shared secrets (s_i) at each TTP, leading to serious security concerns in the case of collusion.

In a second approach, the user sends to each TTP a piece of the destination address. Each TTP computes the same layer on its partial destination. The resulting partial headers of this layer are aggregated to compute the integrity tag. This process is repeated layer by layer. However, computing the integrity tag at the end of each layer requires knowledge of the shared secret s_i. If a TTP does it, it reintroduces the same problem as the first approach. If the user does it, he could potentially cheat.

User could send $h(s_i)$ to TPP such that it could computes the integrity check without getting useful information...

The third and retained approach is similar to the second one. Instead of partial computations at each layer, each TTP computes a full Sphinx header using only its assigned partial destination. These partial headers are then returned to the user, who aggregates them to produce the final Sphinx header. Although this approach requires less interaction with the TTPs, it introduces additional constraints as discussed in section .

ref section

The overall decentralized scheme is illustrated in Figure 5. The user begins by splitting the destination into several parts, such that their combination (e.g., via XOR) reconstructs the original address. Each part is sent to a different TTP, along with the required cryptographic material. The TTPs independently generate Sphinx headers using a modified version of the protocol. These partial headers are then returned to the user, who aggregates them to form the final Sphinx header, ready for transmission through the mixnet.

JT: I think we should only focus on the third approach. There could be a section "Alternative Designs" where we discuss your other approaches.

(see section ...)

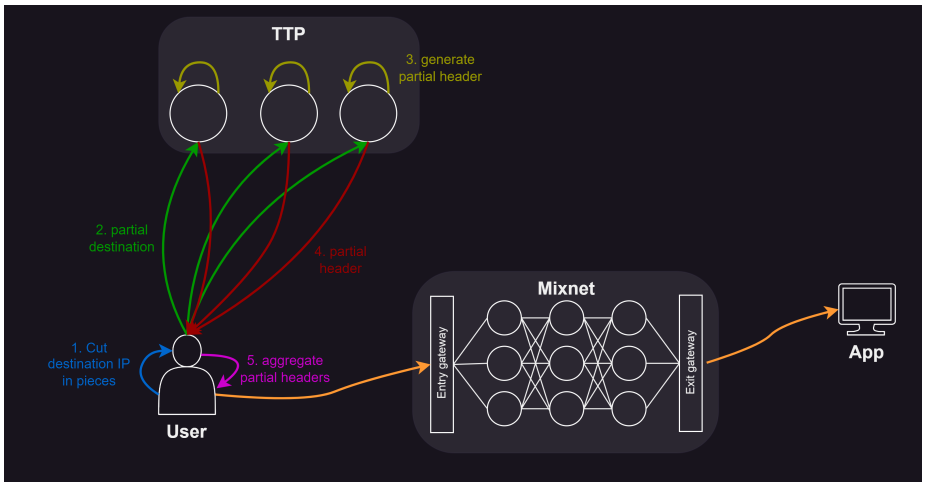


Figure 5: Overview of the decentralized scheme

The main issue with the choosen approach is that we have to compute integrity tag on partial header such that combining those partial integrity tag gives the integrity tag of the final header. However, even if breaking this homomorphic hash is feasible, if it remains computationally hard enough (e.g., requiring several hours), it could still be considered sufficiently secure for our purposes.

I'd rather specify the requirements for the hash here.

References

[1] Danezis, G., and Goldberg, I. Sphinx: A compact and provably secure mix format. In *2009 30th IEEE Symposium on Security and Privacy* (2009), pp. 269–282.

[2] Diaz, C., Halpin, H., and Kiayias, A. The nym network, 2021.

[3] Sonnino, A., Al-Bassam, M., Bano, S., Meiklejohn, S., and Danezis, G. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019* (February 2019), The Internet Society.