Generating trusted sphinx packets

aurelien.chassagne

December 2024

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:

- **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.
- **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.

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

2 Sphinx

As illustrated in Figure 1, the Sphinx packet consists of a header and an encrypted payload. The header contains a cryptographic element, encrypted rout-

ing information, and an integrity tag. The encrypted routing information is encapsulated with an integrity check as shown in Figure 2.

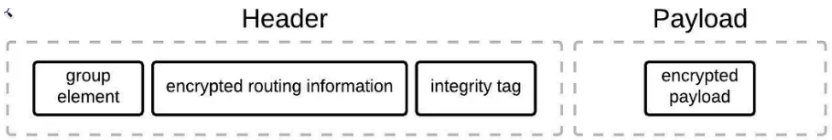

Figure 1: Structure of sphinx packet. [source]

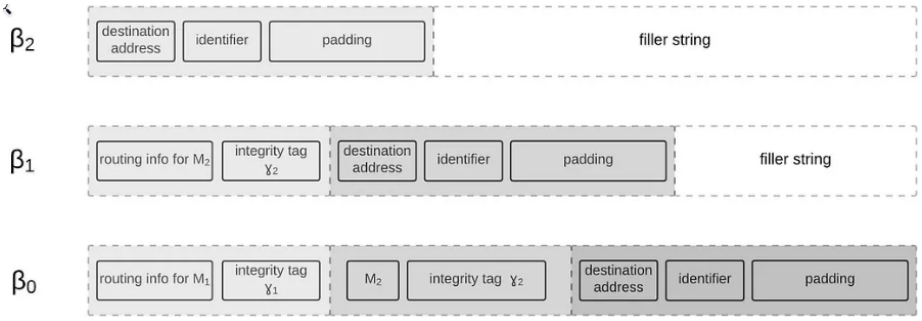

Figure 2: Sphinx encrypted routing information encapsulation. [source]

To encrypt the routing information, the user first chooses a random secret value x and compute 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: $s_i = y_i^x$. Then the mixnode i receiving the packet will get g^x allowing him to compute the shared secret as followed: $s_i = (g^x)^{x_i}$.

The encrypted routing information is computed, as illustrated in Figure 3, by processing the path in reverse order. This involves XORing the current routing information (β_{i-1}) 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).

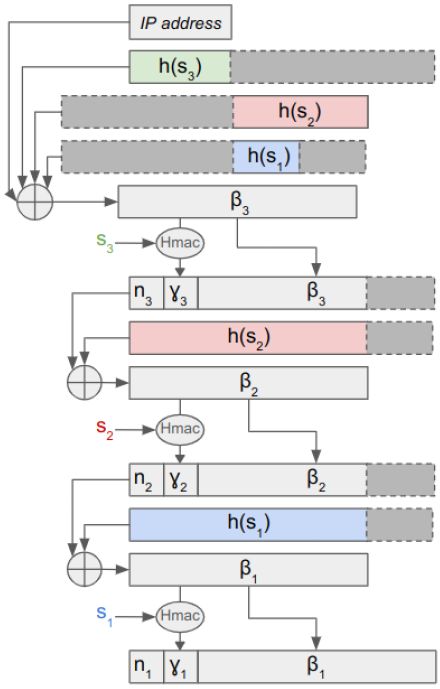

Figure 3: Construction of the Sphinx header (modified from [1])

We repeat the same process at each XOR step (i.e., additional nodes can easily be added to the path by following this pattern). However, the first XOR differs as it requires combining parts of all shared secrets. This deviation is a deliberate design choice to ensure fixed-size headers, enabling fixed-size packets which is a crucial property in mixnets for maintaining anonymity.

3 Zero-Knowledge Proof (ZKP)

If we focus solely on the final encrypted routing information (β), we can simplify the diagram from Figure 3 to the representation in Figure 4.[1] We observe that a small part of the header contains the destination address (δ) XORed with part of each shared secret. In other words, a substring β^* of the final header is equal to $\Delta \oplus h(s_3) \oplus h(s_2) \oplus h(s_1)$.

[1]It is important to note that Figure 4 shows the final representation of the encrypted routing information, not the computation, as the integrity tags (γ) still need to be computed.

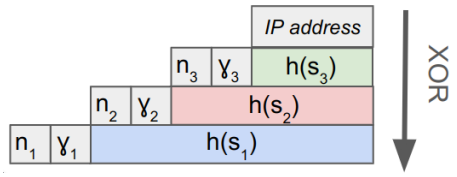

Figure 4: Representation of the final encrypted routing information (β)

Therefore, to user could prooves that the destination address used in the original header is indeed the one he is authorized by its credentials (i.e. to one linked to its credential) without revealing anything about neither the destination, neither the routing path. In other words, prooving that the Δ linked to the credential is the one used in $\beta^* = \Delta \oplus h(s_3) \oplus h(s_2) \oplus h(s_1)$. To recall, with the current Sphinx mixnet architecture, a user can specify any destination address Δ, and the packet will still traverse the entire mixnet before the fraud is detected at the exit gateway.

Idea Proving that the address Δ from the credentials is indeed the one used in the header (i.e. $\beta^* = \Delta \oplus h(s_3) \oplus h(s_2) \oplus h(s_1)$).

Limitations Zero-Knowledge proofs are efficient with addition and multiplication operators, but not with the XOR (bitwise) operator, as XOR requires a separate proof for each bit instead of a single proof for the entire operation.

Solutions To improve the efficiency of the ZK proof, we could replace the XOR in the original schema (Figure 3) with modular addition, measuring performance improvement, and ensuring it does not compromise security.

Discussion If a user want to exhaust mixnode computation, he can provide the right Δ with a wrong $h(s_3)$. It will exhaust computation of the two first mixnodes before beeing detected at the third, just before the exit gateway (1 hop earlier that previously). Thus we should look at a way to proof that $h(s_3)$ is correct as well... Otherwise, we can think of it differently. Is it really a problem if user exhaust mixnodes computation ? It depends, in case of light network traffic is not a big deal since it follows the role of 'dummy' packet, improving the anonymity.

Discussion If a user wants to exhaust mixnode computation, they can provide the correct Δ with an incorrect $h(s_3)$. This strategy would exhaust the computational resources of the first two mixnodes before being detected at the third one, just before the exit gateway (one hop earlier than previously). To address this, we should explore methods to prove the correctness of $h(s_3)$... Alternatively, we could reconsider whether this scenario is truly problematic. If a user exhausts mixnode computation, it may not be critical in cases of light network traffic, as it aligns with the role of 'dummy' packets in enhancing anonymity.

4 Multi-Party Computation (MPC)

The second approach to ensuring trust in the Sphinx header is to prevent user manipulation by decentralizing the scheme through the use of Multi-Party Computaftion (MPC).

One major challenge in decentralizing the current schema (Figure 3) lies in the integrity tag, which is the HMAC of the header β_i computed using the shared secret s_i. To ensure security, no third party should have access to s_i, as collusion among third parties could lead to the recovery of all s_i, enabling them to decrypt the header and payload.

To address this, third parties could compute a portion of the HMAC using a fragment of the shared secret s_i, and then combine these partial HMACs to produce the final HMAC. This approach would require a hash function with homomorphic properties, which inherently weakens the collusion resistance and potentially compromises the second pre-image resistance of a secure hash.

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.

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.