Generating trusted sphinx packets

Anonymous

Abstract We propose a decentralized scheme that prevent mixnets users from sending traffic that does not match the service provider of an anonymous credential. Our scheme is of direct use in the Nym network where users construct an anonymous credential given they receive a certificate of them paying for that service provider. Our scheme prevent users from cheating by using a credential for a free service and send traffic for a paid service. Our solution works even if the majority of the third parties collude. Finally we evaluate the performance of our solution.

Keywords: mixnet · sphinx · malicious users

Target: CBT Workshop on Cryptocurrencies and Blockchain Technology; deadline: 2025-06-17; `https://cbtworkshop.org/`

1 Introduction

Mix network, or mixnet, is an overlay network of servers (called mix nodes) that prevent an adversary from correlating senders with receivers [3, 4, 6, 8, 11, 14, 16, 17]. They achieve this by adding delays to messages inside the different mixnodes in order to mitigate timing analysis attacks. Additionally, unlike Tor [10] where all packets sent by a user follow the same path for the entire session (called circuit-based), in mixnets routing is typically packet-based (meaning that each packet follows a different path). These two techniques ensure that mixnets are resilient against a strong adversary who observes the entire input and outputs of the network typically called a Global Passive Adversary (GPA). During the last few decades several systems have been proposed in the literature and few of them have been implemented. Open research problems such as parameterization, authentication and fake dummies have halted the usage on a large scale. Recently, Nym Technologies a mixnet-based company is being commercialized and proposing a mixnet network for services to be integrated with their network for a fee. Their network is based on Loopix [17]. Users of these services are then allowed to use the Nym mixnet by using Nym credentials (based on the Coconut credential [18]) that they can construct after getting a certificate of paying for a specific service to use the Nym network. However, nothing prevents users from cheating who might exploit a valid Nym credential for another service they did not pay for. This is a particular difficult problem in anonymous communication networks where the mixnodes do not know the traffic type or the final service a user is communicating with by doing layered encryption where the final IP

address is only know by the last node in the path using a packer format such as Sphinx [7].

In this paper we present a scheme that creates the Sphinx header in a decentralized way based on trusted third parties while ensuring that these parties learn nothing about the destination or the path. Our work makes the following contributions:

- ...
- ...
- We assess the impact on availability and economic viability adversaries can have in Sphinx and our solution and conclude that...

Something on prototype and availability of artifacts!

For CBT: highlight relevance of the contribution to Lightning and Nym in intro and contributions.

We highlight related work and motivation in Section 2, then we specify and justify our system model in Section 3, where we also describe the considered threat model. We then present our scheme that decentralize the creation of the Sphinx headers in Section 4 and the evaluation of our proposed solution in Section 5. Finally we conclude and discuss future work in Section 6

2 Motivation and Related Work

Since Chaum's seminal work on untraceable email in 1981 [3], there has been a great amount of research related to mixnets' design [1, 3–6, 11–13, 17]. However most systems that have been deployed and used come with their own system on top of the network meaning that only one type of traffic type is allowed. As beautifully stated by Dingledine et al. in [9], "Anonymity Loves Company" meaning that the more messages there are in the network the more privacy the network provide. This is also shared by Ben Guirat et al. in [2] where the authors show that blending different traffic types on top of a mixnet provide better anonymity, meaning let's imagine an instant messaging system where users do not tolerate latency of more than few seconds and an email app where users tolerate latency of up to 1 minute. The authors show that blending these two types of traffic do actually increase the privacy for both traffic. This only applicable in networks such as Tor or the Nym network that they offer the network for different applications to be integrated on top of the network rather than dictating which application to use the mixnet. However, certain open research problems remain open. For example how can we ensure that certain traffic are not allowed (for whatever reason and we will specify the exact reason for our work) without compromising ? Tor solves the problem with having exit policy that simply drop traffic at the last node. However Tor routing is circuit-based meaning that all traffic from a specific user session follow one path and hence packets are encrypted using symmetric cryptography which is cheap, so even if traffic is dropped that is not a problem.

However, and beyond Tor, mixnets can impose substantial computational overheads due to packet based routing where every packet takes a different path and hence layered public-key cryptography is used. This implies that that, if packets are dropped at the last node of the mix, which is traditionally the only

place where service contracts can be validated, computational resources required by the mix to propagate a packet to its final node are wasted. Additionally, and unlike Tor where relaying traffic happens on a volunteering base, nodes in mixnets such as Nym are economically incentivized through payment per relayed bandwidth. Thus, dropping packets at an the exit node impacts the economic viability of the mixnet service.

3 System and Threat Model

We observed that adversarial interactions on mixnets can increase the use of computational resources and network resources, thus reducing the availability and negatively affecting economic viability of mixnet services. Our work seeks to mitigate these challenges. Below we describe the system model and the adversary we consider, and derive privacy and security requirements for our solution.

3.1 System Model

We abstract the Nym mixnet to the components that are relevant for our work. We consider a mixnet, where each user have a gateway that shows a credential and if accepted their traffic is allowed. To prevent correlation, mixnet relies on fixed-size packet format such as Sphinx packet, making it difficult for external observers to link incoming and outgoing messages at any given node. The Nym credential is constructed by the user and issued by a third decentralized party after obtaining a certificate that proves payment. For example let's say Signal is integrated with Nym, and Signal users who want their traffic to be anonymous instead of sending traffic directly to the Signal server, traffic will be first routed through the mixnet such that an adversary who observes the signal server and/or the device of the user can not correlate the sender with signal server and eventually the final recipient. Signal (service provider) can add an option for user who want to pay and issue a certified attribute to those users. Users then encode this attributes into a credential and sends it to validators. If the proof is valid, validators return partial signatures. Once the user collects a threshold number of these signatures, they aggregate them to form a valid credential and re-randomize it to ensure unlinkability from previous interactions. The user can then present this credential to a verifier to prove their right to access a service to show that the credential meets all necessary payment and authentication conditions. To prevent double-spending, the verifier checks that the credential has not already been used by consulting the blockchain and then commits the credential's serial number to the blockchain upon acceptance. For example, a user can obtain an certification from the Signal service provider, construct a valid credential and then use it to route traffic to another service provider they didn't pay for or simply not allowed (an illegal website). 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

Aurelien: I think we could try to be more concise on this subsection because it feels quite heavy (especially the second half speaking of coconut and blockchain). But for the moment, this part is a bit tricky since we haven't really focus on credentials...

Iness: add a simple graph here of a mixnet; purpose: illustrate system model, benign interactions, adversarial interactions.

wasted computational resources processing an invalid packet. This vulnerability enables Denial of Service (DoS) attack by exhausting mixnodes computational power with illegitimate packets.

Additionally, each encryption layer includes an integrity tag, which prevents tampering and improves the network's resistance against malicious mixnodes and active adversaries.

3.2 Threat Model

Aurelien: I feel like the bullet points are mixing a bit threat model (GPA, malicious user and honest-but-curious TTP) with desired properties (collusion resistant, integrity, unlinkability, ...)

We consider different types of adversaries:

- A GPA: an adversary who is able to observe all the inputs and outputs of the network. This adversary should not be able to correlate an input with an output based on the packet's appearance. This is achieved by the bitwise unlinkability of the sphinx packets.
- When constructing the headers: By using a Trusted Third Party that constructs the headers we need to make sure that even if the majority of the entities collude, no one should know the final destination of the user.
- An adversary who captures the headers is not able to change headers without intervening with the integrity check and hence mixes are able to know that the integrity check has been tampered with.
- Malicious users: Users can not cheat and create their own headers and putting the final destination different from the one they have the credential for.

3.3 Security & Privacy Objectives

O1. Cheating users: Users' traffic is only allowed to be routed if the traffic belongs to the same service provider from the credential
O2. even if the majority of headers issuers are colluding, they do not know which service provider the user is communicating with.
O3. the Spinx headers can not be altered.
O4. Verifiers can verify that the headers has not been altered without revealing the service provider.
O5. Unlinkability between sphinx packets, the original sphinx packet that is constructed in a centralized way provide the *unlinkability* property, meaning that an adversary can not know that two packets are connected to the same user. Our scheme that decentralize the headers creation aim at providing this same property.

Some objectives may need a bit of explanation re. why they matter, for others I'm not sure how they can be evaluated.

4 Our Solution - Multi-Party Computation (MPC)

Our approach to ensure 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 ensures that TTPs cannot infer any information about the shared secrets s_i or the involved mixnodes, even when TTPs collude (except one).

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

To facilitate explanation and illustration, we describe our solution such that each packet goes through a fixed-length path consisting of three mixnodes. However, the scheme is general and can be adapted to support arbitrary path lengths.

4.1 Header structure

In our approach, each piece of header information is encoded as an elliptic curve point. To achieve this, the encoded string is divided into fixed-size chunks, as illustrated in Figure 2, where each chunk is a single EC point. For example, in the case of a path consisting of three mixnodes, the resulting header contains seven EC points: one destination, three mixnodes and three integrity tags.

Therefore a method to encode and decode information to and from elliptic curve points is required. Specifically, we require a pseudo-random mapping between integers field and curve points group.

Traditional methods achieve this by directly mapping integers to the x-coordinates of points on the curve. However, this approach could leaks information by introducing bias since nearby integers result in nearby points.

To address this, we adopt Elligator, which offers stronger privacy guarantees. Unlike the traditional approach, Elligator produces a uniformly distributed output which is computationally indistinguishable from truly random curve points. This uniformity is critical in our context, where preserving anonymity and avoiding linkability are core goals.

4.2 Protocol description

The overall decentralized scheme is illustrated in Figure 1. The client first computes a sequence of shared secrets and then splits these secrets, along with the IP addresses of the mixnodes in the path and the final destination, into m shares. Each set of shares is sent to a different TTP, along with the necessary cryptographic element (α). Each TTP independently computes a *partial* Sphinx header using the received shares. The client then aggregates these partial headers to reconstruct the final header, ready for transmission through the mixnet.

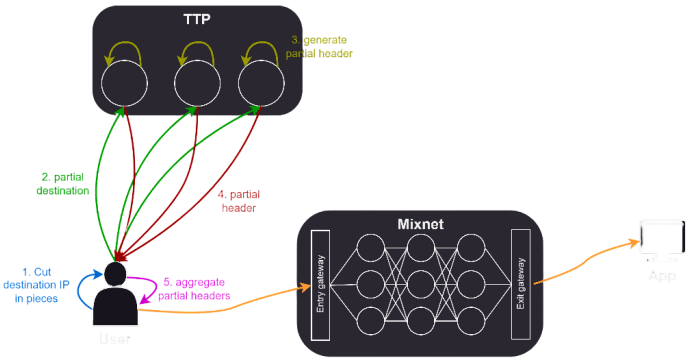

Figure 1. [DRAFT] Overview of the decentralized scheme

draft image

In summary, the protocol is divided into four main steps:

- **Setup**: fixes random points as generators (once but should be refreshed);
- **Client**: splits and shares the necessary information to TTPs;
- **TTP**: encrypts the routing information into a partial header;
- **Mixnode**: decrypts and forwards the header.

Setup In our protocol, routing information is divided into seven chunks, each one encoded into an elliptic curve point (see Figure 2). These points will later be encrypted by adding a masking point of the form $P = sG$, where s is a shared secret and G is the base generator of the curve. However, using the same masking point P for all the seven chunks compromises the *unlinkability* property (see security note in the TTP step 4.2).

To mitigate this, we use a set of seven *independent* generators G_j, one for each chunk. By *independent*, we mean that the scalar relationship between any two generators is unknown. Therefore no one knows the scalars $z_j \in \mathbb{Z}_N$ such that $G_j = z_j G$.

In principle, this setup phase can be executed only once. However, to preserve unlinkability, the set of generators should be refreshed periodically. Every entity (clients, mixnodes, and TTPs) must run the setup phase to ensure they use the same set of generators.

To achieve synchronized and deterministic generator values across the system, we derive them from a common seed. This seed is computed as the hash of the current timestamp, truncated to the chosen refresh interval. To produce distinct seeds for each generator we rehash the seed, or simply increment it since Elligator already provides a uniform (i.e. pseudo-random) integers to points mapping.

In the current implementation, instead of incrementing the seed, we rehash it for each i. However, since Elligator produces uniformly distributed outputs, incrementing may be equally secure and more efficient.

A subtlety with Elligator is that it can map integers to points outside of the base generator's subgroup. Thus, an extra step is required to *clear the cofactor* by multiplying the resulting point by the cofactor (Curve25519's cofactor is 8).

The final formula for generating the j^{th} chunk's generator is:

$$G_j = 8 \cdot H(\text{hash}(\text{timestamp}) + j), \quad \text{for } j = 1, \ldots, 7 \tag{1}$$

Client To send a message to a specific destination, the client first randomly chooses a path through the mixnet and generates a nonce. Using this nonce and the public keys of each mixnode along the path, the client derives a chain of shared secret integers (s_1, s_2, s_3) as follows:

$$(s_1, s_2, s_3) \begin{cases} \alpha_i = x_i\, G \\ S_i = x_i\, \text{PK}_i \\ s_i = h(S_i) \\ r_i = \text{hash}(\alpha_{iy} \parallel S_{iy}) \\ x_{i+1} = x_i\, r_i \pmod{n} \end{cases} \tag{2}$$

where x_1 is the client's nonce, PK_i is the public key of the i^{th} mixnode in the path, n is the subgroup order (i.e. $(n+1)G = G$), $\parallel$ is the concatenation operator and subscript y refers to the y-coordinate of the point. The scalar r_i acts as a pseudo-randomizer to update the secret x_i for the next hop, ensuring that each cryptographic element α_i remains unlinkable across mixnodes.

The IP addresses of the mixnodes and the destination are padded with random bits then mapped to elliptic curve points using Elligator, allowing later recovery of these addresses. Both the resulting IP points and the shared secrets are split into shares and distributed to m different TTPs.

Each TTP then independently computes a partial header from the received shares and returns it to the client. The client aggregates these partial headers by performing chunkwise point additions to reconstruct the complete encrypted header. Finally, the client sends the header along with α_1 to the first mixnode from the path.

Take a look if a schema could be interesting to illustrate Eq. 2

Test code: to verify if the padding is still necessary or optionnal and if limited (max size N)

Do we need to remind that the user need to encrypt his message using these shared secrets (as in original article) or is uninteresting/obvious enough ?

TTP Each TTP receives from the client:

- a share of the destination address (EC point),
- a share of each mixnode address in the path (EC points),
- a share of each shared secrets (integers).

The TTP constructs a partial encrypted header by layering routing information in reverse path order, starting from the last mixnode, as illustrated by Figure 2.

At each layer, two new chunks are introduced: the next mixnode's address (as an EC point) and an integrity tag. To preserve a fixed-length header while allowing mixnodes to reverse the operation (as in the original design), four additional *filler chunks* are appended. These filler chunks are carefully computed such that at each layer, after applying the corresponding masking points ($s_i G_j$),

the last two chunks cancel out and become identity point (i.e. point at infinity). This mechanism allows these two chunks to be safely truncated at each layer, ensuring a consistent header size while preserving reversibility for mixnodes.

To initialize the header, the TTP encrypts the destination point using the shared secret of the last mixnode and append the required filler chunks. Let Δ be the partial destination point, s_{i} the shared secret corresponding to the i^{th} mixnode in the path, and G_j the j^{th} chunk's generator. The chunks for the last layer (assuming a 3-hop path) are computed as follows:

$$\beta_3 \quad \begin{cases} \beta_{31} = \Delta + s_3 G_1 \\ \beta_{32} = -(s_2 G_4 + s_1 G_6) \\ \beta_{33} = -(s_2 G_5 + s_1 G_7) \\ \beta_{34} = -s_2 G_6 \\ \beta_{35} = -s_2 G_7 \end{cases} \tag{3}$$

Subsequent layers (for $i = 2$ and $i = 1$) are computed as:

$$\beta_{\mathrm{i}} \quad \begin{cases} \beta_{\mathrm{i}1} = N_{\mathrm{i}+1} + s_{\mathrm{i}} G_1 \\ \beta_{\mathrm{i}2} = \gamma_{\mathrm{i}+1} + s_{\mathrm{i}} G_2 \\ \beta_{\mathrm{i}3} = \beta_{\mathrm{i}+11} + s_{\mathrm{i}} G_3 \\ \beta_{\mathrm{i}4} = \beta_{\mathrm{i}+12} + s_{\mathrm{i}} G_4 \\ \beta_{\mathrm{i}5} = \beta_{\mathrm{i}+13} + s_{\mathrm{i}} G_5 \end{cases} \tag{4}$$

For each layer, the integrity tag is computed as:

$$\gamma_{\mathrm{i}} = s_{\mathrm{i}} G + \sum_{j=1}^{5} \beta_{\mathrm{ij}} \tag{5}$$

Here, N_{i+1} represents the elliptic curve point corresponding to the $(i+1)^{\mathrm{th}}$ mixnode in the path.

When the first layer is finally computed, it is appended with the integrity tag γ_1 and the first mixnode's point N_1. Finally, the TTP returns the constructed partial header to the client.

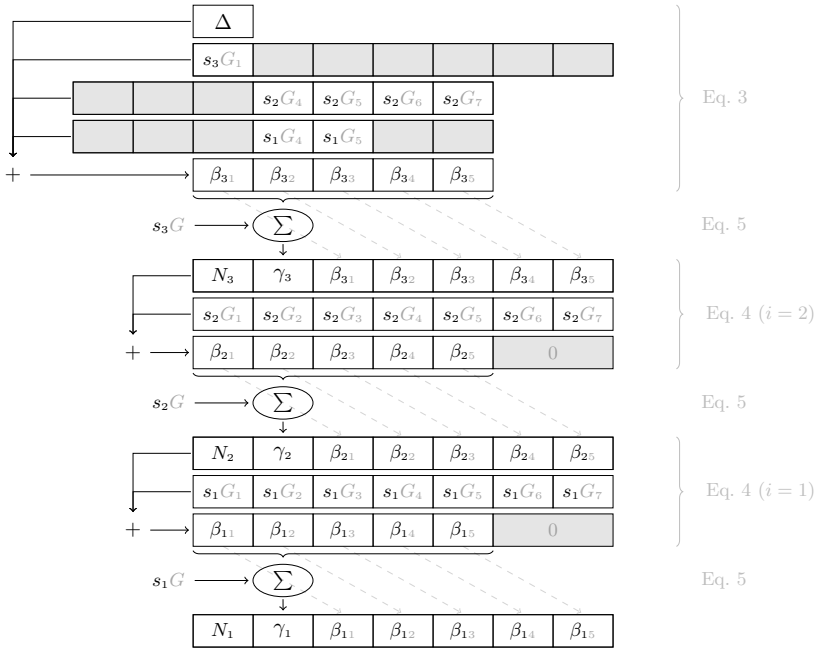

Figure 2. Chunkwise header encryption (TTP side)

Security Note. To preserve unlinkability, it is essential that the generators $(G_1, \ldots, G_7)$ are *independent*, meaning that their scalar relationships are unknown and cannot be derived. If the same generator G were used across chunks (or their scalar relationships known), an adversary could compute the chunkwise difference between consecutive layers: $\beta_{ij} - \beta_{i+1j} = s_i G$. Since the shared secret s_i remains consistent within a layer, the resulting differences would reveal a predictable pattern (uniform or preserved scalar relationships). This consistency could allow an adversary to correlate incoming and outgoing packets at a mixnode, thereby breaking the unlinkability property. Using independent generators for each chunk prevents such correlations and is therefore critical to the protocol's security.

Mixnode

When a packet arrives at mixnode i, it begins by extracting the relevant header fields, namely the cryptographic element α_i and the integrity tag γ_i. The mixnode then recomputes the shared secret point S_i using its private key sk_i and the cryptographic element α_i:

add index notation + dashed cut Beta block in Figure.

$$\begin{aligned} S_i &= \mathrm{sk}_i\, \alpha_i \\ s_i &= \mathrm{Elligator}(S_i) \end{aligned} \tag{6}$$

This point is subsequently mapped to a shared secret integer s_i using Elligator. This shared secret integer is then used to verify the integrity tag:

$$\gamma_i \stackrel{?}{=} s_i G + \sum_{j=1}^{5} \beta_{ij} \tag{7}$$

The integrity tag is constructed as the sum of the chunks, offset by a secret point derived from the shared secret integer. This ensures that only the client and intended mixnode can create, modify, or verify it.

Actually, there is a vulnerability since an active adversary could randomly modify the the header and the integrity tag accordingly...

If the integrity check passes, the mixnode pads the header by appending two *identity* chunks (i.e. point at infinity) and then updates it as:

$$\beta'_{ij} = \beta_{ij} - s_i G_j \qquad \forall j = 1, \ldots, 7 \tag{8}$$

The first chunk (β'_{i1}), encodes the next mixnode's IP address (N_{i+1}). This point is mapped back to an integer using Elligator and the random padding is removed by keeping the last 128 bits (i.e. size of an IP address).
The second chunk (β'_{i2}) is the next integrity tag (γ_{i+1}) for the remaining five chunks that form the new encrypted routing information (β_{i+1}).

To maintain unlinkability, the cryptographic element α_i must be updated for the next hop. As in the client's step, it is computed using the y-coordinates of both α_i and S_i:

$$\alpha_{i+1} = \text{hash}(\alpha_{iy} \parallel S_{iy})\ \alpha_i \tag{9}$$

Finally, the mixnode forwards the updated header (α_{i+1}, γ_{i+1}, β_{i+1}) to the next node.

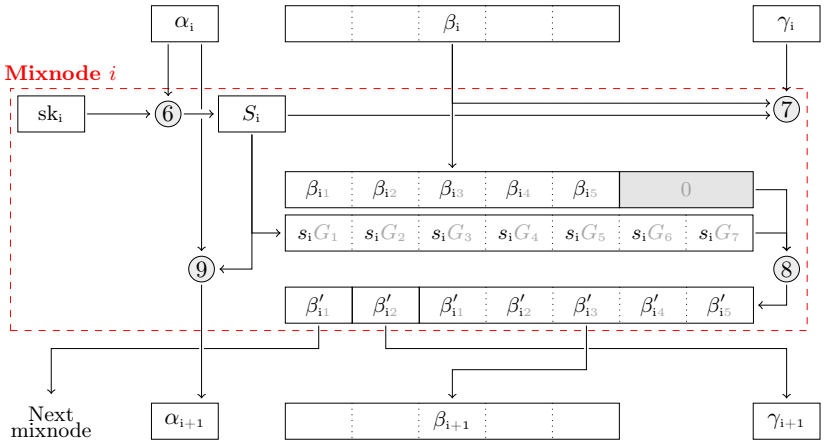

Figure 3. Processing sphinx header at mixnode i, where (x) refers to Eq. x.

Do you think it would be easier to understand/visualized the "mixnode step" with this kind of figure ?

5 Evaluation

Our prototype implementation is available on GitHub[1]. The implementation is written in Python 3.13 using the ECPy library for elliptic curve operations. ECPy is well-suited for prototyping due to its simplicity, but it is significantly slower than many EC python libraries (approximately one order of magnitude slower). Nevertheless, it remains one of the few Python libraries supporting Montgomery Curve25519, which is essential for our implementation.

We evaluate our system on two main axes: computational complexity and unlinkability (i.e. resistance to correlation between incoming and outgoing packets).

5.1 Computational Complexity

Instead of measuring raw execution time, which can vary significantly based on hardware and software environments, we focus on the number of expensive cryptographic operations. Table 5.1 count these operations per party for m TTPs and path of length p. To compute the total cost for sending one message, the TTP cost should be multiplied by m and the mixnode cost by p.

	Client	**TTP**	**Mixnode**
EC Multiplication	$\mathcal{O}(p\,m)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p)$
EC Addition	$\mathcal{O}(p\,m)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p)$
Point → Integer	$\mathcal{O}(p)$	0	2
Integer → Point	$\mathcal{O}(p)$	0	0

Table 1. Number of expensive cryptographic operations per party, where p is the path length (e.g. $p = 3$), and m is the number of TTPs.

Since long paths are typically unnecessary, we can approximate $\mathcal{O}(p)$ as constant (i.e. $\mathcal{O}(1)$). A path of length $p = 3$ is generally sufficient to ensure strong anonymity. Consequently, the overall computational burden is more sensitive to the number of TTPs m rather than the path length. Because our design remains secure as long as a single TTP is honest (i.e. does not collude), only a small number of TTPs may be required. A deeper analysis of how many TTPs are needed to ensure a desired level of trust remains an open question for future work.

Aurelien: Iness do you have some source to support this ? If yes, could you cite here, thanks.

5.2 Unlinkability assessment

Quantifying unlinkability is inherently challenging. However, under the assumption that uniformly random packet headers prevent adversaries from correlating

[1] `https://github.com/AurelienCha/Decentralized-Sphinx`

incoming and outgoing packets via cryptanalysis, unlinkability can be approximated by assessing the statistical randomness of the headers.

We rely on the NIST SP 800-22 statistical test suite [15], which includes 15 tests designed to assess the quality of random number generators. Each test returns a p-value representing how likely the data could be produced by a uniform source. If the outputs are truly random, the distribution of p-values across many samples should itself be uniform.

Add a brief description of these tests (i.e. what is evaluating) -> no, too long (or for appendix)

TODO: For the moment, only the first 4 of the 15 NIST tests have been implemented.

For our prototype, we simulate a network of 20 mixnodes and 3 TTPs. At each iteration:

- A random destination IP and path of 3 mixnodes are selected;
- The client generates shares and distributes them to the three TTPs;
- TTPs independently compute partial headers, which are then aggregated and forwarded through the simulated mixnet.

Each run produces 4 headers (one per hop). We perform 100 000 runs, resulting in a 400 000 x 1792 matrix (400 000 headers of 1792 bits). Then we apply the NIST tests to each:

- header independently (i.e. row-wise) to obtain p-value distributions across headers (Figure 4).
- bit independently (i.e. column-wise) to ensure no bias exists in specific bit positions (Figure 5).

For comparison, we conduct the same evaluation on the original Sphinx implementation by Danezis[2], which produces 400 000 headers of 1634 bits. Figures 4 and 5 compare header randomness of our implementation (in orange) with the original implementation (in blue).

[2] https://github.com/UCL-InfoSec/sphinx

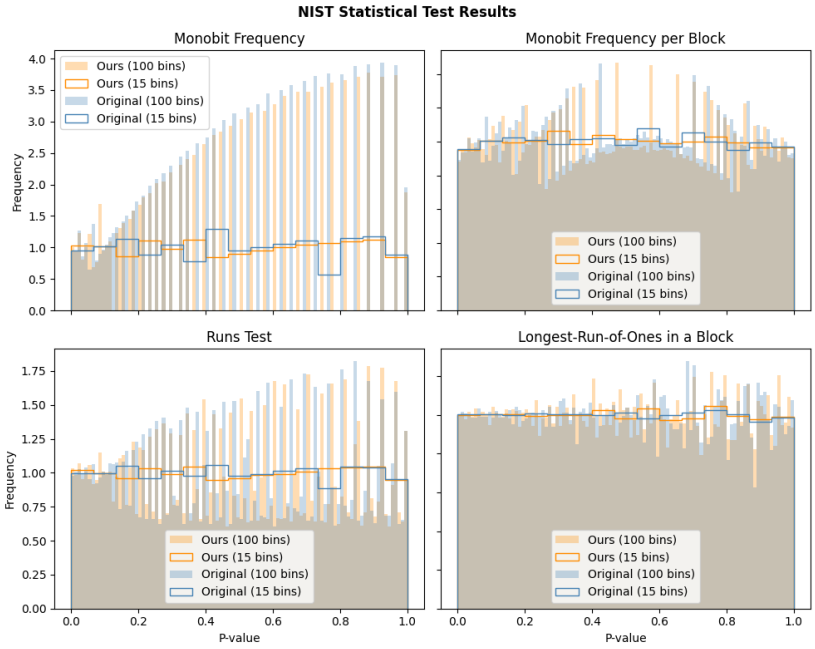

Figure 4. Distribution of NIST SP 800-22 p-values across 400 000 headers (headerwise evaluation).

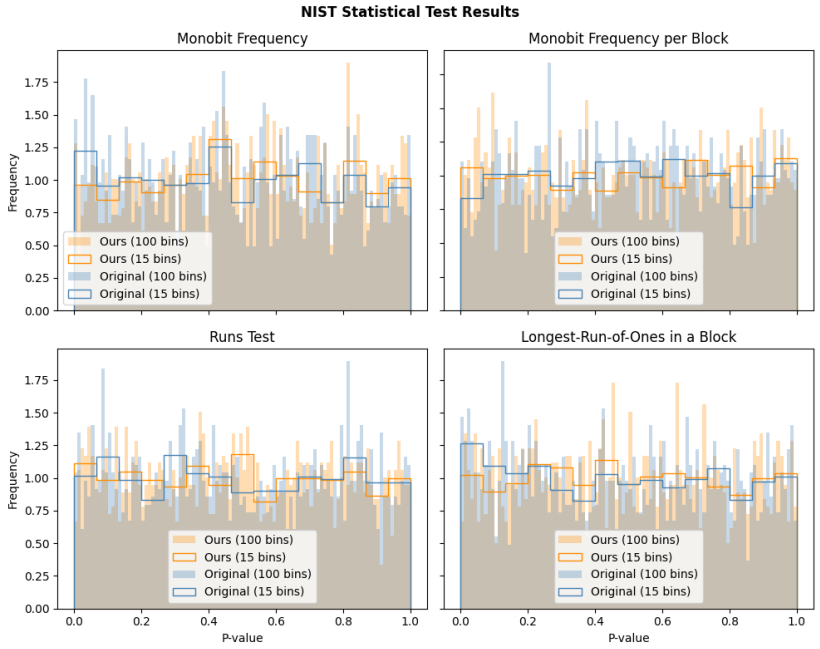

Figure 5. Distribution of NIST SP 800-22 p-values per bit across 400 000 headers (bitwise evaluation).

Some deviations in the histograms arise from the limited size of the bitstrings (1634 or 1792 bits), which introduces quantization effects. In particular, the slight distortion in the first test of Figure 4 reflects the granularity of possible p-values for short inputs. Variations in Figure 5 are due to the small amount of p-values (1634 or 1792). Increasing the histogram bin size mitigates these artifacts and confirms that the underlying distributions remain approximately uniform.

In both evaluations, the distributions of p-values are approximately uniform, which suggests that headers are statistically indistinguishable from random data, and therefore indicates strong linkability resistance, comparable to the original Sphinx implementation.

5.3 Discussion

Come back to 3.3 objectives and explain why they hold.

6 Conclusion

CBT: About 15 pages till here, 16 pages in total.

References

1. Alexopoulos, N., Kiayias, A., Talviste, R., Zacharias, T.: Mcmix: Anonymous messaging via secure multiparty computation. In: 26th {USENIX} Security Symposium ({USENIX} Security 17). pp. 1217–1234 (2017), https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/alexopoulos
2. BEN GUIRAT, I., Das, D., Diaz, C.: Blending different latency traffic with beta mixing. Proceedings on Privacy Enhancing Technologies (2023)
3. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**(2), 84–88 (1981). https://doi.org/10.1145/358549.358563, http://doi.acm.org/10.1145/358549.358563
4. Chaum, D., Javani, F., Kate, A., Krasnova, A., Ruiter, J., Sherman, A.T., Das, D.: cmix: Anonymization by high-performance scalable mixing. Tech. rep., Technical report (2016), http://www.cs.bham.ac.uk/~deruitej/papers/cmix.pdf
5. Cottrell, L.: Mixmaster and remailer attacks (1995), https://www.obscura.com/~loki/remailer-essay.html
6. Danezis, G., Dingledine, R., Mathewson, N.: Mixminion: Design of a Type III Anonymous Remailer Protocol. In: Proceedings of the 2003 IEEE Symposium on Security and Privacy. pp. 2–15. IEEE (May 2003)
7. Danezis, G., Goldberg, I.: Sphinx: A compact and provably secure mix format. In: 2009 30th IEEE Symposium on Security and Privacy. pp. 269–282 (2009), http://research.microsoft.com/en-us/um/people/gdane/papers/sphinx-eprint.pdf
8. Diaz, C., Halpin, H., Kiayias, A.: The Nym Network. https://nymtech.net/nym-whitepaper.pdf (February 2021)
9. Dingledine, R., Mathewson, N.: Anonymity loves company: Usability and the network effect. In: WEIS (2006)
10. Goldschlag, D.M., Reed, M.G., Syverson, P.F.: Hiding routing information. In: Proceedings of the First International Workshop on Information Hiding. p. 137–150. Springer-Verlag, Berlin, Heidelberg (1996)
11. Hooff, J.V.D., Lazar, D., Zaharia, M., Zeldovich, N.: Vuvuzela: Scalable private messaging resistant to traffic analysis. In: Proceedings of the 25th Symposium on Operating Systems Principles. pp. 137–152 (2015), https://people.csail.mit.edu/nickolai/papers/vandenhooff-vuvuzela.pdf
12. Kwon, A., Lu, D., Devadas, S.: {XRD}: Scalable messaging system with cryptographic privacy. In: 17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20). pp. 759–776 (2020)
13. Lazar, D., Gilad, Y., Zeldovich, N.: Karaoke: Distributed private messaging immune to passive traffic analysis. In: 13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18). pp. 711–725 (2018)
14. Möller, U., Cottrell, L., Palfrader, P., Sassaman, L.: Mixmaster Protocol — Version 2. IETF Internet Draft (July 2003)
15. NIST: A statistical test suite for random and pseudorandom number generators for cryptographic applications. Special Publication SP 800-22, National Institute of Standards and Technology (2010). https://doi.org/10.6028/NIST.SP.800-22r1a
16. Parekh, S.: Prospects for remailers. First Monday **1** (August 1996)
17. Piotrowska, A.M., Hayes, J., Elahi, T., Meiser, S., Danezis, G.: The loopix anonymity system. In: 26th {USENIX} Security Symposium ({USENIX} Security 17). pp. 1199–1216 (2017)

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