HOPR - a Decentralized and Metadata-Private Messaging Protocol with Incentives

Amira Bouguera, Robert Kiel, Dr. Sebastian Bürgel, Qianchen Yu

October 2021, v0.1

1 Introduction

Internet privacy is a subset of data privacy and a fundamental human right as defined by the United Nations [UN, 2018]. In order to retain and exercise this right to privacy, each internet user must have full control of the transmission, storage, use, and disclosure of their personally identifiable information. However, the infrastructure and economics of our increasingly technologically driven world put great pressure on privacy, due to the various advantages (economic and otherwise) which knowledge of users' private information provides.

To combat this trend, multiple solutions are being developed to restore internet users' control of their private information. These solutions generally centre on providing truly anonymous communication, such that neither the content of the communication nor information about the communicators can be discovered by anyone, including the communicating parties.

HOPR is a decentralized incentivized mixnet [Chaum, 1981] that employs privacy-by-design protocols. HOPR aims to protect users' transport-level metadata privacy, giving them the freedom to use online services safely and privately. HOPR leverages existing mechanisms such as the Sphinx packet format [Danezis and Goldberg, 2009] and packet mixing to achieve its privacy goals, but adds an innovative incentive framework to promote network growth and reliability. HOPR uses the Ethereum blockchain [Wood, 2021] to facilitate this incentive framework, specifically to perform probabilistic payments via payment channels.

1.1 HOPR Design Ethos

HOPR is built with the following principles in mind: that privacy must be an integral part of the design process (privacy by design); that privacy cannot be reliably achieved in centralized systems; and that decentralized systems must

be properly incentivized to provide privacy at a sufficient scale and level of reliability. The following section explains these three principles in more detail, as well as how HOPR meets them.

1.1.1 Privacy by design

Privacy by design is an approach to systems engineering which considers privacy throughout the entire engineering process, not merely as an afterthought. The internet is a public good – a digital commons that should be safe and secure to use for all users. However, it is impossible to provide such privacy using current internet infrastructure, which attaches little or no importance to metadata privacy, and indeed in many cases relies on copious amounts of metadata to be made public in order to function. Therefore, new infrastructure with a focus on privacy must be created on top of the existing internet. HOPR provides an essential part of that infrastructure, and has been built with privacy as its foremost goal.

1.1.2 Decentralization

The internet is not private by design, although it is decentralized by design. In the early days of the internet, this decentralization provided a certain measure of privacy, by placing control in the hands of individual users. However, internet users must increasingly interact with services provided by central authorities. These central authorities control the privacy of individual users, often without their full informed consent.

In particular, interactions with third-party service providers regularly entail a loss of transport-layer privacy. Links can often be drawn between the user and the service provider, either by the service provider themselves, or by a third party who can observe public metadata revealed during the interaction or pressure the service provider to divulge stored data. With a sufficient number of such links, a profile can be constructed of a user's online activity.

This runs contrary to the requirements of privacy as a fundamental human right, since often the only truly private option is to abstain from using such services entirely. To provide privacy as a fundamental feature on top of the internet, a decentralized infrastructure is required.

The HOPR protocol runs on nodes within a decentralized network, therefore ensuring that the network is independent. No single entity can influence its development or manipulate its performance to their advantage. It also makes the network resilient, able to keep running even if a majority of nodes are damaged or compromised and very difficult, if not impossible, to shut down.

1.1.3 Incentivization

Although users are incentivized to use privacy technologies by the privacy they provide, most privacy technologies have no baked-in incentive framework for infrastructure providers: rewards either flow to a centralized service provider (and thus the service is not truly private, since the service provider will gather transport-level information on user activity) or the providers in a decentralized network receive no direct rewards. This constrains the growth of the network and limits its scope. In systems which leverage privacy through obscurity, this reduced scale compromises the privacy of the entire network, since fewer users means less data to confound observers. Although blockchains provide a way, for the first time, to incentivize a decentralized network without introducing a centralized entity responsible for payment provision, it is challenging to leverage blockchain technology without compromising either the privacy of network members or the reliability of the network. In brief, if node runners can rely on the anonymity of the technology they are providing, they can potentially use it to claim rewards without properly fulfilling their duties as a node runner. On the other hand, if node runners are required to interact with a blockchain to prove their service provision and claim their reward, the resultant on-chain data is liable to provide a permanent and growing source of metadata which can be used to erode the users' privacy.

HOPR's proof-of-relay mechanism threads this needle, and is the major innovation of the HOPR protocol. Every HOPR node can receive payment for each packet they process and forward, but only once their data payload arrives at the next node. Payments are probabilistic and cheat-proof, which ensures nodes do not (de-)prioritize other nodes and/or packets. To maximize received incentives, node operators must ensure good network connectivity, node availability, computational capacity, and committed funds (to reward other nodes). These combined incentives promote a broad, robust, and reliable network. At the same time, the probabilistic nature of payments obscures links between on-chain data and node runners.

1.2 Security Goals

Generally, the HOPR protocol aims to hide the fact that two parties are communicating with each other, as well as the contents of the communication. This information should be hidden from any party external to the network, all intermediaries involved in transferring the data between the communicating parties, and even the communicating parties themselves, if one or both of them desires it.

Specifically, the HOPR protocol aims to build a network with the following features (definitions based on [Backes et al., 2013], [Piotrowska et al., 2017] and [Danezis and Goldberg, 2009]) for senders, indicated by A_n and recipients,

indicated by Z_n.

1.2.1 Sender anonymity

In a network with sender anonymity, an observer is not able to tell if a particular packet was sent by any adversary-selected honest senders A_1 or A_2 to the honest recipient Z. Formally this is denoted as $\{A_1 \rightarrow Z, A_2 \not\rightarrow\}$ or $\{A_1 \not\rightarrow, A_2 \rightarrow Z\}$.

1.2.2 Recipient anonymity

In analogy to sender anonymity, here an observer is not able to tell if a particular packet was received by any adversary-selected honest recipients Z_1 or Z_2 from the honest sender A. Formally this is denoted as $\{A \rightarrow Z_1, \not\rightarrow Z_2\}$ or $\{\not\rightarrow Z_1, A \rightarrow Z_2\}$.

1.2.3 Sender-recipient unlinkability

For any pair of senders, A_1 and A_2, communicating with any pair of recipients, Z_1 and Z_2, an adversary must be unable to determine whether two packets travelled from $\{A_1 \rightarrow Z_1, A_2 \rightarrow Z_2\}$ or $\{A_1 \rightarrow Z_2, A_2 \rightarrow Z_1\}$.

These properties assume senders and recipients are honest, but they should hold for any honest senders and recipients of the adversary's choice.

To provide these features, the HOPR protocol builds on top of the Sphinx packet format [Danezis and Goldberg, 2009], the specifics of which are explained in Section 3. HOPR also employs cover traffic, explained in Section 6. The additional parameters required for HOPR's incentive scheme are encapsulated in the Sphinx header as described in section 5.1 as additional routing parameters and as such do not change the privacy guarantees of Sphinx.

1.3 Threat Model

Although we assume that nodes in the HOPR network can communicate reliably, the network must still be protected from malicious actors and node failures. We assume a threat model with Byzantine nodes with the ability to either observe all network traffic and launch network attacks or to inject, drop, or delay packets as follows:

1.3.1 Global passive adversaries (GPAs)

A global passive adversary (GPA) is an attacker who can observe the entirety of network traffic passing between users. A GPA is considered passive because their attacks are based on observation alone. A theoretical GPA is arbitrarily powerful, and their full abilities are unlikely to be manifest in any real-world attacker. Nonetheless, building a network which is GPA-resistant introduces extra security through redundancy and future-proofing.

Thanks to the properties of Sphinx (see Section 3) and proof of relay (see Section 5.1), HOPR is able to defend against GPAs. Nonetheless, it is worth mentioning some additional attacks that a decentralized mixnet like HOPR is particularly vulnerable to by virtue of its decentralized and anonymous nature.

1.3.2 Sybil attacks

In a Sybil attack, an attacker forges multiple identities in the network, thereby introducing network redundancy and reducing system security. The attacker can potentially de-anonymize packet traffic and thus link the sender and recipient's identities. HOPR mitigates Sybil attacks via the trust assumption of the Sphinx packet format: only a single honest relayer is needed to ensure integrity of the entire transmission chain, and since the traffic is source routed, users can choose routes themselves to ensure this minimal requirement of one honest relayer is met. Since path selection is dependant on a financial stake, launching a Sybil attack would be prohibitively expensive.

1.3.3 Eclipse attacks

Many decentralized networks suffer from a general unavailability of a general understanding of their topology, hence nodes cannot determine whether their respective local views are complete or accurate. As an attacker, it might be attractive to flood a victim with inaccurate information about collaborating nodes while withholding information about honest nodes.

HOPR mitigates this issue by using a different medium to announce entry nodes to the network. This is done using a smart contract on a blockchain and known within HOPR as DEADR (Decentralized Entry Advertisement and Distributed Relaying). DEADR nodes serve two functions: (1) providing access to the distributed hash table (DHT) that is used within the libp2p environment that HOPR leverages and (2) facilitating network address translation (NAT) or relaying traffic that cannot be sent directly between adjacent nodes who reside in separate internet sub-networks (e.g., computers without a public internet-facing internet protocol (IP) address behind typical home routers). A HOPR node only requires access to one honest DEADR node and it is significantly cheaper

in computational terms to check whether a DEADR node is honest than to conduct an eclipse attack, which requires on-chain transactions. A successful eclipse attack therefore requires controlling all announced DEADR nodes or forging a public blockchain, both of which we assume are impossible and beyond the scope of this work.

Thus we are satisfied that HOPR more than adequately protects users against attack. However, security cannot come at the expense of usability. To be appealing to users and node runners alike, it is important for HOPR to find a satisfactory compromise to what is known as the "anonymity trilemma" [Das et al., 2018], which states that a privacy network can provide at most two of the following properties: low latency communication, low bandwidth overhead, and strong anonymity.

The rest of this paper will outline the current state of the art among anonymous communication protocols and layer-2 scalability protocols, describe how the HOPR network is built, and explain how HOPR provides a satisfactory resolution to the anonymity trilemma.

1.4 State of the Art

HOPR has been built on top of, or as a reaction to, various technologies and research projects which try to meet some or all of the design ethos and security goals outlined above. These technologies and research projects can be broadly divided into two groups: (1) anonymous communication protocols which attempt to hide user information; (2) layer-2 scalability protocols which allow systems to perform micro-payments via the Ethereum blockchain. The following sections present the state of the art for each of these groups.

1.4.1 Anonymous Communication Protocols

Virtual Private Networks (*VPNs*) are point-to-point overlay networks used to create secure connections over an otherwise untrusted network, such as the internet [Venkateswaran, 2001]. Clients receive a public IP address from the VPN endpoint, which is then used for all outgoing communication. Since this public IP address can be almost anywhere in the world, VPNs are often used to bypass censorship [Hobbs and Roberts, 2018] and geolocalization blocks. However, VPNs provide dubious privacy, since clients must trust the VPN endpoint. The endpoint provider has full access to users' connection and communication metadata, since they control all communication going through it. In addition, VPN endpoints are often a single point of failure, since they are provided as centralized services. Both factors are major weaknesses when considering VPNs for privacy-preserving communication.

These privacy concerns have motivated the creation of a new model for VPNs, **Decentralized Virtual Private Networks** (*dVPN*s). dVPNs leverage blockchain technology to remove reliance on a central authority. Users act as bandwidth providers, allowing other users to access internet services via their node, in exchange for rewards.

Examples of dVPN projects include Orchid [Cannell et al., 2019], Sentinel [Sentinel, 2021], and Mysterium [Mysterium, 2017]. However, these projects still lack privacy, performance, and reliability guarantees, since bandwidth providers can potentially inspect, log, and share any of their traffic. This harms honest bandwidth providers as well as users: since bandwidth providers can theoretically monitor traffic, nodes whose bandwidth are used to enable illicit activity can potentially be held liable for those activities by government authorities. Indeed, there have been incidents where unaware dVPN users have been (ab)used as exit nodes through which DDoS attacks were performed.

A research project called VPN^0 [Varvello et al., 2019] aims to provide better privacy guarantees. VPN^0 leverages zero-knowledge proofs to hide traffic content to relay nodes with traffic accounting and traffic blaming capabilities as a way to combat the weaknesses of other dVPN solutions.

Onion Routing is another approach to network privacy, implemented by projects such as Tor [Dingledine et al., 2004]. Tor encapsulates messages in layers of encryption and transmits them through a series of network nodes called *onion routers*. However, Tor is susceptible to end-to-end correlation attacks from adversaries who can eavesdrop on the communication channels. These attacks reveal a wide range of information, including the identity of the communicating peers.

The Invisible Internet Project (*I2P*) is an implementation of onion routing with notably different characteristics than Tor [Astolfi et al., 2015]:

- **Packet switched instead of circuit switched** Tor allocates connection to long-lived circuits; this allocation does not change until either the connection or the circuit closes. In contrast, routers in I2P maintain multiple tunnels per destination. This significantly increases scalability and failure resistance since packets are used in parallel.
- **Unidirectional tunnels** Employing unidirectional rather than bidirectional tunnels makes deanonymization harder, because tunnel participants see half as much data and need two sets of peers to be profiled.
- **Peer profiles instead of directory authorities** I2P's network information is stored in a DHT (information in the DHT is inherently untrusted) while Tor's relay network is managed by a set of nine Directory Authorities.

Despite these improvements, I2P is vulnerable to eclipse attacks since no I2P router has a full view of the global network (similar to other peer-to-peer networks). Like Tor, I2P only provides protection against local adversaries, making it vulnerable to timing, intersection, and traffic analysis attacks. I2P has also been shown to be vulnerable to Sybil and predecessor attacks, in spite of various countermeasures implemented to thwart these.

Mixnets are overlay networks of so-called *mix nodes* which route packets anonymously, similarly to Tor [Chaum, 1981]. Mixnets originally used a cascade topology where each node would receive a batch of encrypted messages, decrypt them, randomly permute packets, and transfer them in parallel. Cascade topology makes it easy to prove the anonymity properties of a given mixnet design for a particular mix. However, it does not scale well with respect to increasing mixnet traffic and is susceptible to traffic attacks. Since then, research has evolved to provide solutions with low latency while still providing high anonymity by using a method called *cover traffic*. Cover traffic is designed to hide communication packets among random noise. An external adversary able to observe the packet flow should not be able to distinguish communication packets from these cover traffic packets, increasing privacy.

The application of cover traffic provides metadata protection from global network adversaries, a considerable improvement on projects such as Tor and I2P. However, because mixnets add extra latency to network traffic, they are better suited to applications that are not as sensitive to increased latency, such as messaging or email applications. For applications such as real-time video streaming, Tor may be more suitable, provided the user feels the latency improvements outweigh the increased privacy risks.

Loopix [Piotrowska et al., 2017] is another research project which uses cover traffic to resist traffic analysis while still achieving low latency. To achieve this, Loopix employs a mixing strategy called *Poisson mix*. Poisson mix nodes independently delay packets, making packets unlinkable via timing analysis.

1.4.2 Layer-2 Scalability Protocols

Blockchain technology (mostly public blockchains like Bitcoin and Ethereum) suffers from a major scalability issue: every node in the network needs to process every transaction, validate them, and store a copy of the entire chain state. Thus the number of transactions cannot exceed that of a single node. For Ethereum, this is currently around 30 transactions per second.

Multiple solutions have been proposed to resolve the scalability issue, including sharding and off-chain computation. Both approaches intend to create a second layer of computation to reduce the load on the blockchain mainnet.

Off-chain solutions such as Plasma [Poon and Buterin, 2017], Truebit [Teutsch and Reitwießner, 2019], and state channels process transactions outside the blockchain while still guaranteeing a sufficient level of security and finality. State channels are better known as "payment channels". In models like the Lightning Network [Poon and Dryja, 2016], a payment channel is opened between two parties by committing a funding transaction. Those parties may then make any number of signed transactions that update the channel's funds without broadcasting those to the blockchain. The channel is eventually closed by broadcasting the final version of the settlement transaction. The channel balance is updated by creating a new set of commitment transactions, followed by trade revocation keys which render the previous set of commitment transactions unusable. Both parties always have the option to "cash out" by submitting the latest commitment transaction to the blockchain. If one party tries to cheat by submitting an outdated commitment transaction, the other party can use the corresponding revocation key to take all the funds in the channel.

As soon as closure is initiated, the channel can no longer be used to route payments. There are different channel closure transactions depending on whether both parties agree on closing the channel. If both agree, they provide a digital signature that authorizes this cooperative settlement transaction. In the case where they disagree or only one party is online, a unilateral closure is initiated without the cooperation of the other party. This is done by broadcasting a "commitment transaction". Both parties will receive their portion of the money in the channel, but the party that initiates the closure must wait for a certain delay to receive their money. This delay time is negotiated by the nodes before the channel is opened, for the protection of both parties.

The Raiden network [Raiden Team, 2016] is a layer-2 payment solution for the Ethereum blockchain. The project employs the same technological innovations pioneered by the Bitcoin Lightning Network by facilitating transactions off-chain, but provides support for all ERC-20 compliant tokens. Raiden differs in operation from its main chain because it does not require global consensus. However, to preserve the integrity of transactions, Raiden powers token transfers using digital signatures and hash-locks. Known as **balance proofs**, this type of token exchange uses payment channels. Raiden also introduces "mediated transfers", which allow nodes to send payments to another node without opening a direct channel to it. These payment channels are updated with absolute amounts, whereas HOPR employs relative amounts (more details in Section 4).

1.5 Assessment

Although there are numerous projects attempting to solve the problem of online privacy, most fail to satisfy the design principles and security goals outlined in Section 1.2.

Privacy The metadata privacy properties that a mixnet like HOPR provides are significantly beyond what VPNs, dVPNs, or even Tor can deliver. VPNs and dVPNs have single points of trust and failure. Tor and I2P have been shown to be subject to a large variety of de-anonymizing attacks, which a mixnet like HOPR is resilient against. The HOPR mixnet unlinks sender and recipient even when faced by powerful global passive adversaries (GPAs) which can monitor every packet in the network. This is achieved by packet transformation, mixing, and introducing delays before forwarding a packet, as well as bandwidth overhead in form of cover traffic and packet padding.

Decentralization Most projects presented above have decentralization as a core goal (with the exception of VPNs, where the service is provided by a centralized entity, at the expense of user privacy). However, many projects lack the impetus to scale, which prevents them from leveraging many of the advantages which come with a decentralized network, such as privacy through obscurity, while retaining all of the challenges, such as coordination and consensus issues. It is our opinion that the only way to solve the scalability problem is with the introduction of a robust incentivization scheme for node runners.

Incentivization Many projects referenced in this section lack incentivization of any kind. VPNs incentivize the service provider, but since this is a centralized entity this fails the privacy condition.

Bandwidth providers in dVPNs share their resources and are granted tokens accordingly as payment for their services. For example, Mysterium [Mysterium, 2017], an open-source dVPN built on top of a P2P architecture, uses a smart contract on top of Ethereum to ensure that the VPN service is adequately funded. However, the resultant liability risk for dVPNs means these incentives are likely to be insufficient.

Tor and I2P rely on donations and government funding, which only covers the cost of running a node, not any additional reward. This has discouraged volunteers from joining these networks and the number of relayers in both networks has stayed mostly static in recent years, even as usage and the demand for privacy has risen sharply.

Mixnet designs are generally unconcerned with the problem of incentivization, and most existing implementations rely on a group of volunteer agents who lack incentives to participate. However, it is possible to add incentivization on top of a mixnet design, which is what HOPR does. Some authors have proposed adding digital coins to packet delivery in mixnets [Atkinson and Silaghi, 2007]. However, here the anonymity provided by the mixnet acts as a double-edged sword: since there is no verification for whether a packet arrives at its final destination, and node identities are kept secret by the mixnet, selfish nodes can extract payment without performing their relaying task or without passing on acknoledgements to

the next node. Thus a naive implementation of this approach does not actually provide an incentive at all.

It is in this last area that HOPR provides its main innovation: HOPR's proof-of-relay mechanism ensures that node runners are only paid if they complete their relaying duties. The challenge is to enforce this without publicizing data which would allow an adversary to break the anonymity of the network, and to provide both incentivization and anonymity without introducing unacceptable latency and bandwidth costs. The remainder of this paper will explain how this is achieved.

2 Paths and Path Selection

HOPR currently utilizes free-route sender-selected paths which are known to provide better privacy than other topologies, as well as better packet delivery in the event of partial network failure [Dingledine et al., 2005]. HOPR makes it easy to customize various different routing strategies. In this section, we present the free-routing strategy that HOPR nodes are currently following. Another strategy is later introduced in the cover traffic section.

Packets in the HOPR network are sent via multiple *hops* before they are delivered to their final destination. To meet HOPR's stated security goals, it is crucial for nodes in the network to keep the chosen path secret.

As relay nodes differ in their ability to forward packets, path choices need to consider the nodes' resources and cannot be chosen entirely at random.

Path selection is therefore done in multiple steps: first, there is a preprocessing step to assign each payment channel a score, which by default equals to its balance (the number of HOPR tokens locked in a payment channel). Here we use the balance that a node has staked in a particular payment channel as a signal for how much traffic the node is willing to relay in a fashion that aligns its incentives with the rest of the network.

The score is then multiplied by a random number to yield the channel's final weight. This randomization helps reduce linkability, which is a risk in a purely deterministic selection process. Finally, a path is chosen to the destination based on these scores while considering the relative availability of intermediate hops in a depth-first greedy search.

This chapter explains node score assignment based on a node's payment channel balances, randomization of the edge weights of the graph, and how nodes employ path selection to find a path through the HOPR network based on randomized channel weights and node availability.

2.1 Node score assignment

The HOPR network is kept open and permissionless, hence there is no gatekeeper which grants access to the network and maintains a full list of active nodes. For this reason, it is necessary for network participants to maintain a solid understanding of the network topology to decide which nodes they can use to relay their mixnet packets.

The network topology includes two layers: the payment channel graph and the ability of nodes to establish a network connection to each other. Whilst the former is easily constructed by observing and indexing on-chain interactions, the latter needs active communication with other nodes to check their availability. Since the network status can change abruptly, e.g., due to electricity outages or unstable network links, availability needs to be measured frequently on an ongoing basis.

2.1.1 Node availabilty

The availability of nodes in the network is estimated by regularly pinging other nodes in the network and listening to ping attempts from other nodes. Receiving a *ping* request or a *pong* response from another node means that a node is available and gradually increases their network score.

If a node does not respond to *ping* attempts, their network score decreases and the waiting time to the next *ping* attempt increases exponentially. This exponential backoff occurs until an upper bound is reached. A successful response to *ping* resets the backoff timer.

$$t_{bo} = \exp(t_{base}, \exp(f_{bo}, n_{fail}))$$

where t_{bo} is the backoff time, t_{base} is the initial backoff time, f_{bo} is the backoff factor, and n_{fail} is the number of failed attempts since the last sucessful attempt or the network start.

When the network score is below the network quality threshold, the node is considered offline. All incoming and outgoing channels of this node are ignored and the node will not be utilized in the path selection process.

2.1.2 Payment channel graph and stake

Incentives for relaying mixnet packet require the existence of an open payment channel that is funded towards the relay node. Hence, it is necessary to be in sync with the latest on-chain state.

Funding a payment channel with HOPR tokens means staking these tokens in the HOPR network. The more tokens a node stakes to an outgoing payment channel, the more likely the payment channel is selected as a path, hence the more likely the node is chosen as a relayer by other nodes.

2.2 Randomization

Since the previous step is entirely deterministic and potential adversaries could thus deduce the path of an individual packet from public information, each node score is randomized by multiplying it with a random number, r_i. Over long time scales, the expectation value of the weight is still proportional to the node's score and thus predictable.

$$weight(n_i) := score(n_i) * r_i$$

Note that the random numbers are assigned once at the beginning of the process and reassigned upon the selection of a subsequent packet.

2.3 Path selection

The HOPR network uses source-selected routing. This means a node must sample the entire path the mixnet packet will take before sending it to the first relayer.

To achieve the aforementioned privacy guarantees, paths must include at least one intermediate node. However, using the Sphinx packet format (see Section 3) increases both the size of the header and the computation needed to generate it, which makes it possible for adversaries to distinguish packets on longer paths. Therefore, all nodes in the HOPR network are strongly encouraged to use a $targetLength$ of three. Packets with longer paths are dropped by relayers. It is possible but discouraged for nodes to use paths of one or two hops.

Nodes can only be chosen once per path. For example, when choosing the third node in the path $A \rightarrow B$, if A is found to be the only node with an open channel to B, the search will fail and a new path will be generated.

The algorithm terminates once a path with $targetLength$ is found. To prevent the algorithm from visiting too many nodes, the number of iterations is bound by $maxIterations$ and the longest known path is returned if no path of length

targetLength was found.

Algorithm 1: Path selection

Input: nodes V, edges E

$queue \leftarrow$ **new** $\mathsf{PriorityQueue}_{\mathsf{pathWeight}}()$
$queue.addAll(\{(x, y) \in E \mid x = self)$
$closed \leftarrow \emptyset$
$iterations \leftarrow 0$

while $size(queue) > 0$ and $iterations < maxIterations$ **do**
  $path \leftarrow queue.peek()$
  if $length(path) = targetLength$ **then**
    return $path$
  end

  $current \leftarrow last(path)$
  $open \leftarrow \emptyset$

  foreach $next \in \{y \in V \mid (x, y) \in E \ \wedge \ x = current\}$ **do**
    if $score(next) > threshold$ **and** $y \notin closed$ **and** $y \notin path$ **then**
      $open.push(y)$
    end
  end

  if $size(open) = 0$ **then**
    $queue.pop()$
    $closed.add(current)$
  else
    foreach $node \in open$ **do**
      $queue.push(\ (...path, node)\ )$
    end
  end

  $iterations \leftarrow iterations + 1$
end
return $queue.peek()$

3 Sphinx Packet Format

HOPR uses the Sphinx packet format [Danezis and Goldberg, 2009] to encapsulate and route data packets in order to ensure sender and recipient unlinkability. The Sphinx packet format determines how mixnet packets are created and transformed before relaying them to the next downstream node in a way that does not

leak path information to relayers or other parties. Each Sphinx packet consists of two parts, a header and an onion-encrypted payload:

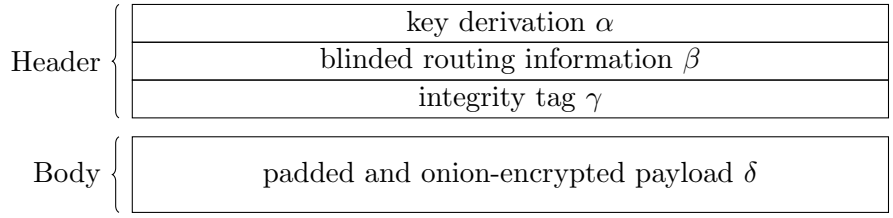

Figure 1: Schematic overview of a SPHINX packet

3.1 Construction

The following explains how a Sphinx packet is created and how it is transformed at each hop before arriving at its final destination. At first, the sender chooses a path (see Section 2), and derives shared keys with each node on the path. The shared keys serve as a master secret to derive subkeys. These subkeys are used to blind the routing information in such a way that nodes can solely determine the next downstream node, and also to create an authentication tag to check the integrity of the header. In addition, the sender applies one layer of encryption for each node along the chosen path to the payload. This accomplished, the sender finally sends the packet to the first hop.

Once a node receives a mixnet packet, it first derives the key that it has shared with the sender of the packet and checks the integrity of the header. Next, it unblinds the routing information to determine the next downstream node and removes one layer of encryption from the encrypted payload. At this point, the node is able to decide whether it is the final recipient of the message or it is supposed to forward the packet to the next hop.

3.1.1 Key derivation

The sender A picks a random $x \in \mathbb{Z}_q^*$ that is used to derive new keys for every packet.

A randomly picks a path consisting of intermediate nodes B, C, D, and the packet's final destination, Z.

A performs an offline Diffie-Hellman (DH) key exchange with each of these nodes and derives shared keys with each of them.

A computes a sequence of tuples $(\alpha_i, s_i, b_i) \in \{B, C, D, Z\}$ with $\alpha_A = g^x$ and $b_A = h_b(\alpha_A, y_B^x)$ as follows:

$$
\begin{aligned}
(\alpha_B, s_B, b_B) &= (g^{xb_A}, y_B^{xb_A}, h_b(\alpha_B, s_B)) \\
(\alpha_C, s_C, b_C) &= (g^{xb_Ab_B}, y_C^{xb_Ab_B}, h_b(\alpha_C, s_C)) \\
(\alpha_D, s_D, b_D) &= (g^{xb_Ab_Bb_C}, y_D^{xb_Ab_Bb_C}, h_b(\alpha_D, s_D)) \\
(\alpha_Z, s_Z, b_Z) &= (g^{xb_Ab_Bb_Cb_D}, y_Z^{xb_Ab_Bb_Cb_D}, h_b(\alpha_Z, s_Z))
\end{aligned} \tag{1}
$$

where y_B, y_C, y_D, and y_Z are the public keys of the nodes B, C, D, and Z, respectively, which we assume to be available to A.

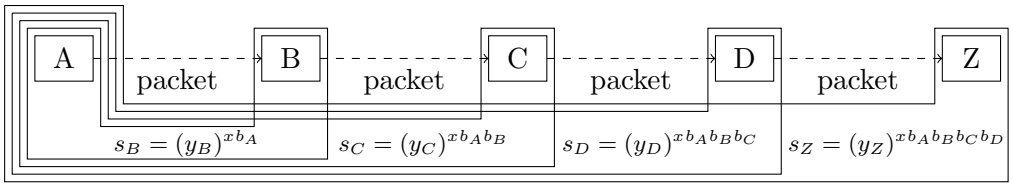

Figure 2: A derives shared keys s_B, s_C, s_D, s_Z with node B, C, D, Z using their public keys y_B, y_C, y_D, y_Z.

3.1.2 Routing information

Each node on the path needs to know which is the next downstream node. Therefore, the sender A generates routing information β_i for B, C, and D, as well as a message END to tell Z that it is the final recipient of the message. Whilst the keys for B, C, and D are given as a compressed elliptic, the END message occurs as a distinguished prefix.

In general, compression of elliptic curve points given as (x, y) happens by taking the x-coordinate and the sign of y. If y is above the x-axis, `0x02` is added, otherwise `0x03` is used. The END message is given by `0x04` and is therefore easily distinguishable from public keys.

The routing information is computed as follows:

$$
\beta_{v-1} = (y_Z \| 0_{(2(r-v)+2)\kappa - |y_Z|} \oplus \rho(h_\rho(s_{v-1}))_{[0....(2(r-v)+3)\kappa-1]}) \| \phi_{v-1} \tag{2}
$$

and

$$
\beta_i = y_{i+1} \| \gamma_{i+1} \| \beta_{i+1_{[0....(2r-1)\kappa-1]}} \oplus \rho(h_\rho(s_i))_{[0....(2r+1)\kappa-1]} \tag{3}
$$

$0 \leq i < v - 1$

such that y_Z is the destination's public key in compressed form (since this is only the x-coordinate, it is 33 bytes instead of 64) and $|y_Z|$ is its length. ρ is a

pseudorandom generator (PRG) and h_ρ is the hash function used to key ρ. $v \leq r$ is the length of the path traversed by the packet, where $|y_Z| \leq (2(r-v)+2)$. ϕ is a filler string such that

$$\phi_i = \{\phi_{i-1} \| 0_{2\kappa}\} \oplus \rho(h_\rho(s_{i-1}))_{[\,(2(r-i)+3)\kappa..(2r+3)\kappa-1\,]} \tag{4}$$

where $\phi_0 = \epsilon$ is an empty string. ϕ_i is generated using the shared secret s_{i-1} and used to ensure the header packets remain constant in size as layers of encryption are added or removed. Upon receiving a packet, the processing node extracts the information destined for it from the route information and the per-hop payload. The extraction is performed by deobfuscating and left-shifting the field. Ordinarily, this would make the field shorter at each hop, allowing an attacker to deduce the route length. For this reason, the field is pre-padded before forwarding. Since the padding is part of the HMAC, the origin node will have to pre-generate an identical padding (to that generated at each hop) in order to compute the HMACs correctly for each hop.

β_i is computed as the concatenation of y_Z and a sequence of padding which is then encrypted by XORing with the output of a PRG seeded with shared key s_{v-1} of node $v-1$. The result is finally concatenated with ϕ to ensure the header packets remain constant in size.

In the original Sphinx paper, y_Z is concatenated with an identifier I and 0 padding sequence, where I is used for SURBs (single-use reply blocks) such that $I \in \{0,1\}^\kappa$. We do not use I since HOPR does not currently employ SURBs. We do, however, include *hint* and *challenge* values in β, defined in the proof of relay section. These values are not included in the original Sphinx paper but are needed for the HOPR protocol. Since A has a shared secret with each of the nodes along the path, it is able to derive blindings for each of them. Each node along the path receives an authentication tag γ_i in the form of a message authentication code (MAC), which is encoded in the header.

Padding is added at each mix stage in order to keep the length of the message invariant at each hop.

The mix header is constructed as follows:

$$M_i = (\alpha_i, \beta_i, \gamma_i) \tag{5}$$

A sends the mix header M_0 to B. Once B receives the packet, it derives the shared key s_0 by computing

$$s_0 = (\alpha_0)^b = (g^x)^b = (g^b)^x = y_B^x$$

and removes its blindings. Here b is the private key of node B. This allows B to unblind the routing info that tells B the public key of the next downstream node, C. The process happens in the same fashion for all further downstream nodes after B.

3.1.3 Integrity check

The integrity check allows the node to verify whether or not the header has been modified. By using the derived shared secret s_i, each node is able to recompute the authentication tag and check the integrity of the received packet as follows:

$$\gamma_i = HMAC(s_i, \beta_i) \tag{6}$$

B computes the keyed hash of the encrypted routing information β_0 as

$$\gamma_0 = HMAC(s_0, \beta_0)$$

and compares with the integrity tag γ_0 attached in the packet header. If the integrity check fails, it is assumed the header has been tampered with and the packet is dropped. Otherwise, the mix node proceeds to the unblinding step. The HOPR packet header contains one integrity tag γ_i for each node along the path.

3.1.4 Unblinding

The unblinding works as follows: B decrypts the attached β_0 in order to extract the routing instructions. First, B appends a zero-byte padding at the end of β_0 and decrypts the padded block of routing information β by XORing it with $PRG(s_0)$ as follows:

$$(\beta_0 \| 0_{2\kappa}) \oplus \rho(h_\rho(s_0)) \tag{7}$$

B parses the routing instructions from A in order to obtain the address of the next mix node, C, as well the new integrity tags γ_1 and β_1, which should be forwarded to the next hop.

3.1.5 Delete and shift

After B extracts the public key of C, it deletes the routing information from the packet. B then fills the empty space with its own blinding (which is different from the one received from A) by setting the key share α_0 to $\alpha_1 = g^{xb_0}$. B also computes β_1 as follows:

The first κ bits of β_0 will be n_1 itself, the next κ bits will be γ_1, and the remaining $(2r-1)\kappa$ bits of β_0 are shifted left to form the leftmost $(2r-1)\kappa$ bits of β_1; the rightmost 2κ bits of β_1 are simply a substring of an output of the PRG function.

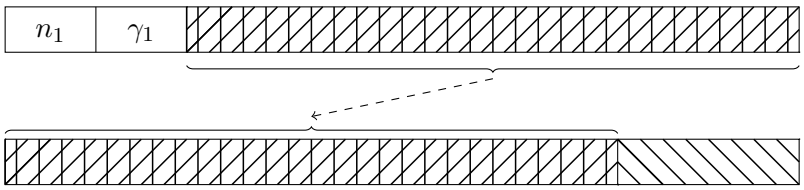

Figure 3: Shifting in the header

The new mix header is now ready to be sent to C, defined as the node with public key y_1:

$$M_1 = (\alpha_1, \beta_1, \gamma_1)$$

where α, β and γ are defined in equations 1, 3 and 6

3.1.6 Encrypt and decrypt

In contrast to the header, the integrity of the payload is not directly protected by the protocol. To ensure potential manipulations of the message remain visible to the final recipient, the content of the payload is hidden using a pseudorandom permutation scheme (PRP) and its inverse is used to undo the transformation. This comes with the property that if there were any modifications to the payload, such as a bit flip, the probability that the decoded message contains any relevant information is expected to be negligible.

To implement the PRP, HOPR uses the LIONESS [Anderson and Biham, 1996] wide-block cipher scheme, instantiated by using Chacha20 as a stream cipher and BLAKE2s as a hash function as suggested by Katzenpost. See appendix B for a detailed description and the chosen parameters.

As seen in Section 3.1.1, while creating the packet the sender derives a shared key s_i with each node along the chosen path and uses them to create subkeys s_i^{prp} to key the PRP. See Appendix A for more details about the key derivation.

To allow the final recipient to determine whether a message is meaningful content or not, each message is padded by a protocol-specific tag τ and 0s to fit the packet size of 500 bytes, yielding m_{pad}. Decoded payloads that do not include τ are considered invalid and should be dropped.

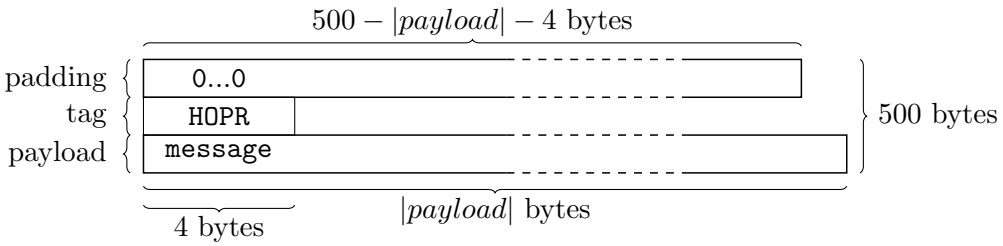

Figure 4: Padded message consisting of 0-padding, protocol tag τ (`0x484f5052`, ASCII-encoded "HOPR"), and payload m.

The sender takes the padded message and encrypts it using PRP.permutate with the derived subkeys in reverse order $(..., s_{i+1}^{prp}, s_i^{prp}, s_{i-1}^{prp}, ...)$

$$\delta_i = \mathsf{PRP.permutate}_{s_i^{prp}}(\delta_{i+1})$$

where $\delta_4 = m_{pad}$ and δ_0 is the ciphertext that is sent to the first relayer when using three intermediate hops.

Each node n_i along the chosen path then removes one layer of encryption by setting

$$\delta_{i-1} = \mathsf{PRP.inverse}_{s_i^{prp}}(\delta_i)$$

yielding $\delta_0 = m_{pad}$ in case the node is the final recipient.

3.2 Implementation choices

HOPR employs the following cryptographic primitives:

- **Cyclic group** HOPR's Sphinx implementation uses an elliptic curve group on the secp256k1 curve. Operations are therefore performed on the elliptic curve.
- **Hash function** HOPR uses the BLAKE2s hash function, a cryptographic hash function faster than SHA-2 and SHA-3, yet at least as secure as SHA-3. It produces digests of 32 bytes.
- **MAC** HOPR uses HMAC based on the BLAKE2s hash function.
- **Encryption scheme** HOPR uses the LIONESS [Anderson and Biham, 1996] implementation, using BLAKE2s as a hash function and ChaCha20 as a stream cipher.
- **Padding** The original Sphinx paper uses a sequence of 0s for padding. However, this allows the last mix node in the path to infer information

about the length of the path and the final destination, hence breaking one of the security properties promised by Sphinx. In order to prevent this attack, HOPR replaces the 0-padding with randomized padding for the final mix node when $v < r$. This ensures the exit node cannot identify where the padding starts and thus will not be able to determine the path length. In the case where $v = r$ there is no need to add padding as the length of the path is the maximum length, and thus no additional information is being revealed.

4 Tickets

In the HOPR protocol, nodes that have staked funds within a payment channel can issue tickets that are used for payment to other nodes. Tickets are used for probabilistic payments; every ticket is bound to a specific payment channel and cannot be spent elsewhere. Tickets are redeemable at most once and lose their value when the associated payment channel is closed or when the commitment is reset. A commitment is a secret on-chain value used to verify whether a ticket is a winner or not when an attempt is made to redeem it.

4.1 Ticket Issuance

A ticket can be issued when two nodes have established a payment channel with each other. By definition this means at least one of them has staked HOPR tokens.

The ticket issuer A (who could also be the packet creator) sets a winning probability and relay fee to use and sets the amount to:

$$\sigma = \frac{L \times F}{P_w}$$

where σ is the amount of HOPR tokens set in the ticket, L is the path length, F is the relay fee, and P_w is the ticket's winning probability. These are currently static values which apply network wide.

A issues a ticket for the next downstream node. The challenge is given together with the routing information by the packet. A does not know whether the ticket is a winner or not.

A sets the content of the ticket to:

$$t = (R, \sigma, P_w, \alpha, I, T_c, \zeta),$$

where t has the following components, in addition to those already defined above:

- **Recipient's Ethereum address** R: a unique identifier derived from the ticket recipient's public key.
- **Ticket epoch** α: used as a mechanism to prevent cheating by turning non-winning tickets into winning ones. This is done by increasing the value of α whenever a node resets a commitment, which helps keep track of updates to the on-chain commitments and invalidates tickets from earlier epochs.
- **Ticket index** I: set by the ticket issuer and increases with every issued ticket. The recipient verifies that the index increases with every packet and drops any packets where this is not the case. Redeeming a ticket with index n invalidates all tickets with index $I < n$, hence the relayer has a strong incentive to not accept tickets with an unchanged index.
- **Ticket challenge** T_c: set by the ticket issuer and used to check whether a ticket is redeemable before the packet is been relayed. If it is not redeemable, the packet is dropped.
- **Channel epoch** ζ: used to give each incarnation of the payment channel a new identifier such that tickets issued for previous instances of the channel become invalid once a channel is reopened (α's count restarts again). This is due to the fact that ζ increments whenever a closed channel is (re)opened.

A then signs the ticket with its private key and sends $T = (t, Sig_I(t))$ to the recipient together with a mixnet packet.

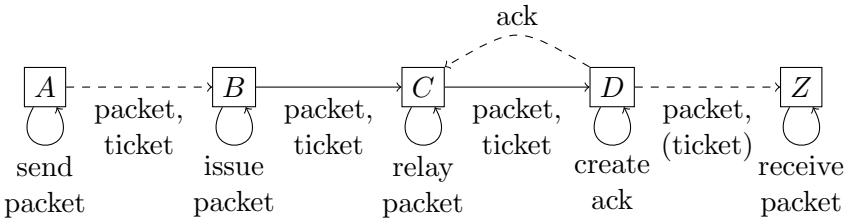

Figure 5: Ticket workflow

4.2 Ticket Validation

Tickets are received together with packets. To this end, the recipient and the next downstream node share a secret, s, whose key shares s_i and s_{i+1} are derivable by those nodes as introduced in section key derivation. Ticket validation requires the following steps:

Validate response Once A receives $s_{i+1}^{(1)}$ from B via secret sharing, it can compute

$$r_i = s_i^{(0)} + s_{i+1}^{(1)}$$

where r_i is the response r at iteration i such that it verifies

$$r_i * G = T_{c_i}$$

Validate hint Once the recipient transforms the packet, it can compute s_i. The recipient can now also extract the routing information from the packet. This includes a hint to the value s_{i+1}, given as

$$H_i = s_{i+1}^{(1)} * G,$$

which is stored in the Sphinx packet header.

The unacknowledged ticket is stored in the database under the hint to the promised value to ensure the acknowledgement can be later linked to the unacknowledged ticket.

Together with $s_i^{(0)}$, the node can verify that

$$T_{c_i} = s_i^{(0)} * G + H_i$$

with

$$s_i^{(0)} * G + H_i = s_i^{(0)} * G + s_{i+1}^{(1)} * G = (s_i^{(0)} + s_{i+1}^{(1)}) * G$$

This allows the recipient to verify that the promised value $s_{i+1}^{(1)}$ indeed leads to a solution for the challenge given in the ticket. If this is not the case, the node should drop the packet. Without this check, the sender could intentionally create false challenges which lead to unredeemable tickets.

4.3 Ticket Redemption

In order to unlock a ticket, the recipient node stores the ticket within its database until it receives an acknowledgement containing s_{i+1} from the next downstream node. HOPR uses acknowledgements to prove the correct transformation of mixnet packets as well as their delivery to the next downstream node.

The challenge can be computed from the acknowledgement as $T_{c_i} = Ack_i * G$. The node checks the following in order to redeem its ticket:

- **Stored ticket** Once the node receives an acknowledgement, it checks whether it is storing an unacknowledged ticket corresponding to the received acknowledgement. If it does not, the node should drop the acknowledgement.

The node then computes the response to the challenge (also referred to as the *proof of relay secret*) given in the ticket as

$$r_i = (s_i^{(0)} + s_{i+1}^{(1)}) * G$$

- **Sufficient challenge information** The node checks whether the information gained from the packet transformation is sufficient to fulfil the challenge sent along with the ticket,

  $$r_i * G = T_{c_i}$$

  The node then replies with an acknowledgement which includes a response to the challenge. If this is not the case, the node should drop the packet.

Both the node and the smart contract then perform the following checks:

- **Channel existence** The node checks that the appropriate channel **exists** and is **open** or **pending to close**. If these checks do not pass, the node should drop the packet and the smart contract check will revert. To prevent metadata leakage, the check happens locally rather than on-chain using the blockchain indexer. If the node has no record of the channel or considers the channel to be in a state different from **open** or **pending to close**, the ticket is dropped and receipt of the accompanying packet is rejected.
- **Commitment value check** Additionally, the node and the smart contract retrieve the next commitment value, $comm_{i-1}$. They then check that this value is not empty and that the commitment is the opening of the next commitment as follows:

  $$comm_{i-1} \neq 0 \; and \; comm_i = h(comm_{i-1})$$

- **Commitment verification** The node then verifies that r_i and $comm_{i-1}$ lead to a winning ticket. This is the case if

  $$h(t_h, comm_{i-1}, r_i) < P_w$$

  where $t_h = h(t)$ is the ticket hash. The values are first ABI encoded, then hashed using keccak256 and last but not least converted to uint256.

  If the check is not valid, the node should drop the packet. The final recipient of the packet does not receive a ticket because packet receipt is not incentivized by the HOPR protocol.

- **Issuer identity** The ticket signer must be same as the ticket issuer. Both node and smart contract verify whether the public key associated to the private key used to sign

  $$T = (t, Sig_I(t))$$

  is the issuer's public key. The packet is dropped if this test fails and ticket redemption reverts.

- **Prior redemption** The ticket must not have been already redeemed and the ticket index must be strictly greater than the current value in the smart contract (replay protection).

$$I_c < I$$

where I_c is the current value in the smart contract and I is the ticket index.

- **Valid ticket amount** The amount of ticket must be greater than 0:

$$\sigma > 0$$

where σ is the ticket amount.

- **Liquidity check** The channel must have enough funds to cover the value transfer for the ticket:

$$C_b > \sigma$$

where C_b is the channel balance.

Finally, the smart contract also performs an epoch check:

- **Epoch check** The ticket epoch, α, and channel epoch, ζ, must be equal to the current values in the smart contract

$$\alpha = \alpha_c \ \ and \ \ \zeta = \zeta_c$$

where α_c and ζ_c represent the current values in the smart contract.

5 Incentivization Mechanism

The HOPR protocol provides incentives to nodes in the network to achieve correct transformation and delivery of mixnet packets. This is accomplished through a combination of proof of relay, a novel mechanism which is cost effective and privacy preserving, and probabilistic payments, together with an on-chain commitment. The high-level overview of the motivation behind this incentive scheme was covered in the introduction. This section focuses on the technical details used to implement the mechanism.

Construction

- Every packet is sent together with a ticket.
- Each ticket contains a challenge.
- The validity of a ticket can be checked on receipt of the packet, but the on-chain logic enforces a solution to the challenge stated in the ticket.
- The challenge can be solved after the packet has been forwarded.

5.1 Proof of Relay

HOPR incentivizes packet transformation and delivery using a mechanism called **proof of relay**. This mechanism guarantees that a node's relay services are verifiable.

Secret sharing Each node derives two keys, s_i^{own} and $_i^{ack}$, by using the s_i as given by the SPHINX packet (see the key derivation section for more details). s_i^{own} and s_{i+1}^{ack} serve as key shares of a 2-out-of-2 secret sharing between a node n_i and the next downstream node n_{i+1} along the chosen path. Once a node knows *both* key shares, s_i^{own} and s_{i+1}^{ack}, it is able to reconstruct $s_i^{response}$ to redeem the received ticket on-chain.

Whilst s_i^{own} is derivable upon reception of a packet, s_{i+1}^{ack} requires the cooperation of the next downstream node n_{i+1} and is sent as an *acknowledgement* if n_{i+1} has received the transformed packet and considers both the packet and embedded ticket valid.

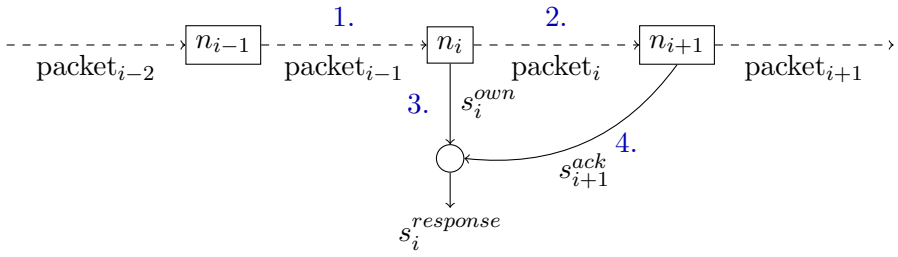

Figure 6: 1. Node n_i receives $packet_{i-1}$, validates it, transforms it and 2. sends it to node n_{i+1}. 3. While processing $packet_i$, node n_i derives s_i^{own} and 1. once node n_{i+1} considers $packet_i$ valid, it *acknowledges* the receipt of $packet_i$ and thereby reveals s_{i+1}^{ack} to node n_i which allows node n_i to reconstruct $s_i^{response}$.

Challenge response Tickets sent next to a mixnet packet include a challenge C_i which is computed as

$$C_i = s_i^{response} \cdot G = (s_i^{own} + s_{i+1}^{ack}) \cdot G$$

where G refers to the base point and $\cdot$ means scalar multiplication on the curve. Hence, in order to solve the challenge, it is necessary to know s_i^{own} as well as s_{i+1}^{ack}.

Challenge and Hint Once a node receives a packet, it is able to derive s_i^{own} but it is unable to decide whether s_{i+i}^{ack} will ever lead to $s_i^{response}$ that solves the challenge. Since the underlying field preserves the distributivity, it holds that

$$C_i = s_i^{response} \cdot G = (s_i^{own} + s_{i+1}^{ack}) \cdot G = s_i^{own} \cdot G + s_{i+1}^{ack} \cdot G$$

Hence by knowing $hint_i = s_{i+1}^{ack} \cdot G$, the node can verify that $s_i^{own} \cdot G + hint_i = C_i$ and thereby check that the creator of the challenge must have known a value $\tilde{s}_{i+1}^{ack}$ that led to $hint_i$. Due to the infeasibility of inverting scalar multiplication on the chosen curve, knowing $hint_i$ does not reveal s_{i+1}^{ack}. Hence, by embedding $hint_i$ into the part of β within the SPHINX packet that is readable by node n_i, the sender makes the validity of the embedded challenge verifiable.

As the next downstream node would not accept a packet without a ticket[1], the node n_i not only needs to transform the packet but must also issue a ticket to the next downstream node n_{i+1}. Therefore, it needs to know which challenge $\tilde{C}_{i+1}$ to put into the ticket issued for node n_{i+1}. As in the previous section, this is done with the help of the creator of the packet, who embeds C_{i+1} into the part of the SPHINX packet that is readable by node n_{i+1}.

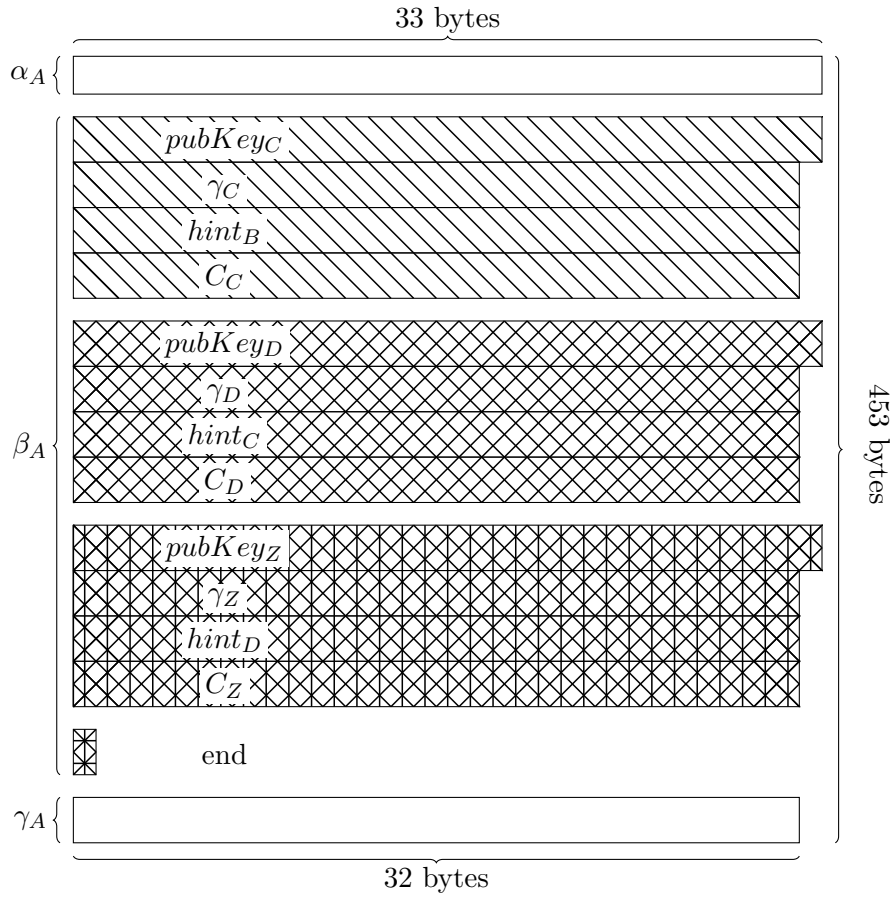

Figure 7: Mixnet packet header with PoR fields that is sent from the sender A to B and supposed to be forwarded through nodes C, D to Z.

[1]By default, nodes only forward packets that include incentives. Nevertheless, the protocol does not prevent them from processing packets without enforcing an incentive.

By chaining this principle, nodes are forced to *always* issue a ticket to the next downstream node because they are unable to claim their own incentive without the help of the next downstream node.

Since the very last node, namely the final receiver, of the packet, does not need to forward the packet to anyone else, it has no direct incentive to acknowledge tickets, hence there are no direct consequences for not acknowledging packet. In its current version, the protocol does not prevent this kind of behaviour, but research is being conducted into the necessity and feasibility of solving this issue via a reputation system.

5.2 On-chain Commitment

HOPR uses a commitment scheme to deposit values on-chain and reveal them once a node redeems an incentive for relaying packets. This comes with the benefit that the redeeming party must disclose a secret that is unknown to the issuer of the incentive until it is claimed on-chain. The **opening** Open and the **response** r to the proof-of-relay challenge (called *nextCommitment* and *proofOfRelaySecret* in the smart contract) are then used to prove to the smart contract that the node has a legitimate claim to the funds. HOPR uses a computationally hiding and binding commitment scheme:

Commitment phase Once a node is the destination of a HOPR unidirectional channel, it generates a master key $comm_0$ randomly and uses it to create an iterated commitment $comm_i$ such that for every $i \in \mathbb{N}_0$ and $i > 0$ it holds that

$$\mathsf{Open}(comm_i, comm_{i-1}) = \top,$$

which means opening $comm_i$ with $comm_{i-1}$ holds true.

The iterated commitment is computed as

$$comm_n = h^n(comm_0),$$

where h is a pre-image resistant hash function (we use the keccak256 hash function, which is also used in Ethereum). The master key should be pseudorandom, such that all intermediate commitments $comm_i$ for $i \in \mathbb{N}_0$ and $0 < i \leq n$ are indistinguishable for the ticket issuer from random numbers of the same length. This is necessary to ensure that the ticket issuer is unable to determine whether a ticket is a winner or not when issuing the ticket. This makes it infeasible for the ticket issuer to tweak the challenge to such that it cannot be a winner.

When dispatching a transaction that opens the payment channel, the commitment $comm_n$ is stored in the channel structure in the smart contract and the smart contract will force the ticket recipient to reveal $comm_{n-1}$ when redeeming a ticket issued in this channel. The number of iterations n can be chosen as

a constant and should reflect the number of tickets a node intends to redeem within a channel.

Opening phase In order to redeem a ticket, a node must reveal the opening to the current commitment $comm_i$ that is stored in the smart contract for the channel. Since the opening $comm_{i-1}$ allows the ticket issuer to determine whether a ticket is going to be a winner, the ticket recipient should keep $comm_{i-1}$ until it is used to redeem a ticket. Tickets lead to a win if:

$$\mathsf{h}(t_h, r_i, comm_{i-1}) < P_w,$$

where P_w is the ticket's winning probability and

$$t_h = \mathsf{h}(t) \; and \; \mathsf{Open}(comm_i, comm_{i-1}) = \top.$$

Since $comm_0$ is known to the ticket recipient, the ticket recipient can compute the opening as $comm_{n-1} = \mathsf{h}^{n-1}(comm_0)$. On redeeming a ticket, the smart contract verifies that

$$\mathsf{Open}(comm_i, comm_{i-1}) = \top$$

and sets $channel.comm[redeemer] \leftarrow comm_{i-1}$. Thus the next time the node redeems a ticket, it must reveal $comm_{i-2}$. In addition, each node is granted the right to reset the commitment to a new value. This is particularly necessary once a node reveals $comm_0$ and therefore is with high probability unable to compute a value r such that

$$\mathsf{Open}(comm_0, r) \neq \bot,$$

where $\bot$ represents the truth value "false". Since this mechanism can be abused by the ticket recipient to tweak the entropy used to determine whether a ticket is a winner or not, the smart contract keeps track of resets of the on-chain commitment and sets

$$channel[redeemer].ticketEpoch \leftarrow channel[redeemer].ticketEpoch + 1,$$

thereby invalidating all previously unredeemed tickets.

5.3 Probabilistic Payments

In traditional payment channels, two parties A and B lock some funds within a smart contract, make transactions off-chain and only commit the aggregation on-chain. Thus, a payment channel is bidirectional, which means both A and B can send and receive transactions within the same payment channel. The HOPR protocol uses unidirectional payment channels to implement bidirectional payment channel behaviour, where one payment channel is created from A to B and another from B to A. The payment channel creator is the sole owner of funds in the payment channel and the only one able to create **tickets**, encapsulated

funds which are described in detail in Section 4. A payment channel created from party A to party B is different from a payment channel created from party B to party A.

$$A \rightarrow B \neq B \rightarrow A$$

This separation reflects the directional nature of packets flowing through the network. It also brings the advantage that each payment channel's logic is easier to verify.

Acknowledgements are messages which allow every node to acknowledge the processing of a packet to the previous node. This acknowledgement (ACK) contains the cryptographic material needed to unlock the possible payout for the previous node. Note that an acknowledgement is always sent to the previous node, and using acknowledgments with vanilla payment channels results in accumulated incentives, where the latest acknowledgement contains all previous incentives plus the incentive for the most recent interaction, as explained below:

$$value(ACK_n) = \sum_{i=1}^{n} fee_{packet_i}, \tag{8}$$

where n is the total number of mixnet packets transformed.

An issue arises when B receives ACK_n for $packet_n$ before sending $packet_{n-1}$. At this point B would have no incentive to process $packet_{n-1}$ rather than $packet_n$. To avoid such false incentives, the HOPR protocol utilizes probabilistic payments. A *ticket* can be either a win or a loss, determined based on some winning probability lower than 1. This means nodes are incentivized to continue relaying packets, as they do not know which tickets will result in a payout. From a node's perspective, each ticket has the same value until it is claimed; therefore, the HOPR protocol encourages nodes to claim tickets independently from each other.

$$value(ACK_i) = value(ACK_j) \quad for \quad i, j \in \{1, n\} \tag{9}$$

If we assume constant costs, there is no added value in pretending packet loss or intentionally changing the order in which packets are processed. On the contrary, a node would reduce its potential payouts if it were to forward packets slowly or not at all.

5.4 Payment Channel Management

For node A to transfer packets to node B, it must first open a payment channel. There are four distinct payment channel states, represented in the following scheme:

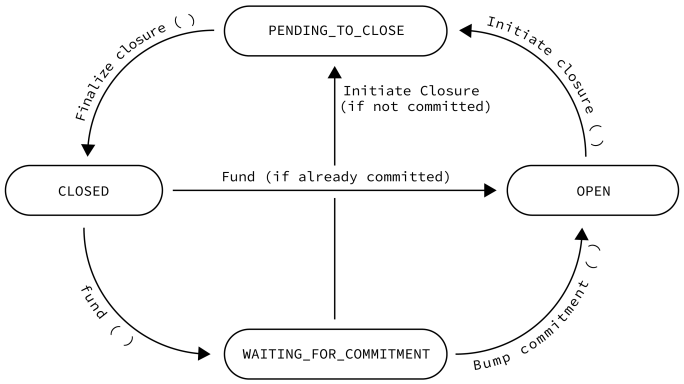

Figure 8: Payment channel states

Initially, each payment channel is *Closed*.

Opening a channel Node A can open a channel by transferring funds to the payment channels contract *HoprChannels* and including the following *userdata*:

$$[A : address, B : address, \lambda : uint8, \mu : uint8], \mu = 0,$$

where λ is the amount to be staked by A. This call will trigger an on-chain event *ChannelFunded* and open a unidirectional payment channel from A to B. The payment channel will start in state *Waiting for commitment*. The destination address of the payment channel must now set an on-chain commitment in order for the payment channel between both parties to become *Open*. This is done by B calling the *bumpChannel()* function to make a new set of commitments towards this payment channel. This call will trigger an on-chain event *ChannelOpened* and bumps the ticket epoch to ensure tickets with the previous epochs are invalidated. Every time the channel changes its state, an on-chain event *ChannelUpdated* is emitted.

Redeeming tickets As long as the channel remains open, nodes can claim their incentives for forwarding packets via tickets. Tickets are redeemed by dispatching a *redeemTicket()* call to an *Open* payment channel.

If B tries to redeem a ticket from the channel $A \rightarrow B$ (spending channel),

but there is an open channel $B \rightarrow A$ (earning channel), B's rewards will be transferred to $B \rightarrow A$ (earning channel). Otherwise, rewards will be sent directly to B.

Closing a channel Nodes can close a payment channel in order to access their previously staked funds. Only the payment channel creator can initiate the process by calling *initiateChannelClosure()*. This changes the state to *Pendingtoclose* and triggers a grace period during which the destination node can redeem any unredeemed tickets. Nodes should actively monitor blockchain events to be aware of this payment channel state change.

Once the grace period has elapsed, the payment channel creator can call *finalizeChannelClosure()* which changes the payment channel to *Closed*. When a payment channel is closed, the remaining funds are automatically transferred to the payment channel creator. The channel epoch increments, meaning any unredeemed tickets can no longer be redeemed.

6 Cover Traffic

Cover traffic (CT), also known as chaff, is randomly generated packets injected into a mixnet to increase its anonymity set. Since cover traffic is transported and mixed using the same procedures as 'real' packets, cover traffic is indistinguishable from real traffic. Deploying cover traffic makes it more difficult for an attacker to conduct passive attacks against the network. This is a direct result of there being more traffic: passive attacks such as traffic analysis attacks become more expensive the more traffic there is to analyse.

Cover traffic also increases the anonymity level of the mix network by improving sender-recipient unlinkability. This is particularly important in the early stages of HOPR's lifecycle, since it can be reasonably expected that traffic through the network will initially be low. Without cover traffic, it would be easier for a global passive adversary to link the sender and recipient of a particular packet.

A drawback of cover traffic is that it increases bandwidth overhead, which ultimately reduces the capacity of the network. This trade-off between capacity, latency, and anonymity is a result of the "anonymity trilemma" [Das et al., 2018]. Since HOPR is a privacy network, it is reasonable to prioritise anonymity here. However, it is important to mitigate the deleterious effects of cover traffic as far as possible, particularly avoiding wasted transmissions. This is achieved through judicious choice of which nodes to open cover traffic channels to and select paths to, based on node importance, and closing covert traffic channels to nodes which are found to be offline.

6.1 Cover traffic nodes

Cover traffic nodes (CT nodes) are HOPR nodes with a cover traffic service enabled, initially run by the HOPR Association (or third parties sponsored by the HOPR Association) for the purpose of generating cover traffic. In the future, anyone who fulfils the requirements for running a CT node in the HOPR network will be able to start one.

6.2 Opening cover traffic channels

CT nodes must open and fund payment channels before they can send cover traffic packets via that counterparty. The number of channels a CT node can open is predefined and cannot be exceeded. Whenever payment channels are closed, the CT node replaces them by opening new channels, provided it still has funds. Channel opening is randomly weighted according to the importance of a node. A node's importance, $\Omega(N)$, is defined as follows:

$$\Omega(N) = st(N) * sum(w(C), \forall\ outgoingChannels(N))$$

where

$$st(N) = uT(N) + tB(N)$$

and

$$w(C) = \sqrt{(B(C)/st(Cs) * st(Cd))}$$

where N is the node, w is the channel's weight, st is the number of HOPR tokens staked by the node, and C is a payment channel between two nodes in the HOPR network. For this channel, Cs is the node which opened the payment channel, while Cd is channel counterparty. $uT(N)$ is the balance of unreleased tokens for a node N. B is the channel balance and tB is the sum of the outgoing balances of each channel from node N.

In accordance with this definition, a node's importance score increases with the channel balances of channels that node shares with other nodes with high total stake. This means that the odds of channel being opened to a particular node is not simply proportional to that node's stake. Cover traffic packets are allocated in proportion to a node's importance rather than channel stake as a way to mitigate against selfish node operators seeking to maximize their own profits.

The way importance and weight are calculated prevents nodes from opening a few minimally funded channels to nodes with large stake and incentivizes nodes to stake in as many channels as possible, including re-allocating their stake to their downstream nodes to receive cover traffic there. This discourages centralization and prevents "thin traffic", which limits the size of the anonymity set and increases the risk of traceable packets, even with high per-hop latency.

Since distributing cover traffic comes at a cost to network bandwidth, it is important to try and minimise wasted transmission. Therefore, CT nodes distribute cover traffic based on node importance rather than node uptime. This is because individual nodes are liable to suddenly lose connectivity. Of course, this distribution method will still result in sending cover traffic to offline nodes. If a mix node is found to be offline, the channel to that node is closed and new one to a different node is opened. The offline node will have to wait until the CT node is ready to open a new channel to stand a chance of being reselected.

6.3 Path selection and payout

After opening channels, the CT node sends cover traffic packets at regular intervals. The path selection algorithm used for cover traffic is the same one used to select nodes for real traffic (see Section 2 for more details) where nodes are selected at random but weighted by importance, starting at the CT node.

The first relay node is chosen with a probability proportional to the amount funded by the CT node, so every node has an equal chance of getting selected. For subsequent hops, nodes are selected with a probability weighted proportional to their importance. We use a weighted priority queue of potential paths to choose the next node. If the queue is empty then it fails.

As with real traffic, rewards are distributed to all selected nodes in a path as tickets which can be redeemed for HOPR tokens with a certain probability (see Section 4). Thus there is a direct correlation between a node's importance and the cover traffic rewards it receives, ensuring that rewards are distributed fairly. These cover traffic rewards also serve as an incentive for node operators to open meaningful payment channels instead of, e.g., Sibyls or nodes with insignificant stake who would thus not be able to relay a lot of traffic.

6.4 Closing cover traffic channels

To minimize wasted cover traffic, CT nodes will close any channel deemed likely to result in failed distribution and open new channel in its place. A cover traffic channel is closed and a new one opened if any of the following factors falls below its threshold:

- **Channel balance** The channel balance must remain higher than the minimum stake value, defined as:

  $$B(C) < L * \sigma$$

  where $B(C)$ is the channel balance, L is the path length, and σ is the ticket payout amount, which is currently hardcoded as 0.1 HOPR per ticket.

- **Connection quality** Connection quality, q, is defined as the fraction of 'pings' the node has responded to within a 5 second cut-off. q must not drop below the defined upper bound.
- **Number of failed packets** The number of failed packets must stay above the failed packet threshold.

If a node is detected as having fallen below any of these thresholds, the CT node calls *initiateChannelClosure()* and emits two events: *ChannelUpdate* and *ChannelClosureInitiated*.

7 Conclusion

7.1 Privacy, Reliability, and Scalability

This paper explained how the HOPR protocol, particularly its proof-of-relay mechanism, provides a first solution to building a scalable and robust incentivized network which provides full anonymity with minimal trade-offs in latency and bandwidth. Although HOPR is indebted to previous developments in the space, particularly the Sphinx packet format and the Ethereum blockchain, we believe its unique combination of probabilistic payments and verifiable but unlinkable cryptographic tickets is the first viable approach to creating a global privacy network which does not rely on any centralized entity to provide data transmission or incentive management.

The HOPR protocol is still under active development, with much research and implementation work still to be completed, but a working proof of concept is already live and available to use.

7.2 Future work

7.2.1 Path position leak

In HOPR, payments are performed at each hop along a packet's route. These incentives weaken the unlinkability guarantees inherited from the Sphinx packet format [Danezis and Goldberg, 2009] as they reveal the identity of the packet sender, who must transfer these incentives using their signature. To solve this problem, HOPR forwards incentives along with the packet.

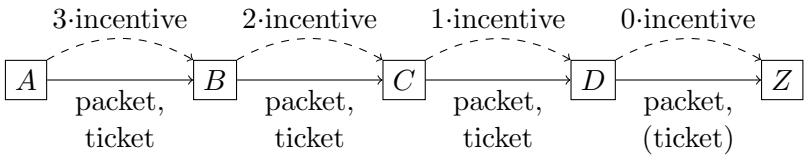

Figure 9: Incentive workflow

However, this leaks the relayer's position within the selected path, since the value of the ticket is set according to the current relay fee and the number of intermediate hops, more precisely

$$amount := \frac{(hops - 1) * relayFee}{winProb}$$

This leak is considered low severity, but further research will be conducted on the subject.

7.2.2 Reputation (aggregated trust matrix)

In HOPR, we assume the majority of nodes are honest and act properly. Nevertheless, there might be nodes who actively try to attack the network by:

- Dropping packets or acknowledgements
- Sending false packets, tickets, or acknowledgements

Since nodes need to monitor the network to select paths, they need to filter nodes that behave inappropriately. To help nodes achieve this, HOPR is considering implementing a transitive reputation system which assigns a score to each node that acts as a relayer. A node's behaviour will affect its reputation, which will in turn affect its probability of being chosen as a relayer.

Transitive trust evaluation

Reputation within a network can be defined as "a peer's belief in another peer's capabilities, honesty, and reliability based on other peers recommendations" [Wang and Vassileva, 2003]. Trust is represented by a triplet (trust, distrust, uncertainty) where:

- Trust: $td^t(d, e, x, k) = \frac{n}{m}$ where m is the number of all experiences and n are the positive ones

- Distrust: $tdd^t(d, e, x, k) = \frac{l}{m}$ where l stands for the number of the trustor's negative experience.
- Uncertainty = 1 - trust - distrust.

Commitment derivation

In the future $comm_0$ will be derived from the private key of the node as

$$comm_0 = \mathsf{h}(privKey, chainId, contractAddr, channelId, channelEpoch)$$

And there will be further research to determine whether such a design leaks information about the private key or not.

References

[Anderson and Biham, 1996] Anderson, R. J. and Biham, E. (1996). Two practical and provably secure block ciphers: BEARS and LION. In Gollmann, D., editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 113–120. Springer. https://doi.org/10.1007/3-540-60865-6_48.

[Astolfi et al., 2015] Astolfi, F., Kroese, J., and Van Oorschot, J. (2015). I2p - the invisible internet project. Web technology report, Media Technology, Leiden University. https://staas.home.xs4all.nl/t/swtr/documents/wt2015_i2p.pdf.

[Atkinson and Silaghi, 2007] Atkinson, T. and Silaghi, M. (2007). Guarantees for customers of incentive anonymizing networks. Cryptology ePrint Archive, Report 2007/460. https://ia.cr/2007/460.

[Backes et al., 2013] Backes, M., A., K., Manoharan, P., Meiser, S., and Mohammadi, E. (2013). Anoa: A framework for analyzing anonymous communication protocols. *Computer Security Foundations Symposium*, pages 163–178. https://eprint.iacr.org/2014/087.pdf.

[Cannell et al., 2019] Cannell, J. S., Sheek, J., Freeman, J., Hazel, G., Rodriguez-Mueller, J., Hou, E., Fox, B. J., and Waterhouse, S. (2019). Orchid: A decentralized network routing market. https://www.orchid.com/assets/whitepaper/whitepaper.pdf.

[Chaum, 1981] Chaum, D. (1981). Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88. http://doi.acm.org/10.1145/358549.358563.

[Danezis and Goldberg, 2009] Danezis, G. and Goldberg, I. (2009). Sphinx: A compact and provably secure mix format. In *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*, pages 269–282. IEEE Computer Society. https://doi.org/10.1109/SP.2009.15.

[Das et al., 2018] Das, D., Meiser, S., Mohammadi, E., and Kate, A. (2018). Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency - choose two. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 108–126. IEEE Computer Society. https://doi.org/10.1109/SP.2018.00011.

[Dingledine et al., 2004] Dingledine, R., Mathewson, N., and Syverson, P. F. (2004). Tor: The second-generation onion router. In Blaze, M., editor, *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 303–320. USENIX. http://www.usenix.org/publications/library/proceedings/sec04/tech/dingledine.html.

[Dingledine et al., 2005] Dingledine, R., Shmatikov, V., and Syverson, P. (2005). Synchronous batching: From cascades to free routes. In Martin, D. and Serjantov, A., editors, *Privacy Enhancing Technologies*, pages 186–206, Berlin, Heidelberg. Springer Berlin Heidelberg. http://web.mit.edu/arma/Public/sync-batching.pdf.

[Hobbs and Roberts, 2018] Hobbs, W. R. and Roberts, M. E. (2018). How sudden censorship can increase access to information. *American Political Science Review*, 112(3):621–636. https://www.cambridge.org/core/journals/american-political-science-review/article/how-sudden-censorship-can-increase-access-to-information/A913C96E2058A602F611DFEAC43506DB.

[Mysterium, 2017] Mysterium (2017). Mysterium network project. https://mysterium.network/whitepaper.pdf.

[Piotrowska et al., 2017] Piotrowska, A. M., Hayes, J., Elahi, T., Meiser, S., and Danezis, G. (2017). The loopix anonymity system. *CoRR*, abs/1703.00536. http://arxiv.org/abs/1703.00536.

[Poon and Buterin, 2017] Poon, J. and Buterin, V. (2017). Plasma: Scalable autonomous smart contracts. https://plasma.io/plasma.pdf.

[Poon and Dryja, 2016] Poon, J. and Dryja, T. (2016). The bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf.

[Raiden Team, 2016] Raiden Team (2016). Raiden network - high-speed asset transfers for ethereum. https://raiden.network.

[Sentinel, 2021] Sentinel (2021). Sentinel: A blockchain framework for building decentralized vpn applications. https://sentinel.co/whitepaper.

[Teutsch and Reitwießner, 2019] Teutsch, J. and Reitwießner, C. (2019). A scalable verification solution for blockchains. *CoRR*, abs/1908.04756. http://arxiv.org/abs/1908.04756.

[UN, 2018] UN (2018). The right to privacy in the digital age : report of the united nations high commissioner for human rights. page 17 p. http://digitallibrary.un.org/record/1640588.

[Varvello et al., 2019] Varvello, M., Querejeta-Azurmendi, I., Nappa, A., Papadopoulos, P., Pestana, G., and Livshits, B. (2019). VPN0: A privacy-preserving decentralized virtual private network. *CoRR*, abs/1910.00159. http://arxiv.org/abs/1910.00159.

[Venkateswaran, 2001] Venkateswaran, R. (2001). Virtual private networks. *IEEE Potentials*, 20(1):11–15. https://ieeexplore.ieee.org/document/913204.

[Wang and Vassileva, 2003] Wang, Y. and Vassileva, J. (2003). Trust and reputation model in peer-to-peer networks. *Peer-to-Peer Computing*. https://ieeexplore.ieee.org/document/913204.

[Wood, 2021] Wood, G. (2021). Ethereum: A secure decentralised generalised transaction ledger (istanbul version). https://ethereum.github.io/yellowpaper/paper.pdf.

A Key Derivation

During a protocol execution, a node derives a master secret s_i from the SPHINX header that is then used to derive multiple sub-secrets for multiple purposes by using the BLAKE2s hash function within HKDF. HKDF is given by two algorithms: extract and expand, where extract is used to extract the entropy from a given secret s, such as an elliptic-curve point, and produces the intermediate keying material (IKM). The IKM then serves as a master secret to feed expand in order to derive pseudorandom subkeys in the desired length.

A.1 Extraction

As a result of the packet creation and its transformation, the sender is able to derive a shared secret s_i given as a compressed elliptic-curve point (33 bytes) with each of the nodes along the path.

$$s_i^{master} \longleftarrow \mathsf{extract}(s_i, 33, pubKey)$$

By adding their own public key $pubKey$ as a salt, each node derives a unique s_i^{master} for each s_i.

A.2 Expansion

Each subkey s_i^{sub} is used for one purpose, such as keying the *pseudorandomness generator* (PRG).

$$s_i^{sub} \longleftarrow \mathsf{expand}(s_i^{master}, length(purpose), hashKey(purpose))$$

Usage	Purpose	Length	Hash Key (UTF-8)
SPHINX packet	PRG	32	`HASH_KEY_PRG`
	PRP	128 + 64	`HASH_KEY_PRP`
	Packet tag	32	`HASH_KEY_PACKET_TAG`
Proof of Relay	acknowledgement	32*	`HASH_KEY_ACK_KEY`
	ownKey	32*	`HASH_KEY_OWN_KEY`

The values marked with a * are treated as field elements, hence there exists a non-zero probability that expand produces a value outside of the field. In this specific case, the utilized hash key is repeatedly padded by "_" until expand returns a field element.

B LIONESS wide-block cipher scheme

HOPR uses the LIONESS [Anderson and Biham, 1996] wide-block cipher scheme with ChaCha20 as stream cipher and BLAKE2s as hash function, resulting in a key length of 128 bytes and 64 bytes for the initialisation vector.

The key k is split into four chunks of 32 bytes $k = k_1 \mid\mid k_2 \mid\mid k_3 \mid\mid k_4$, where k_1, k_3 are used as keys for the stream cipher and k_2, k_4 are used to key the hash function. The same applies to the initialisation vector iv which is split into four chunks of 16 bytes $iv = iv_1 \mid\mid iv_2 \mid\mid iv_3 \mid\mid iv_4$ with iv_1, iv_3 as an initialisation vector for the stream cipher and iv_2, iv_4 as an initialisation vector for the hash function.

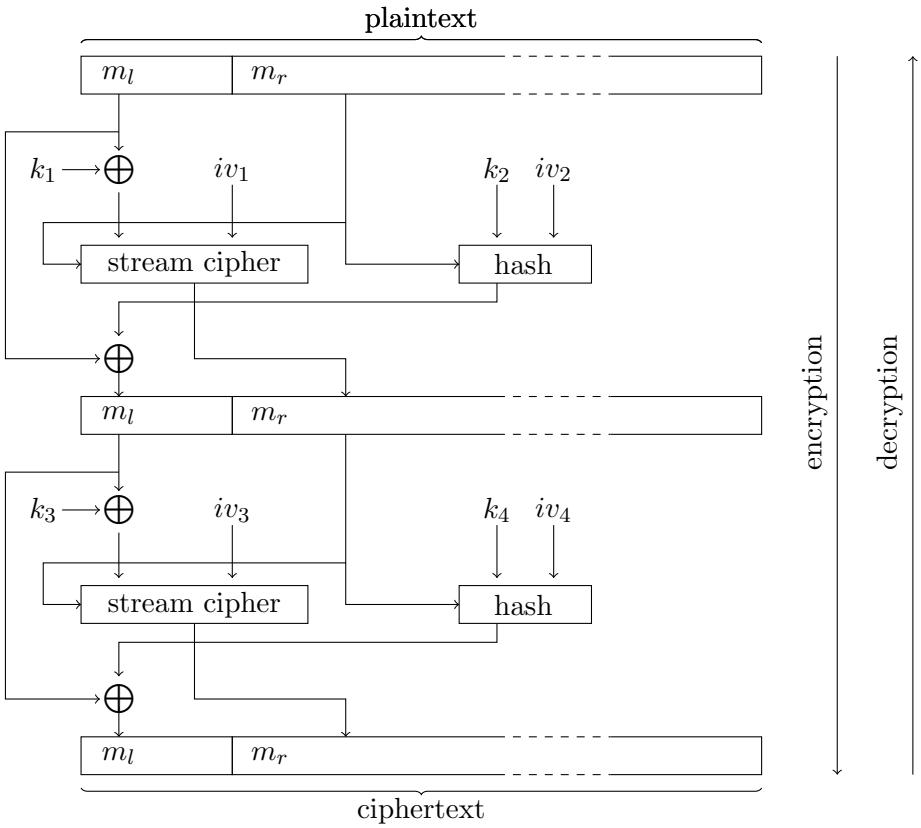

Figure 10: Schematic overview of encryption and decryption of the LIONESS [Anderson and Biham, 1996] scheme

LIONESS is an unbalanced Feistel scheme, hence decryption is achieved by applying the operations used for encryption in the inverse order. In contrast to other Feistel schemes, the plaintext m is not split equally and the size of the parts depend on the output length of used hash function.

As HOPR payloads have a size of 500 bytes, messages are split as $m = m_l \mid\mid m_r$ where $|m_l| = 32$ bytes and $|m_r| = 468$ bytes.