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

Amira Bouguera, Robert Kiel, Dr. Sebastian Bürgel

September 2021, v2.0

Abstract

1 Introduction

Internet privacy, commonly referred to as online privacy, is a subset of data privacy and a fundamental human right as defined by the UN [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, because of 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 other than the communicating parties.

HOPR is a decentralized incentivized mixnet that employs privacy-by-design protocols. HOPR aims to protect users' 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 The 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 reliability. The following section explains these three principles in more detail, as well as how HOPR meets them.

- **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's impossible to provide such privacy using current internet infrastructure. 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.

- **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. 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 services. 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.

- **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) 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. 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 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.

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 which ensures nodes don't (de-)prioritize other nodes and/or packets. To maximize the 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.

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, as well as all intermediaries involved in transferring the data between the communicating parties.

Specifically, the HOPR protocol aims to build a network with the following features:

- **Sender-receiver unlinkability**: An adversary should be unable to distinguish whether $\{S_1 \to R_1, S_2 \to R_2\}$ or $\{S_1 \to R_2, S_2 \to R_1\}$ for any concurrently online honest senders S_1, S_2 and honest receivers R_1, R_2 of the adversary's choice.
- **Resistance to active attacks**: Adversaries must not be able to conduct active attacks such as tagging and replay attacks, where the adversary modifies and re-injects messages to extract information about their destinations or content.

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 the Sphinx Packet Format section.

1.2.1 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 messages.

In particular, to provide reliable privacy and data transmission the HOPR network needs to be resistant to both Sybil and eclipse attacks:

- **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 receiver'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.
- **Eclipse attacks:** In an eclipse attack, the attacker seeks to isolate and attack or manipulate a specific user in the network. This is a common attack in peer-to-peer networks since nodes do not have a global view of the entire network, making it difficult to identify malicious nodes. HOPR mitigates this by employing a smart contract known as DEADR (Decentralized Entry Advertisement and Distributed Relaying). This smart contact employs a distributed hash table (DHT) to advertise honest entry nodes to the network.

However, security cannot come at the expense of usability. To be appealing to users and node runners alike, it is important for HOPR to satisfy what is known as the "anonymity trilemma":

The Anonymity Trilemma [Das et al., 2017] To be successful, a privacy network must provide low-latency communication, low bandwidth overhead and strong anonymity.

1.3 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) Privacy Protocols attempt to hide user information; (2) Layer-2 Scalability Protocols allow systems to perform micro-payments via the Ethereum blockchain.

1.3.1 Privacy 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. Orchid [Cannell et al., 2019], Sentinel [Sentinel, 2021], and Mysterium [Mysterium, 2017] are well-known dVPN projects. However, these projects still lack privacy, performance, and reliability guarantees, since bandwidth providers can potentially inspect, log, and share any of their traffic. This potential 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 promising project called VPN^0 [Varvello et al., 2019] has been started by the Brave team 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 [I2P-Team, 2003]:

- **Packet switched instead of circuit switched**: Tor allocates connection to long-lived circuits; this allocation does not change until either the connection or circuit closes. On the other hand, 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.

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 messages 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 and active 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 messages among random noise. An external adversary able to observe the message flow should not be able to distinguish communication messages from random noise messages, 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 a well-known 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 messages, making packets unlinkable via timing analysis.

1.3.2 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 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 the Tickets section).

1.4 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 above.

The Anonymity Trilemma [Das et al., 2017] Most projects presented above share the goal of satisfying the anonymity trilemma of providing low latency communication, low bandwidth overhead and strong anonymity. However, most fail to satisfy one or more horns of the trilemma. In general, strong anonymity for both users and node runners is the hardest condition to satisfy. Where approach come close, it usually comes at the expense of latency.

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 Most projects 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. Some solutions have proposed adding digital coins to messages, such that each volunteer can extract only the digital coin designated as a payment for them. However, here the anonymity provided by the mixnet acts as a double-edged sword: since there is no verification for whether a message arrives at its final destination, and node identities are kept secret by the mixnet, selfish nodes can extract payment without performing their relaying task. Thus 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. The remainder of this paper will explain how this is achieved.

2 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 receiver unlinkability. The Sphinx packet format determines how mixnet packets are created and transformed before relaying them to the next downstream node. In HOPR terminology, each journey from one node to the next downstream node is known as a "hop". Each Sphinx packet consists of two parts:

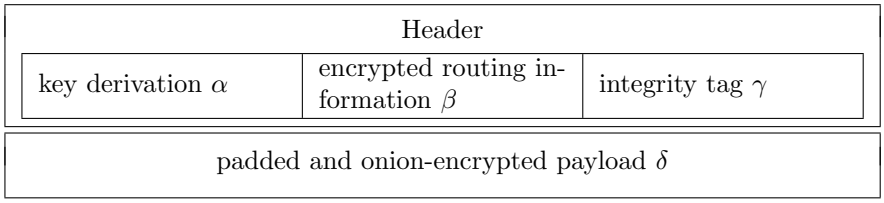

Figure 1: Sphinx packet format

2.1 Construction

The following explains the steps a Sphinx packet goes through before arriving at its final destination. We start with key derivation to extract shared keys for the relay nodes chosen by the sender. These shared keys will be used to unblind the routing information and find the public key of the next relay node.

The integrity of this routing information is checked before sending. At each step along the route, the relaying node replaces the routing information with their own blinding and decrypts one layer of the payload. Similar to onion routing, the payload has several layers of encryption, chosen such that the final layer is removed as the packet reaches its destination. However, as layers of encryption are removed, the packet is padded to ensure the message length remains constant. This ensures that an adversary cannot draw any conclusions from message length about a transmission's position along the route.

Notation: Let $\kappa = 128$ be the security parameter. With non-negligible porbability, an adversary must perform around 2^κ operations to break the security of Sphinx.

Let r be the maximum number of nodes that a Sphinx mix message will traverse before being delivered to its destination.

G is a prime order cyclic group satisfying the Decisional Diffie-Hellman Assumption. We use the secp256k1 elliptic curve [Brown, 2010]. The element g is a generator of G and q is the (prime) order of G, with $q \approx 2^{\kappa}$.

$G^*$is the set of non-identity elements of G. h_b is a pre-image resistant hash function used to compute blinding factors and modelled as a random oracle such that $h_b : G^* \times G^* \rightarrow \mathbb{Z}_q^*$, where $\mathbb{Z}_q^*$ is the field of non-identity elements of $\mathbb{Z}_q$ (field of integers). We use the BLAKE2s hash function [Aumasson et al., 2013].

Each node i has a private key $x_i \in \mathbb{Z}_q^*$ and a public key $y_i = g^{x_i} \in G^*$. α_i is the group elements which, when combined with the nodes' public keys, allow a shared key to be computed for each via Diffie-Hellman (DH) key exchange. This ensures that each node in the user-chosen route can forward the packet to the next, and only the receiving mix node can decrypt it. s_i are the DH shared secrets, b_i are the blinding factors.

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 key exchange with each of these nodes and derives shared keys with each of them.

A computes a sequence of r tuples (in our case r=4)

$$(\alpha_0, s_0, b_0), ..., (\alpha_{r-1}, s_{r-1}, b_{r-1})$$

as follows:

$$\alpha_0 = g^x, s_0 = y_B^x, b_0 = h_b(a_0, s_0)$$

and

$$\begin{cases} \alpha_i = g^{x\Pi_{i=1}^{i=r-1} b_i} \\ s_i = y_i^{x\Pi_{i=1}^{i=r-1} b_i} \\ b_i = h_b(a_i, s_i) \end{cases} . \tag{1}$$

for $1 \leq i < r-1$,

where y_1, y_2, y_3 and y_4 are the public keys of the nodes B, C, D, and Z, which we assume to be available to A .

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 message END to tell Z that it is the final receiver

of the message. The END message is a distinguished prefix byte which is added to the final receiver's compressed public key. For ECDSA public key compression, only the x coordinate is used and is prepended by 02.

The y coordinate is extracted from x by resolving the elliptic curve equation $Y^2 = X^3 + aX + b$, using the given parameters a and b. A square root extraction will yield Y or $-Y$. The compressed point format includes the least significant bit of Y in the first byte (the first byte is 0×02 or 0×03, depending on that bit).

The routing information is computed as follows:

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

and

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

$0 \leq i < v - 1$

such that y_Z is the destination's public key in compressed form (since this is only the x-coordinate, it is 32 bytes instead of 64) and $|y_Z|$ is its length. ρ is a pseudo-random number 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 $|Z| \leq (2(r - v) + 2)$. ϕ is a filler string such that

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

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 preformed 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 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, 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 also 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. This allows B to unblind the routing info that tells B the public key of the next downstream node, C.

Integrity check 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 because the header has been tampered with, the packet is dropped. Otherwise, the mix node proceeds to the next step.

This integrity check allows to verify whether or not the header was modified.

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 $PRNG(s_0)$ 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.

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 PRNG function.

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

Encrypt and decrypt The encrypted payload δ is where the actual message is hidden. This is computed in different layers using a wide block cipher encryption algorithm and is decrypted at each stage of mixing. δ is repeatedly encrypted via keys derived from the Diffie-Hellman key exchange between the packet's group element α_i and the public key of each node in the path y_i.

Notation Let j be the block size in bits, which will typically be large. Let H_k be a keyed hash function with the key k for the payload. k consists of four independent keys, k_1, k_2, k_3, and k_4, which A derives from the master keys s_0, s_1, s_2, and s_3. Of the four keys, k_1 and k_3 will be used to key the stream cipher, while k_2 and k_4 are used to key the hash function. Sphinx uses the LIONESS wide block cipher scheme for encryption and decryption purposes. Let S_k be a pseudo-random function (stream cipher) which given the input m will generate an output of arbitrary length. We add an authentication tag τ to m before encryption and a 0-padding string as follows:

$$m \leftarrow 0_l \| \tau \| m \tag{8}$$

where τ is generated arbitrarily and $|\tau| = 4$ is its length (τ = "$HOPR$" in ASCII). l is the 0 padding length where $0 <= l <= |m| - 4$.

m is divided into two blocks, left m_l and right m_r, whose sizes are $|m_l| = w$ and $|m_r| = j - w$. We thus get $m = m_r \| m_l$.

Encryption The blocks m_l and m_r are transformed using a four-round Feistel structure:

$$m_r \leftarrow m_r \oplus S_{k1}(m_l), m_l \leftarrow m_l \oplus H_{k2}(m_r), m_r \leftarrow m_r \oplus S_{k3}(m_l), m_l \leftarrow m_l \oplus H_{k4}(m_r)$$

The updated $m_l \| m_r$ constitutes the ciphertext δ.

Decryption Decryption happens as follows:

$$m_l \leftarrow m_l \oplus H_{k4}(m_r), m_r \leftarrow S_{k3}(m_l), m_l \leftarrow m_l \oplus H_{k2}(m_r), m_r \leftarrow m_r \oplus S_{k1}(m_l)$$

Integrity The payload is encrypted using a bidirectional error propagating block cipher and protected with an integrity check for the receiver, but not for processing relays. Packet integrity is authenticated by prepending a tag set so that any alteration to the encrypted payload as it traverses the network will result in an irrecoverably corrupted plaintext when the payload is decrypted by the recipient.

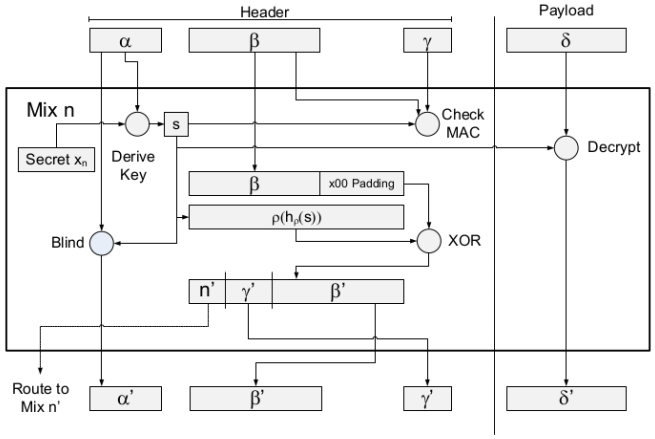

Figure 2: Processing a Sphinx message [Danezis and Goldberg, 2009]

The same process of key derivation, unblinding, deleting, shifting, integrity check, decryption, and blinding occurs at C and D.

2.2 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 last 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 last exit-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.

3 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 **probabilisitic payments**. 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.

3.1 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.

Construction

- Every packet is sent together with a ticket.
- Each ticket contains a challenge.
- The validity of a ticket can only be checked on reception of the packet but the on-chain logic enforces a solution to the challenge stated in the ticket.

3.1.1 Challenge

A creates a shared group element α_i with all the relay nodes in the channel (B-C-D-Z) by using an offline version of the Diffie-Hellman key exchange. This shared key is a session key generated from the master DH Sphinx key and will be used to derive new keys $s_i^{(0)}$ and $s_i^{(1)}$. The first key, $s_i^{(0)}$, will be used as the node's *own key share* and the second key, $s_i^{(1)}$, will be used as the *acknowledgement key*. The acknowledgement key will be embedded in the acknowledgement for a packet and thereby unlocks the reward for the previous relayer, earned for transforming and delivering the packet.

Key derivation A first creates a "session secret", s_i, then uses it as a seed to derive subkeys $s_i^{(0)}, s_i^{(1)}$ for each node on the route. This is done using an HKDF (HMAC-based Key Derivation Function), a cryptographic hash function that derives one or more secret keys from a secret value using a pseudorandom function. The key derivation works as follows:

- **Extract**: Creates a pseudo-random key s_i

$$HKDF.extract(h_b, |h_b|, (\alpha_i * privKey)||pubKey)$$

- **Expand**: This step takes the output of the previous one, s_i, as a seed and creates output key material $s_i^{(0)}, s_i^{(1)}$ which is expanded from hashes of s_i and an optional info message (salt). The process occurs as follows:

$$HKDF.expand(h_b, |h_b|, s_i, |s_i|, hashKey)$$

where $||$ is concatenation and $*$ is scalar multiplication on the curve. h_b is the BLAKE2s 256 hash algorithm, $|h_b|$ is its length, s_i is the pseudo random key used as a seed and $|s_i|$ is its length. $hashKey$ is an identifier used to create a"virtual" hash function for each purpose, e.g., one for the PRG, one for PRP, etc. and also for the proof-of-relay scheme. If the result of HKDF does not lead to a field element, $hashKey$ is padded until it does.

A provides a hint to the expected value $s_{i+1}^{(1)}$ that a node n_i is expected to get from the next downstream node. The hint value, H, is computed as

$$H_i = s_{i+1}^{(1)} * G, \tag{9}$$

where $*$ is the curve multiplication operation and G is a generator of the curve (the same used in the 2 section).
The hint for party n_i is used to check whether the returned value $s_{i+1}^{\prime(1)}$ matches the promised value $s_{i+1}^{(1)}$ by checking whether H_i equals $s_{i+1}^{\prime(1)} * G$.
The sender A also creates a challenge T_{c_i}, such that

$$T_{c_i} = (s_i^{(0)} + s_{i+1}^{(1)}) * G \tag{10}$$

Since proof of relay is used to ensure the relay services of nodes are verifiable, it is the duty of each node to check that given challenges are derivable from the given and expected information. Packets with inappropriate challenges should be dropped, as they might not result in winning tickets with the expected probability.

The values H_i and T_{c_i} are sent with the routing information β_i as follows:

$$\beta_i = y_{i+1} \| H_i \| T_{c_i} \| \gamma_{i+1} \| \beta_{i+1}{}_{[0 \ldots (2r-1)\kappa - 1]} \oplus \rho(h_\rho(s_i))_{[0 \ldots (2r+1)\kappa - 1]}$$

By decrypting β_i, each mix node n_i will retrieve the public key of the next downstream node and both the hint and challenge required by proof of relay to ensure relay services are verifiable.

3.2 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:

Definition 3.2.1. *A commitment scheme* $\mathsf{C_m} = (\mathsf{Commit}, \mathsf{Open})$ *is a protocol between two parties, A and B, that gives A the opportunity to store a value* $comm = \mathsf{Commit}(x)$ *at B. The value x stays unknown to B until A decides to reveal it to B. For any* $m \in \mathbb{M}$, $(c, d) \leftarrow \mathsf{Commit}(\mathsf{m})$ *is the commitment/opening pair for m where* $c = c(m)$ *serves as the commitment value, and* $d = d(m)$ *as the opening value.*

Hiding: *A commitment scheme is called* ***computationally hiding*** *if the following holds:*

$$\forall\, x \neq x' \; \{\mathsf{Commit}_k(x, U_k)\}_{k \in \mathbb{N}} \approx \{\mathsf{Commit}_k(x', U_k)\}_{k \in \mathbb{N}}.$$

This means both probability ensembles are computationally indistinguishable, such that U_k *is the uniform distribution over the* 2^k *opening values for the security parameter* k*.*

Binding: *A commitment scheme is called* ***computationally binding*** *if for all bounded polynomial adversary Adv algorithms that run in time t with output* $x, x', open, open'$*, the following holds:*

$$P[\, x \neq x' \text{ and } \mathsf{Commit}(x, open) = \mathsf{Commit}(x', open')] \leq \epsilon$$

A computationally bounded adversary has limitations on their computational resources.

3.2.1 Commitment phase

Once a node is the destination of a HOPR unidirectional channel, it derives a master key $comm_0$ from its private key 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 preimage-resistant hash function (we use the keccak256 hash function, which is also used in Ethereum) and $comm_0$ is derived as

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

This will be implemented in the future after further research determines whether such a design leaks information about the private key or not.

The master key should be pseudo-random, 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.

3.2.2 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.

3.3 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 behavior, 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 Tickets. 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:

$$value(ACK_n) = \sum_{i=1}^{n} fee_{packet_i}, \tag{11}$$

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 100 percent. 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.

$$value(ACK_i) = value(ACK_j) \quad for \quad i, j \in \{1, n\} \tag{12}$$

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.

3.4 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:

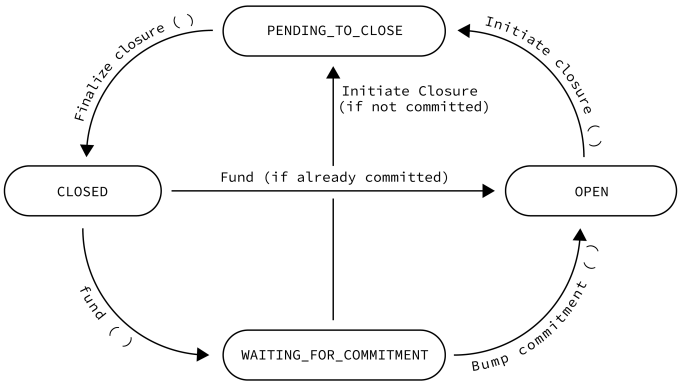

Figure 3: Payment channel states

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 will 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 *ChannelIsOpened* and bumps the ticket epoch to ensure tickets with the previous epochs are invalidated.

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.

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) selects the winning probability of the ticket and the 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.

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 adress 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.

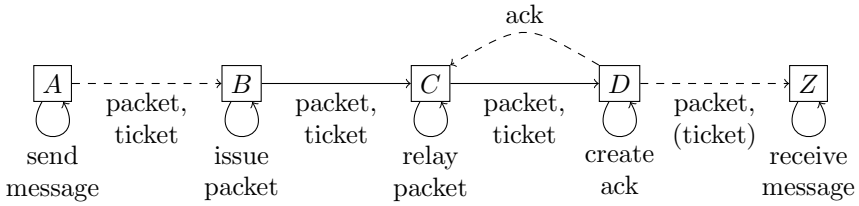

Figure 4: Ticket workflow

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. 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 of 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 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 drops 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:

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

- **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 reception 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 if 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.

- **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 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 Path Selection

To send packets through the HOPR network, each node must decide which path a packet is sent via. This decision, called **path selection**, must be performed for every packet. HOPR uses a random selection algorithm to determine the identities of nodes participating in the network relaying service, both generally and for each particular packet.

The selection process is divided into two steps:

1. **Pre-Selection:** Initially, a subset $m << n$ nodes will be selected based on the following factors:
   - Node availability
   - Payment channel graph
   - Payment channel stake

   Each node receives a weight that is proportional to these factors.

2. **Random Selection:** Each edge (from node A to B) within the subset m is assigned a random number r_i. Edges are then sorted by

$$r_i * weight(edge_i).$$

   The path with the largest weight is extended next, using a priority queue mechanism.

Algorithm 1: Path selection in HOPR

$V := Nodes$
$M := MaxSelectionIterations$
$A := MinNodeAvailability$
$P := RequiredPathLength$

$openChannels \leftarrow \{(x, y) \in V \times V \mid x \neq y \wedge getChannel(x, y).state = OPEN\}$
$queue \leftarrow new\ PriorityQueue()$
$queue.addAll(\{(x, y) \in openChannels \mid x = self\})$
$deadEnds \leftarrow \emptyset$
$iterations \leftarrow 0$

while $!queue.isEmpty() \wedge iterations < M$ **do**
  $current \leftarrow queue.peek()$
  if $|current| = P$ **then**
    return $current$
  end

  $currentNode \leftarrow lastNode(current)$
  $open \leftarrow \{(x, y) \in E \mid x = currentNode \wedge y \notin deadEnds \wedge y \neq currentNode \wedge availability(y) > A\ \}$
  $open \leftarrow open.sort(weight_fun)$

  if $open = \emptyset$ **then**
    $queue.pop()$
    $deadEnds \leftarrow deadEnds \cup currentNode$
  else
    $newPath \leftarrow \{(current_0, ..., current_{|current|-1}, o) \mid o \in open\})$
    $queue.push(newPath)$
  end

  $iterations \leftarrow iterations + 1$
end
return $\perp$

5.1 Node Availability

Node availability is estimated using a heartbeat protocol. Each node maintains a list of its neighbouring nodes within the network. A node will *ping* neighbours and respond to *pings* from other nodes to determine whether they are online or offline. A node is considered online if the ping response ("pong") returns within a certain timeframe. Otherwise, the node is considered offline and the waiting time before the next ping attempt to that node is doubled. Newly working pings will gladually improve a neighbour's availability score.

5.2 Payment channel graph

Every node that intends to send messages needs to have a basic understanding about the topology of the network, meanings whether a payment channel to the node is open and funded with sufficient HOPR tokens for the relaying service. If no such payment channel exists, the sender creates a new channel and funds it with sufficient HOPR tokens.

5.3 Payment channel stake

As payment channels are opened with other nodes, HOPR tokes are staked in those channels and therefore the network as a whole. Nodes use these HOPR tokens to cover transaction costs when interacting with the blockchain. The higher the stake a node has locked, the higher probability that it will be chosen as a relayer.

6 Conclusion

6.1 Privacy, Reliability and Scalability

This paper explains how the HOPR protocol, particularly its proof-of-relay mechanism, provides a first solution to building a scalable and robust incentivized network which satisfies all three components of the anonymity trilemma. 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.

6.2 Future work

6.2.1 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 to using their signature. To solve this problem, HOPR forwards incentives along with the packet.

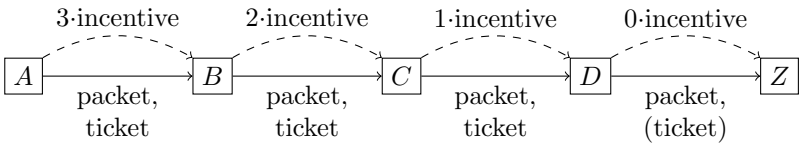

Figure 5: Incentive workflow

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.

6.2.2 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 plans to implement 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 with a network can be defined as:"a peer's belief in another peer's capabilities, honesty, and reliability based on the other peers recommendations." 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.

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.

[Aumasson et al., 2013] Aumasson, J., Neves, S., Wilcox-O'Hearn, Z., and Winnerlein, C. (2013). BLAKE2: simpler, smaller, fast as MD5. volume 2013, page 322. http://eprint.iacr.org/2013/322.

[Brown, 2010] Brown, D. R. L. (Version 2.0, 2010). Sec 2. standards for efficient cryptography group: Recommended elliptic curve domain parameters. https://www.secg.org/sec2-v2.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. volume 24, pages 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., 2017] Das, D., Meiser, S., Mohammadi, E., and Kate, A. (2017). Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency—choose two. https://eprint.iacr.org/2017/954.pdf.

[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.

[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.

[I2P-Team, 2003] I2P-Team (2003). I2p: The invisible internet project. https://geti2p.net/en/about/intro.

[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. volume 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.

[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. volume abs/1908.04756. http://arxiv.org/abs/1908.04756.

[UN, 2018] UN (2018). The right to privacy in the digital age: report. https://www.ohchr.org/EN/Issues/DigitalAge/Pages/ReportDigitalAge.aspx.

[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. volume 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.

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