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

Dr. Sebastian Bürgel, Robert Kiel

November 2019, V1

Abstract

1 Introduction

1.1 Sphinx Packet Format

HOPR uses the SPHINX packet format to encapsulate and route data packets in order to provide privacy features such as sender and receiver unlinkability. Sphinx is a cryptographic message format used to relay anonymized messages within a mix network. A sphinx packet consists of two parts:

1. Header:
    - Key derivation
    - Routing information
    - Integrity protection
2. Body:
    - Onion-Encrypted payload

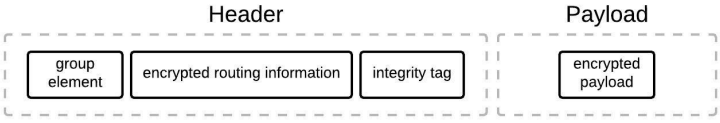

Figure 1: The structure of Sphinx

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) [see section path-finding] and the final destination of the packet (E)
(A) performs an offline Diffie-Hellman key exchange with each of these nodes and derives shared keys with each of these nodes.
(A) computes a sequence of r tuples (in our case r=4)

$$(a_0, s_0, b_0), ..., (a_{r-1}, s_{r-1}, b_{r-1})$$

as follows:

- $a_0 = g^x, s_0 = y_B^x, b_0 = h(a_0, s_0)$
- $a_1 = g^{xb_0}, s_1 = y_C^{xb_0}, b_1 = h(a_1, s_1)$
- $a_2 = g^{xb_0b_1}, s_2 = y_D^{xb_0b_1}, b_2 = h(a_2, s_2)$

Where y_B, y_C, y_D, y_E are the public keys of the nodes B, C, D which we assume are available to A . The a_i are the group elements which, when combined with the nodes' public keys, allows computing a shared key for each via Diffie-Hellman key exchange, and so the first node in the user-chosen route can forward the packet to the next, and only that mix-node can decrypt it. The s_i are the Diffie Hellman shared secrets, and the b_i are the blinding factors.

Routing information Each node on the path needs to know 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 (E) that it is the final receiver of the message.
As (A) has a shared secret with each of the nodes along the path, it is able to derive blindings for each of them which is symbolised as different hatchings.
Once (B) receives the packet, it derives the shared key s_0(for simplicity we call it s_B as it is the shared key with B) by computing

$$s_0 = (a_0)^b = (g^x)^b = (g^b)^x = y_B^x$$

and removes its blindings. This allows (B) to unblind the routing info that tells (B) the public key of the next downstream node (C).

(B) also removes one layer of encryption from the payload. Same happens at (C) and (D): key derivation, unblinding, deleting, shifting, decryption and blinding. In addition to all these steps, (E) finds a message that symbolizes the end of the path and tells that it's the recipient.

Integrity 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 and allows to check whether or not the header was modified which guarantees integrity.

1.2 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. They are redeemable at most once and they lose their value when the channel is closed.

1.2.1 Ticket issuance

A ticket can be issued once two nodes have established a payment channel with each other which means that at least one of them has locked HOPR tokens.
The ticket issuer A (A could also be the packet creator) selects the winning probability of the ticket and the relay fee to use and sets amount to:

$$amount := \frac{pathLength * relayFee}{winProb}$$

If the issuer (A) receives the packet from another node (O) (packet creator) and attempts to issue a ticket for the next downstream node, the challenge is given together with the routing information by the packet. We assume (A) and (O) are not the same.

(A) does not know whether the ticket is a win or not.

(A) sets content of a ticket to:

$t = (recipient, challenge, account.counter, amount, winProb, channel.teration, index)$

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

The data for a ticket is signed by the issuer, and a ticket is the data followed by the signature:

$$ticket := (ticketData, Sig_Issuer(ticketData))$$

where

$ticketData := (amount, winProb, recipient, index, challenge, iteration, chainId, tag, version)$

Challenge

(O) creates a shared secret s_i with all the relay nodes in the channel (A-B-C-Z) by using an offline version of the Diffie-Hellman key exchange.
The shared secret s_i is used as a seed for a PRG (Pseudo Random Generator) to create secret shares s_i, s_i' for each node along the route. Relayers compute s_i and get $s_i + 1'$ from the next downstream node.
The sender (A) creates $challenge_i := (s_i + s_i + 1') * G$ and a hint for B,C,D,Z (let's suppose these are the relay nodes and Z will be the final destination) a "hint" how the promised value s'_C, s'_D, s'_Z is going to look like. The value "hint" is computed as $hint_i := s_i + 1' * G$

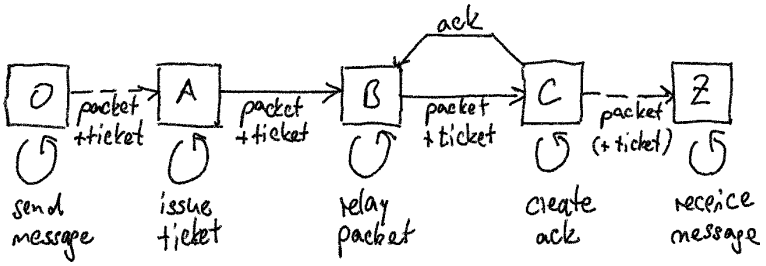

Figure 2: Ticket workflow

1.2.2 Ticket validation

Tickets are received together with packets which means that 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.
Once the recipient (A) receives $s_{i+1}^{(1)}$ from (B) by the secret sharing, it can compute $response := s_i^{(0)} + s_{i+1}^{(1)}$ such that it verifies

$$response * G = ticket.challenge$$

Once the recipient transforms the packet, it is able to compute s_i. The recipient is now also able to extract the routing information from the packet. This includes a hint to the value s_{i+1} given as $hint_i := s_{i+1} * G$
Together with s_i, the node can verify that

$$ticket.challenge = s_i * G + hint_i$$

with

$$s_i * G + hint = si * G + s_{i+1} * G = (s_i + s_{i+1}) * G$$

This allows the recipient to verify that the promised value s_{i+1} indeed leads to a solution of the challenge given in the ticket. If this is not the case, then the node should drop the packet.
Without this check, the sender is able to intentionally create falsy challenges that lead to unredeemable tickets.

1.2.3 Ticket redemption

In order to unlock the ticket, the node stores it within its database until it receives an acknowledgement containing s_{i+1} from the next downstream node. The challenge can be computed from acknowledgement as $challenge := ack * G$.
Once it receives an acknowledgement, it checks whether it stores an unacknowledged ticket for the received acknowledgement. If this is not the case, the node should drop the acknowledgement.
The node then computes the response to the challenge given in the ticket as

$$response := (s_i + s_{i+1}) * G$$

Additionally, the node retrieves the opening value open to the current on-chain commitment and checks whether response, open leads to a winning ticket. This is the case if

$$hash(ticketHash, response, open) < ticket.winProb$$

where

$$ticketHash := hash(ticketData)$$

If this is not the case, the node should drop the ticket. The final recipient of the packet does not receive a ticket because message reception is not incentivized by the HOPR protocol.
The node checks whether the information gained from the packet transformation is sufficient to fulfill the given challenge sent along with the ticket. It then replies with an acknowledgement that includes a response to the challenge.

1.3 Tickets aggregation

The word aggregation means the process of combining things or amounts into a single group. HOPR adds an additional scaling layer that aggregates multiple tickets and redeems them within one transaction.

Since probabilistic payments cannot scale arbitrarily and ethereum gas fees are only increasing it makes sense to aggregate multiple tickets and redeem them within one transaction. The preimage of a winning ticketis stored in the node's database.

The node sends ticket or several winning tickets $ticket_A$ and $ticket_B$ with the corresponding pre-images to the issuer who computes their aggregation $ticket_C$.

The issuer stores on chain $hash(ticket)$ of the redeemed tickets in a space-efficient data storage called Bloom Filters. The issuer computes the modified Bloom Filter by XORing the previous version with the one after inserting the ticket.
Once $ticket_C$ is sent to the smart contract, the intended version of the modified Bloom Filters is recovered by $bloom_{n+1} = bloom_n \vee diff$ and thereby invalidates $ticket_A$, $ticket_B$ as well as $ticket_C$.

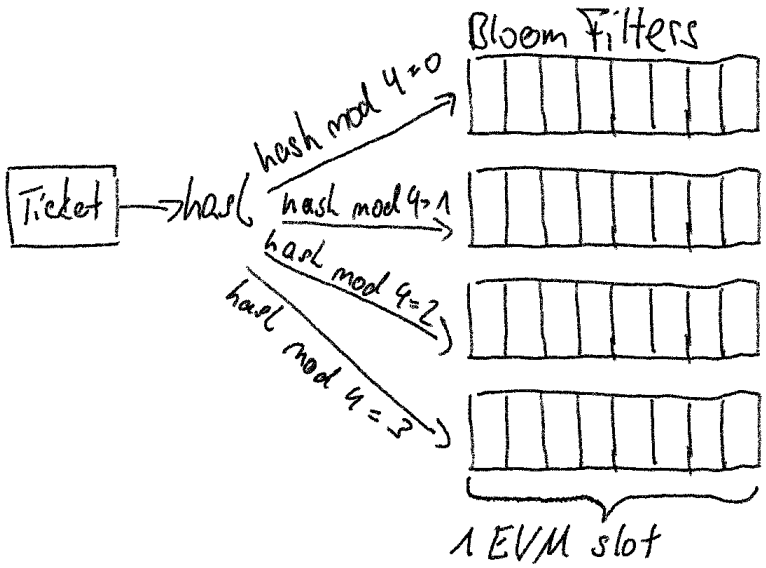

Figure 3: Ticket aggregation

1.4 HOPR incentivization mechanism

HOPR incentivizes nodes in order to achieve correct transformation and delivery of mixnet packets. This is accomplished using a mechanism called "Proof-Of-Relay" with the following second-layer solutions which are both cost effective and privacy preserving.

1.4.1 Probabilistic payments within payment channels

In payment channels, two parties A and B lock some funds within a smart contract, make multiple transactions off-chain and only commit the aggregation on-chain. This implies that the last HOPR acknowledgement contains all previous incentives plus the incentive for the most recent interaction

$$value(ACK_n) = \sum_{i=1}^{n} fee(packet_i)$$

If B received ACK_n before sending $packet_{n-1}$, it has no incentive to process $packet_{n-1}$ rather than $packet_{n-2}$.

Probabilistic payments

The payouts use a concept called "tickets". In this scheme, there is a known selection rate s (assuming $s = 1000$). A ticket can be a win or a loss with $\frac{1}{1000}$ probability which means nodes are incentivized to continue relaying packets as they don't know which ticket is a win.

HOPR uses a custom-made second-layer solution. It is inspired by payment channels and probabilistic payments where incentives can be claimed independently:

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

Hence, there is no added value in pretending packet loss or intentionally changing the order in which packets are processed.

Privacy challenges

The incentives break the unlinkability guarantees inherited from the SPHINX packet format as they reveal the identity of the packet origin who transfers those incentives in the channel using their signature.
To solve this problem, HOPR forward incentives next to the packet.

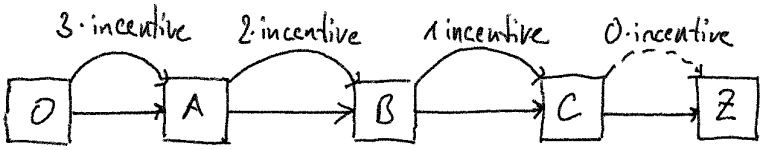

Figure 4: Incentive flow

This however 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 leakage is considered to have a low severity but further research will be conducted on the subject.

1.4.2 Proof Of Relay

HOPR incentivizes packet transformation and delivery using a mechanism called "Proof-Of-Relay". This mechanism makes sure nodes 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.

Since "Proof-Of-Relay" is used to make the relay services of nodes verifiable, it is the duty of each node to check that given challenges are derivable from the given and the expected information. Packets with inappropriate challenges should be dropped as they might not lead to winning tickets. Therefore, the sender of the packet also provides a hint of the expected value that a node is supposed to get from the next downstream node (as explained in the ticket section).

1.4.3 The Importance of Stake

In order to incentivize reliable connections, we require payment channels to be staked. We also assume stake is correlated with availability and thus with reputation. Therefore, weighing by stake the selection of which mix nodes to include in the channel graph results in a mixnet populated by nodes that have the reputation of offering a high quality of service. Staking is also a countermeasure for sybil attacks, as creating many node identities becomes expensive.

1.5 Path Selection

HOPR uses a pseudo-random selection algorithm to determine the identities of nodes participating in the network relaying service.
For each round, a subset k-of-n nodes will be chosen. The selection is divided into two steps:

1. Pre-selection: During this phase, a subset m-of-n nodes will be selected based on different factors:

   - Availability
   - Payment channel graph
   - Reputation
   - Stake

   Each node gets a score that is proportional to the previously mentioned 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 $r_i * score(edge_i)$

Once an edge is selected, it is added to the current path. All paths are then sorted according to their weight and the path with the highest score is expanded next.

1.6 On-chain Commitment

Commitment schemes

A commitment scheme is a protocol between usually two parties A and B and fulfills two properties (hiding and biding).

Hiding: The ability to commit a value only known by the sender.
Binding: The committed value must be the only one that the sender can compute and that validates during the reveal phase.

1.7 Setup phase

Once a node joins the HOPR network, it creates an iterated commitment and stores the opening key in the database. Iterated commitment scheme means openn opens cm_n whereas $open_{n-1}$ opens $cm_{n-1} = open_n$ and so on.
The final commitment is computed as $cm_n := hash_n(r)$ where hash is a preimage-resistant hash function and r is chosen uniformly at random by the node in order to prevent the issuer from knowing whether the ticket will be a win or not.

Furthermore, it will prevent it from tweaking the given challenge such that the ticket cannot be a win. The number of iterations n can be chosen as a constant and should reflect the number of tickets a node intends to redeem.

1.8 Opening phase

In order to redeem a ticket, a node has to reveal the opening to the current commitment cm_n that is stored in the smart contract.
The opening shouldn't be revealed otherwise since it discloses whether a ticket is a win or not. The ticket is a win if:

$$hash(ticketHash, response, open) < ticket.winProb$$

where

$$ticketHash := hash(ticketData)$$

The opening is computed as $open_{n-1} = hash_{n-1}(r)$ such that $cm_n = hash(open_{n-1})$.

The on-chain logic verifies whether the latest opening $open_n$ indeed opens the current on-chain commitment cm_n. If this is the case, the current on-chain commitment is replaced by the given opening.
The on-chain keeps track of updates to the on-chain commitments to prevent double-spending. So whenever a node resets a commitment, a counter, namely *account.counter* is increased and a call to *updateCommitment* invalidates all previously unredeemed tickets.

1.9 Event propagation

HOPR uses a decentralized event propagation method to allow information to be aggregated at many points in the network and shared with other nodes. This method prevents single point of failure vulnerabilities that service providers like infura or Alchemy could create. It also allows HOPR end users to use the HOPR network without needing any additional computational and bandwidth resources.

HOPR decentralized trustless event propagation

HOPR uses Ethereum full nodes to fetch events and forward them to their HOPR instance. HOPR then aggregates the events and store them chronologically:

- $Open_i := (channelId, open, balance, balance_a)$
- $Close_i := (channelId, close)$

The nodes then publish the update independently to the DHT and allow HOPR end users to download it and verify the validity of the data.

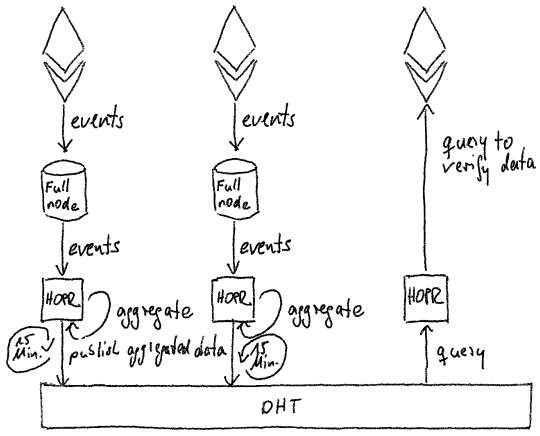

Figure 5: Event propagation

HOPR is a privacy preserving protocol that guarantees sender-receiver unlinkability while still providing high availability.
In order to achieve the aforementioned properties, nodes use DHT to place information in the network and ask selected other nodes to replicate it and update that list every 5 minutes.
Event logs are set to

$$log_i := hash(log_{i-1}||Open) \quad or \quad log_i := hash(log_{i-1}||Close)$$

where $log_0 := hash(0)$.
When a node receives log_i from another node, the validity can be checked by recreating the hash and comparing the value with the on-chain value.

1.10 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 attract the network by:

- Dropping packets or acknowledgements

- Sending falsy packets, tickets or acknowledgements

Since nodes need to monitor the network to select paths, they need to filter nodes that behave inappropriately. In order to do so, we use a reputation system which gives a score to each node that acts as a relayer.
The node's reputation either increases or decreases its probability of being chosen depending on its behavior.

1.10.1 Transitive trust evaluation

The reputation 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.

Sequential propagation

Assume that the trustor a trusts b's believes with: $tr(a,b,x,k) =< (td^b, dtd^b) >$ and b trusts c's performance with: $tr(b,c,x,k) =< (td^p, dtd^p) >$, then the trust relationship from a to c can be derived as follows:

$$td^p(a,c,x,k) = td^b(a,b,x,k).td^p(b,c,x,k) + dtd^b(a,b,x,k).dtd^b(b,c,x,k)$$

$$dtd^p(a,c,x,k) = dtd^b(a,b,x,k).td^p(b,c,x,k) + td^b(a,b,x,k).dtd^b(b,c,x,k)$$

Parallel propagation

Assume that entity a trusts (directly or indirectly) entities $b_1, ..., b_n$ with a certain degree and that entities $b_1, ..., b_n$ trust the entity c with some degree, the aggregation of trust from a to the entity c is:

$$td_g(a,c) = 1 - \prod_{i=1..n} (1 - td_i) \quad and \quad dtd_g(a,c) = \prod_{i=1..n} dtd_i$$

where $< td_i, dtd_i >$ is the sequential aggregation of trust degree between a and c throughout the path $a -> b_i -> c$.

1.10.2 Evaluation of trust

Step1: Partial Trust Nodes can create small groups of neighbors, inside each group, nodes challenge one another using different testing techniques to figure out which members of the group are honest or malicious. These challenges will allow every node to construct trust knowledge about its neighbors.

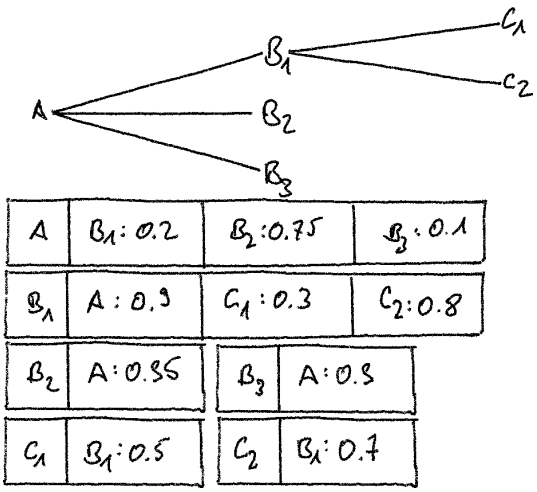

Figure 6: Evaluation of trust

Step2: Global Trust In order to compute the global trust, all nodes must collaborate and share their collective partial trust values with everyone on the network. The reputation score is updated continuously on the HOPR network (once every week or so).
We create a "matrix of trust" that contains the trust degrees given by each entity of the network towards its neighbors. The final trust given to a by the network is the average of all inputs given by the other nodes in the network.

2 Threat Model

Although we assume that nodes in the HOPR network can communicate reliably, the network can still be damaged by malicious attacks and node failures. We assume byzantine nodes with either the ability to observe all network traffic and launch network attacks or inject, drop or delay messages.

There are different attack vectors which could threaten the security of HOPR network, in the following section we mention these attacks and the mitigation methods used by HOPR to resist them:

- **Sybil attacks:** An attacker uses a single node to forge multiple identities in the network, thereby bringing network redundancy and reducing system security. This attack is expensive to conduct since they must stake a lot of HOPR tokens within each malicious node they create in order to increase their probability of being chosen as a relayer and thus attacking the network.
- **Eclipse attacks:** The attacker seeks to isolate and attack or manipulate

a specific user that is part of the network. This is a common attack in peer-to-peer networks since nodes have a hard time identifying malicious ones as they don't have a global view of the whole network. The cost of launching an eclipse attack is high since HOPR nodes are constantly challenging other peers and keeping a reputation score for each node.

- **Camouflage attacks:** A malicious node pretends to be an honest one most of the time. When its reputation value reaches a high level, it occasionally attacks the system. Since the attacker needs a long time to gain enough reputation score and be selected. Based on this, the system can still perform well.

- **Observe-Act Attack:** The attacker observes the reputation score distribution of honest nodes, then control malicious nodes to act and have the same reputation score in order to increase the probability that most malicious nodes are chosen as relayers. This attack however reveals the identities of malicious nodes which conduct this attack and their reputation will be reduced if not loose their stake.

- **Circuit position leaks:** In HOPR, payments are performed hop-by-hop along a packet's route which leaks the relayer's position within the selected path. This leakage is considered to have a low severity but further research will be conducted on the subject.

Security Goals

In addition to resisting the previous attacks, the HOPR protocol has been defined to meet these security goals which are inherited from the Sphinx packet format:

- Sender-receiver unlinkability: The inability of the adversary to distinguish whether $\{S_1 \rightarrow R_1, S_2 \rightarrow R_2\}$ or $\{S_1 \rightarrow R_2, S_2 \rightarrow R_1\}$ for any concurrently online honest senders $S1, S_2$ and honest receivers R_1, R_2 of the adversary's choice.

- Resistance to active attacks: Resistance to active attacks like tagging and replay attacks where the adversary modifies and re-injects messages to extract information about their destinations or content.