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

Dr. Sebastian Bürgel, Robert Kiel

November 2019, V1

1 Introduction

The Web3 comprises a range of technologies that enable truly decentralized applications (dApps) featuring increased privacy and resilience for the next wave of web applications. dApps on the Web3 do not rely on central infrastructure with central points of failure but instead support self-custody of wealth and data. Beyond technical innovation, the Web3 ecosystems strives to innovate the business and organizational status quo of today's web2.0. Are rent-seeking for-profit corporations with shareholder meetings and CEOs the only possible way to generate value for users? Are shareholders necessarily the only financial beneficiaries of a project while most work and value might be community-created? The ICO wave of 2017 has not just created an alternative funding scheme for high risk early investments via tokens but also led to an increasing number of projects that aspire to the Web3 vision and build its infrastructure. HOPR believes in that privacy-first, decentralized and self-empowering Web3 vision and delivers a privacy foundation for the Web3.

1.1 Pillars of the web3

Test While several components of the web3 are still heavily in the making, some projects are already today giving a glimpse of that future web architecture. Early examples working towards dApps are decentralized organizations (DAOs, e.g. by Aragon), social media platforms such as Akasha or financial products like Maker's DAI or non-custodial trading venues like Uniswap. Currently, we see technological pillars emerging that enable developers to build true dApps which do not rely on central infrastructure anymore:

- Financial asset management systems enabled by blockchains such as Ethereum or ZCash
- Data storage solutions like Filecoin or NuCypher
- Computation providers like Golem or Enigma

1.2 The web3 stack

In this new ecosystem, multiple decentralized applications (dApps) interact with one another as well as with these core technologies. All pillars feature projects that focus on privacy within that domain. For example, we see on-chain privacy solutions such as AZTEC and MatterLabs on Ethereum or ZCash, private data storage by re-encryption in NuCypher and privacy-preserving computation in Enigma. At the same time, the ecosystem is lacking a go-to solution for network-level privacy enabling communication between separate networks, applications and users. Some dApps make use of Whisper which is developed by the Ethereum community but which - similar to other broadcast schemes - suffers from scalability restrictions when used for point-to-point communication and unclear delivery behavior. We build HOPR as a metadata-private communication foundation for the Web3 and the web of today. As such, HOPR solves the privacy fallacy of end-to-end encryption that we see in today's web applications: While the encrypted message itself might not be accessible to third parties, metadata such as "Who are you talking to?", "How often are you talking to them?", "From where?", "At what time?", "How many and long messages do you exchange?" and more, are leaking and accessible to various third parties including wifi-providers, internet service providers, device manufacturers and other low-level hard- and software vendors.

dApps		
Assets Bitcoin, ZCash, tokens	**Storage** IPFS, NuCypher	**Computation** Golem, Enigma
Messaging **HOPR**		

1.3 Protocol layers of the web3

HOPR fills the gap between peer-to-peer (P2P) networks and dApps that exchange sensitive information. It adds metadata privacy on top of an existing P2P layer that is used in form of libp2p or WebRTC in decentralized architectures today. It is compatible with underlying network protocols such as TCP/IP or QUIC. Depending on the application, one layer above HOPR could be an optional storage / sync layer like Matrix which then enables e.g. chat application with longer-term message caching.

Layer	**Purpose**	**Example**
Application	Application logic	Chat app, M2M comms
Storage / sync	Synchronization of data, version management, medium-term message caching	Matrix
Privacy	**Scalable & decentralized metadata protection, incentivization for packet relayers, short-term caching**	**HOPR**
P2P	Overlay routing, NAT traversal	libp2p, WebRTC
Network	Underlay routing, congestion control	TCP/IP, QUIC

In the blockchain world, HOPR complements technologies that provide on-chain privacy. Data packets produced by dApps may not just contain valuable data but may also reveal metadata that can be linked to real-world identities. On-chain privacy, for example, is of limited impact if a network observer can link metadata to a social media account in order to attack the person because of the fact *that* they used privacy-preserving financial networks - without knowing *what* exactly they used them for.

2 Architecture Overview

The HOPR protocol outlined in this paper comprises various modules which are detailed in later sections. A reference implementation of the HOPR protocol is the HOPR node software. A first version of the HOPR node is written in Javascript / NodeJS to optimize for time to release and a steep learning curve. At a later time, a more efficient and secure implementation, possibly written in Rust, will be provided. The HOPR node comprises a core module of the payment and message layer as well as a REST-JSON API which is interfaced either directly or via libraries by applications that run on top of HOPR.

<table>
<tr><td>Application</td><td colspan="2">CLI / UI</td></tr>
<tr><td rowspan="2">HOPR</td><td colspan="2">APIs & libraries</td></tr>
<tr><td>Message layer</td><td>Payment layer</td></tr>
<tr><td>Infrastructure</td><td>libp2p</td><td>Ethereum</td></tr>
</table>

HOPR provides privacy by relaying packets from sender via multiple relay hops who receive a payment for their services to the recipient. The message layer presented in section 3 ensures security, integrity and metadata privacy of the data packet as it is sent from the sender via relay hops to the recipient. It is based on libp2p and thus supports various lower-level transport modules such as TCP, WebRTC, WebSocket or UDP. The payment layer detailed in section 4 allows relay nodes to get paid for providing privacy for sender and recipient in a fashion that does not expose critical metadata about the packet. The payment happens via deterministic micropayment channels outlined in section 5. These payment channels are interfacing an existing public ledger, the current proof-of-concept implementation relies on the main Ethereum blockchain. The fairness of this payment mechanism relies on our Proof-of-Relay secret sharing mechanism described in section 6. Many of these processes rely on cryptographic keys, the derivation of which is discussed in section 7. The payments that are carried out between nodes in the HOPR network are performed in HOPR tokens which is introduced in section 8.

3 Message Layer

Each relay node is caching packets for a customizable duration, mixes their ordering before passing them on to the next downstream relay node (or recipient) so that packets are hard to link for an outside observer. In addition to this mixnet functionality, HOPR leverages onion encryption: Each packet is encrypted by the sender multiple times such that only the chosen relay nodes can decrypt it, read the next destination and pass on the payload which now has one onion encryption layer less. This does not only increase security by requiring all nodes and the recipient to decrypt the plain-text message but it also makes incoming and outgoing packages indistinguishable for a passive observer. Beyond traditional onion routing, HOPR packets do not only contain information of where to send the packet but also contain a payment. The payment is credited for the node that successfully relaid the packet to the next downstream node in a privacy-preserving fashion.

4 Payment Layer

HOPR nodes get paid for their privacy-enabling packet relay services via customized payment channels. In contrast to on-chain (layer 1) payments, payment channels have a number of properties that make them particularly suitable for the purpose of paying for privacy-preserving network traffic:

1. **Cheap**: Opening and closing payment channels are the only on-chain interactions that cost transaction fees. All intermediate update transactions are exchanged in a P2P fashion without incurring settlement fees.
2. **Fast**: After opening a payment channel, update transactions are not slowed down by the consensus layer (blockchain) as these transactions

are signed off between both parties of the payment channel. Thus, the transaction throughput is decoupled from the blockchain and only limited by minimal processing and network latencies.

3. **Minimal counter-party risk**: Either side of the payment channel can choose to close the channel at any time. However, the counter-party has some time to provide proof to the smart contract that they have a later update transaction with the signature of the other side. This allows each party to always settle on the latest balance on-chain. If the discrepancy between the closing transaction and the actual balance is less than the transaction fee of the closing transaction (gas) then it is not economically rational to send the later update transaction. This counter-party risk is however limited to the cost of the closing transaction which is a few cents today on the public Ethereum chain.

4. **Minimal public metadata**: Since payments are not settled on-chain on a packet-by-packet basis but in bulk, the metadata is limited to the bulk information of how heavily a certain route in the network was used, e.g. on a monthly basis or whatever settlement interval the parties choose. Importantly, no linking between individual packets and payments, time or path is possible.

Traditional payment channel implementations such as those used by Raiden are having a number of shortcomings that HOPR improves upon. Specifically, HOPR requires a payment channel architecture that allows for the following:

1. **Pay for relaying**: Relayer should not be able to get paid for non-existing messages, i.e. they should not cheat the upstream node from which they receive the payment. This means that the payment and package delivery needs to be tightly coupled and one payment needs to be submitted per packet.

2. **Proof of relay**: Relayer should not be able to get paid unless they have actually forwarded the package to the next downstream node. This requires cooperation between the receiving relayer and the next downstream node. Only upon confirmation of the next downstream node should the relayer receive their payment.

3. **Partial payout**: Relayer should not be unduly punished for not being able to relay a package to the next downstream node. I.e. while they should not be able to get the corresponding payment for the packet that is thus lost, they should be able to get paid for the remainder of the successfully delivered packets.

4. **Efficient settlement**: The relayer should have an efficient means of closing the payment channel. They cannot be required to submit individual proofs for each packet that they relayed as the on-chain transaction fees would be prohibitively high for relaying millions of packets.

4.1 Pay for Relaying

HOPR overcomes the limitations mentioned above by embedding a customized payment channels. The update transaction of the payment channel is embedded in the header of a packet. The payment is, however, not redeemable for the receiving relayer without cooperation of the next downstream node. Consider the following example:

Alice is the sender of a private message and chooses a route via Bob and Charlie who are both relayers to Dave as the recipient.

In this case Alice will pay the entire amount for both relayers (Bob and Charlie) to Bob. Bob now needs to forward part of the payment (and the packet) to Charlie in order to get his payment. Charlie in turn needs to deliver the message to Dave in order to get her payment.

5 Probabilistic Micropayments

6 Proof-of-Relay

Alice employs a secret sharing mechanism so that Bob will only get Alice's payment when he delivered the message to Charlie. Charlie, in turn, only receives her payment from Bob with the cooperation of Dave. The mechanism relies on a secret sharing and elliptic curve multiplications as a cryptographic one-way function. Bob receives the curve point S which Alice (sender of a message) computed as $S = s * G$ where $*$ denotes an elliptic curve multiplication and G is the base point of the curve. Bob also receives the public key half S_b from Alice. He can then obtain the message secret key half s_a from intermediate key material in the SPHINX message header. Bob will relay the entire packet to Charlie and in return he will receive the second half of the message secret key s_b back from Charlie. After Charlie passes the second key-half s_b to Bob, Bob can compute the message secret key $s = s_a + s_b$ where $+$ denotes addition over a finite field of the elliptic curve. After Bob relayed N transactions from Alice and also got all the key-halves from the next downstream node, he can thus submit the sum of all secrets $s_{total} = \sum_{i=0}^{N} s_{i,a} + s_{i,b} = \sum_{i=0}^{N} s_i$ to release his payments.

In order to prevent Bob from faking s_{total} and thereby extracting a larger amount from a payment channel than he should, a signature is required from Alice, similar to traditional payment channel implementations. Therefore, every update transaction from Alice contains a signature over $S_{total} = s_{total} * G$. Now the smart contract

- ensures validity of the signature of S_{total} from Alice $sig(S_{total}, Alice)$
- ensures validity of the pre-image $S_{total} = s_{total} * G$ and
- facilitates the payout for the successfully delivered packages.

6.1 Partial Payout and Efficient Settlement

In reality, not all packages are successfully delivered. As a consequence, Bob misses some s_b from the packet that got lost or for which the downstream relay node was not responsive. In turn, Bob is not able to submit a correct s_{total} matching the corresponding S_{total} for which he has a valid signature from Alice.

For settlement of payment channels for which not all packages have been successfully delivered, a partial settlement needs to be available. Assume that for a total of N packages, the first X were successfully delivered and the last Y failed so that $N = X + Y$, ordering of successful and unsuccessful messages is irrelevant and only serves as a simple example. In analogy to the above we then have

$s_{success} = \sum_{i=0}^{Y} s_{i,a} + s_{i,b} = \sum_{i=0}^{Y} s_i$,
$s_{fail} = \sum_{i=Y}^{Y+X} s_{i,a} + s_{i,b} = \sum_{i=Y}^{Y+X} s_i$,
$S_{success} = s_{success} * G$,
$S_{fail} = s_{fail} * G$

Bob can calculate $s_{success}$ in order to get the corresponding payout for each of those X successfully delivered packages. He then needs to submit the remainder S_{fail} for which he will not receive a payout such that Alice's signature can be validated.

Then the smart contract:

- ensures validity of the signature of S_{total} from Alice $sig(S_{total}, Alice)$
- ensures validity of the pre-image $S_{total} = s_{success} * G + S_{fail}$
- facilitates a payout for successfully delivered packages corresponding to $s_{success}$

7 Key Derivation

The key halves s_a and s_b are derived via the HKDF key derivation function which is based on HMAC-SHA256. The input key material (IKM) is obtained from the SPHINX packet header and different salt values are used for the different types of keys (e.g. s_a or s_b).

The sender of a packet (Alice in the example above) generates a secret pseudo random number x_{b_0} which she then turns into a curve point $X_{b_0} = g^{x_{b_0}}$. This allows Bob to derive the input key material $IKM = (X_{b_0})^{b_0}$. Finally Bob can then use this IKM to generate the different types of keys $key = HKDF(IKM, salt)$.

The IKM can be derived both by Alice (from the left side of the equation below) and also from Bob (from the right side in the equation below):

${B_0}^{x_{b_0}} = (g^{b_0})^{x_{b_0}} = g^{x_{b_0} * b_0} = (g^{x_{b_0}})^{b_0} = (X_{b_0})^{b_0}$

x_{b_0} : Alice generates this (random number) and keeps it secret

b_0: private key of Bob

B_0: pub key of Bob, Alice knows that

X_{b_0}: This is what Alice stores in SPHINX header for Bob and from which he derives the input key material

$(X_{b_0})^{b_0}$: This is the input key material that Bob derives with his private key

8 HOPR Token

9 Objectives

The HOPR team engaged with various contributors and organizations in the blockchain and web3 ecosystem to establish the objectives towards a decentralized and privacy-preserving communication protocol. In the following sections we detail how the identified objectives are being addressed by HOPR.

9.1 Ensuring metadata protection

The HOPR message layer comprises a Chaumian mixnet. As such, no node in the network and no passive observer can tell if a certain node was sender or relayer of a message. Likewise, they cannot tell if a particular node was receiver or relayer of a message. This works as long as sender, relayer and recipient are subject to sufficient traffic so that they can mix their packets into the existing background packet traffic. As several nodes are relaying the traffic between sender and receiver, it is hard to link the two and thus establish who is talking to who.

The payment channels that HOPR leverages on the payment layer, are not settled on-chain after every packet and thus privacy that was established by the message layer is maintained by the payment layer. The channels between sender-relayer, multiple relayers and between relayer and recipient are settled infrequently, e.g. on a monthly basis, and therefore make it difficult to link payment- and message layer activities. Relayers can choose when they want to settle, they might choose to do so frequently in order to have a constant revenue but on the other hand they would not settle too often as that comes at the cost of on-chain transaction fees. In addition, when settling for a very low number of relayed packets (worst case is all relayers of a path are settling after just one single packet) closing the payment channel might leak some information about the path along which a certain packet was routed. To prevent this and guarantee sufficient privacy, a certain amount of traffic should be routed along the relay nodes that are chosen for a particular packet before the corresponding payment channels are settled. It is the task of the sender of a packet to choose a route via relay nodes that fulfill such conditions which are also deemed sufficiently trustworthy. As trust might be subjective, HOPR does not impose a strategy for establishing a path and instead allows the sender to choose relay nodes.

Objectives that HOPR achieves:

1. Sender anonymity (who sent a message?)
2. Receiver anonymity (who read a message?)

3. Sender-receiver unlinkability (who is talking to whom?)

9.2 Convenience, Usability

HOPR does not make assumptions about latency or anonymity and instead lets applications define these parameters. Higher latency provides for more efficient mixing of packets and thus increased anonymity but might not be suitable for all applications (e.g. instant messaging needs lower latencies than e.g. email services). The SPHINX packet format that HOPR utilizes provides for high anonymity guarantees and at the same time contains overhead bandwidth. While traffic through HOPR will be significantly slower than direct communication due to the involvement of intermediate relay hops as well as additional artificial latencies to mix packets, the throughput of HOPR should allow for reasonable bandwidth to at least send several Kilobytes of traffic per second per sender. The payment layer aims at implementing efficient cryptography so that even low-energy devices are capable of sending, relaying and receiving traffic. Therefore, HOPR does not involve cryptographic building blocks that are currently en vogue in various web3 projects such as zk-SNARKs or trusted execution environments which require heavy computational resources or specialized hardware that is unlikely to be found in e.g. low-power internet-of-things (IoT) devices that need a metadata-private machine-to-machine (M2M) communication protocol.

Objectives that HOPR achieves:

4. Reasonable latency (under 5 seconds, to allow for instant messaging)
5. Reasonable bandwidth (not specified, ability to work with mobile data plan in undeveloped countries)
6. Adaptable anonymity (adjustable pricing and resource consumption depending on how anonymous you want to be)

9.3 Decentralization

HOPR is a decentralized network without central points of failure and it allows anyone to join and use the services. It specifically does not rely on mailbox providers or other trusted parties. The message layer does require some on-chain activities for opening or closing/settling payment channels but existing public blockchains today (e.g. Ethereum) are easily capable of handling traffic of up to 1M nodes which would lead to several million transactions per month which arise from a few channel open and channel close transactions per node.

Objectives that HOPR achieves:

7. Scalable (up to approx. 1M active nodes)
8. No specialized service providers (pure peer-to-peer protocol)

9.4 Incentives

The payment layer is an integral part of HOPR and provides incentives for relayers to get paid in proportion to the number of packets that they relayed. The payment layer is detailed in depth in a later section.

Objectives that HOPR achieves:

9. Incentivization for relayers

10 Future Work

HOPR is by no means complete and there are still various aspects that need further thought, design and implementation work. This section lists the known limitations that need further significant work.

10.1 Get amount from payment channel

We need to find out not just that S was correct but also how much money was associated with it. An approach based on polynomials should work but needs to be defined in more details.

10.2 Economics

Make sure everyone gets sufficiently incentivized and disincentivize bad behavior (and define what that exactly is, e.g. lots of dropped packages, spam, etc). How much does one packet cost? We are envisioning a dynamic fee pricing that is updated, e.g. every 10 days. Similarly to the Ethereum block gas limit adjustment, relayers that did settle channels within that time frame get a vote on the fee. Maybe it is better to have a general token-based voting, not just for the relayers.

10.3 Cover Traffic

We plan to finance cover traffic through an inflationary token model. However, delivering this cover traffic needs to be specified. At first it might be done by HOPR AG, later it might be fully decentralized with path establishment in TEE without leaking information and without ability to cheat.

10.4 QoS / Slashing

If nodes are not online or do not want to support an upstream node then they should get punished by slashing some of their staked funds. Potentially an extra amount needs to be staked separate from the channels to ensure that all connected parties get their unsettled channel funds even if the counter-party got slashed. However, the upstream node should not be able to troll the downstream

node with attempting to slash them although they are online. The downstream node should have a way to respond to such on-chain accusations. One way to reach fairness might be to slash both nodes half of the normal slashing amount in case the downstream node does respond within some time frame. This time frame should be short so that nodes cannot be offline for e.g. one day without getting punished but also cannot be too short (e.g. one minute) so that a transaction did not get mined in time.

10.5 On-Demand NAT traversal

Currently we are routing all traffic between nodes via the bootstrap node to be on the safe side. This does not significantly decrease privacy as HOPR is resistant against passive observers but it is a waste of resources and adds latency unnecessarily. Instead, nodes should detect when STUN or TURN is required and use some other HOPR node in reach as a relay server. That relay service might at a later time also be incentivized.

10.6 Tooling & Documentation

We need minimally:

- HOPR management GUI, probably web-based running on localhost similar to IPFS
- public analytics website of public data (showing channels open/close, cover traffic, some DHT data etc)
- REST JSON API
- JS wrapper library
- libraries in other languages
- tutorials
- detailed documentation
- an integrated example on the website