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

Amira Bouguera, Robert Kiel, Dr. Sebastian Bürgel

July 2021, V2

Abstract

1 Introduction

Internet privacy, also commonly referred to as online privacy, is a subset of data privacy and a fundamental human right. We consider privacy to revolve around control, use and disclosure of one's personally identifiable information. However, our increasingly technologically-driven world puts great pressure on privacy. This is why several actors in the space are continuously developing different solutions to achieve anonymous communication and thus leveraging internet privacy.

1.1 State of the art

1.1.1 Privacy protocols

VPN or Virtual Private Network is a privacy technology that allows creating a secure connection on the internet by building an encrypted tunnel between the client and server. VPN also provides a new IP address (the one of the proxy) to help bypass censorship and geolocalization blocks. Due to the centralized trust model VPNs has, they suffer from inherent weaknesses like being a single point of failure. They are also vulnerable to powerful network eavedropers who can track the routed network traffic.
Those privacy concerns have motivated the creation of a new model of VPNs, dVPNs (Decentralized Virtual Private Networks) with no central authority and backed by blockchain technology. Orchid [1], Sentinel [2] and Mysterium [3] are the most known projects in the blockchain ecosystem providing dVPN solutions. These projects however lack privacy, performance and reliability guarantees due to the fact that regular internet users can be both bandwidth providers and normal users. One of the main reasons behind using decentralized VPN instead of a centralized one is to implement a no traffic logging policy as a way to increase

privacy, this however introduces the risk of using the service illegally (terrorism, drug smugling, child pornography...etc) which holds service providers reliable to government authorities without them having any legal protection as large VPN providers would have. There have also been some incidents reported, where unaware dVPN users have been (ab)used as exit nodes through which DDoS attacks were performed. Similarly to VPN, users have no guarantee whether a dVPN might inspect, log, and share any of their traffic. A promising project called VPN^0 [4] started by the Brave team which provides better privacy guarantees by leveraging 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. This project is still research oriented though.

Another existing solution for network privacy is Tor [5], an open-source software for enabling anonymous communication. Tor is based on onion routing which encapsulates messages in layers of encryption and transmits them through a series of network nodes called onion routers. Tor however is susceptible to end-to-end correlation attacks conducted by an adversary who can eavesdrop the communication channels.These attacks reveal a wide range of information like the identity of the communicating peers. Another project based on onion routing is I2P peer-to-peer network [6]. I2P has different design choices from those of Tor:

- 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 which increases significantly the scalability and resilience against failures since packets are used in parallel.
- Unidirectional instead of bidirectional tunnels: which makes deanonymization harder since tunnel participants see half as much data in unidirectional tunnels 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 are vulnerable to eclipse attacks since no I2P router has a full view of the global network (similar to other peer-to-peer networks) and they also protect against only local adversaries (like Tor) and thus vulnerable to timing, intersection and traffic analysis attacks. I2P have also showed to be vulnerable to sybil and predecessor attacks inspite of the different contermeasures implemented to defeat them.

Mixnets are overlay networks of mix nodes that route messages anonymously similarly to Tor. First mixnet paper in 1981 by David Chaum [7] used a cascade topology where each node receives a batch of encrypted messages, decrypts,

randomly permutes packets, and transfers 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 also 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 discriminate communication messages from random noise messages which increases privacy. What differentiates mixnets from Tor is that mixnets are designed to provide metadata protection from global network adversaries by using cover traffic. 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 while applications like real-time video streaming are better suited for Tor.
One of the well known projects is Loopix [8]. Loopix leverages cover traffic to resist traffic analysis while still achieving low- to mid-latency. To this end Loopix employs a mixing strategy that we call a Poisson Mix that is based on the independent delaying of messages, which makes the timings of packets unlinkable.

The goal of each one of these projects is to acheive low latency, low bandwidth overhead and strong anonymity or as we call it the anonymity trilemma. All the previous mentioned projects except VPN and dVPNs lack the economic incentive which could result in scaling issues and poor performance. Tor and I2P for example rely on donations and government funding only which covers the cost of running a node. This has discouraged volunteers to join the network and the number of routers in both networks hasn't increased much for the last few years. Mixnets are also based on a group of volunteer agents who lack incentives to participate. 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, malicious volunteers can sabotage the system by extracting and using their coins without performing their task which consists of forwarding anonymized messages since there is no verification whether the message arrives to its final destination or not. Bandwidth providers in dVPNs share their resources and are granted tokens accordingly as way of payment for their services. This is done using the blockchain technology. A good example of such technologies is Mysterium: an open source dVPN completely built upon a P2P architecture. Mysterium runs a smart contract on top of Ethereum to make sure that the VPN service is paid adequately.

1.1.2 Scalability Layer 2 protocols

Blockchain technology (mostly public blockchains like Bitcoin and Ethereum) suffers from a major scalability issue which is due to the fact that every node in the network needs to process every transaction, validate it and stores a copy of

the entire state. The number of transactions Ethereum can process for example cannot exceed that of a single node which is currently 15 transactions per second.

There have been multiple solutions proposed to treat the scalability issue such as sharding and off-chain computation. Both of these solutions intend to create a second layer of computation in order to reduce the load on the blockchain mainnet.
Off chain solutions like Plasma [9], Truebit [10] 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" [11], a payment channel is opened between two parties by committing a funding transaction, followed by making any number of transactions that update the channel's funds without broadcasting those to the blockchain, then closing the channel by broadcasting the final version of the settlement transaction. Updating the channel balance is done by creating a new set of commitment transactions, then trade revocation keys that 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. But if one of them 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.
A channel can no longer be used to route payments from the moment that the close is initiated. There are different closing channel transactions depending on whether both parties agree or disagree to close 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 close 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 close must wait a certain delay to receive their money, this delay time is negotiated by the nodes beforehand to receive their funds.
The Raiden network is a layer 2 payment solution for the Ethereum blockchain. The project builds off the same technological innovations pioneered by the Bitcoin Lightning Network by facilitating transactions off-chain, but focuses on support for all ERC-20 compliant tokens. Raiden differs in operation from its main chain because it doesn't require global consensus. However, to preserve the integrity of transactions, Raiden powers token transfers using digital signatures and hash-locks. Referred to as balance proofs, this type of token exchange uses payment channels. Raiden network also introduces a concept called "mediated transfers" which allows nodes to send payments to another node without opening a direct channel to it.

1.2 The HOPR vision

HOPR is a decentralized incentivized mixnet that leverages privacy by design protocols. HOPR aims to protect people's metadata privacy and give them the

freedom to use internet services safely and privately. HOPR runs on top of the Ethereum blockchain and uses these mechanisms to ensure users privacy via incentivisation: sphinx packet format and packet mixing, proof-of-relay and probabilistic payments.

- *Privacy by design:* is an approach to systems engineering and calls for privacy to be taken into account throughout the whole engineering process. HOPR believes that the Internet is a public good – a digital commons that should be safe and secure for all its users. However, it is impossible to provide such privacy using the current Internet infrastructure. What is needed is a new privacy infrastructure on top of the existing Internet. This is what HOPR has built using the sphinx packet format and packet mixing.

- *Decentralization:* At the heart of HOPR, there is a global, decentralised network of nodes running the HOPR protocol. Decentralisation ensures that the network is independent, with no one single entity to influence its development or manipulate outcomes 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:* This is biggest innovation of the HOPR design. Previous privacy technologies like Tor didn't incentivize node runners to provide service which discouraged new users to join. By incentivising users, HOPR will lower the barrier to adoption. Incentivisation also encourages good behavior as the only way to earn token rewards is to behave correctly and follow the protocol.

2 Threat Model

Although we assume that nodes in the HOPR network can communicate reliably, the network can still be damaged by malicious actors and node failures. We assume a threat model with 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 in practice since the attacker must stake lots 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.

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 S_1, 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.

3 Sphinx Packet Format

HOPR uses the SPHINX packet format [12] to encapsulate and route data packets in order to achieve sender and receiver unlinkability. The SPHINX packet format determines how mixnet packet are created and transformed in order to relay them to the next downstream node. Each sphinx packet consists of two parts:

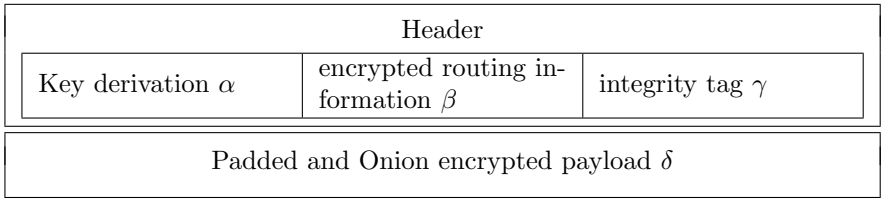

Figure 1: Sphinx packet format

3.1 Construction

We will explain in the following all the steps a sphinx packet goes through in order to arrive to its final destination starting from the key derivation in order to extract new shared keys with all the relay nodes to unblinding the routing information using those shared keys in order to find the public key of the next relay node and checking the routing information's integrity before sending it. Each node then deletes that information and replaces it with their own blinding and decrypts one layer of the payload which has several layers of encryption similar to onion routing.

Notation: Let $\kappa = 128$ be the security parameter. An adversary will have to do about 2^{κ} operations to break the security of Sphinx with non negligible probability. 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 [13]. 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 modeled 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 [14].

Each node i has a private key $x_i \in \mathbb{Z}_q^*$ and a public key $y_i = g^{x_i} \in G^*$. The α_i are the group elements which, when combined with the nodes' public keys, allow computing a shared key for each via Diffie-Hellman (DH) 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 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 final destination of the packet (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 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 that's added to the final receiver's compressed public key. For ECDSA public key compression, the x coordinate would be the only one used and would be prepended by 02.
The y coordinate will be extracted from x by resolving the elliptic curve equation $Y^2 = X^3 + aX + b$, with two given a and b parameters. 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 looks as the following:

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

and

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

for $0 \leq i < v-1$

Such that y_Z is the destination's public key in a compressed form (only the x-coordinate with size 32bytes instead of 64) and $|y_Z|$ is its length. ρ is a pseudo-random 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

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

where $\phi_0 = \epsilon$ is an empty string. ϕ_i is generated using the shared secret s_{i-1} and used to ensure the header packets remain constant in size as layers of encryption

are added or removed. Upon receiving a packet, the processing node extracts the information destined for it from the route information and the per-hop payload. The extraction is done by deobfuscating and left-shifting the field. 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 which each hop will generate) 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 pseudo-random number generator 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 don't use I since we don't currently use SURBs. We also include a *hint* and *challenge* values in β that are defined in Proof Of Relay section. These values aren't included in the original sphinx paper but will be 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.
Some 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 follow:

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

(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 follow:
(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 B 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 tag γ_1 and β_1, which should be forwarded to the next hop.

Delete and shift After that (B) extracts the public key of (C), it deletes the routing information from the packet. Afterwards, it 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:

β_0		
y_1	γ_1	β_1

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) or as defined 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 & Decrypt The encrypted payload δ is where the actual message is hidden and 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 where k_1, k_3 will be used to key the stream cipher and k_2, k_4 to key the hash function. The sphinx uses 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 $|m_r| = j - w$ so we get $m = m_r \| m_l$.

Encryption The blocks m_l and m_r are transformed using a 4-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 The decryption happens as follow:

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

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. Authentication of packet integrity is done 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.

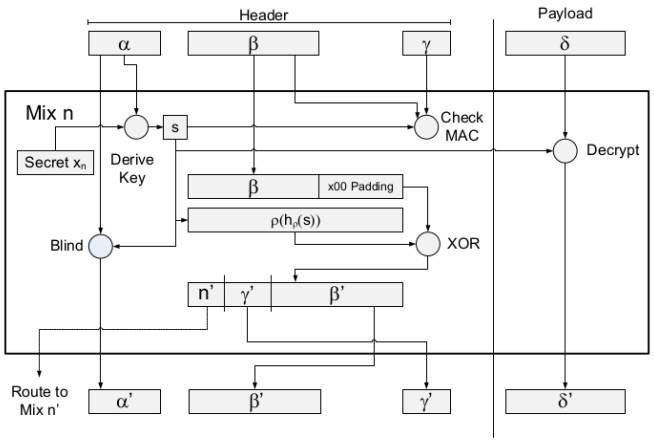

Figure 2: The processing of a Sphinx message [12]

Same happens at (C) and (D): key derivation, unblinding, deleting, shifting, integrity check, decryption and blinding.

3.2 Implementation choices

Within HOPR, the following cryptographic primitives were used:

- **Cyclic group:** The cyclic group used in the HOPR Sphinx implementation is an elliptic curve group on the secp256k1 curve and thus operations will be done on the elliptic curve.
- **Hash function:** BLAKE2s hash function which is a cryptographic hash function faster than SHA-2 and SHA-3, yet is at least as secure as SHA-3 and produces digests of any size between 1 and 32 bytes.
- **MAC:** HMAC based on a hash function BLAKE2s.
- **Encryption scheme:** LIONESS [15] implementation, using BLAKE2s as a hash function and ChaCha20 as a stream cipher.
- **Padding:** In the original SPHINX paper, a sequence of only 0s is used for padding, 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, We replace the 0-padding by a randomized padding for the last exit-mix node when $v < r$. This way, the exit node can't identify where the padding starts and thus won't be able to find the path length that preceeds the padding. 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.

3.3 Packet serialization

4 Cover Traffic

5 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 layer 2 solutions which are both cost effective and privacy preserving.

5.1 Probabilistic payments

In traditional 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. By default a channel is bidirectional which means that both A and B can send and receive transactions within the channel no matter who created it. There is another type of channels which we use in HOPR called unidirectional channels where the channel creator is the sole owner of funds in the channel and the one able to create tickets. A channel created from A to B is different from a channel created from B to A.

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

HOPR uses *acknowledgements* which allow every node to create a message that acknowledges the processing of the packet to the previous node. This acknowledgement contains the cryptographic material to unlock the payout for the previous node. Note that acknowledgement is always sent to the previous node - even if there was no payment.
The fact that we are using payment channels implies that the last HOPR acknowledgement contains all previous incentives plus the incentive for the most recent interaction

$$value_(ACK_n) = \sum_{i=1}^{n} fee_{packet_i} \tag{9}$$

where n is the total number of mixnet packets transformed.

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

To avoid this limitation of traditional payment channels, HOPR utilizes probabilistic payments

In probabilistic payments, the payouts use a concept called "tickets" (see section Tickets), a ticket can be either a win or a loss with a certain winning probability. This means nodes are incentivized to continue relaying packets as they don't know which ticket is a win.

HOPR uses a custom-made layer 2 solution. It is inspired by payment channels and probabilistic payments where incentives can be claimed independently:

$$value(ACK_i) = value(ACK_j) \quad for \quad i, j \in \{1, n\} \tag{10}$$

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

5.1.1 Channel management

Initially, each payment channel in HOPR is *closed* which means that in order to transfer packets, those channels have to be opened. There are four distinct channel statuses which are represented in the following scheme:

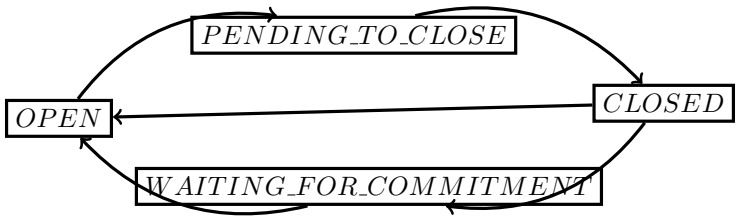

Figure 3: Channel statuses

Opening a channel Nodes can open channels to other nodes by the following: A calls the method *fundChannelMulti()* such that:

$$fundChannelMulti(A :< \lambda >, B :< \mu >)$$

where λ and μ are the amounts to be staked by the sender A for A and B. Both values can also be equal or any of them could be zero. This function opens two unidirectional channels $A \rightarrow B$ and $B \rightarrow A$ if both staked amounts are specified. If not, only the channel $A \rightarrow B$ is opened where A is the sender and B is the recipient. Once the call has succeeded, an on-chain event *WAITING_FOR_COMMITMENT* is emited.

The destination address of the channel must now commit in order for the channel between both parties to be open. The channel destination (let's say it's B) can now call *bumpChannel()* function to make a new set of commitments towards this channel. This call will trigger an onchain event *ChannelIsOpened* and bumbs the ticket epoch to ensure tickets with the previous epoch are invalidated

Redeem tickets As long as the channel remains open, nodes can claim their incentives for forwarding packets which is represented as tickets (see ticket section). Tickets are redeemed by dispatching a *redeemTicket()* call.

If/when B tries to redeem a ticket from the channel $A \rightarrow B$ (spending channel) but also there is an open channel $B \rightarrow A$ (earning channel), B's rewards will go to $B \rightarrow A$. Otherwise, rewards will be sent directly to B.

Closing a channel Nodes can close a payment channel in order to access their funds. The way to do so is using a timeout. Only the channel creator (let's say it's A) can initiate the process by calling *initiateChannelClosure()*. This changes the state to *pending_timeout* and triggers an on-chain event. Other nodes should monitor blockchain events to be aware of this change and thus will have now time to claim not yet claimed tickets.

Once the timeout is done, (A) can call *finalizeChannelClosure()* which turns the channel state into *closed*. When channel is closed, funds (stake) are transfered automatically back to (A). Every ticket that wasn't redeemed while channel was open can't be redeemed after closure.

5.2 On-chain Commitment

HOPR uses a commitment scheme to deposit values on-chain and reveal them once a node redeems an incentive for relaying packets. This comes with the benefit that the redeeming party discloses a secret that is unknown to the issuer of the incentive until it is claimed on-chain. The **opening** and the **response** to the PoR challenge (or as we call them in the smart contract *nextCommitment* and *proofOfRelaySecret*) are then used by the smart contract to determine whether the ticket has been redeemed or not.

Definition 5.2.1. *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. Such that 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* ***computational hiding*** *if the following holds:*

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

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

Binding: *A commitment scheme is called* ***computational binding*** *if for all*

bounded polynomial adversary Adv algorithms that run in time t and 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 means that the adversary has limitations on their computational resources.

5.2.1 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 keccak256 hash function which is used in Ethereum) and $comm_0$ is derived as

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

The master key is supposed to 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 in order to ensure that the ticket issuer is unable to determine whether a ticket is a win 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 win.

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.

5.2.2 Opening phase

In order to redeem a ticket, a node has to 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 win, 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

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

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

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

and sets $channel.comm[redeemer] \leftarrow comm_{i-1}$. Hence next time, the node redeems a ticket, it has to reveal $comm_{i-2}$. In addition, each node is granted the right to reset the commitment to a new value which is necessary especially 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 that is used to determine whether a ticket is a win or not, the smart contract keeps track on resets of the on-chain commitment and sets

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

and thereby invalidates all previously unredeemed tickets.

5.3 Proof Of Relay

HOPR incentivizes packet transformation and delivery using a mechanism called "Proof-Of-Relay". This mechanism guarantees that nodes relay services are verifiable.

Construction

- Every packet is sent together with a ticket (check ticket section).
- 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 (check ticket section).

5.3.1 Challenge

(A) creates a shared secret s_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 that's generated from the master DH key sphinx key.

Key derivation We first create a "session secret" s_i then we use it as a seed to derive subkeys $s_i^{(0)}, s_i^{(1)}$ for each node on the route. We do so using a HKDF (HMAC-based Key Derivation Function) which is 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 and optional salt

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

- Expand: Creates an output key material $s_i^{(0)}, s_i^{(1)}$ of a specific length which is expanded from hashes of the pseudorandom key s_i and an optional info message (salt). The process goes as follows:

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

  where $||$ is concatenation and $*$ is scalar multiplication on the curve. h_b is blake2s256 hash algorithm, $|h_b|$ is its length, s_i is the pseudo random key used as a seed and $|s_i|$ is its length.

Relayers compute $s_i^{(0)}$ and get $s_{i+1}^{(1)}$ from the next downstream node.
The sender (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 value "hint" or H is computed as

$$H_i = s_{i+1}^{(1)} * G \tag{11}$$

where $*$ is the curve multiplication operation and G is a generator of the curve (the same used in the sphinx 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

$$T_{c_i} = (s_i^{(0)} + s_{i+1}^{(1)}) * G \tag{12}$$

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

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

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 which will be used in the Proof Of Relay to make sure relay services are verifiable.

6 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 or when the commitment is reset. A commitment is a secret on-chain value that is used to verify whether a ticket is a win or not when it's revealed in order to redeem it.

6.1 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:

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

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 win or not.
(A) sets the content of a ticket to:

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

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

Recipient's Ethereum adress R: is a unique identifier that is derived from the ticket recipient's public key.
Ticket Index I: is set by the ticket issuer and increases with every issued ticket. The recipient verifies that the index increases with every packet and drops packets if 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 unchanged index.
Ticket challenge T_c: is set by the ticket issuer and used to check whether a ticket is redeemable before the packet is been relayed. The packet is dropped if that's not the case.
Ticket Epoch α: is 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 keeping track of updates to the on-chain commitments and invalidates tickets with the previous epoch.
Channel Epoch ζ: is used to give each reincarnation of the payment channel a new identifier such that tickets issued for previous instances of the channel lose their validity 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).

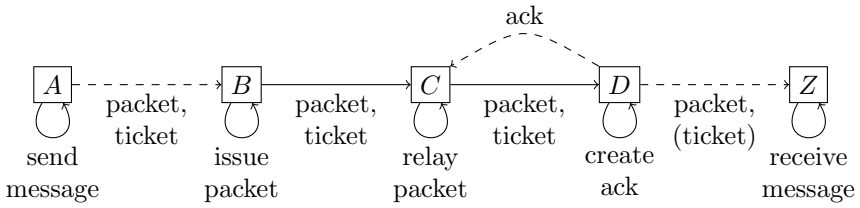

Figure 4: Ticket workflow

6.2 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 (A) receives $s_{i+1}^{(1)}$ from (B) by the 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

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

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

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

which is stored in the sphinx packet header.
The unacknowledged ticket is stored in the database under the hint to the promised value to make sure that the acknowledgement can be afterwards 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, then the node should drop the packet.
Without this check, the sender is able to intentionally create falsy challenges that lead to unredeemable tickets.

6.3 Ticket redemption

In order to unlock the ticket, the node (ticket recipient) 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 $T_{c_i} = ack_i * G$. The node checks the following in order to redeem tickets:

- Channel **exists** and is **open** or **pending to close**. The check happens locally and not on-chain using the blockchain indexer in order not to reveal any metadata. If the node does not have a record about the channel or considers the channel to be in a state different from **open** or **pending to close**, the ticket is dropped and the reception of the accompanying packet is rejected.

- 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 (which we can also refer to as *proof of relay secret*) given in the ticket as

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

- Additionally, the node retrieves the next commitment value $comm_{i-1}$, checks that this value is not empty and that commitment is the hash of next commitment as follows:

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

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

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

  where $t_h = h(t)$ is the ticket hash and h is a hash function. 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.

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

  It then replies with an acknowledgement that includes a response to the challenge.

- Ticket signer must be the ticket issuer. This is done using the

$$ECDSA.recover()$$

ethereum function which recovers the private key used to sign $T = (t, Sig_I(t))$ and verifies if the associated public key is the issuer's public key. The packet is dropped if this test fails.

- Ticket Epoch α and channel Epoch ζ must be equal to the account nonces (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.

- The ticket hasn't been already redeemed and ticket index is 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.

- Channel must have enough funds to cover the costs for the ticket:

$$C_b > \sigma$$

where σ is the ticket amount and C_b is the channel balance.

7 Path Selection

HOPR uses a random selection algorithm to determine the identities of nodes participating in the network relaying service. The selection is divided into two steps:

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

   - Availability
   - Payment channel graph
   - Stake

   Each node gets a weight that is proportional to the previously mentioned factors. Once an edge is selected, it is added to the current path. All paths are then sorted according to their weight.

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 expanded next using a priority queue mechanism.

Algorithm 1: Path selection in HOPR

$V := Nodes$
$E := \{(x, y) \in V \times V \mid channel(x, y).state = OPEN\}$

$queue \leftarrow new\ PriorityQueue()$
$queue.addAll(\{(x, y) \in E \mid x = self\})$
$deadEnds \leftarrow \emptyset$
$iterations \leftarrow 0$
while $(!queue.isEmpty()) \wedge iterations < maxIterations$ **do**
  $current \leftarrow queue.pop()$
  $currentNode \leftarrow lastNode(current)$
  if $|current| = l$ **then**
    return $current$
  end
  $open \leftarrow \{(x, y) \in E \mid x = currentNode \wedge y \notin deadEnds\ \}$
  $open \leftarrow open.sort(weight)$
  if $open = \emptyset$ **then**
    $deadEnds \leftarrow deadEnds \cup currentNode$
  else
    $toPush \leftarrow \{(current_0, ..., current_{|current|-1}, o) \mid o \in open\})$
    $queue.push(toPush)$
  end
  $iterations \leftarrow iterations + 1$
end
return $\perp$

7.1 Availability

Availability is estimated using the heartbeat protocol. Each node maintains a list of neighbor nodes in the network and either ping or passively listen to them in order to determine whether they are online or offline. A node is considered online if the ping response ("PONG") comes back within a certain timeframe. Otherwise, the node is considered offline and its waiting time for the next PING attempt is doubled.

7.2 Payment channel graph

Every node that intends to send messages needs to have a basic understanding about the topology of the network which means whether the channel is open and funded with enough HOPR tokens for the relaying service. In case no existing payment channel is open, the sender creates a new channel and funds it with enough HOPR tokens

7.3 Stake

The HOPR tokens are used to create payment channels with other nodes in the network and thereby staked in the HOPR network. The node will then use the HOPR token to cover transaction costs when interacting with the blockchain. The more stake a node locks the higher probability that it would be chosen as a relayer.

8 Conclusion

8.1 Future work

8.1.1 Path position leak

In HOPR, payments are performed hop-by-hop along a packet's route. 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.

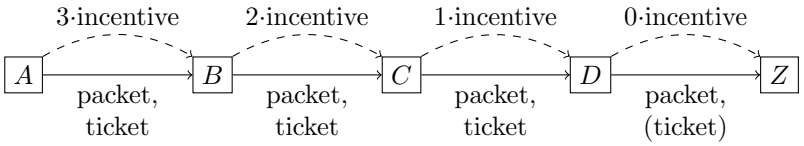

Figure 5: Incentive workflow

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.

8.1.2 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 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, HOPR plans to implement a transitive 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.

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.

References

[1] Jake S Cannell, Justin Sheek, Jay Freeman, Greg Hazel, Jennifer Rodriguez-Mueller, Eric Hou, Brian J Fox, and Steven Waterhouse. Orchid: A decentralized network routing market. 2019.

[2] Sentinel: A blockchain framework for building decentralized vpn applications. 2020.

[3] Mysterium network project. 2017.

[4] Matteo Varvello, Iñigo Querejeta-Azurmendi, Antonio Nappa, Panagiotis Papadopoulos, Goncalo Pestana, and Benjamin Livshits. VPN0: A privacy-preserving decentralized virtual private network. volume abs/1910.00159, 2019.

[5] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In Matt Blaze, editor, *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 303–320. USENIX, 2004.

[6] I2p: The invisible internet project.

[7] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. volume 24, pages 84–88, 1981.

[8] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The loopix anonymity system. volume abs/1703.00536, 2017.

[9] J. Poon and V. Buterin. Plasma: Scalable autonomous smart contracts. 2017.

[10] Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. volume abs/1908.04756, 2019.

[11] J. Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. 2016.

[12] George Danezis and Ian Goldberg. 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, 2009.

[13] Daniel R. L. Brown. Sec 2. standards for efficient cryptography group: Recommended elliptic curve domain parameters. Version 2.0, 2010.

[14] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: simpler, smaller, fast as MD5. volume 2013, page 322, 2013.

[15] Ross J. Anderson and Eli Biham. Two practical and provably secure block ciphers: BEARS and LION. In Dieter Gollmann, 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, 1996.