BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

OPTIMIZING THROUGHPUT OF INCENTIVIZED MIX NETWORKS

OPTIMALIZACE PROPUSTNOSTI INCENTIVIZOVANÝCH MIXOVACÍCH SÍTÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. MARTIN JACKO**
AUTOR PRÁCE

SUPERVISOR **Mgr. KAMIL MALINKA, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

162640

Institut:	Department of Intelligent Systems (DITS)
Student:	**Jacko Martin, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Cybersecurity
Title:	**Optimizing throughput of incentivized mix networks**
Category:	Security
Academic year:	2024/25

Assignment:

1. Get familiar with the current state-of-the-art concepts of mixnets and incentivized mixnets.
2. Research cryptographic proving techniques compatible with blockchain, including ZK-based (Zero Knowledge) and non-ZK based approaches.
3. Compare the existing approach of HOPR incentivized mix net with possible new approaches in terms of transport efficiency (e.g., packet size increase, throughput) and on-chain execution costs.
4. Design and implement selected approaches.
5. Compare them and discuss the benefits and drawbacks of proposed solutions.

Literature:

- Kiel, R. et al. "HOPR - a Decentralized and Metadata-Private Messaging Protocol with Incentives." https://github.com/hoprnet/hoprnet/blob/56d6ca3ffbdeb753c0594f2cd4e7a2808e718939/docs/yellowpaper/yellowpaper.pdf
- G. Danezis and I. Goldberg, "Sphinx: A Compact and Provably Secure Mix Format," *2009 30th IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 2009, pp. 269-282, doi: 10.1109/SP.2009.15. https://cypherpunks.ca/~iang/pubs/Sphinx_Oakland09.pdf
- https://zk-learning.org/

Requirements for the semestral defence:
Items 1 to 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Malinka Kamil, Mgr., Ph.D.**
Consultant:	Pohanka Lukáš, Ing.
Head of Department:	Kočí Radek, Ing., Ph.D.
Beginning of work:	1.11.2024
Submission deadline:	21.5.2025
Approval date:	31.10.2024

Abstract

The topic of this thesis is designing an optimization of the HOPR incentivised mixnet. HOPR network aims to solve the lack of incentive for providing a service in overlay networks by distributing on-chain tokens. This work explores and evaluates the examples of other incentivized mixnets and the specifics of HOPR. The thesis further explores the cryptographic proving techniques, with the focus on state-of-the-art zero-knowledge proofs, and discusses the feasibility and challenges of their integration into the HOPR network. Additionally, the thesis designs an optimization of the HOPR network, which is tested, and its positive impact is measured and discussed along with its advantages and disadvantages.

Abstrakt

Témou tejto diplomovej práce je návrh a optimalizácia incentivizovanej mixovacej siete HOPR. Sieť HOPR sa snaží vyriešiť problém nedostatku motivácie pre poskytovanie služieb v overlay sieťach prostredníctvom distribúcie tokenov cez blockchain. Táto práca skúma a hodnotí príklady iných incentivizovaných mixnetov a špecifiká siete HOPR. Ďalej sa venuje kryptografickým dokazovacím technikám, so zameraním na najmodernejšie dôkazy s nulovým únikom informácií (zero-knowledge proofs), a diskutuje o ich vhodnosti a výzvach pri integrácii do siete HOPR. Okrem toho práca navrhuje optimalizáciu siete HOPR, ktorá je následne testovaná a jej pozitívny dopad je zmeraný a diskutovaný spolu s výhodami a nevýhodami navrhnutého riešenia.

Keywords

mixing network, incentivization, optimization, HOPR, blockchain, zero-knowledge proofs, anonymity, privacy, onion routing

Kľúčové slová

mixovacia sieť, motivácia, optimalizácia, HOPR, blockchain, dôkazy s nulovým únikom informácií, anonymita, súkromie, vrstvené smerovanie

Reference

JACKO, Martin. *Optimizing throughput of incentivized mix networks.* Master's thesis. Supervisor Mgr. Kamil Malinka, Ph.D. Brno: Brno University of Technology, Faculty of Information Technology, 2025.

Rozšírený abstrakt

Digitálna doba zmenila spôsob akým ľudia komunikujú, interagujú medzi sebou a zdieľajú informácie. Tento neobmedzený prístup ku informáciám nás však stojí naše súkromie. Aj napriek tomu, že súkromie je základným ľudským právom, je jeho dosiahnutie na internete čím ďalej tým ťažšie.

Takmer všetka aktivita na internete za sebou zanecháva metadáta, ktorými sú napríklad navštívené ip adresy, časové značky, veľkosti paketov a smerovacie informácie. Obsah samotnej internetovej komunikácie je chránený, no metadáta sú odhalené.

Na prvý pohľad sa voľný prístup k metadátam nemusí javiť ako problém, no sofistikované postupy ako napríklad analýza internetovej prevádzky vedia odhaliť mnoho citlivých informácií, ktorými sú napríklad komunikačné vzorce, navštívené služby a stránky, kto s kým komunikuje, ako často a kedy. Jednotlivci môžu byť sledovaní, profilovaní a identifikovaní na základe ich sieťových metadát.

Samozrejme existujú postupy a technológie, ktoré boli navrhnuté za účelom ochrany metadát užívateľov. Najpoužívanejšími takými technológiami sú virtuálne privátne siete (VPN) a overlay siete, no obe však majú svoje nedostatky. VPN typicky poskytujú vysokú kvalitu služby, no metadáta užívateľov zostávajú viditeľné pre prevádzkovateľa služby. Overlay siete naopak poskytujú plnú ochranu metadát, no kvalita služby často kolíše, nakoľko jej prevádzka je často na báze dobrovoľnosti jednotlivcov. Tento problém sa snaží vyriešiť veľa projektov, medzi ktorými je aj sieť HOPR.

Cieľom tejto práce je optimalizácia siete HOPR, za využitia zero-knowledge dôkazov. Sieť HOPR je incentivizovaný mixnet, ktorý sa od väčšiny overlay sietí líši tým, že obsahuje mechanizmus pre odmeňovanie svojich účastníkov. Sieť HOPR pozostáva z mnohých súčastí, na ktoré by sa mohla upriamiť naša pozornosť, no základe dohody s konzultantmi z HOPR sa budeme zameriavať na incentivizačnú vrstvu. Táto vrstva je založená na systéme tiketov, ktoré účastníci dostávajú za preposielanie paketov v sieti. Tieto tickety si následne môžu uplatniť pomocou smart kontraktu, čím dostanú tokeny.

Nami navrhnutá optimalizácia musí zachovať všetky doterajšie bezpečnostné vlasnosti siete HOPR, ktorými sú neprepojiteľnosť odosielateľa a príjemcu, anonymita odosieľateľa a anonymita príjemcu. Zároveň musia byť zachované vlastnosti incentivizačného mechanizmu: vynútenie platby za preposlanie paketu, možnosť overenia existencie zdrojov pre platbu a obdržanie platby až po preposlaní paketu.

Na základe analýzy siete HOPR a získaných znalostí o zero-knowledge dôkazoch sme diskutovali ich integráciu do navrhovanej optimalizácie a aj to akými vlastnosťami by optimalizácia mala disponovať. Boli identifikované dve cesty, ktorými by sa táto integrácia mohla uberať.

Prvou z nich bola integrácia jedného zero-knowledge dôkazu do jedného paketu. Toto sa však ukázalo ako nevyhovujúce, kvôli príliš vysokému nárastu veľkosti paketu spojenému s príliš vysokou náročnosťou výpočtov spojených s obsluhou dôkazu (predpríprava, vytvorenie, verifikácia).

Druhou z identifikovaných ciest bolo rozdelenie jedného zero-knowledge dôkazu medzi viaceré pakety. Tento spôsob ich použitia nevyhnuteľne vedie ku logickej spojitosti medzi určitou skupinou paketov, obsahujúcich rovnaký zero-knowledge dôkaz. Toto spojenie spôsobuje, že uzly vedia identifikovať pakety z rovnakého zdroja a tým pádom majú prístup ku časovým metadátam, čo je nežiaduce. Z tohto dôvodu sa táto cesta, a celkové využitie zero-knowledge dôkazov, ukázalo ako nevhodné.

Na základe predošlých zistení a rozboru siete HOPR bola úspešne navrhnutá optimalizácia procesu agregácie tiketov. Optimalizácia tohto procesu spočíva v zefektívnení a zjednodušení procesu uplatnenia agregovaného tiketu, ktorý prebieha v smart kontrakte.

Navrhnutá optimalizácia bola implementovaná a úspešne otestovaná s prihliadnutím na funkčnosť a výkon. Nebolo zistená absencia niektorej z požadovaných vlastností siete HOPR a jej incentivizačnej vrstvy. Z hľadiska výkonnosti bola zistená úspora efektívnych nákladov na vykonanie v reťazci oproti predošlej verzii o viac ako 50%.

Ako hlavný prínos dosiahnutých výsledkov tejto práce vidím reálnu využiteľnosť navrhnutej optimalizácie, čiže možnosť jej integrácie do siete HOPR bez veľkých úprav existujúcej funkcionality.

Optimizing throughput of incentivized mix networks

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Mgr. Kamil Malinka, Ph.D and Ing. Lukáš Pohanka. I have listed all the literary sources, publications and other sources that were used during the preparation of this thesis. This assignment of this thesis was a joint effort between the Brno University of Technology, Faculty of Information Technology, and the HOPR Association. AI tools were used during the preparation of this thesis, namely Grammarly and Writeful for text styling enhancements and ChatGPT for brainstorming and text rephrasing.

.....
Martin Jacko
May 19, 2025

Acknowledgements

I would like to thank my supervisor Mgr. Kamil Malinka, Ph.D, and my consultant from HOPR Ing. Lukáš Pohanka for their help, constructive feedback, and encouragement.

Contents

List of Figures

Chapter 1

Introduction

The digital age has transformed the way people communicate, interact, and share information. The internet provides users with global connectivity, facilitating everything from social interactions and business transactions to educational resources and political activism. However, this unprecedented access to information has come at a cost, which is our privacy.

Even though the right to privacy is considered a human right, reliably achieving it on the internet has become increasingly difficult. Most online communications, whether they occur over social media, email, or other forms of digital communication, involve the exchange of metadata. This metadata—data about data—is non-content information that could be used to reveal critical user behavior patterns.

While encryption can protect the content of communication, metadata remains largely exposed. Network metadata includes a range of data points that provide insight into communication patterns without revealing the actual content of the messages. Common metadata elements include IP addresses, timestamps, message size, and routing information. Although this metadata may seem harmless, sophisticated network analysis tools can infer substantial information from it. For instance, by correlating timestamps and message sizes, an observer can infer a user's communication habits or relationships. IP addresses can be used to pinpoint geographic locations or associate particular activities with specific users.

One of the most concerning developments is traffic analysis, a method used by adversaries to study metadata and discover communication patterns, even when the content is encrypted. By monitoring packet flows and comparing traffic at different points in a network, adversaries can deduce who is communicating with whom, how frequently, and when. The individuals can be tracked, profiled, and identified even if the adversaries cannot read the actual messages.

This ability to extract detailed information from metadata introduces significant risks to users' privacy and anonymity. Surveillance agencies, internet service providers, and malicious actors could exploit metadata to monitor user activities, raising concerns about the erosion of internet privacy. Thus, achieving internet privacy is not simply about keeping message content secure but also includes protecting user identity from exposure via metadata.

There are, of course, tools and technologies that were designed with protecting users' metadata in mind. Some of the most popular among these are onion-routed overlay networks and virtual private networks. Each of these technologies takes a different approach and therefore has different tradeoffs. Their end goal is to protect user metadata and secure communication channels, offering varying levels of protection against metadata analysis.

First of such technologies are virtual private networks or VPNs, which provide a straightforward solution to hiding user metadata by encrypting traffic and routing it through a secure server operated by the VPN provider. This masks the user's IP address and creates a private tunnel for internet traffic. However, VPNs typically only conceal metadata from external observers, while the VPN provider itself has full access to user metadata and traffic, making the user's trust in the provider critical. VPNs generally offer great quality of service at the cost of trusting the VPN provider.

The other such technology are overlay networks, which promise truly secure and private access to the internet by obfuscating the traffic via a network of nodes. They generally provide a high level of security and anonymity, but often suffer from downsides, such as unstable performance and service quality, as they are mostly provided on a voluntary basis.

There are many projects that are trying to address these downsides, with one of them being the HOPR network. HOPR network's goal is to incentivize network participation with on-chain currency. Such an approach inevitably faces many challenges, one of which is performance.

In this thesis, we will cooperate with the HOPR team directly, and our goal will be to design and implement a new approach that will lead to the optimization of the HOPR network. The designed approach should either increase performance by decreasing the computational overhead or lowering the space overhead (packet size), or decrease the on-chain execution costs. We will also attempt to utilize the latest zero-knowledge proofs in the designed optimization.

The designed optimization will be considered a success if it provides a measurable performance improvement over the current version of the HOPR network, with little to no introduced tradeoffs.

This thesis consists of several chapters, of which the following chapter, Mixed Networks, introduces the fundamentals of mixnets and outlines the key challenges they face, with a particular focus on incentivization. Alongside HOPR, we also explore the Nym mixnet to gain insight into other possible approaches. The subsequent chapter examines cryptographic proving techniques that are compatible with blockchain, with an emphasis on zero-knowledge proofs. The Analysis chapter then outlines the specific constraints of the optimization and evaluates the feasibility of integrating zero-knowledge proofs. The final chapters present the design, implementation, and testing of the proposed optimization, followed by an evaluation of the results and conclusion.

Chapter 2

Mixed networks

Firstly, we need to look at how established, state-of-the-art mixnets of today operate and to understand fundamental concepts such as onion routing that underpin their functionality.

2.1 Onion Routing

Onion routing is a routing scheme designed to enhance privacy and security by ensuring that no single intermediary knows both the origin and the destination of the data being transmitted. Developed to provide secure, private communication across a network, onion routing enables users to send messages without revealing their identity or the content of their communication to any single intermediary or eavesdropper.

The core concept of onion routing involves encrypting the message in layers using public keys. Each layer of encryption is peeled away by successive intermediary nodes in the network, with each node only knowing the identity of the previous and next hop. This ensures that no node in the route knows the full path of the message or its content.

The sender first selects a sequence of nodes, known as onion routers, and encrypts the message multiple times, once for each node along the path. When the message reaches the first node, it removes the outermost encryption layer and forwards the partially decrypted message to the next node, which removes its own layer of encryption, and so on, until the message reaches its final destination. The primary downside of onion routing is increased latency due to encryption and decryption operations on messages [23].

2.2 Overview

Mixnet, or mixing network, is an autonomous communication network whose goal is to provide a means of anonymous and private communication through the modern internet. It builds upon the concept of Onion routing and further enhances it, introducing some tradeoffs [34].

When used for communication, mixed networks provide sender and receiver anonymity, achieved by mixing messages as they go through the network. The messages between the sender and receiver are relayed using a series of servers called mixing nodes or relay nodes, as can be seen Figure 2.1. Mixing networks use previously introduced layered encryption and hop-by-hop routing known from onion-routed networks. The messages in mixed networks are source-routed, which means that the message originator chooses the path that the message will follow through the network, and he is the only one to know this path.

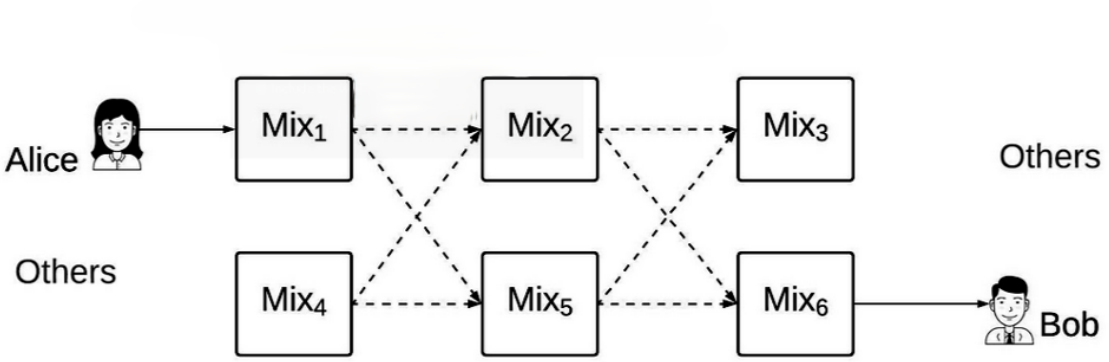

Figure 2.1: Basic mixnet [27].

Where the mixnets differ from simple onion-routed networks is mostly in the way messages are relayed. As messages follow a pre-determined path, they are susceptible to timing-based traffic analysis. In onion-routed networks, messages are decrypted and sent to the next server immediately, while in mixed networks, order shuffling and time delays are introduced. At each relay node, the messages are buffered and sent out in a different order. This introduces random delays between messages in respective sender-to-receiver streams, which makes them less exposed to timing-based traffic analysis.

There are conditions that need to be met in order for mixing to achieve the desired outcome. The first of them is a large enough volume of messages flowing through the network, so that individual streams of messages are not identifiable.

Due to the possibility of a lack of number of messages flowing through a network at a given moment, mix networks often introduce cover traffic. Cover traffic consists of dummy packets that are sent through the network alongside legitimate user traffic in order to preserve the security properties of the network, even at times with lower network load.

Mixing networks also take into account a scenario where some of the relay nodes are malicious. In this case, privacy-preserving properties are retained as long as at least one relay node along the route is honest [11, 26].

2.3 Incentivization

Incentivization mechanisms in the context of mixing networks aim to address two main challenges:

- ensuring sufficient participation,
- deterring malicious actors while simultaneously encouraging honest behavior.

The first challenge is related to participation in the mixing network, where one of the common problems of these networks is their availability and reliability. This is primarily due to the fact that participating nodes are often operated on a voluntary basis, where volunteers donate their resources. Entities that run and maintain relay nodes are not incentivized financially, and they are motivated mainly by a sense of community and belief.

However, the amount of donated resources is quickly outgrown by the increasing demands of mixed networks. One solution for this phenomenon could be some form of incentivization of network entities that donate said resources to networks. This kind of reimbursement for node operators should lead to an increase in the overall number of nodes and, therefore, increased network availability and reliability.

The other challenge that incentivization solves is the deterrence of malicious bad actors and the encouragement of honest behavior. With a stake-based incentivization mechanism present in the network, the nodes that provide reliable service are rewarded more, while misbehaving nodes are penalized, and can be identified and flagged, enhancing the network's robustness.

Although the incentivization should theoretically be a benefit to a mixed network, it must be introduced thoughtfully, with original network aspects in mind. The introduced incentivization mechanism can not function at the expense of network security or privacy, and it must also balance those aspects with performance [28, 3].

2.4 Nym mix network

To gain insight into other possible approaches to incentivized mixing networks, we will study the Nym network [15]. Nym is an overlay network that supports private access features for its users and services. It consists of a decentralized mixnet, an anonymous credentials cryptosystem, and the NYM blockchain. It supports incentivization through its "Proof of mixing" scheme.

2.4.1 Network design

Nym network uses a token-based incentivization principle in combination with "right-to-use" credentials to access the network. The schematic of the Nym network can be seen in Figure 2.2. There are several types of entities that participate in the network, which include:

- three types of nodes: validators, gateways, and mixing nodes,
- service providers,
- end-users.

End users

The users of the Nym network are individuals who choose to communicate through its privacy-preserving infrastructure, as well as those who make use of various services built on top of the network.

Gateways

They serve as an access point to the network, their main purpose is to check the "right-to-use" of individual users and, therefore, to protect the network from use without payment. Individual users can choose which gateway they want to use, and they can even split their traffic among multiple gateways. Gateways also cache messages for users who are offline for up to 24 hours.

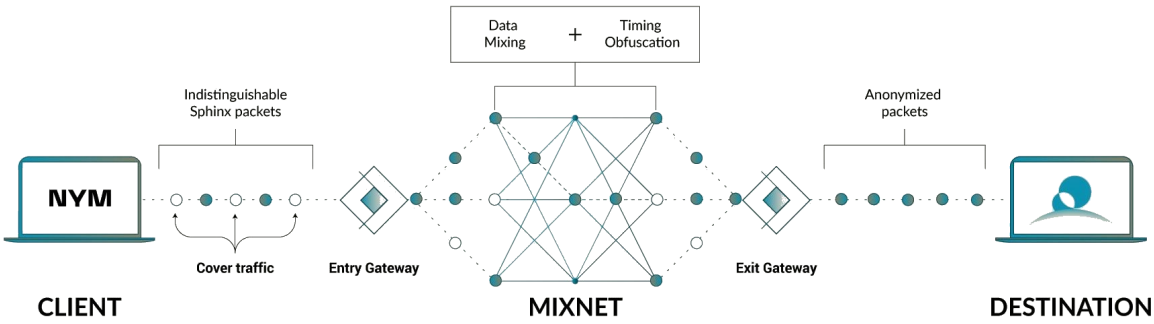

Figure 2.2: Nym network schematic [33].

Gateways can also implement various censorship-resistant measures. As the IP addresses of the users are not hidden from the Gateway, and therefore from adversaries, the users can be identified to be using the Nym network, which could cause problems in some regions. In this case, Gateways are able to disguise the traffic from the user as another type of traffic. They are identified by IP address and public key.

Mixing nodes

Network nodes that provide communication privacy, relay data packets by transforming them and mixing them among each other.

They are arranged into layers, which can be scaled up to increase privacy and scaled to the side to handle more traffic. This means that the first layer nodes only receive traffic from Gateways, the second layer nodes from the first layer nodes, and so on.

The routing algorithm selects a path for the packet according to the current load of individual nodes.

Periodically, the pool of nodes for each layer is selected. The probability of a node being selected is proportional to its stake, which serves as an indicator of its reputation.

The layer to which the node is assigned is randomized. The process of assignment is decentralized, publicly verifiable, and carried out by the Validator nodes in combination with the output of the Verifiable Random Function executed by each candidate mixing node. This also helps network security, as adversaries cannot predetermine which layer the node will belong to, thus making control of a specific subset of nodes difficult.

Validators

They perform two core functions required for the Nym network to function: the maintenance of the NYM blockchain and the issuance of “right-to-use” credentials.

The maintenance of the NYM blockchain consists of the storage of network-wide configuration information: the list of active nodes and their public keys, stake and delegated stake of participants, state of reward pool and reward distribution data, and others.

Their second function is to issue “right-to-use” credentials to network participants. There are two types of these credentials: Bandwidth credentials and Service credentials.

Bandwidth credentials encode a proof of payment into the reward pool in exchange for data allowance to be sent through the network. The individual users present these credentials to gateways in order to gain access to the network.

Service credentials provide proof of "right-to-use" of the specific service accessible via a Nym network, including proof of payment for the service. They also encode arbitrary data required by the service.

Service providers

Third-party service providers utilize the Nym network as an access medium for their service. Such services can also utilize the Nym network's credentials infrastructure to obtain payments.

2.4.2 Privacy enhancing features and technologies

Sphinx packet format

Nym network, as well as the HOPR network, utilizes the Sphinx packet format [14]. Sphinx is an efficient packet format for hop-by-hop source-routed networks. It adds very little overhead to the payload, only a few hundred bytes.

Sphinx allows for encrypting the message in such a way that provides per-hop unlinkability by cryptographic blinding, tamper resistance by using MAC, and resistance to size-based attacks by always being padded to the same length.

As it supports source-based routing, the packet creator performs the offline key derivation with all of the nodes along the path. After obtaining the shared key, the encryption is performed, starting with the innermost layer.

Single-use reply blocks

SURBs are an optional part of a Sphinx header. They offer a crucial functionality of any network that uses Sphinx format: an option to reply.

SUBR is a precomputed Sphinx header, which is indistinguishable from the standard Sphinx header. It also contains a key that the original message recipient can use to encrypt the reply message. The validity period is limited as the public keys and network topology change over time.

SURBs also solve the issue of reliable transport in these types of networks, where the user is notified via the acknowledgment that his message was delivered successfully, and said acknowledgment was sent using the SURB. Individuals can also publish a means to be contacted in the form of a SURB, without revealing their true identity or location.

Mixing strategy

There are multiple approaches to mixing messages in the network. So-called mixing strategies have an impact on the network's latency, effective use of bandwidth, and, most importantly, its security.

Early mixing strategies used message shuffling and batching, where the node waits for a certain number of messages, reorders them, and then sends them out in a burst. This approach came with many disadvantages, namely, little to no control over message latency and poor use of available bandwidth due to its bursty nature. This strategy was also later found out to provide insufficient anonymity properties [16].

The Nym network uses a mixing strategy called continuous mixing, where messages are sent out independently after each is held for a pre-determined amount of time. Independently delaying each incoming message results in an unlinkable sequence of output messages. This approach also gives users control over tradeoffs between end-to-end latency and a higher degree of privacy, as they are the ones who determine the delay time of the message at each node.

Cover traffic

During times when there is low traffic in the network or when adversaries observe the network for extended periods of time, a leakage of private information is still possible. The best example of this is the time pattern of users accessing the network. The adversaries should not know when or how much traffic is being sent by users.

The solution to this issue is to add artificial traffic to the network to hide the possible observable patterns. The principle of so-called cover traffic utilized by the Nym network is to create loops of "dummy" traffic in the network. This "dummy" traffic is indistinguishable from normal traffic in the network. It can be initiated either by mixing nodes, to ensure that there is a sufficient number of messages in the network at any given time, or by the users themselves, in order to hide their actual communication patterns.

Cover traffic can also have benefits regarding network properties, for example, nodes can utilize cover traffic loops to maintain knowledge of the actual network state, detect when the nodes are offline or performing poorly.

2.4.3 Incentivization mechanism

This section will take a look at the way incentivization works in the Nym network.

Stake and reputation

Although the Nym network utilizes staking of the NYM token, the individual nodes are rewarded proportionately to their work rather than their stake. In the Nym network, the node's stake represents its reputation.

For a node to participate in the network, it must stake a certain amount of NYM tokens in order to be selected to participate in a given network epoch. The probability of being selected is proportional to the amount of the node's staked NYM token.

This staking mechanism is utilized in order to prevent Sybil attacks and to motivate nodes to behave according to the network rules. In case of misbehavior, the node's stake can be slashed.

This system also allows for the delegation of stake, where other network entities can delegate their stake to a certain node, therefore increasing its chances of being selected to participate in the network. The entities are entitled to a share of the node's reward, proportional to their delegated stake. This also motivates entities to delegate stake only to reliable nodes.

Proof of mixing

The presence of incentivization in the network creates a requirement for a mechanism that will be able to determine the quality of service of individual nodes in order to fairly compensate them.

The Nym network utilizes the Proof-of-mixing scheme, which uses a measurement message sample system to provide a verifiable and decentralized way to measure latency.

Generation of measurement messages

The first step of the scheme is to generate messages used to measure the quality of service. These messages are generated by both mixing nodes and end users in a verifiable manner. The path through the network, delay at each hop, time of dispatch, and the rest of the routing information are computed using a verifiable random function using a pseudorandom seed provided by the Validator nodes. This ensures that the messages are not crafted maliciously to bias the measurement. The only variable that remains in the hands of nodes or users is the choice of the Gateway. Measurement messages always form a loop and are indistinguishable from regular network traffic.

Routing of measurement messages

The measurement messages are pre-generated locally before the start of the network epoch. Once the epoch starts, the measurement messages are sent out at a designated time.

As all types of messages travel through the network, mixing nodes remember which messages were already seen by keeping records of Sphinx message tags. These tags are recorded to prevent replay attacks and to evaluate the quality of service of a node. In addition to keeping these tags, the nodes also construct a Merkle tree of seen message tags together with their timestamps.

After the epoch ends, the mixing nodes broadcast the root of this Merkle tree to validator nodes, which then commit it into the NYM blockchain. The commitment of the Merkle tree root is necessary to obtain the reward for mixing.

Once all of the roots are committed, validator nodes publish a pseudorandom seed, which was used during the creation of measurement messages. This enables mixing nodes to recreate the measurement messages and to select those that have the node's public address in their path. Mixing nodes then check whether the tags of such messages are a part of the node's Merkle tree root. When the match is found, the node opens a corresponding leaf, revealing the message tag and timestamp, which serve as proof that the node has seen the message.

Evaluation of quality of service

Using the data from the previous step, all network participants can see which messages were sent, received, reported lost, or incorrectly delayed. This data is then used to determine the quality of service of the individual mixing nodes. The algorithm must take into account data from many measurement message paths to accurately measure the quality of service.

For example, this system does not differentiate between the case when the message tag is added to the Merkle tree and then dropped by the same node, and the case when the message tag is added to the Merkle tree, successfully sent out, and then lost by the following node. In order to mitigate this, the evaluation algorithm considers data from many measurement message paths to pinpoint misbehaving nodes.

Distribution of rewards

At the end of a network epoch, there is a certain amount of NYM tokens collected in the reward pool. These rewards belong not only to the mixing nodes but also to stakeholders.

The objective of reward distribution is to encourage mixing nodes that behave according to the protocol and those who stake NYM tokens on behalf of such nodes.

The mixing nodes get a share from the reward pool that is proportional to their declared bandwidth and their quality of service. For new nodes, a grace period is given where the quality of service is not taken into account. The nodes with large downtime or some forms of misbehavior lose their stake.

The reward for the delegated stake is distributed according to the overall performance of delegated nodes, as well as the amount of delegated stake. There is a soft cap built into this reward system in order to prevent centralization of stake on a handful of nodes.

Observations

The Nym is an evolved, feature-rich, incentivized mixing network. From the performance point of view, the network is very reliant on the NYM blockchain. Many operations necessary to use the network, besides the minimum of acquiring a NYM token, such as credential issuance, are executed with the participation of the NYM blockchain.

The network has many self-governance operations, such as mixing node selection, measuring the quality of service, reward distribution, and others, which are also dependent on blockchain, and many of them require write operations. The magnitude of this overhead results in the need for Validator nodes that do not handle any traffic, which could be seen as a waste of resources.

Some of this overhead is necessary due to features that are not found in other mixnets, such as direct integration of services, buffering of messages at gateways, and the presence of arbitrary credentials.

The incentivization mechanism takes a more probabilistic approach, as not every message is checked for delivery. The majority of its overhead resides in operations between epochs, such as staking, reward distribution, Merkle tree root broadcasts, searching for measurement messages in the tree, and proving their presence.

2.5 HOPR

In this section, we will look at the basics of the HOPR mixnet. More focus is given to parts deemed relevant for this thesis, e.g., throughput optimization. The official HOPR yellowpaper [40] is missing descriptions on some of the parts of the version (v2.2.0) of the HOPR network, which was used as a reference for this thesis. The missing information was obtained via the codebase[1] and knowledge sharing sessions with the HOPR team.

HOPR is a decentralized incentivized mixnet that offers its users transport layer data security. It uses existing standards, such as Sphinx packet format and packet mixing, and adds an incentivization layer to aid network reliability and growth. HOPR's blockchain of choice is Ethereum, which is mainly used to perform incentivization operations, such as managing payment channels. It also utilizes cover traffic.

The HOPR network was developed with privacy by design in mind, and its security goals are sender and recipient anonymity and sender-recipient unlinkability.

[1] https://github.com/hoprnet/hoprnet

2.5.1 Network design

HOPR network consists of 3 entities: blockchain, relay nodes, and users. The network uses a homogeneous node model, where all the nodes serve the same purpose.

Packets in the network follow a free-route sender-selected path, with various customizable routing strategies possible.

2.5.2 Ticket payment scheme

Incentives in the HOPR network are carried out by a per-packet micropayment system of „tickets“, pictured in Figure 2.3, which utilizes payment channels and probabilistic payments. This system allows nodes to initiate asset transfers without requiring on-chain interactions for every packet. Tickets are initially sent in a locked state and are unlocked once proof is provided that a packet has been successfully relayed.

Nodes receiving a locked ticket can validate its authenticity. After obtaining the necessary cryptographic material to unlock the ticket, the node can claim the incentive by submitting the ticket to the smart contract.

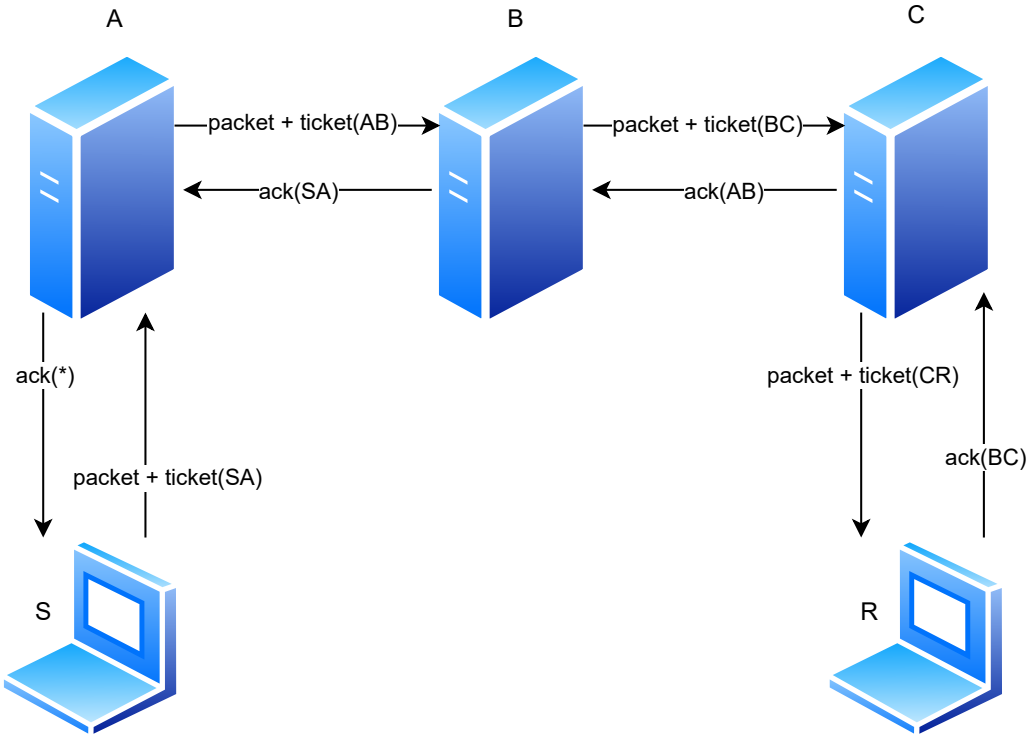

Figure 2.3: Ticket flow in HOPR network between sender S, relay nodes A, B, C, and receiver R.

Ticket issuance

Before a node can send and receive packets in the network, it needs to commit funds on-chain. This requirement applies to both relay nodes and users. This process is executed by a smart contract and creates multiple payment channels, one towards everyone with whom the node intends to communicate. This serves as proof of eligibility to issue tickets.

Tickets are bundled and sent together with a packet and include an incentive for processing and sending the packet to the next node. As the ticket issuance is not an on-chain operation, the ticket recipient must check whether sufficient funds are locked on-chain, including the funds for previously received tickets. If this check is unsuccessful, the packet should be discarded.

The ticket structure consists of the following attributes:

Recipient The Ethereum address of a recipient is derived from their public key. This binds a ticket to a concrete issuer-recipient pair. The address is computed as the last 20 bytes of the `keccak256` hash of the uncompressed ECDSA public key.

Challenge Tickets are issued locked, but the ticket's validity can be verified. The challenge is a piece of a cryptographic puzzle that needs to be solved in order to claim the incentive.

Value States the amount of claimable funds for a given ticket. The minimum ticket value is set by the network, and tickets with lower values shall not be accepted.

Winning probability Not every ticket in a HOPR network is a winning one. The percentage of tickets that lead to a payout is denoted by the winning probability. The ticket itself carries an inverse of this value to account for rounding errors. The value is normalized with the Ethereum base to get the original value.

$$ticketData.invWinProb := winProb * (2^{256} - 1).$$

This phenomenon is introduced to lower the amount of on-chain operations required to obtain the incentive.

Ticket index Each ticket is given an incremental index, while the index of the last redeemed ticket is stored in the smart contract to ensure that each ticket is redeemed only once. Once a new ticket is redeemed, the value in the smart contract will be updated. As ticket issuance is not an on-chain operation, the ticket recipient must check incoming ticket indices to ensure their validity. Additionally, he must redeem tickets in order.

Channel epoch Payment channels undergo multiple open and closed state changes during their existence. Each cycle increases the channel epoch counter. The current channel epoch is stored smart contract. In order to differentiate to which epoch does ticket belongs, the structure carries this information. Once the channel is reopened, all the tickets from previous epochs are invalidated.

Signature Each ticket is signed by the issuer. The signature consists of a `keccak256` hash of ticket attributes. Both the ticket recipient and the smart contract check this signature.

Ticket validation

Tickets are used to prove to their recipient that they will receive an incentive for processing an incoming packet.

Initially, a ticket is received in a locked state, meaning that the ticket recipient does not know a response to the challenge in a given ticket, which is used to unlock it. The response that unlocks a ticket is provided to the recipient only after processing the packet. Additional checks must take place, such as checking the payment channel for sufficient funds and checking the ticket index, epoch, value, winning probability, and signature.

Whether a ticket is a winner can only be determined after obtaining the response. This is determined by computing the ticket luck as follows:

$$keccak256(keccak256(ticketData)||V||response||ticket.signature) < ticket.winProb$$

where V is an output of VRF computed by the ticket recipient. The value V is present in order to prevent the ticket issuer from tampering with the ticket's luck, as it is dependent on this value, which can only be calculated by the ticket recipient. The VRF is used to prevent the ticket recipient from trying to brute force the ticket's luck value.

If the ticket's luck is arithmetically lower than the ticket's winning probability, the ticket is considered a winner and stored for later redemption. If the ticket is a losing one, it will be discarded. The losing tickets are an important part of making the payment scheme efficient, as given that the ticket amount is set according to the network, the asymptotic payout stays the same.

Ticket redemption

After the checks from the previous section are passed, the node may proceed with ticket redemption. The node first creates an ordered set of winning tickets, as they must be redeemed in order.

The redemption is executed by the smart contract, which acts as a trusted third party. The node needs to prove to the smart contract that each ticket is a winning one and that it knows a response that solves the ticket challenge.

Challenge Solving a ticket challenge C means finding a value $r \in \mathbb{F}$ such that $r \cdot G = C$ is an elliptic curve point. To verify this, the smart contract must perform scalar multiplication of an elliptic curve point, which is not directly supported by Ethereum at the time. However, Ethereum does provide an efficient implementation of the following function:

$$x \in \mathbb{F} \rightarrow \text{ethAddr}(x \cdot G)$$

where $\text{ethAddr} : \{0,1\}^{64} \rightarrow \{0,1\}^{20}$ maps uncompressed elliptic curve points to Ethereum addresses. Smart contract then uses this function to verify the provided response against the challenge from the ticket.

Signature and payment channel validation Once the challenge is confirmed, the smart contract will verify the ticket signature and recover the public key of its issuer. The smart contract then checks the channel ID of the ticket. If such a channel in the blockchain exists, it checks if the public key of the ticket issuer and the recipient match with the public keys in the payment channel. Lastly, the smart contract checks whether the channel is open and whether it contains sufficient funds. If all of the checks above succeed, the ticket is accepted.

Relay protection Despite the ticket having a valid signature and correct response to a challenge, it must be valid exactly once, as it initiates an asset transfer. This is guarded by the ticket index and the ticket epoch.

Each ticket is given an incrementing index and when being redeemed, the ticket index must be higher than index of the latest successfully redeemed ticket, otherwise it is rejected.

In the same fashion, the ticket carries an epoch, which refers to a channel epoch that is stored in the smart contract. After each open and close cycle, the epoch increments. Upon redemption, the smart contract will only accept the tickets with an epoch that matches the current channel epoch.

Ticket luck The last check performed by the smart contract is whether a ticket is a winner. As mentioned previously, ticket luck is calculated using the ticket hash, response, signature, and the VRF output V. The smart contract checks whether the provided value V was computed correctly and then computes whether the ticket is a winner.

Asset transfer Once all of the previously mentioned checks pass, the smart contract changes the state of the payment channel accordingly, e.g., credits the recipient side with the amount of funds stated in the ticket.

2.5.3 Incentivization principle

The nodes in the HOPR network receive incentives for transforming and delivering packets in the network. In order to make these steps verifiable, the HOPR network uses a custom trustless Proof-of-Relay scheme. This scheme ensures that the incentive is paid out only after the node transforms and sends out the packet according to the network protocol.

Proof of Relay

By using the Sphinx packet format, it is guaranteed that the packet can only be processed in the order chosen by its creator. Therefore, when a downstream node receives a packet and attempts to decode it, it can end up with either a random bitstring or a valid, readable packet. If the latter is the case, the downstream node can be certain that the previous upstream node performed the packet transformation correctly. Upon this verification, an acknowledgment, which is required to unlock an incentive, is sent to the upstream node.

The relation between the tickets and the acknowledgments can be seen in Figure 2.3, where, for example, node B receives the acknowledgment from node C, and this acknowledgment unlocks the ticket issued by node A.

Construction Each ticket includes a Challenge, which then requires a Response in order to claim the incentive. Each relayer is engaging in the proof-of-relay scheme with the next downstream node, using the challenge selected by the packet creator. In order to prove that the packet creator knows the valid response, a node is given a hint. This hint can be used to verify knowledge of the response computationally.

Secret sharing The proof-of-relay is a 2-out-of-2 secret sharing scheme where each node derives two keys, s_i^{own} and s_i^{ack}, using s_i from the Sphinx packet header. These two keys act as shares in a sharing scheme between node n_i and the next downstream node n_{i+1} along the path. s_i^{own} can be derived immediately after receiving a packet, obtaining s_{i+1}^{ack} is

possible only after correctly transforming and sending the packet to the next downstream node n_{i+1}. Once the node n_{i+1} verifies the correctness of the packet, it sends s_{i+1}^{ack} to the node n_i.

Once the node n_i obtains both key shares, s_i^{own} and s_{i+1}^{ack}, it can compute s_i^{response} and redeem the ticket on-chain.

Challenge, Response, and Hint The relationship between challenge C_i and response s_i^{response} is as follows:

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

where G refers to the base point, and $\cdot$ means scalar multiplication on the curve. Therefore, to solve the challenge, both s_i^{own} and s_{i+1}^{ack} are required.

The s_i^{own} is derivable upon packet reception, but, at that time it cannot be decided whether s_{i+1}^{ack}, will lead to a $s_{\text{i}}^{response}$ that solves the challenge. Since the distributivity is preserved:

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

By knowing $\text{hint}_i = s_{i+1}^{\text{ack}} \cdot G$, the node can verify that $s_i^{\text{own}} \cdot G + \text{hint}_i = C_i$, and thereby check that the creator of the packet must have known a value s_{i+1}^{ack}. Due to the computational complexity of inverting scalar multiplication, including hint_i in the packet header does not aid in finding s_{i+1}^{ack}, its only contribution is making the embedded challenge verifiable.

By default, the nodes in the HOPR network should not accept a packet without a ticket. Node n_i not only needs to transform the packet but must also issue a ticket for the next downstream node, n_{i+1}. To do this, it must know the challenge C_{i+1} to include in the ticket for node n_{i+1}, which is embedded in the portion of the Sphinx packet that is accessible to n_{i+1} by the packet creator.

By following this chaining mechanism, nodes are compelled to issue a ticket to the next downstream node because they cannot claim their own incentive without the cooperation of the downstream node.

The final node in the message's path is not strictly forced to send acknowledgments, as there are no direct consequences for doing so. In the current version, the protocol does not prevent such behavior.

Incentive payout

The incentive is transferred to the recipient on demand after obtaining all parts of the Proof-of-Relay scheme through an unidirectional payment channel and a smart contract.

Payment channel The channels utilized in the HOPR network are unidirectional and offer an opportunity to lock the funds on-chain and withdraw them after fulfilling certain conditions. The change of the payment channel state, and therefore the movement of funds, is carried out by the smart contract after a successful ticket redemption.

The paying node can initiate the payment closure to retrieve the leftover funds. After the closure initiation, the channel's state is changed from `Open` to `PendingToClose`. The channel will stay in `PendingToClose` state for a given grace period. The node on the receiving end of the channel can see the state change and will act accordingly. Specifically, it will redeem all the outstanding winning tickets tied to the channel in question. After the grace period expires, the channel's state is changed to `Closed`.

Ticket aggregation As the ticket redemption is an on-chain operation, it requires additional funds to be executed. The number of tickets that are being redeemed is slightly lowered by the winning probability mechanism.

In addition to this, there is an option to ask the node that issued the tickets to aggregate all of the outstanding winning tickets into one, in order to lower the on-chain ticket redemption costs. In the form that aggregation exists in HOPR now, the ticket issuer node can refuse this, as the protocol does not enforce ticket aggregation.

If the issuer node agrees to perform the ticket aggregation, it checks whether all of the tickets that the receiving node wishes to aggregate are winning tickets. This check consists of performing checks that the ticket recipient performs upon the initial ticket reception mentioned in 2.5.2 and 2.5.2, with the benefit of being provided all the necessary values. If that is the case, the issuer node then issues an aggregated ticket, which has a winning probability of 1, and its value is equal to the sum of the outstanding tickets.

The ticket aggregation exists as an aid to the time when the HOPR network runs in an undesirable state. The ideal state of the HOPR network is a high number of packets in the network, therefore many issued tickets, with low winning probability and high price per ticket, which equals less to no need for aggregation as the tickets are „aggregated by default“.

The minimal ticket winning probability and minimal price per ticket are determined by the network and increase and decrease respectively, with the decrease in the amount of packets in the network, to assure the steady reward distribution for packet relay.

On the other hand, the undesirable state occurs when there is a low number of packets in the network, therefore low number of issued tickets, with high winning probability and low price per ticket, which result in expensive ticket redemption with respect to its price and therefore the aggregation is needed.

Chapter 3

Cryptographic proving techniques compatible with blockchain

Cryptographic proofs in the context of blockchain are used to authentically, undisputably prove something. Their specific properties, advantages, and disadvantages tend to be considered and leveraged upon usage.

The subject of the proof, e.g., what is being proven, can vary by occasion. It can have a form of data where we prove that it is genuine, and it has specific properties or adheres to certain constraints. There can also be a need to prove that a computation has been performed correctly or at all.

Various types of proofs also differ by the amount of information that needs to be revealed for the proof to take place. This amount varies from revealing all the data to not revealing any useful information.

The main purpose of this chapter is to explore proving techniques and other technologies related to proving and to assess what they offer, their strengths, and weaknesses.

3.1 Proof of Work

This type of proof, mostly popular in early cryptocurrencies, is utilized to prove that certain computations (work) have been performed by the prover. The subsequent proof is then checked by the verifier or verifiers and accepted.

The key property of this protocol is the asymmetry of proving and verification. The process of verifying the proof has to be significantly less computationally expensive than the process of proving. Probably the most popular example of this is the mining mechanism in Bitcoin. Here, the miners are trying to find a hash of the new transaction block by varying an input in the block header. This hash is required to have a specific number of leading zeroes. The more zeroes, the more difficult it is to find a suitable hash. Once the hash is found, the solution is broadcast to the Bitcoin network, checked, and accepted by the verifiers.

This type of proof is based on the difficulty of certain computations and, therefore, will always be computationally expensive. For verification to occur, there is always a need for the result of computations to be revealed, therefore it does not offer any secrecy properties [29].

3.2 Proof of Stake

Proof of stake is commonly used as an energy-efficient, and therefore computationally efficient, alternative to the previously mentioned proof of work. Its main use case is cryptocurrencies.

Proof of stake is based on the idea that the network participants are required to possess an amount of chain currency in order to participate in the blockbuilding of a given blockchain. The strength of this proof resides in the fact that to take over a large portion of validation capacity, adversaries would have to possess rather large amounts of on-chain currency.

This type of proof lacks general versatility and only offers a narrow use case as it is mainly used to prove ownership and commitment of a certain amount of on-chain currency [18].

3.3 Homomorphic encryption

This form of encryption allows computations to be performed on encrypted data without the need for decryption. The result of operations is again encrypted data, which, when decrypted, will be the same as if the computations were performed on the unencrypted data. This scheme allows for enhanced privacy since the unencrypted data does not need to be sent anywhere.

While not usable as proof standalone, a homomorphic encryption scheme offers useful privacy-preserving properties [21, 31].

One downside is that homomorphic encryption carries a significant computational overhead when encrypting compared to traditional encryption schemes.

3.4 Multi-party computation

Another example that is not a proof, but rather a privacy-preserving technique, is multi-party computation (MPC). MPC allows a group of participants to collaboratively compute a function over their private inputs without ever revealing those inputs to one another. Each participant only learns the final result of the computation, not the individual data contributed by others.

MPC is mostly used in applications like secure electronic voting, secure auctions, key custody management, joint market analysis, and compliance with data privacy regulations.

While MPC ensures input privacy during computation, it does not inherently provide a proof that the computation was carried out correctly by its participants [10].

3.5 Digital signatures and PKI

Asymmetric cryptography and digital signature schemes such as RSA or ECDSA can generally be utilized in a proof of identity type of protocol or to prove the authenticity and integrity of certain data. In combination with public key infrastructure, they are a strong and well-known proving tool.

There are also some disadvantages, for example, the public key infrastructure requires trust in a third party to store and sign public keys. Another issue is performance, both size-wise and computation-wise. The amount of data that can be encrypted is limited,

usually by key size, and the process of encryption and decryption is slower compared to symmetric encryption. Performance can be improved by using hybrid encryption, but this requires additional usage of symmetric encryption.

The quantum resistance should also be mentioned. With the recent advancements in quantum computers, it is most likely only a matter of time until most of the widely used signature algorithms, such as RSA and ECDSA, become vulnerable to quantum attacks. To address this, there has been extensive research and development of post-quantum cryptographic algorithms, such as CRYSTALS-Dilithium [17], Falcon [20], and SPHINCS+ [30].

3.6 Hash-based commitment schemes

Hash-based commitment scheme offers a way to commit to a certain value without revealing said value straight away. In the case of hash-based schemes, the hash itself is used as a commitment, and the preimage represents the so far unrevealed value. Assuming up-to-date algorithms are used, this scheme is binding and offers integrity, as the hash commitment cannot be changed after the verifier obtains the commitment. When the time comes to reveal the commitment, the prover reveals the preimage, and the verifier checks if the hash output matches the original commitment. One of the advantages of this scheme is that it is not very computationally intensive. Hashing algorithms used in both the commitment and verification phases are rather fast. Despite the hash algorithm itself being fast, forging a commitment scheme remains difficult due to preimage resistance.

The disadvantage of this kind of scheme is its lack of reusability. After the commitment value is revealed, it cannot be used again. A new value needs to be created for every instance of the proving protocol [13].

3.7 Merkle tree proofs

Proofs based on the Merkle tree provide an efficient method to prove the presence of data in a large dataset. Every piece of data in a given dataset is hashed and arranged into leaves of a Merkle tree structure, where each node carries hashes of its children, which are then hashed together and stored in its parent, as is shown in Figure 3.1.

To prove the presence of data in a Merkle tree structure, we find a leaf node, which contains the data in question, and work our way up to the root. While traversing the tree toward the root, we hash every node along the way and compare the resulting hash to the hash of the root node. If they match, the presence of the particular data in a given dataset is proven. This approach is more computationally efficient compared to the alternative, which is traversing the whole dataset, although it introduces some initial overhead in creating a Merkle tree itself.

The cryptographic security and strength of this approach reside in the second preimage resistance. Overall, it is a fairly efficient proving tool, but its versatility is limited [43].

3.8 Zero-knowledge proofs

For a proof to be considered zero knowledge, it has to have specific properties. Besides having the ability to reliably prove something, zero-knowledge proofs have to allow a prover to execute the proving process without revealing any additional useful information to the verifier or any third party, besides the fact that a statement is true.

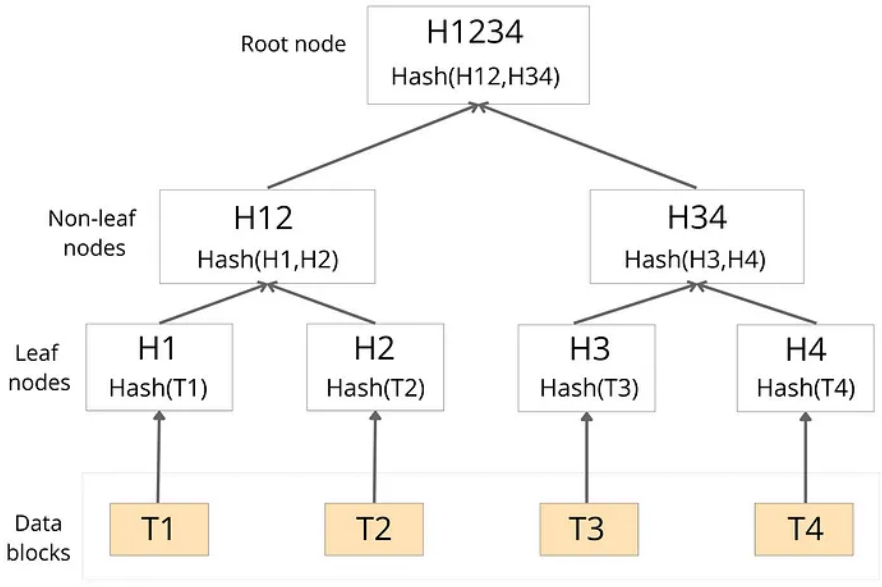

Figure 3.1: Merkle tree [43]

Although first mentioned some time ago[24], Zero-knowledge proofs are still in their early days in regards to their proper exploration, standardization, and wider use. Zero-knowledge proofs must satisfy three main conditions, the first two of which are standard for proof systems:

- Completeness: If the proof that the prover provides is legitimate, an honest verifier must be able to verify the proof successfully every time
- Soundness: If the proof provided by the verifier is false, the verifier must reject this proof with a high probability
- Zero-knowledge: the proof must reveal to the verifier only the fact that the statement is true, but nothing else

As zero-knowledge proofs are a recently rediscovered field, their taxonomy is still very ambiguous. One property, according to which Zero-knowledge proofs can be divided into two categories, is the required amount of interaction with the verifier [1].

3.8.1 Interactive zero-knowledge proofs

The first category is the interactive zero-knowledge proofs. Proofs in this category require an exchange of protocol messages between the prover and the verifier during the proving phase. Usually, the verifier provides some kind of self-generated entropy in order to be sure that the proof is authentic. This introduces multiple shortcomings. The obvious one is the fact that the need for communication during the proving phase introduces additional overhead. The second shortcoming is that the proofs are verifier-specific, which means that if, for example, the prover wants to prove a statement to a large group of entities, he has to go through the proving process with each one of them, which requires a lot of resources [41].

3.8.2 Non-interactive zero-knowledge proofs

These shortcomings are, however, solved by a second category of zero-knowledge proofs, the non-interactive ones. As the name suggests, this type of zero-knowledge proof does not require communication between the prover and the verifier during the proving phase of the protocol; the prover can create a proof independently, and the verifier can also verify the proof independently [41, 6].

Typical non-interactive zero-knowledge proofs have the following phases:

- Trusted setup
- Proof generation
- Verification

Trusted setup During the setup phase, the data that is necessary for proof creation and verification is created. It uses a computation function to create common knowledge that is shared between the prover and the verifier, which is called a common reference string. This common reference string is what allows the proof to be non-interactive and also bears a huge part of its security. The common reference string must be generated in a way that is trusted by both the prover and the verifier. It must be generated from truly random data. If the generation is not secure, if the adversary could have an influence on the data so that it would not be truly random, then the soundness and zero-knowledge properties could be jeopardized. Since having an actual third party provide the random data would defeat the whole purpose of using zero-knowledge proofs, the most common way of generating this data is through the previously mentioned Multi-party computation. There are proofs that require a trusted setup for every separate proof, only one trusted setup per many proofs, or no trusted setup at all [5, 24].

Proof generation Proof generation typically requires encoding the statement into a mathematical representation and then using specialized algorithms to compute the proof based on private witness data and a public statement. Private witness is a secret piece of information that only the prover knows and is used in the generation of the proof. The public statement is the claim being proven to the verifier. Witness and statement could have many forms, depending on the specific zero-knowledge proof used and the mathematical representation used.

Proof verification In this process, the verifier confirms that a given statement is true based on a cryptographic proof provided by the prover, without learning any information about the underlying private witness. The verifier uses the public statement to run a verification algorithm, which outputs a simple true or false result. This allows the verifier to be confident that the prover possesses a valid witness, without needing to trust the prover or perform the full computation themselves.

3.8.3 State-of-the-art zero-knowledge proofs

As the advancements in this field continued, some of the proposed zero-knowledge proofs emerged as industry leaders. Some of the most prominent protocols include zk-SNARKs, zk-STARKs, and Bulletproofs. They are the most widespread and advanced examples of zero-knowledge proofs [39].

Zk-SNARKs

Short for zero-knowledge succinct non-interactive proofs of knowledge. This type of zero-knowledge proof is arguably one of the most advanced recent ones. For a proof to be considered zk-SNARK, it must have certain properties, with zero-knowledge and non-interactivity already introduced previously.

For a proof to be succinct, it means that it is small and can be verified very quickly. Normally, proving knowledge of a complex computation would require large, time-consuming proofs. Succinctness suggests that a proof is packed into a very small size that can be verified in a fraction of the time it would take to run the computation itself.

Argument refers to a proof system where the prover can convince the verifier of the truth of a statement, assuming that the prover has limited computational power.

The knowledge part ensures that the prover actually knows the specific secret or data they are claiming to prove knowledge of. It's not just that they can generate a valid proof by chance, but that they must have the actual information required to generate the proof [22].

They were designed to deliver small proofs, even at the expense of other properties, where one such expense is a need for a trusted setup. A trusted setup is an initial phase where secret parameters, such as randomness, are generated to create public proving and verification keys. If the setup is compromised and the secret values are not securely discarded, a malicious party could create fake proofs that appear valid.

As they utilize elliptic curves, the zk-SNARKs are not quantum safe.

Mathematical principle - zk-SNARKs

The zk-SNARK proof setup phase takes the subject of the proof and transforms it into a common reference string. This transformation requires a number of mathematical operations.

The first step in transforming a proof subject into a mathematical form is to break down its logical steps. This breakdown reduces the proof subject into its simplest operations, forming a structure called an arithmetic circuit. The arithmetic circuit represents a program decomposed into basic steps involving trivial arithmetic operations. It consists of inputs, wires, and gates, as shown in Figure 3.2.

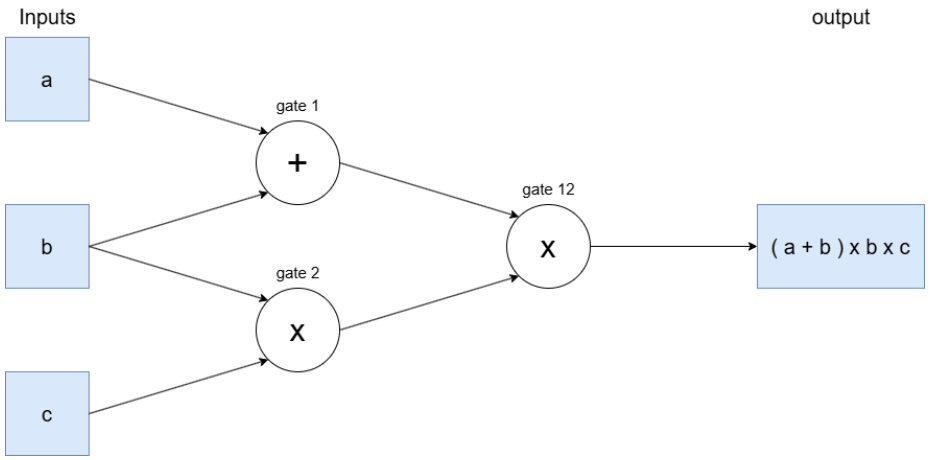

Figure 3.2: Arithmetic circuit [22]

Once the arithmetic circuit is computed, the next step in constructing zk-SNARKs is creating a Rank 1 Constraint System. Rank 1 Constraint System ensures that the values move correctly through the circuit by introducing a vector of constraints based on dot products, which represent the subject of the proof. The witness in this case is another vector, which, when used as an evaluation, satisfies the vector of constraints. This representation requires the verifier to check constraints on each circuit wire, which is resource-consuming.

There is, however, an efficient way to bundle all constraints into one using a Quadratic Arithmetic Program to represent the circuit. In this case, the single constraint is evaluated between polynomials rather than vectors. Rank 1 Constraint System can be transformed into a Quadratic Arithmetic Program (QAP) using Lagrange interpolation or a more efficient Fast Fourier Transformation [8, 12, 38].

Elliptic curves play a crucial role in zk-SNARKs. If a prover knew the exact point the verifier would use for evaluation, they could craft invalid polynomials that satisfy the identity at that point. To address this, zk-SNARKs use techniques such as elliptic curve pairing and homomorphic encryption, enabling blind evaluation. In blind evaluation, neither the prover nor the verifier knows the point under assessment. Public parameters determine this point, while elliptic curve pairing ensures that the evaluation is secure.

Some of the particular examples of zk-SNARKs include The Pinocchio [37], which is one of the earlier examples, the widely used Groth16 [25], and PLONK [19], which introduces a universal setup.

Groth16 Perhaps the prime example of the zk-SNARKs is Groth16. It is one of the most efficient known constructions in terms of proof size and verification time. It is especially known for its succinctness: the resulting proofs are constant-sized (roughly 200–300 bytes) regardless of the complexity of the computation.

While it builds upon the earlier zk-SNARK constructions, its contribution is the improvement in reducing the proof size by using only three group elements and requiring only a single product equation using three pairings for verification [25].

PLONK Stands for Permutations over Lagrange bases for Oecumenical Noninteractive arguments of Knowledge. It is a universal, efficient, and transparent zk-SNARK, which was developed to overcome key limitations in older zk-SNARKs like Groth16, specifically the need for a trusted setup per circuit and lack of modularity.

It is mostly similar to previous zk-SNARKs, with the notable differences being the usage of polynomials over a Lagrange basis, instead of QAP, and using a permutation check to ensure that the wiring between gates in the circuit is valid. These allow PLONK to feature a universal setup, where a single trusted setup can be reused across many circuits [19].

zk-STARK

Another member of the state-of-the-art zero-knowledge proofs are the zk-STARKs. First introduced in 2018, the acronym zk-STARK stands for zero-knowledge scalable transparent argument of knowledge [4].

Similar to zk-SNARKs, they are a cryptographic zero-knowledge proof system, consisting of a prover and verifier, where the prover can convince the verifier that a computation was performed correctly, without revealing any details about the inputs or intermediate steps of the computation.

As denoted by the „T“ as transparent in zk-STARK, the zk-STARKs do not require a trusted setup. Instead, they only need a transparent setup, meaning there is no need for a trusted third party to initialize the proof system. The randomness used by verifiers is publicly available. No requirement for a trusted initialization simplifies the setup from both the security and computational point of view.

Another one of their advantages is quantum resistance, as the only cryptographic part of zk-STARKs is the hash function.

Zk-STARKs are also more efficient at handling large proofs and scale well. As the complexity of the proof statement increases, the size and the time required for verification rise far less significantly than in the case of zk-SNARKs. This improvement from zk-SNARKs, however, comes at a cost. Zk-STARKs use different cryptographic principles that allow this, but in exchange, the proofs are larger, usually 10 to 100 times larger compared to zk-SNARKs. They were initially developed with scalability in mind, where a large part of the computation can be taken off-chain, or utilizing Layer-2 networks, therefore reducing the costs [32, 2].

Mathematical principle - zk-STARKs

It is similar to zk-SNARKs, where the prover expresses the computation as a sequence of logical steps, which are transformed into a structured format known as an algebraic execution trace. This trace is then interpolated as low-degree polynomials over finite fields. The prover then commits to these polynomials using hash functions and performs the Fast Reed-Solomon Interactive Oracle Proofs of Proximity to show that they satisfy a set of constraints representing the computation. The verifier checks a few randomly chosen parts of the committed data to be convinced that the whole computation is valid. The probability of catching a dishonest prover is extremely high, even with only partial checking [42].

Bulletproofs

Introduced in 2017 [9], Bulletproofs are another cryptographic zero-knowledge proof system, with their primary application being use in cryptocurrencies.

Bulletproofs are designed to be short, efficient, and do not require a trusted setup. They are especially well-suited for range proofs, but their capabilities include creating proofs for general-purpose arithmetic circuits.

Mathematical principle - Bulletproofs

In the Bulletproof protocol, the prover first creates a Pedersen commitment of a vector of values, which serves two purposes: it hides the original value and allows for certain arithmetic operations to be performed on the hidden value. The process of finding a value hidden by creating a Pedersen commitment would require solving a discrete logarithm problem. Therefore, the security of Bulletproofs relies on the general assumption about discrete logarithms, which means that it is not quantum safe. The next step in the protocol is constructing an inner product argument, where the inner product polynomial is defined. The inner product polynomial is derived from the two vector polynomials in such a way that evaluating the inner product polynomial yields the same result as evaluating the two vector polynomials and then computing the inner product. The biggest innovation of the Bulletproofs is the improved inner product argument [7], where the argument is constructed using recursion, which results in a logarithmic reduction in its size. The protocol must also

ensure non-interactivity, which is done by replacing the random challenges using the Fiat-Shamir heuristic and hash function.

Notable feature of Bulletproofs is their support for the aggregation of proofs, both from a single party and multiple parties. In the case of a single party, one proof can be used to prove that more than one value lies in the given range, whereas multiple parties can use the protocol to create a single range proof, without revealing the inputs to each other.

3.8.4 Comparison

In this section, we will compare the properties of the zk-SNARKs (Groth16), zk-STARKs, and Bulletproofs. The findings were studied from the work by Oude Roelink, B. [36], where the three Rust implementations of the mentioned zero-knowledge proofs, and the additional Go implementation of Groth16 were tested, and also from the work by Oude Roelink, B.; El Hajj, M. and Sarmah, D [35], where a systematic review of other papers about the zero-knowledge proofs was conducted.

Quantum resistance

This aspect is worthy of consideration mainly in the context of future-proofing. The quantum resistance mainly depends on cryptographic assumptions of a given protocol. Both Bulletproofs and zk-SNARKs depend on the discrete logarithm problem and elliptic curve cryptography, which are not quantum safe, rendering the proofs themselves also not quantum safe. Zk-STARKs remain the only quantum-resistant protocol, which was mentioned, due to their reliance solely on cryptographic hashes.

Performance

Arguably, for the purpose of this thesis, the most important aspect of a given zero-knowledge proof is performance. Performance in the context of zero-knowledge proofs evaluates two main parameters: the size and the computational intensity. The size parameter includes mainly the size of the proof itself, but also the size of the prover and verifier keys is taken into account. The computational intensity focuses mainly on the proof verification time and the proof verification time, but the need for a trusted setup, or lack thereof, was also taken into account.

Size The observations about the proof size concluded that zk-SNARKs consistently provided the smallest proof size, as the succinctness in their name suggests, closely followed by the Bulletproofs, which were approximately 10 times larger. In the case of zk-SNARKs, the proof size remained constant even with larger proof subjects, while with the Bulletproofs, there was a slight increasing trend. The zk-STARK proof size was the largest, approximately 100 to 1000 times larger than zk-SNARKs, depending on the size of the proof subject.

However, as mentioned previously, the size of the proof is not the only factor contributing to the overall size parameter of a given proof. In the case of zk-SNARKs, the verifier tools are also required in order to verify the proof. The respective size of the verifier tools varied based on implementation, from a constant size to a linearly increasing size depending on the proof subject. In the case of the first examined implementation, which had the constant verifier tools size, the total size when counting proof and verifier tools increased to a similar level as that of Bulletproofs. In the case of the other examined implementation, the size of

the verifier tools was linearly increasing with the size of the proof subject, with total size starting on a similar level as Bulletproofs and ending with an even larger total size than zk-STARKs.

The zk-SNARKs verifier tools are designed to be reusable, meaning every proof computed using the same arithmetic circuit can be verified by the same set of verifier tools. Therefore, this overall size increase is only in the form of an initial overhead of including the verifier tools with the proof and will become negligible with time and the number of proofs produced with the same set of verifier tools.

Proof generation time Generally, the zk-STARK implementation offered the fastest proof generation times, with zk-SNARKs closely in second place. The worst performing in this metric were Bulletproofs, which were approximately 10 times slower than zk-STARKs. All of the tested zk-proofs showed a linear increase with a larger proof subject, with Bulleproofs showing the largest increasing trend.

Proof verification time This metric followed a similar pattern to proof generation times. The overall fastest times were offered by zk-STARKs across all measured proof subject sizes, although with a slightly increasing tendency. Zk-SNARKs were the second fastest, about 10 times slower than the zk-STARKs. They also showed an increasing trend with larger proof subject sizes, but much less so than in the case of zk-STARKs. The slowest were Bulletproofs, be it only when considering the average times across all tested proof subject sizes. When looking at the smallest proof subject size, the Bulletproofs were slightly faster than zk-SNARKs, but their verification time increased more significantly with larger proof subject size, almost 1000-fold.

Conslusion The performance comparison brought insight into the strengths and weaknesses of the examined zero-knowledge proofs. It is difficult to select an outright winner as every zero-knowledge proof fits a different use case.

The zk-STARKs excelled at proof generation and verification times, but at the cost of significantly larger proof sizes. The zk-SNARKs offered the smallest proof size combined with very competitive generation and verification times, which were not far from the zk-STARKs. The Bulletproofs were in the middle, although a bit closer to zk-SNARKs when comparing size, suffered with the worst proof generation times, and the worst average proof verification times. Performance testing results from [36] are available in table 3.1.

Protocol	**Size (B)**	**Setup time (ms)**	**Proof generation time (ms)**	**Verification time (ms)**
Bulletproof	993.0	-	849.705	275.701
zk-SNARK #1 (Rust)	192.0	128.947	29.878	1.695
zk-SNARK #2 (Go)	484.0	131.650	18.155	2.351
zk-STARK	28388.4	-	14.384	0.232

Table 3.1: Protocol comparison using the average performance over the five default benchmarks with different proof subject sizes [36].

Chapter 4

Analysis

In this chapter, we will discuss the findings and conclusions that resulted from the analysis of the HOPR network.

4.1 What should the optimization look like

Firstly, we need to assess what the properties of the proposed optimizations should look like and what constraints must be satisfied. The core goal of a HOPR network is to provide transport-layer security to its users. Proposed optimizations must not compromise this in any way. As discussed previously, the effort will be to optimize the incentivization layer.

There are also core incentive principles that the incentivization mechanism has to possess:

- payment enforcement - the node that is paying must not be able to deny the payment
- payment existence - there has to be a way for a node that is supposed to receive the payment to check that the payment exists, and it needs to be sure that it will actually receive it
- Proof of Relay - the node gets the payment only after relaying the packet

4.1.1 Participation of entities in the protocol

Let's consider packet sender S, HOPR nodes A, B, C, and packet recipient R. The packet is sent from sender S to node A, which then relays it to node B, which then relays it to node C, which then delivers the packet to its recipient R. In the examined version of the HOPR network's incentivization mechanism, the participation of all of the entities is necessary to achieve the incentive principles.

The first incentivization principle

In the current version of the HOPR network, it is achieved by a combination of sender participation and packet hash.

Let's assume that node A is malicious and wants to avoid paying the next node, B, for relaying the packet. One way this could be achieved is by sending all the packets with incorrect challenges. Node A could send the ticket containing a challenge that does not match the acknowledgment. The node B would find out only after relaying the packet, so the malicious intentions of node A would be achieved by then.

HOPR incentivization mechanism mitigates this by including the original packet sender, S, in the process of creating a ticket. When the sender S creates a ticket, he generates all the parts of the HOPR's Proof-of-Relay challenge-response scheme in such a way that they are derivable by the nodes along the path.

More specifically, node A has the ability to derive the challenge for node B, node B can derive its own challenge, shared key and hint; and node C can derive the acknowledgment for node B. Node A then uses the derived challenge in the ticket for node B. If node A used a different challenge, node B would know since it can derive the challenge too. Tampering with the layer intended for the next node is not possible since the next node always checks the hash of the packet after receiving it.

The key takeaway here is that the reason why node A cannot deny payment to B in the previously mentioned way is the participation of the sender S. In other words, in order for the first incentive principle to be satisfied, the sender S has to participate in any mechanism used in the payment scheme.

The second incentivization principle

The second incentivization principle is the verification of payment existence. This is currently assured by the presence of Hint among the data, which can be derived from the packet. The node that receives a ticket can check for the existence of the solution to an embedded challenge, as described in paragraph „Challenge, Response, and Hint“ in section 2.5.3. Fulfilling this incentivization principle is possible again due to the participation of the sender node and the specific properties of the curve operations used in the Proof-of-Relay scheme.

The third incentivization principle

This principle assures that the payment can only be obtained after the relay is successfully finished. This is currently assured by the participation of the sender node and the next downstream node. The sender node initially generates an acknowledgment, which is a key to solving the Proof-of-Relay scheme. The next downstream node can derive the acknowledgment and, after it checks that the packet arrived as it should, it sends the acknowledgment back to the upstream node.

The key takeaway In order to be able to guarantee these principles, the sender node must always participate in the node-to-node incentivization protocol, as it is the only entity that can share information with all of the nodes along the path. This means that any designed incentivization principle must keep the participation of the sender node.

4.2 Zero-knowledge proof integration

After studying the specifics of HOPR's incentivization layer and studying the properties, benefits, and drawbacks of the zero-knowledge proofs, we will discuss the possible ways of their integration into the HOPR incentivization layer for the purpose of optimization.

Two main possible directions of their integration were identified: one zero-knowledge proof per packet and one zero-knowledge proof spread among many packets. The main considerations here are the memory overhead in the form of the size of the ticket, which is currently 180 bytes, and the computational overhead.

4.2.1 One zero-knowledge proof per packet

Such theoretical protocol, which utilizes one zero-knowledge proof per packet, whatever its specifics would be, would always have to carry all the necessary cryptographic data required for proof verification. The nodes would also have to perform all the necessary calculations related to that specific zero-knowledge proof. Let's have a look at the feasibility of both with different zero-knowledge proofs used.

zk-SNARKs

The memory overhead, in the best-case scenario, would be at least 720 bytes (528 bytes for the verification key, and 192 bytes for the proof itself [36]), which is already much higher than the whole routing header of the current HOPR solution (the parts of zero-knowledge proofs would have to be a part of it).

Another problem occurs with the need for a trusted setup, which takes a minimum of approximately 3.5 milliseconds [36]. This, coupled with the best-case scenario for proof generation time of 1.299 milliseconds [36], makes the overall computational overhead very significant.

The zk-SNARKs have been shown to be unsuitable for integration.

zk-STARKs

The case is similar with zk-STARKs. The trusted setup is not an issue here, and the overall computational overhead is lower (but not significantly enough) than in the case of zk-SNARKs. However, this time it is the size of the proof, which is thousands of bytes, that makes them unsuitable for integration.

Bulletproofs

The phenomenon of unsuitability repeats with Bulleproofs also, as their very large proof generation times (hundreds of milliseconds) rule them out completely.

4.2.2 One zero-knowledge proof spread among many packets

This direction consists of designs where a single zero-knowledge proof would be used by more than one packet, either by dividing the proof directly and embedding it into packets, or by reusing the same set of verifier tools, e.g., verification key, common/universal reference string, or witnesses.

All of the variations of this direction would have one property in common, and that is the logical coupling of several packets, among which is the proof spread, together. As the nodes would process such packets, they would be able to group them by the set of verifier tools necessary to operate with the embedded proofs. This property is not always detrimental, but we will show that in our case, it will be.

As was shown in section 4.1.1, in order to preserve the incentivization principles, the packet creator must always participate in the incentivization protocol. This, together with the previously mentioned logical coupling of packets, means that if the packet creator is to participate in the protocol, then the packets from one creator will inevitably be logically coupled together.

This will effectively allow nodes to group packets by their creator. When considering a standard 3-hop route of the packet, this becomes a problem from the second node onwards,

as the second node is not in direct contact with the packet creator, and therefore it should not be able to distinguish which incoming packets belong to the same origin, but since they are coupled together, it can be done.

Such a situation creates a security flaw, as the node now has access to metadata about the time and frequency of communication from a given origin, albeit without knowing its direct identity.

One possible solution to this problem would be to design a protocol in a way that the coupling is by the paying node, rather than the packet creator. This is again unfeasible, as incentivization participation of only the payer node cannot guarantee the payment enforcement.

As the coupling likely cannot be removed, this direction is also not suitable for integration.

4.3 Room for improvement in ticket aggregation

Upon closer examination of the ticket aggregation process, described in 2.5.3, some room for improvement was found. In the examined version of the HOPR network, the ticket aggregation process was found to execute unnecessary operations. The process at its current state operates as showcased in the figure 4.1:

1. Firstly, a node that wants to have its winning tickets aggregated reaches out to the node that issued said tickets with an aggregation request
2. The requesting node then sends all of the winning tickets, including responses to their challenges, to the aggregating node
3. The aggregating node checks whether all of the received tickets are, in fact, winning ones
4. If the check is successful, the aggregating node will create one aggregated ticket with a 100% winning probability and send it to the requesting node
5. The requesting node then redeems the aggregated ticket using the smart contract
6. The smart contract then adjusts the state of the payment channel pointing from the aggregating node to the receiving node

The smart contract will treat the aggregated ticket as any other ticket, meaning that in order to redeem it, all the checks described in section 2.5.2 must be satisfied. These checks are: validation of funds and payment channel existence, ticket index, ticket signature, winning status, VRF value correctness, and Proof-of-Relay Response correctness.

Since the aggregated ticket is a special case, as it was not created as a reward for relaying a packet, but rather by a special request, its redemption should not be conditioned by knowing the solution to the Proof-of-Relay scheme. Therefore, all checks related to the Proof-of-Relay scheme are redundant, specifically checks of winning status, VRF value correctness, and Proof-of-Relay Response correctness.

The second problematic area of the ticket aggregation process is its enforcement. At the current state, there is little to no motivation for the aggregating node to aggregate the tickets for the requesting node. If the aggregating node refuses to perform the aggregation, the requesting node can still redeem its winning tickets one by one, which will consume more gas.

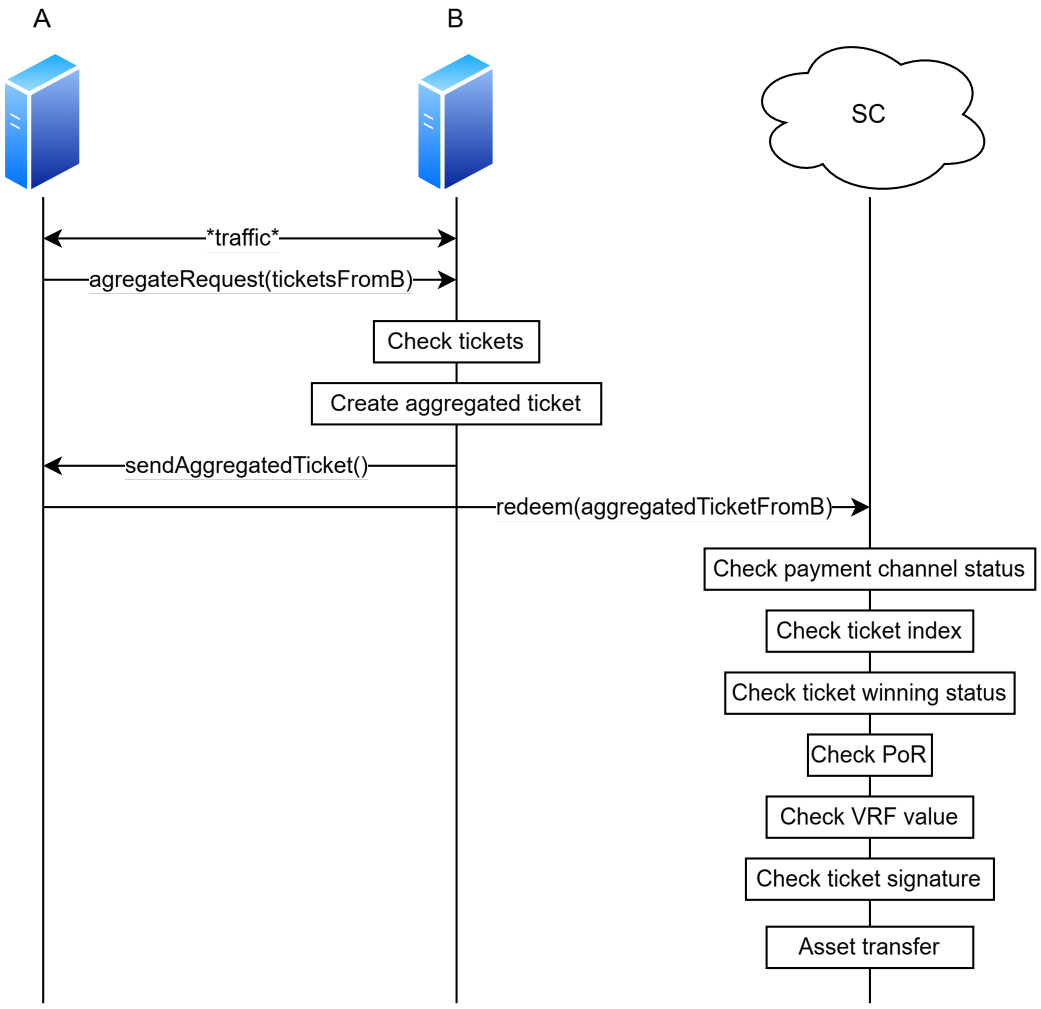

Figure 4.1: Existing ticket aggregation protocol between nodes A, B, and Smart Contract

Chapter 5

Design

This chapter will go over the design process of HOPR network optimization. The designed optimization must maintain the core principles and security of the HOPR network, and its goal is to improve its performance or cost effectiveness.

5.1 Proposal

Based on the findings from the previous chapter, we propose the following optimization of the ticket aggregation process:

1. Motivate nodes to perform the ticket aggregation.
2. Remove the unnecessary checks from the smart contract.

5.1.1 Adding motivation

One of the simplest ways to convince someone to do something is to offer something in return. If we apply this to our case, we must find something that will incentivize the nodes to perform aggregation.

At its current state, the aggregation is a one-sided operation, meaning that in order for both nodes to obtain aggregated tickets from one another, they must both perform the aggregation process, depicted in Figure 4.1.

The proposal is to transform the aggregation process into a joint operation between the two nodes, where the nodes will cooperate to create an aggregated ticket, which will then be redeemed by one of the nodes using the smart contract.

The benefit to the payer node, compared to the original aggregation process, is the cost saving on the smart contract execution fees, since it will be the receiving node that calls the smart contract. Since the receiving node is owed a larger amount than it owes, it is in its interest to redeem the aggregated ticket.

The payer node can still refuse to perform the aggregation process (as it loses a larger amount of tokens than it gains), but with the proposed aggregation process, it would be to the node's own detriment. As an alternative to the aggregation process, is to redeem the tickets one by one, which would result in the same amount of tokens sent and received, but at additional redemption costs for everyone.

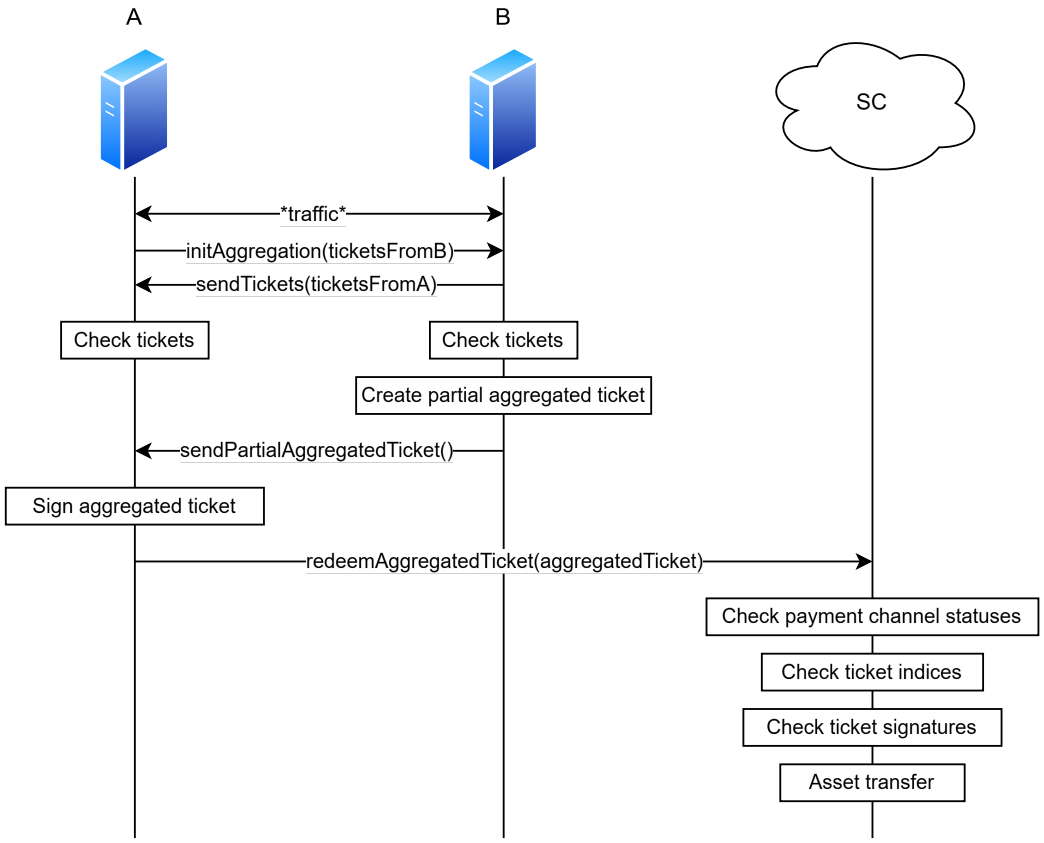

Figure 5.1: Proposed ticket aggregation protocol between nodes A, B, and Smart Contract

5.1.2 Optimizing the smart contract

As was mentioned in the previous chapter, the current version of the smart contract for redeeming an aggregated ticket includes some redundant operations, which we aim to eliminate. In our proposed version, depicted in 5.1, the smart contract will only check the necessary items: validation of funds and existence of payment channels, ticket indices, ticket offsets, ticket epochs, and ticket signatures, which should result in reduced execution costs. In the proposed version of the smart contract, the checks for the channel status, ticket index, ticket offset, ticket epoch, and signature must be performed twice, once for each node, which will inevitably increase the cost of these changes. However, the expectation is that the omission of unnecessary checks will make up for this increase.

5.1.3 Protocol description

This subsection will go over the exact steps of the proposed protocol.

Prerequisities The execution of the new ticket aggregation process requires certain conditions to be met.

- Payment channel(s) - existence of payment channel in the direction of the funds transfer is mandatory, while the existence of a payment channel in the opposite direction is not compulsory

- Previous traffic - the protocol assumes that a substantial amount of traffic has occurred between the two participating nodes, resulting in the emission of a number of winning tickets. The sum value of winning tickets should be large enough for it to be economically feasible to perform aggregation and, therefore, on-chain redemption

Protocol steps - nodes

1. Once a node (let's call it the „requesting node“) gathers a large enough number of winning tickets from another node (let's call it the „issuer node“ - as it is the node that issued the tickets which the requesting node wants to aggregate), the requesting node can initiate the aggregation process by sending the aggregation request to the issuer node

2. The aggregation request from the requesting node contains all of the winning tickets issued by the issuer node, along with the solutions of their respective challenges

3. The issuer node replies by sending all of its winning tickets, issued by the requesting node, to the requesting node

4. Both nodes will now check the validity and the winning status of the received tickets and calculate the sum of their values, which will indicate how much each node is owed.

5. The difference of these sums is calculated, and the node that is owed more now becomes the „receiving node“, and the node that owes more now becomes the „payer node“.

6. The payer node creates a partial aggregated ticket that contains a set of:

   - ticket indices
   - ticket offsets
   - channel IDs
   - channel epochs

   for both the payer and receiver nodes, and the amount, which is the difference between the sum of the payer's and the receiver's tickets.

7. The payer node then signs the partial ticket to make it verifiable that it agrees with the transaction and sends it to the receiver node.

8. The receiver node then also signs the partial ticket, thus also expressing its agreement with the transaction, which makes the ticket complete

9. The receiver node then redeems the complete aggregated ticket using the smart contract

10. The smart contract then performs the appropriate checks and transfers the amount of tokens to the receiver node, described in detail in the following section

Protocol steps - smart contract

1. The process begins by retrieving the current states of both the payer's and receiver's payment channels and verifying that the payer's channel is not closed
2. The provided epochs are compared against the current channel epochs (stored on-chain), ensuring consistency and preventing double spending
3. Similarly, the provided ticket indices and offsets are compared against the current channel indices and offsets to prevent double spending
4. The available balance in the payer's channel is checked to ensure it can cover the amount being claimed
5. Using the ticket hash, the signatures of the payer and receiver are verified
6. The indices and offsets of both payment channels are updated
7. The balance of the payer's channel is lowered by the amount, while the balance of the receiver's payment channel is credited with the amount
8. The notifications about the ticket redemption and change in balance of each channel are recorded

Chapter 6

Implementation

This chapter will discuss the implementation specifics of the proposed optimization of the HOPR network. The optimization is based on the existing HOPR network source code and can be integrated. The Smart contract is written in the Solidity programming language.

6.1 Data structure

As the proposed optimization removes checks related to the Proof-of-Relay scheme and performs the checks for two nodes instead of one, some adjustments needed to be made to the structure of the aggregated ticket.

```
struct AggregatedTicket {
     Balance amount;
     TicketIndex ticketIndexSource;
     TicketIndex ticketIndexDest;
     ChannelEpoch epochSource;
     ChannelEpoch epochDest;
     TicketIndexOffset indexOffsetSource;
     TicketIndexOffset indexOffsetDest;
     bytes32 channelIdSource;
     bytes32 channelIdDest;
     HoprCrypto.CompactSignature signatureSource;
     HoprCrypto.CompactSignature signatureDest;
 }
```

Figure 6.1: Aggregated Ticket struct

As can be seen in the figure 6.1, the `ticketIndex`, `ChannelEpoch`, `TicketIndexOffset`, `channelId`, and `signature` are all included twice, one set of values belonging to each node. The winning probability and Proof-of-Relay response are removed from the structure completely.

6.2 Ticket redemption

The ticket redemption is implemented by the `redeemAggregatedTicket` function, which is based on the `_redeemTicketInternal` function by HOPR.

The function begins by fetching the source (`spendingChannel`) and destination (`earningChannel`) channels using the provided identifiers. It verifies that the spending channel is in a valid state, either `OPEN` or `PENDING_TO_CLOSE`. If not, the function reverts with an error, enforcing state consistency.

Additionally, the function ensures that the epochs associated with the ticket match the current epochs of the channels. Epochs are used to prevent replay attacks.

Next, the function validates the index intervals of the aggregated ticket. Each ticket specifies a base index and an offset, which together define the range of tickets it represents. The function ensures that the offset is greater than zero and that the base index is not older than the current ticket index of the respective channel to prevent double-spending or replay. This is done for both the spending and earning channels

The function further ensures the spending channel has enough balance to cover the amount being redeemed. If not, the redemption is rejected.

Both source and destination addresses are recovered from the ticket hash and signatures using ECDSA. The recovered addresses are validated by recomputing their associated channel IDs and comparing them against the values provided in the ticket to ensure their authenticity.

If all checks pass, the function proceeds to update the state, where the `ticketIndex` of the spending channel is advanced and the balance of the spending channel is reduced by the ticket amount.

If the destination channel is closed, the funds are transferred directly to the recipient's wallet via a direct token transfer. Otherwise, the `earningChannel` is updated by incrementing its `ticketIndex` and increasing `Balance` by the `amount` redeemed.

The following events are emitted:

1. `ChannelBalanceDecreased` for the sender's channel
2. `TicketRedeemed` indicating the sender's new ticket index
3. `ChannelBalanceIncreased` for the receiver's channel
4. `TicketRedeemed` for the receiver's new ticket index

6.3 Ticket hash function

In order to sign the ticket, its hash must be obtained. As the ticket structure was changed, the hashing function must be adjusted accordingly.

The hash function utilizes `keccak256` algorithm and a custom way of hashing structured data, which deviates from the EIP-712 standard to provide compactness. The function goes through several rounds of packing and encoding of the structure's members and two rounds of hashing, while adding the two control bytes, domain and function-specific identifiers.

Before the final round of hashing, the fixed EIP-191 header and the domain separator are added. The use of a domain separator binds the hash to a specific contract and chain, mitigating replay attacks across different domains or forks.

6.4 Tests

The implementation includes several test cases. These test cases were developed to ensure the correct behaviour of the implementation, and will later be used to measure performance.

The tests are developed with Forge, which is an Ethereum-native testing framework for smart contracts written in Solidity, developed as part of the Foundry toolkit[1].

6.4.1 Functional

The aim of this test case is to check whether the smart contract behaves as expected when executed in the correct setting with the correct parameters.

The test function `test_redeemAggregatedTicket()` begins by setting up internal variables, namely bounding private keys, `privKeyA` and `privKeyB`, within the valid `secp256k1` field range to ensure they are usable for signature generation. A key assumption is enforced to ensure that the two keys are distinct.

Next, the amount to be redeemed is constrained within a valid operational range defined by the protocol constants `MIN_USED_BALANCE` and `MAX_USED_BALANCE`. The total `channelAmount` is then bounded to be at least as large as the ticket amount, ensuring redemption is feasible. The `indexOffset` is enforced to be strictly positive, and the ticket indices, `channelTicketIndex` and `maxTicketIndex`, are carefully bounded to prevent integer overflow during aggregation while still allowing index advancement within expected limits.

With these internal variables initialized, the test proceeds to simulate the sender (`src`) and receiver (`dest`) addresses using the private keys.

The addresses are then funded with the `amount` of the token.

The test constructs a `RedeemAggregatedTicketArgBuilder` object, encapsulating all inputs necessary to generate a valid, signed `AggregatedTicket` off-chain.

Subsequently, two payment channels are preloaded into the system to simulate the open payment channels between the sender and receiver. Each channel is initialized with the agreed balance, epoch, ticket index, and open status, as they would be in a real-world scenario.

An aggregated ticket is generated using an auxiliary function called `CryptoUtils.getRedeemableAggregatedTicket`. This function generates the aggregated ticket as it would be generated by the two nodes in the real-world scenario.

Before redemption, the test sets expectations for emitted events during the redemption process:

1. `ChannelBalanceDecreased` for the sender's channel
2. `TicketRedeemed` indicating the sender's new ticket index
3. `ChannelBalanceIncreased` for the receiver's channel
4. `TicketRedeemed` for the receiver's new ticket index

The redemption is then triggered by the `dest` address via `vm.prank`, which simulates the transaction sender context.

After verifying the redemption under an `OPEN` channel, the test repeats the process with the destination channel set to `PENDING_TO_CLOSE`. This ensures the smart contract correctly allows redemptions during channel shutdowns, provided all other validation criteria are met.

[1] https://book.getfoundry.sh/

6.4.2 Cost measurement

The aim of this test part is to measure the deployment costs and execution costs of the implemented ticket aggregation protocol and its comparison to the previous version.

The Forge tool has a feature to measure the gas cost of contract deployment and execution of a function. However, as the whole test case is written as a Solidity function, a creative approach must be taken to filter out the cost of parameter generation and crafting of the aggregated ticket to accurately measure the true cost of the redemption function itself.

Three separate functions, all based on `test_redeemAggregatedTicket()`, were implemented. The functions were adjusted so that the only difference in their respective logic would be the number of calls to `redeemAggregatedTicket()` function, which would be zero, one, and two. The rest of the function logic is identical.

The true execution cost of the `test_redeemAggregatedTicket()` can then be obtained as the difference between the execution cost of the test functions with zero `test_redeemAggregatedTicket()` calls and one `test_redeemAggregatedTicket()` call.

The same procedure was used to obtain the execution cost of the HOPR's original aggregated ticket redemption function. The contract deployment cost was measured by the built-in Forge inspect contract functionality.

Chapter 7

Test results and Evaluation

This chapter will provide the results of testing, both functional and performance, and discuss the pros and cons of the implemented optimization. The tests were executed on a local development Ethereum node[1], inside the Nix[2] development environment running on Ubuntu 22.04.5 LTS inside WSL.

7.1 Test results

7.1.1 Functional tests

The results of the functional testing indicate that the smart contract behaves as expected in the tested scenario. The test case passed consistently and produced repeatable results, suggesting that the implementation is deterministic and correctly handles the specific input and produces correct output.

7.1.2 Execution cost measurement

The execution cost measurements were performed for both the old and the new aggregated ticket redemption functions. The execution gas cost of the functions themselves was obtained as explained in subsection 6.4.2. The measured costs of the test cases execution, seen in Table 7.1, were used to calculate true execution costs seen in Table 7.2.

Executed action	Average Gas	Median Gas
zero redemptions old	3,463,037	3,355,000
one redemption old	3,548,359	3,429,596
zero redemptions new	190,577	190,493
one redemption new	267,496	267,378

Table 7.1: Execution Gas cost of the test functions across 257 runs.

From Table 7.2, we can see that the gas execution cost of the new function was approximately 11% lower when looking at the average value and approximately 3% higher when looking at the median value. We can also see larger differences between the average and the median cost of the old function execution. This is most likely due to the input-dependent intensity of the Proof-of-Relay scheme calculations and the VRF value verification.

[1] https://book.getfoundry.sh/anvil/
[2] https://nixos.org/

Redemption function	Average Gas	Median Gas
old	85,322	74,596
new	76,919	76,885

Table 7.2: Execution Gas cost of the test functions across 257 runs.

The overall improvement is around a 50% reduction in the overall cost of executing aggregation on-chain, as the designed protocol requires only one smart contract execution, compared to two executions, which were required by the original version.

7.1.3 Deployment cost measurements

For the sake of completeness, the additional deployment cost of the function `redeemAggregatedTicket()` was measured and can be seen in Table 7.3. The original function used to redeem aggregated tickets must be preserved, as it also provides functionality to redeem the standard tickets. Therefore, there are no cost savings here.

Deployed subject	Gas
Original `MyHoprChannels` contract	4,207,177
Modified `MyHoprChannels` contract	4,702,177
`redeemAggregatedTicket()` function	495,000

Table 7.3: Deployment Gas cost.

7.2 Evaluation

The proposed aggregation optimization successfully passed the functional test, which covered the basic functionality. The general goals of the assignment were to optimize the HOPR network by increasing performance or reducing costs. The proposed solution focused on the latter, with the specific goals being to lower the cost of the aggregation process and, additionally, to add motivation for the nodes to perform aggregation. The fulfillment of these goals and a couple of additional advantages and disadvantages will be discussed in the following section.

Optimizing the smart contract The goal of lowering the cost of the aggregation process was met by successfully optimizing the smart contract, which yielded a 50% reduction in the overall execution cost for both nodes, by requiring only one on-chain execution, instead of two previously.

Adding motivation The goal of adding motivation was also successful, although how effective the added motivation will be is subject to question. The proposed ticket aggregation process is still not strictly enforced by the protocol, however, it is now clearly the most efficient way for all the nodes to redeem their ticket. The node theoretically could still refuse to participate in the aggregation process, with the malicious intention to increase costs for the requesting node by forcing it to redeem its tickets one by one, but such behaviour would be to its detriment. Whereas previously the node could behave maliciously for free, assuming that it had its own tickets already aggregated, by not returning this

favor. A potential improvement in this area could be the introduction of the reputation system, where the nodes would keep track of other nodes' behaviour, namely, willingness to perform aggregation. The nodes could then choose which other nodes they will interact with, based on a given node's reputation.

CryptoEconomics 1 The execution of the proposed aggregation protocol is only feasible when a sufficient amount of traffic and, therefore, a sufficient amount of tickets are being generated. The sum value of the tickets must be large enough to justify aggregating and on-chain redemption. The precise threshold for this value is up to the individual nodes to decide, and will be highly influenced by the current state of the network, see subsection 2.5.3, paragraph „Ticket aggregation" on the HOPR network states and their impacts on winning probability and ticket price.

CryptoEconomics 2 The question of who should bear the on-chain redemption costs is also up for debate. The proposed protocol assumes that the node that is owed more will bear these costs. This setup was partially used as motivation for the nodes to perform the aggregation at all, but can easily be adjusted. Arguably, the fairest distribution of costs would be proportional to the amounts owed by the nodes, which would, however, slightly harm the motivation factor, but it would still be better than the original version of ticket aggregation.

Not everyone benefits One of the prerequisites, see subsection 5.1.3, for the ticket aggregation taking place is a substantial amount of previous traffic between participating nodes. The presence of this traffic is, however, not always the case, specifically when looking at the network entry nodes. This type of node does not relay any packets, and therefore does not earn any tickets; it only sends the packets that it creates and pays for itself. This imbalance causes the motivation aspect of the proposed aggregation to be irrelevant, as one side does not have any tickets to aggregate. Such a node can still perform the aggregation, however, it will be doing so only out of its own goodwill. The existence of such a situation shows that the improved properties of the proposed aggregation are not applicable everywhere.

Trust but verify In a scenario between two nodes, after the aggregation is agreed upon and a partial aggregated ticket is created by the payer, the receiver has the option to act maliciously by not redeeming the aggregated ticket. Instead, they could redeem the original non-aggregated tickets and then trigger an epoch advancement in the payment channel from themselves back to the payer. This advancement would invalidate all outstanding tickets from the payer, allowing the receiver to avoid paying their portion of the aggregated value. To defend against this exploit, the payer must ensure that the receiver actually redeems the aggregated ticket before considering the aggregation complete.

Usability I personally consider the simplicity of the proposed aggregation as its main advantage. As the proposed aggregation does not change any fundamental principle of the HOPR network, it can be easily integrated. If such a change were to occur in the HOPR network, it would be minor in scale but could have an immediate impact.

Chapter 8

Conclusion

In this work, we have designed and implemented an optimization of the incentivization layer of the HOPR incentivized mixnet. We have reviewed and studied existing incentivized mixing network solutions, including HOPR. We have studied cryptographic proving techniques compatible with blockchain with a focus on the latest state-of-the-art zero-knowledge proofs.

Using the gained knowledge, we attempted to integrate the zero-knowledge proofs into the design of the optimization. This, however, proved to be infeasible due to the properties of the zero-knowledge proofs not being a good fit for the HOPR's incentivization layer and due to some of the major flaws, which their possible integration would introduce.

Instead, an optimization of the HOPR's ticket aggregation process was proposed. The proposed adjustment aimed to optimize the smart contract cost and increase motivation to perform aggregation.

The testing achieved satisfactory results. The optimized version of the aggregation was able to demonstrate the intended functionality without reservation and cost improvements of approximately 50%. The motivation increase part can only be marked as improved upon rather than completely solved by our proposal. Nevertheless, the proposed optimization has been shown to offer an improvement of the HOPR network, and with its simplicity and ease of integration, it could be a valuable addition.

Some further opportunities for improvement of the proposed optimization include extending the testing scenarios, tackling its cryptoeconomic aspects, or making it applicable to all types of nodes.

The HOPR team does not plan to integrate the proposed optimization at the time due to the ticket price increase in version 3 of the HOPR network, which will be released in late Q2 of 2025, and subsequent lack of need for ticket aggregation. However, they recognize the idea behind the proposed optimization and will consider future integration if the need arises.

Bibliography

[1] Aad, I. Zero-Knowledge Proof. In: Mulder, V.; Mermoud, A.; Lenders, V. and Tellenbach, B., ed. *Trends in Data Protection and Encryption Technologies*. Cham: Springer Nature Switzerland, 2023, p. 25–30. ISBN 978-3-031-33386-6. Available at: https://doi.org/10.1007/978-3-031-33386-6_6.

[2] Academy, B. *ZK-STARKs*. 2025. Available at: https://academy.binance.com/en/glossary/zk-starks. Accessed: 2025-01-29.

[3] Androulaki, E.; Raykova, M.; Srivatsan, S.; Stavrou, A. and Bellovin, S. M. PAR: Payment for Anonymous Routing. In: Borisov, N. and Goldberg, I., ed. *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)*. Springer, July 2008, p. 219–236.

[4] Ben Sasson, E.; Bentov, I.; Horesh, Y. and Riabzev, M. *Scalable, transparent, and post-quantum secure computational integrity* Cryptology ePrint Archive, Paper 2018/046. 2018. Available at: https://eprint.iacr.org/2018/046.

[5] Benarroch, D. *Diving Into the SNARKs Setup Phase*. 2019. Available at: https://medium.com/qed-it/diving-into-the-snarks-setup-phase-b7660242a0d7. Accessed: 2025-01-29.

[6] Bitansky, N.; Canetti, R.; Chiesa, A. and Tromer, E. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. New York, NY, USA: Association for Computing Machinery, 2012, p. 326–349. ITCS '12. ISBN 9781450311151. Available at: https://doi.org/10.1145/2090236.2090263.

[7] Bootle, J.; Cerulli, A.; Chaidos, P.; Groth, J. and Petit, C. *Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting* Cryptology ePrint Archive, Paper 2016/263. 2016. Available at: https://eprint.iacr.org/2016/263.

[8] Buterin, V. *Quadratic Arithmetic Programs: From Zero to Hero*. 2016. Available at: https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649. Accessed: 2025-01-29.

[9] Bünz, B.; Bootle, J.; Boneh, D.; Poelstra, A.; Wuille, P. et al. *Bulletproofs: Short Proofs for Confidential Transactions and More* Cryptology ePrint Archive, Paper 2017/1066. 2017. Available at: https://eprint.iacr.org/2017/1066.

[10] Catalano, D.; Cramer, R.; Di Crescenzo, G.; Darmgård, I.; Pointcheval, D. et al. Multiparty computation, an introduction. *Contemporary cryptology*. Springer, 2005, p. 41–87.

[11] Chaum, D. L. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*. ACM New York, NY, USA, 1981, vol. 24, no. 2, p. 84–90.

[12] Chen, T.; Lu, H.; Kunpittaya, T. and Luo, A. *A Review of zk-SNARKs*. 2023. Available at: https://arxiv.org/abs/2202.06877.

[13] Costan, I. *Commitment Schemes*. 2020. Available at: https://medium.com/@icostan/commitment-schemes-8b523d48aa1e. Accessed: 2025-01-29.

[14] Danezis, G. and Goldberg, I. Sphinx: A Compact and Provably Secure Mix Format. In: *2009 30th IEEE Symposium on Security and Privacy (SP)*. Oakland, CA, USA: IEEE, 2009, p. 269–282. Available at: https://cypherpunks.ca/~iang/pubs/Sphinx_Oakland09.pdf.

[15] Diaz, C.; Halpin, H. and Kiayias, A. *The Nym Network: The Next Generation of Privacy Infrastructure*. Nym Technologies SA, February 2021. Available at: https://nym.com/nym-whitepaper.pdf.

[16] Diaz, C. and Preneel, B. Taxonomy Of Mixes And Dummy Traffic. *IFIP Int. Fed. Inf. Process.*, july 2004, vol. 148.

[17] Ducas, L.; Lepoint, T.; Lyubashevsky, V.; Schwabe, P.; Seiler, G. et al. *CRYSTALS – Dilithium: Digital Signatures from Module Lattices* Cryptology ePrint Archive, Paper 2017/633. 2017. Available at: https://eprint.iacr.org/2017/633.

[18] Foundation, E. *Proof-of-Stake (PoS)*. Ethereum Foundation, 2024. Available at: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/. Accessed: 2025-01-29.

[19] Gabizon, A.; Williamson, Z. J. and Ciobotaru, O. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge* Cryptology ePrint Archive, Paper 2019/953. 2019. Available at: https://eprint.iacr.org/2019/953.

[20] Gajland, P.; Janneck, J. and Kiltz, E. *A Closer Look at Falcon* Cryptology ePrint Archive, Paper 2024/1769. 2024. Available at: https://eprint.iacr.org/2024/1769.

[21] GeeksforGeeks. *Applications of Homomorphic Encryption in Blockchain*. 2024. Available at: https://www.geeksforgeeks.org/applications-of-homomorphic-encryption-in-blockchain/. Accessed: 2025-01-29.

[22] Geroni, D. *An Introduction to zkSNARKs*. 2021. Available at: https://101blockchains.com/zksnarks-introduction/. Accessed: 2025-01-29.

[23] Goldschlag, D.; Reed, M. and Syverson, P. Onion Routing. *Communications of the ACM*. ACM, February 1999, vol. 42, no. 2, p. 39–41. ISSN 0001-0782.

[24] Goldwasser, S.; Micali, S. and Rackoff, C. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, 1989, vol. 18, no. 1, p. 186–208. Available at: https://doi.org/10.1137/0218012.

[25] Groth, J. *On the Size of Pairing-based Non-interactive Arguments* Cryptology ePrint Archive, Paper 2016/260. 2016. Available at: https://eprint.iacr.org/2016/260.

[26] Gwaro, E. *What Is a MixNet and How Does It Work?* 2023. Available at: https://www.makeuseof.com/what-is-a-mixnet/. Accessed: 2025-03-26.

[27] Hrycyszyn, D. *A simple introduction to mixnets.* 2020. Available at: https://constructiveproof.com/posts/2020-02-17-a-simple-introduction-to-mixnets/. Accessed: 2025-05-19.

[28] "Johnny" Ngan, T.-W.; Dingledine, R. and Wallach, D. S. Building Incentives into Tor. In: Sion, R., ed. *Financial Cryptography and Data Security.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, p. 238–256. ISBN 978-3-642-14577-3.

[29] Karaivanov, D. *Proof of Work Explained in Simple Terms.* The Chain Bulletin, 27. October 2019. Available at: https://chainbulletin.com/proof-of-work-explained-in-simple-terms. Accessed: 2025-01-29.

[30] Kiktenko, E. O.; Bulychev, A. A.; Karagodin, P. A.; Pozhar, N. O.; Anufriev, M. N. et al. SPHINCS+ post-quantum digital signature scheme with Streebog hash function. In: *FIFTH INTERNATIONAL CONFERENCE ON QUANTUM TECHNOLOGIES (ICQT-2019).* AIP Publishing, 2020, vol. 2241, p. 020014. ISSN 0094-243X. Available at: http://dx.doi.org/10.1063/5.0011441.

[31] Labs, C. *Homomorphic Encryption.* 2024. Available at: https://chain.link/education-hub/homomorphic-encryption. Accessed: 2025-01-29.

[32] Labs, C. *Understanding the Difference Between zk-SNARKs and zk-STARKS.* 2025. Available at: https://chain.link/education-hub/zk-snarks-vs-zk-starks. Accessed: 2025-01-29.

[33] Nym. *Step-by-step guide to the Anonymous Mode.* 2024. Available at: https://nym.com/blog/anonymous-mode-in-nymvpn. Accessed: 2025-05-19.

[34] Nym. *What is a mixnet? Unparalleled online privacy with a VPN.* 2024. Available at: https://nym.com/blog/what-is-a-mixnet. Accessed: 2025-01-29.

[35] Oude Roelink, B.; El Hajj, M. and Sarmah, D. Systematic review: Comparing zk-SNARK, zk-STARK, and bulletproof protocols for privacy-preserving authentication. *SECURITY AND PRIVACY*, 2024, vol. 7, no. 5, p. e401. Available at: https://onlinelibrary.wiley.com/doi/abs/10.1002/spy2.401.

[36] Oude Roelink, B. *Benchmarking the zk-SNARK, zk-STARK, and Bulletproof Non-Interactive Zero-Knowledge Proof Protocols in an Equivalent Practical Application.* July 2024. Available at: http://essay.utwente.nl/100612/.

[37] Parno, B.; Gentry, C.; Howell, J. and Raykova, M. *Pinocchio: Nearly Practical Verifiable Computation* Cryptology ePrint Archive, Paper 2013/279. 2013. Available at: https://eprint.iacr.org/2013/279.

[38] Petkus, M. *Why and How zk-SNARK Works*. 2019. Available at: https://arxiv.org/abs/1906.07221.

[39] Responsible, D. I. Berkeley Center for. *ZK Learning*. 2023. Available at: https://rdi.berkeley.edu/zk-learning/. Accessed: 2025-01-29.

[40] Robert Kiel, D. S. B. and Yu, Q. *HOPR - a Decentralized and Metadata-Private Messaging Protocol with Incentives*. HOPR Association, September 2024. Available at: https://github.com/hoprnet/hoprnet/blob/master/docs/yellowpaper/yellowpaper.pdf.

[41] Shoemaker, P. *Zero Knowledge Proofs*. 2025. Available at: https://www.identity.com/zero-knowledge-proofs/. Accessed: 2025-01-29.

[42] Szepieniec, A. *Anatomy of a STARK*. October 2021. Available at: https://neptune.cash/learn/stark-anatomy/. Accessed: 2025-04-29.

[43] Yadav, S. *Merkle Proofs Explained*. 2024. Available at: https://medium.com/@swastika0015/merkle-proofs-explained-208a72971a50. Accessed: 2025-01-29.