DEPARTMENT OF
COMPUTER SCIENCE

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

RICARDO DELGADO ROMÃO

Bachelor in Computer Science and Engineering

PRIVACY-AWARE MULTICAST ON THE INTERNET

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
September, 2024

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

PRIVACY-AWARE MULTICAST ON THE INTERNET

RICARDO DELGADO ROMÃO

Bachelor in Computer Science and Engineering

Adviser: Kevin Gallagher
Assistant Professor, NOVA University Lisbon

Co-adviser: João Leitão
Associate Professor, NOVA University Lisbon

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
September, 2024

Privacy-Aware Multicast on the Internet

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.0.3) [73].

To family and friends.

Acknowledgements

The conclusion of this thesis would not be possible without the help of so many people. I would like to thank my advisors, Professor Kevin Gallagher and Professor João Leitão for their incredible guidance and this incredible learning experience. I would also like to thank my family and friends for their support during this hard-working journey. I would also like to thank the NOVA School of Science and Technology and all the professors that I encountered in my path for providing me with the opportunity to pursue this research and for helping me building the foundations to complete such a difficult work. It was a pleasure to work with all of you, thank you. Last but not least, I would like to share, that after so many sweat and tears, if there is one thing that I have learned is that the journey is as important as the destination. I can say that I learned so much during the process of writing and designing this thesis and in the future (PhD who knows) I will take what I learned in this experience and I will apply it to my future work. It is so rewarding to be proud of something you built, do not buy it complete, do it yourself, that is my main message here. I can only say thank you for everything and to everyone than contribute to make this happen in one way or the other. Without you, this would not be possible. Thank you.

”

“The more I learn, the more I realize how much I don’t know.”

— **Einstein**, Somewhere in a book or speech
(Theoretical physicist)

Abstract

The Internet, and computer networks in general, are essential in modern societies, having a direct impact on the everyday lives of citizens across many domains, from interaction with government services, health, work, and even cultural purposes. A rising concern about the use of the Internet is related to privacy and censorship resistance, with multiple works exploring mechanisms that ensure the privacy of both users and service providers, potentially considering powerful antagonists such as entire countries.

The most adopted solution for privacy on the Internet is the Tor anonymity system which takes advantage of Onion Encryption and Multi-Hop circuits to ensure the ability of users to privately access websites and web services. This privacy guarantees also extends to web service operators who also wish to remain anonymous. However, existing solutions are mostly restricted to point-to-point private interactions and lack effective mechanisms to support point-to-multipoint private interactions - such as multicast or publish/subscribe - in a decentralized way, a functionality that can be relevant for whistle-blowers or to coordinate activities within social activist groups.

In this thesis, we explore different solutions to help us understand the challenges and opportunities for this type of communication. We propose, design, and implement a prototype solution that supports private point-to-multipoint communication, having similar privacy guarantees and latency as the Tor network - called Anonycast. We evaluate Anonycast in different network conditions and analyse the performance through different configurations. This thesis is part of the research efforts of the TaRDIS European Project.

Keywords: Multicast Communication, Tor, Privacy and Anonymity, Cybersecurity

Resumo

A Internet, e as redes informáticas em geral, são essenciais nas sociedades modernas, tendo um impacto direto na vida quotidiana dos cidadãos em muitos domínios, desde a interacção com os serviços governamentais, a saúde, o trabalho e até fins culturais. Uma preocupação crescente sobre o uso da Internet está relacionada com a privacidade e a resistência à censura, com vários trabalhos a explorar mecanismos que garantam a privacidade tanto dos utilizadores como dos prestadores de serviços, considerando potenciais antagonistas poderosos como países inteiros.

A solução mais adotada para a privacidade na Internet é o sistema de anonimato Tor que aproveita os circuitos Onion Encryption e Multi-Hop para garantir a capacidade dos utilizadores acederem de forma privada sites e serviços web. Estas garantias de privacidade estendem-se também aos operadores de serviços web que também pretendam permanecer anónimos. No entanto, as soluções existentes estão principalmente restritas a interações privadas ponto-a-ponto e carecem de mecanismos eficazes para apoiar interações privadas ponto-a-multiponto - como multicast ou publicação/subscrição - de forma descentralizada, uma funcionalidade que pode ser relevante para denunciantes ou para coordenar atividades dentro de grupos de ativistas sociais.

Nesta tese, exploramos diferentes soluções para nos ajudar a compreender os desafios e as oportunidades deste tipo de comunicação. Propomos, concebemos e implementamos uma solução protótipo que suporta comunicação privada ponto-a-multiponto, tendo garantias de privacidade e latência similares à rede Tor - denominada Anonycast. Avaliámos o Anonycast em diferentes condições de rede e o seu desempenho através de diferentes configurações. Esta tese faz parte dos esforços de investigação do Projeto Europeu TaRDIS.

Palavras-chave: Comunicação Multicast, Tor, Privacidade e Anonimato, Cibersegurança

Contents

List of Figures

List of Tables

Acronyms

ABE Attribute-Based Encryption *(p. 13)*

BGW Boneh-Gentry-Waters *(p. 13)*
bLSAG Back's Linkable Spontaneous Anonymous Group *(pp. 11, 12)*

CS Cipher Sequence *(p. 21)*
CSPRNG Cryptographically Secure Pseudorandom Number Generator *(p. 7)*

DDoS Distributed Denial of Service *(p. 19)*
DEK Data Encryption Key *(p. 23)*
DEP Dual Encryption Protocol *(p. 23)*
DoS Denial of Service *(p. 19)*

G Cyclic Group *(p. 22)*
GC Group Controller *(p. 23)*
GDH Group Diffie-Hellman *(p. 22)*
GSA Group Security Agent *(p. 22)*

IBBE Identity-Based Broadcast Encryption *(p. 13)*
IP Introduction Point *(pp. 16, 17)*

KDC Key Distribution Center *(pp. 20, 21)*
KEK Key Encryption Key *(pp. 20, 21, 23)*

LCG Linear Congruential Generator *(p. 7)*
LSAG Linkable Spontaneous Anonymous Group *(p. 11)*

PGP Pretty Good Privacy *(p. 24)*
PKBE Public Key Broadcast Encryption *(p. 13)*

PKI Public Key Infrastructure *(p. 13)*
PRNG Pseudorandom Number Generator *(pp. 6, 7)*
Pub/Sub Publish/Subscribe *(p. 5)*

RP Rendezvous Point *(pp. 6, 17)*
RSA Rivest-Shamir-Adleman *(pp. 10, 14, 17)*

SD Subset Difference *(pp. 12, 13)*
SGM Subgroup Manager *(p. 23)*
SSM Source-Specific Multicast *(pp. 5, 6)*

TRNG True Random Number Generator *(pp. 6, 7)*

1 Introduction

The integration of the Internet and computer networks in our society has become ubiquitous, being present in many facets of everyday life, including in government services, health, work, and culture. However, this raises concerns regarding privacy aspects and censorship resistance, namely concerns related to safeguarding these individual rights in the face of potential adversaries at the national level.

The high need for the development of the Internet has led to little supported growth in terms of privacy, giving rise to several solutions that do not take the privacy of users into account. More recently, some studies and deployed systems emerged to improve the lack of anonymity on the Internet. Namely, the most used solution for this purpose is the Tor system [42], which is based on techniques such as onion routing [91] and multi-hop circuit establishment [42]. This system allows users to anonymously access websites and services while also preserving the anonymity of web service operators if they wish so. Despite the effectiveness of this solution, its focus lies predominantly on facilitating client-server interactions; this leaves ground to explore other solutions in this area.

Regarding anonymous communication systems, other systems were developed [45] before Tor , such as Mix-Net [16], Babel [58], and Mixminion [37]; however, similar to Tor, these systems only support the unicast communication model, where a message is sent from one publisher to one receiver. Notably, there is a gap in solutions capable of supporting private point-to-multipoint interactions, such as publish/subscribe, in a decentralized manner ensuring both privacy and censorship resistance. The existence of this concept is extremely useful in emergencies, as well as in the military, politics, journalism, health, and other areas where these properties are essential to protect users.

Let's consider the case of a journalist who lives in a country where freedom of expression is not allowed. By wanting to share the situation experienced in their country, the journalist is risking their life. Without protection, such a journalist could receive the death penalty, but with an anonymous multicast system, they would be able to disseminate their information to multiple parties simultaneously without fear of retribution. Tor fits this use case if the journalist publishes the information to one source, but it does not allow the journalist to publish the information to multiple sources and therefore, have more

guarantees that the information will reach a wide audience and have an impact.

1.1 Problem Statement and Contributions

With our work, we strive to answer the following research question: *Can we design and implement a decentralized solution that allows secure and private multicast communication?* We answer this question by developing a solution that allows a publisher to send a message to multiple receivers revealing as little information as needed. This solution should allow a receiver to receive a message from a publisher without revealing their identity. More specifically, the contributions of this research are:

- The design, implementation, and evaluation of **Anonycast**, a decentralized system that supports anonymous multicast communication, while being private and secure for both publishers and receivers.
- The development of a protocol that orchestrates all the elements needed to achieve a secure and private multicast communication channel.

1.2 Objective

The objective of this thesis was to design and implement a prototype of a decentralized solution that addresses current limitations in the area of private multicast communication.

As we discuss further ahead on Chapter 3, there are several techniques used to achieve privacy in the context of unicast communication [42, 101, 86, 70, 5, 75, 40] that do not apply to multicast. Though other multicast solutions exist, they aim to solve different problems from ours [96, 65, 66, 48, 64, 90, 63, 43, 17, 79, 78]. The objective of this thesis is to create a solution that provides privacy to both the publishers and receivers of messages and simultaneously fit the multicast communication model. This solution can be used in application domains related to journalism, military forces, health, and natural disaster recovery, among others.

To understand the requirements of our solution we should understand the concept of deaddrops; these are servers similar to mailboxes, where the publishers can publish their messages and receivers, on the other hand, can retrieve them.

1. Anonycast **MUST**[12] allow a publisher/receiver to publish/retrieve a message(s) anonymously. The identity of both the publisher and the receivers must remain concealed throughout the communication process.
2. Anonycast **MUST NOT**[12] disclose the IP addresses of publishers, receivers, or deaddrops to any other publisher, receiver or deaddrop during the communication process. This privacy measure is crucial for encouraging widespread adoption of our system by providing robust anonymity for users and their activities.

3. Anonycast **MUST** verify the integrity of all messages to prevent unauthorized tampering.

4. Anonycast **SHOULD**[12] implement defences against denial of service attacks, ensuring high availability.

5. Anonycast **MUST NOT** be vulnerable to replay attacks.

6. A publisher or receiver **MUST** be able to ensure that it is communicating with the intended deaddrop(s).

7. A receiver of a message **MUST** be able to ensure that the message was sent by an allowed publisher.

8. Only allowed receivers **MUST** be able to observe the content of the messages.

9. Anonycast **MUST** be scalable to support a large number of users and messages.

10. Anonycast **MUST** provide low latency, similar to the current privacy unicast solutions.

11. Both receivers and deaddrops **MUST** validate the messages independently of previous validations.

These are the requirements that we aim to fulfil with our solution. We discuss in detail how we achieve these different requirements in Chapter 5.

1.3 Research Context

All the work presented here was partially motivated by the Trustworthy and Resilient Decentralized Intelligence for Edge Systems (TaRDIS) [100] European project aimed at "supporting the correct and efficient development of applications for swarms and decentralized distributed systems, by combining a novel programming paradigm with a toolbox for supporting the development and executing of applications".

1.4 Thesis Structure

The remainder of this document is organized as follows:

- Chapter 2 presents the necessary background for understanding the mechanisms central to our solution. It includes an overview of the relevant theories, technologies, and methodologies, along with a discussion of related work to contextualize our research.

- Chapter 3 analyzes existing solutions in detail, examining their foundations, strengths, and limitations. This chapter also provides the groundwork for understanding the specific challenges that our research seeks to address.
- Chapter 4 introduces and details the design of Anonycast. We discuss the rationale behind our approach, addressing the challenges identified in related work and explaining the core concepts and methodologies that guide our solution.
- Chapter 5 covers the implementation details of our solution, including the technologies, tools, and libraries utilized. It also presents an extensive evaluation of our solution, focusing on scalability, performance, security, and privacy.
- Chapter 6 summarizes our thesis and suggests potential directions for future work.

2

Background

In this chapter, we present the necessary background information to understand the concepts and technologies used in the development of our prototype solution.

2.1 Multicast Communication

There are four different types of network communication: Unicast, Multicast, Anycast and Broadcast. Unicast is the most common type of network communication, where a single sender communicates with a single receiver, Broadcast communication is a one-to-all communication model, where a single sender communicates with all receivers, Anycast communication is a one-to-one communication model, where a single sender communicates with any one of the elements of the receiver group. Finally, Multicast communication is a one-to-many communication model, where a single sender communicates with a group of receivers. This model is particularly useful for applications that need to deliver the same data to multiple recipients simultaneously, such as video streaming, online gaming, and real-time data feeds.

There are different implementations of Multicast communication, such as Group Addressing [38], Publish/Subscribe Model [49], Source-Specific Multicast (SSM) [61], Multicast with Rendezvous Points (RP) [50], Application-Layer Multicast [18], and Multicast Trees [2]. In Group Addressing, a multicast group is represented by a unique address. All the devices that want to receive the data sent to that group, listen in the address of the multicast group. The sender transmits the data to the multicast address, and the network infrastructure ensures that the data is delivered to all members of the group. This approach is widely used in IP Multicast.

In Publish/Subscribe (Pub/Sub) model, publishers send messages to a topic or channel without needing to know the subscribers. Subscribers express interest in one or more topics and receive messages published on those topics. The Pub/Sub model is highly decoupled, meaning that publishers and subscribers do not need to know about the existence of each other, making it scalable and flexible. This model is used in systems like MQTT [3], and Apache Kafka [69].

In SSM, receivers join a multicast group but also specify the particular source from which they want to receive data. This approach allows for more control and security, as it ensures that only data from the specified source is delivered to the receivers. SSM reduces the risk of receiving unwanted or malicious data and is particularly useful in scenarios where multiple sources might be broadcasting to the same group address, such as in content distribution or live streaming events.

In some multicast networks, a Rendezvous Point (RP) is used to facilitate the joining of receivers to a multicast group. The RP acts as an intermediary where senders initially send their data before it is forwarded to the receivers. This model is useful in larger networks to efficiently manage multicast traffic and ensure that it reaches all interested receivers without overwhelming the network.

In Application-Layer Multicast, multicast groups are formed and managed within the application, and data is forwarded between members of the group using unicast links. This model is beneficial when native multicast support is unavailable or when additional flexibility is needed, such as in peer-to-peer streaming or collaborative applications; systems like BitTorrent use this type of multicast.

In Multicast Trees, the sender transmits the data through a hierarchical network of nodes to reach all the receivers. The most common type of multicast tree is the shared tree, where all receivers of a group use the same tree, rooted at a central node. Alternatively, source-specific trees can be used, where each source has its tree, providing more efficient routing but also increasing the complexity of the solution.

2.2 Randomness Generators

Randomness generators [7] produce sequences of numbers that exhibit no discernible patterns, thereby ensuring unpredictability which is relevant for various applications. Randomness generators are broadly categorized into True Random Number Generators (TRNGs) and Pseudorandom Number Generators (PRNGs). TRNGs derive their randomness from unpredictable physical phenomena. This category includes generators that utilize quantum phenomena, such as radioactive decay or photon polarization, which are fundamentally random and cannot be predicted. Additionally, electronic noise, specifically thermal noise resulting from the random motion of electrons in a conductor, and atmospheric noise, are other possible sources for TRNGs. The primary advantage of TRNGs lies in their unpredictability, rendering them appropriate for security applications where randomness must not be reproducible. However, they tend to be slower and more complex to implement in a digital environment compared to their pseudorandom counterparts.

On the other hand, PRNGs employ mathematical algorithms to generate number sequences that simulate the properties of truly random sequences. These generators need an initial value, known as a seed, and will consistently produce the same sequence of numbers given the same seed. Despite their deterministic nature, PRNGs are designed

to be unpredictable without knowledge of the seed. Among the examples of PRNGs are Linear Congruential Generators (LCGs) [68], which are among the oldest and simplest PRNGs techniques, and the Mersenne Twister [74]. PRNGs offer advantages in terms of speed and ease of implementation, making them suitable for applications where the true unpredictability of TRNGs is not a requirement. However, for cryptographic applications, only PRNGs are designed to be cryptographically secure, known as Cryptographically Secure Pseudorandom Number Generators (CSPRNGs), and are deemed safe. CSPRNGs are specifically designed to withstand attacks aimed at predicting future output based on previously generated numbers.

The selection between TRNGs and PRNGs depends on the specific needs of the application, including considerations of security, speed, and practicality. For cryptographic keys or secure token generation, TRNGs or CSPRNGs are preferred to ensure the highest level of unpredictability. Conversely, for simulations or games where speed is a critical factor, PRNGs often suffice. Another important factor is that TRNGs requires specialized hardware to capture and process the physical phenomena they rely on, whereas PRNGs can be implemented on any general-purpose computing device. Within the context of Randomness Generators, there is a specific type that is of interest, the Distributed Randomness Generators; unlike traditional approaches, which rely on a single source or algorithm, these systems use the power of a network of independent participants to generate randomness that is resistant to manipulation and publicly verifiable.

2.2.1 Drand: A Distributed Randomness Beacon

Drand [47] is a distributed randomness generator that uses a decentralized method for producing random numbers; this system is designed to create unbiased and unpredictable random numbers through a collaborative network of nodes. Each node in the network contributes to the generation process, ensuring that no single participant can control or predict the outcome. This decentralized structure enhances the security and trustworthiness of the randomness produced.

One of the distinctive features of Drand is its commitment to public verifiability. Drand generates values publicly available, allowing anyone to independently verify their authenticity. Drand's randomness is also characterized by its unpredictability. Even the nodes involved in generating the random numbers cannot anticipate the outcome before it is collectively revealed by the network. This unpredictability is important in applications where the integrity of the random values must be maintained against potential manipulation.

Because it is publicly available, the randomness generated by Drand is a versatile resource that can be integrated into various systems without requiring specialized hardware.

2.3 Cryptopuzzle

To understand the concept of Cryptopuzzle, we need to understand the concept of Hashing [62]. A Hash function $H : \{0,1\}^+ \rightarrow \{0,1\}^n$ is a deterministic function that receives an arbitrary length bit string and outputs a fixed n-length bit string. Independently of the input size, the output generated has always the same size. The Hashing algorithm determines how the output is obtained; the output is called a hash. A Hash needs to have the following properties to be considered secure:

- **Pre-image Resistance**: It's infeasible to reverse the hash to find the original input.
- **Second Pre-image Resistance**: It's infeasible to find a different input with the same hash as a given input.
- **Collision Resistance**: It's infeasible to find two distinct inputs that produce the same hash value.
- **Deterministic**: The same input always produces the same hash output.
- **Avalanche Effect**: Small input changes produce significantly different hashes.

The first property makes it infeasible to find the original message through the hash value avoiding exposing the content of the original message. The second property ensures that is not possible to create a different message with the same hash value, avoiding the possibility of a malicious actor changing the content of the message without being detected. When a sender sends a message, by sending the hash of the message along with the message, the receiver can verify if the message was tampered with by hashing the message and comparing the hash with the received hash. If the hashes are different, the message was tampered with. The third property ensures that it should be very improbable for the hashing algorithm to generate the same hash for different inputs. The fourth property is the base for hash verification otherwise, we could not verify the integrity of the message because the output was not stable. The last property makes reverse engineering the hash more difficult, as small changes in the input produce significantly different hashes. SHA-256 [97] is an example of a secure hashing algorithm [83]; it generates a 256-bit hash.

A more familiar term is the **Puzzle** term. This is something, such as a game, toy, or problem, that requires ingenuity and often persistence in solving or assembling [1]. A Cryptopuzzle [80] is a term that combines both the Hashing and Puzzle concepts. A Cryptopuzzle is a cryptographic puzzle that requires a significant amount of computational effort to solve it. It can be used to prevent spam and denial-of-service attacks by forcing the sender to perform a computationally expensive task before sending a message. An example of a cryptopuzzle would be starting with an initial value $solution$, a byte array $v1$ and a $difficulty$. Continuously compute the SHA-256 hash of the concatenation of $v1$ and the current $solution$. Check if the resulting hash has $difficulty$ number of leading zeros. If the hash meets the criteria, return the current solution as it solves the

puzzle. If not, increment the solution value and repeat the hashing process. Continue this until you find a solution that satisfies the difficulty requirement. Algorithm **??** shows the pseudocode of the Cryptopuzzle solving process.

Algorithm 1 Solve Cryptopuzzle

Require: $v1$ (input byte array), $difficulty$ (difficulty level)

Initialize $nonce \leftarrow 0$
Initialize $solved \leftarrow$ False
while not $solved$ **do**
    Initialize hasher with SHA256
    Update hasher with $v1$
    Update hasher with bytes of $nonce$
    $result \leftarrow$ finalized hasher output
    if all bits in $result$ up to $difficulty$ are 0 **then**
        $solved \leftarrow$ True
    end if
    Increment $nonce$ by 1
end while
return $nonce$

2.4 Encryption

Another concept used in our solution is encryption [62], a cornerstone of modern information security. Encryption is the process of encoding data using a key in such a way that only authorized parties who possess the decryption key can access the original content. Encryption plays a crucial role in ensuring the confidentiality and integrity of data in transit and at rest. As our society became increasingly digital, the importance of ensuring that the protection of digital data was as robust as that of the most advanced physical data also increased. By referring to protection, we refer to confidentiality, which guarantees that only authorized users can access the actual content of the data.

There are two main types of encryption: symmetric and asymmetric encryption [62]. In symmetric encryption, the same key is used for both encryption and decryption processes, while in asymmetric encryption, different keys are used for encryption and decryption. In asymmetric encryption, the public key, as the name suggests, is public and can be shared with anyone, while the private key is kept secret. In some algorithms, the content encrypted with the public key can be decrypted with the private key and vice versa. One of the most popular encryption algorithms, allowing the encryption of the content with both the public and the private key of the sender is the Rivest-Shamir-Adleman (RSA) algorithm [92]; it is based on the difficulty of factoring large prime numbers. This algorithm has the property of allowing the encryption of a message with either the public key or the private key, supporting both encryption for confidentiality and digital signatures for authenticity. However, some encryption algorithms such as the Ed25519 algorithm [13], only allow encrypting with the private key; these are used for digital signature validation.

2.4.1 Cryptographic Signatures

In this section, we explore the concept of cryptographic signatures [62] that ensure the non-repudiation of data; non-repudiation is the ability to prove that a specific party participated in a transaction or communication and cannot deny it. Digital signatures [82] are a cryptographic mechanism used to verify the authenticity and integrity of a message, software, or digital document. In the context of asymmetric encryption, the sender creates a digital signature by encrypting a hash of the message with their private key. The receiver can then verify the signature by decrypting it with the sender's public key and comparing it to a freshly computed hash of the received message. If the two hashes match, it proves that no one altered the message and confirms that the owner of the private key was the sender of the message, effectively acting as a digital equivalent of a handwritten signature.

More complex signature schemes have been developed, such as ring signatures and group signatures, which allow multiple parties to sign a message. In the case of group signatures [94], a group manager coordinates the group and its keys. In case of ring signatures [94], the signer can sign the message without the need for coordination with other group members, ensuring that the signer remains anonymous within the group.

2.4.1.1 Ring Signatures

Ring signatures, introduced by Rivest, Shamir, and Tauman in 2001 [93], are a cryptographic method that allows a member of a group to sign a message anonymously on behalf of the group. In a ring signature scheme, the actual signer forms a "ring" of possible signers by selecting a set of public keys, including their own. The signature can be verified using the set of public keys that compose the ring. Additionally, it is computationally infeasible to determine which member's private key was used providing strong anonymity and unlinkability for the signer. A ring signature scheme is composed of three main algorithms [15]:

1. KeyGen, which generates a keypair
2. RSign, which produces a ring signature using a set of public keys, the signer's index, the signer's private key, and the message itself
3. RVerify, which validates the signature using the set of public keys, the message, and the message signature

The core properties expected of ring signatures include correctness, unforgettability, and anonymity. Correctness ensures that the verifier accepts all validly generated ring signatures, Unforgettability guarantees that adversaries cannot produce a ring signature for a message and ring unless they possess the private key of one of the ring members and lastly, Anonymity ensures that it is computationally infeasible for adversaries to identify the actual signer within the group. In 2004 Liu et al. [72] introduced the concept of Linkable Linkable Spontaneous Anonymous Group (LSAG) signatures, which builds on top of the work from Rivest et al. [93]. LSAG signatures are characterized by three properties: Anonymity, Linkability and Spontaneity. Anonymity ensures that the actual signer remains anonymous within the group, Linkability ensures that if a private key signs two different messages, the messages become linked and Spontaneity ensures that there is no group secret, therefore no group manager or group secret sharing setup is needed. Linkability is especially important in scenarios where it's necessary to track the usage of the same private key across different transactions or messages; it also helps identify whether a specific key has been used in multiple instances. Audit purposes and detecting potential fraudulent activities are some of the use cases where these features are extremely valuable.

Back's Linkable Spontaneous Anonymous Group (bLSAG) A more advanced version of the LSAG algorithm is the bLSAG signatures [84]. The bLSAG signature scheme is an enhanced version of the LSAG algorithm; one difference between the two algorithms is that in the LSAG algorithm, the owner of a private key could only produce one anonymous signature per ring member per ring. bLSAG addresses this limitation by allowing for multiple anonymous signatures for a single ring member within the same ring. Moreover,

signature anonymity is independent of the ring's decoy members. A decoy member does not represent an actual potential signer within the ring; instead, it functions as a placeholder, incorporated solely to enhance the number of potential signers, thereby increasing the complexity and ambiguity of the signing process. The other differences are in the main properties of the bLSAG algorithm; the bLSAG algorithm is built on top of the following properties:

1. **Signer Ambiguity**: An observer should be able to determine the signer must be a member of the ring, but not which member.

2. **Linkability**: If a private key is used to sign two different messages then the messages become linked.

3. **Unforgeability**: It is computationally infeasible for an attacker to create a valid signature without knowledge of the corresponding private key. This ensures that only the legitimate owner of the private key can produce a valid signature within the ring, preventing unauthorized parties from impersonating the signer.

2.4.2 Broadcast Encryption

Despite the two main types of encryption, symmetric and asymmetric, other types of encryption are used in different scenarios, such as Broadcast Encryption. Broadcast encryption [55] is a cryptographic technique that allows to securely transmit data to multiple recipients. It ensures that only allowed receivers can decrypt and access the transmitted content. This method is particularly beneficial in scenarios where content needs to be distributed to a large audience, such as digital television, online streaming services, or software updates while preventing unauthorized access. Broadcast encryption schemes can be based on different types of cryptography.

Tree-Based Broadcast Encryption Tree-based Broadcast Encryption [81] is a cryptographic technique that organizes users in a hierarchical tree structure to manage keys efficiently. In this scheme, the broadcaster maintains a key tree where each node represents a key, and each user is assigned to a leaf node. When content needs to be securely distributed, the broadcaster encrypts the message using a minimal set of keys corresponding to the internal nodes of the tree. This approach reduces the number of encryptions required and optimizes storage and computational costs, making it particularly effective for large groups. Tree-based schemes are widely used in scenarios where scalability and efficiency are critical, such as digital content distribution and secure group communication.

Subset Difference (SD) Method The SD Method [81] is a broadcast encryption scheme that enhances tree-based approaches by defining subsets of users who share certain keys based on their position in the tree. This method is designed to efficiently handle user revocation by allowing the exclusion of unauthorized users without needing to rekey the entire

group. In the SD Method, the broadcaster only needs to encrypt the content with keys that cover the intended recipients, reducing the overall computational overhead. This scheme is particularly useful in dynamic environments where user memberships frequently change, such as subscription-based services or secure multicast communications.

Attribute-Based Encryption (ABE) ABE [57] is a flexible broadcast encryption technique where access to the encrypted content is determined by a set of attributes rather than individual user identities. In ABE, the broadcaster defines an access policy based on attributes, and users are granted keys corresponding to their attributes. A user can decrypt the content only if their attributes satisfy the access policy. ABE is particularly useful in scenarios where access control needs to be enforced based on user roles or characteristics, such as enterprise data sharing, cloud storage security, or healthcare information systems.

Identity-Based Broadcast Encryption (IBBE) IBBE [39] simplifies key management by using user identities, such as email addresses, as public keys. This scheme allows the broadcaster to securely transmit content to multiple recipients without requiring a traditional Public Key Infrastructure (PKI). In IBBE, the broadcaster encrypts the content for a set of users based on their identities, and each user can decrypt the content using their private key, which is derived from their identity. IBBE is particularly beneficial in environments where user identities are easily managed and the overhead of maintaining a PKI is not desirable.

Boneh-Gentry-Waters (BGW) Scheme The BGW Scheme [11] is a method for securely sharing information with a large group of people, where you can easily remove or revoke access for specific users when needed. It uses a special mathematical tool called bilinear pairings [76] to keep the size of the encrypted messages and the keys used to encrypt them small and manageable. This makes it practical and efficient for distributing content or updates to many users, such as in digital content services, software updates, or cloud-based applications. The BGW Scheme is well-suited for situations where the audience is large and may change over time.

Public Key Broadcast Encryption (PKBE) PKBE schemes [10] use public key cryptography to enable secure content distribution to a dynamic group of receivers. In PKBE, the broadcaster encrypts the content using a public key that corresponds to the group of intended recipients; each recipient can then use their private key to decrypt the content.

An example of a Broadcast Encryption algorithm that assumes that public keys of allowed receivers are known. Operate as follows: the broadcaster generates a symmetric encryption key that is used to encrypt the content. Remember that this key is symmetric, meaning that this key is capable of encrypting and decrypting the unencrypted/encrypted content. The broadcaster now encrypts the symmetric key with the public key of each

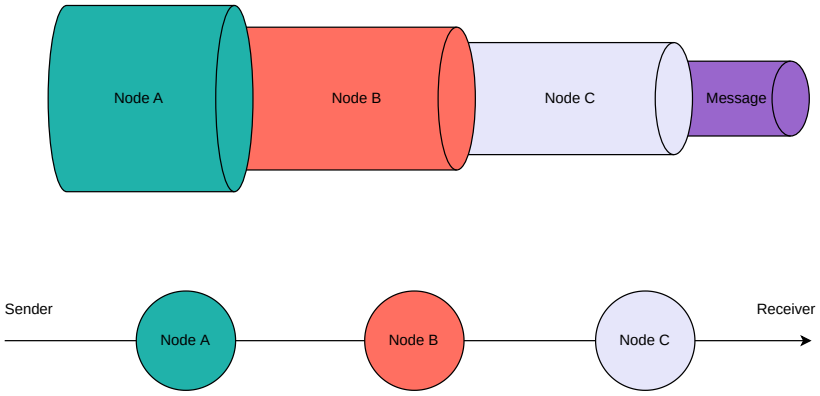

Figure 2.1: Onion Routing example

receiver and sends the encrypted symmetric key along with the encrypted content. The receiver decrypts the symmetric key with his private key if is one of the allowed receivers and then decrypts the content with the symmetric key. The Algorithm 2, shows the pseudocode of the Broadcast Encryption using public keys. This algorithm uses the RSA algorithm; the symmetric key is encrypted with the public key of the receiver and decrypted with the private key of the receiver.

Algorithm 2 Broadcast Encryption Using Public Keys

Input: Content C, Set of allowed receivers R, Public keys PK_r for each $r \in R$
Output: Encrypted content C_{enc}, Encrypted symmetric keys for each receiver
Generate a symmetric encryption key K_{sym}
for each receiver $r \in R$ **do**
    Encrypt K_{sym} with r's public key PK_r to get $K^r_{sym,enc}$
    Send $K^r_{sym,enc}$ to receiver r
end for
Encrypt content C with K_{sym} to get C_{enc}
Broadcast C_{enc} to all receivers

2.5 Onion Routing

Onion Routing [42] is a technique that uses encryption to achieve confidentiality even in cases, where the data is transmitted through intermediaries which can not be fully trusted; we can only trust that the intermediaries will deliver the messages but not that they will not read or tamper with the messages. It operates based on the principle of routing data through a series of intermediate servers, called Nodes. This data is encrypted by different keys, forming different layers of encryption; an analogy is often made with an Onion, hence the name.

Let us consider a communication path where the data is transmitted from a sender S to a receiver R through three intermediate nodes: $OR1$, $OR2$, and $OR3$. The message to

be sent is denoted by M, and each party involved has its own encryption key, represented as K_i.

After an initial exchange of information, the sender S obtains the necessary keys of all parties involved in the path. Thereafter, the sender encrypts the message M in a layered manner, starting with the key of the final receiver K_R, followed by the keys of the intermediate nodes in reverse order: K_{OR3}, K_{OR2}, and finally K_{OR1}.

Figure 2.5 illustrates the Onion Routing process; as the data packets travel through the network from S to R, each node decrypts one layer, revealing the address of the next node. This process continues until the data reaches its final destination R, where the last layer is removed. This method ensures that the true origin of the message remains concealed throughout its path, thus safeguarding the user's privacy from potential observers. The most well-known system that incorporates this technique is the Tor network [42].

2.5.1 Tor: The Second-Generation Onion Router

Tor [42] is the most popular system for secure and anonymous digital communication. In this section, we provide an overview of the Tor system, focusing on its threat model, structure, and key features including Onion Services [88].

2.5.1.1 Threat Model

The threat model for Tor considers a practical adversary with the ability to observe, manipulate and compromise a fraction of the network traffic, including operating its relays (nodes). In low-latency systems with layered encryption, the adversary's goal is typically to observe both the sender and the receiver, potentially confirming connections through distinct timing and volume patterns. Tor focuses on preventing traffic analysis attacks [42], where the adversary exploits traffic patterns to identify vulnerable network points for attack. The adversary can attempt to link a sender (say Alice) to communication partners, build a behavioural profile of Alice, and execute passive attacks by observing the edges of the network. Active attacks [42] involve compromising relays or keys, replaying traffic, selectively denying service to trusted relays, or introducing detectable patterns. Adversarial actions extend to subverting directory servers (servers that store information about the circuits available, among other information) and/or reducing network reliability through attacks on nodes.

2.5.1.2 Overview

In the Tor system, a sender and a receiver communicate through a circuit, typically composed of three nodes. There are three types of nodes: entry nodes that know who the sender is but do not know who they want to communicate with, the middle node that knows neither the sender nor the destination of their messages and the exit node that knows the destination of the messages but don't know who the sender is. By default, the three-node circuit used by Tor has one node of each type.

After the circuit establishment, the sender sends a message using the Onion Routing technique. It sends the messages encrypted in layers, a first layer that only the exit node can remove, a second layer that only the middle node can remove and the outermost layer that only the entry node can decrypt. All these layers result from the use of symmetric keys that are only known by each $(Sender, CircuitNode)$ pair. These keys are generated and set up during the circuit setup.

The system is relevant to our study as it is a low latency system with the greatest acceptance in the real world and whose demand has been increasing with improvements that have been made by the Tor community. If the receivers of the requests also desire privacy, they can use Onion services described in Subsection 2.5.1.4, a mechanism that allows both ends of the circuit to be anonymous.

2.5.1.3 Circuit Setup

We consider the establishment of a circuit with two hops (nodes) between *Alice* and *Bob*. Note that, the normal case is three nodes; in this case with 2 nodes, we only have an entry node and an exit node, and the middle doesn't exist. The process of circuit establishment uses Diffie-Hellman [41], an algorithm that allows two entities to generate a symmetric key securely and privately without anyone but them becoming aware of the key value. Figure 2.5.1.3 illustrates the sequence of actions of the Tor circuit establishment process and it works in the following manner:

1. The sender establishes a symmetric key with $EntryNode$ by sending the first part of the Diffie-Hellman encrypted with the public key of $EntryNode$. $EntryNode$ completes the handshake by sending the second part of Diffie-Hellman as well as the hash of the generated key.

2. Now that the communication with $EntryNode$ has been established and verified, the sender forwards an *Extend* cell to create the connection with $ExitNode$. This cell includes the indication that is an extension, the address of $ExitNode$ and again, the first part of Diffie-Hellman. Note that the address of $ExitNode$ is necessary because otherwise, $EntryNode$ couldn't redirect the message to $ExitNode$. Then the process repeats to the $ExitNode$ as in the $Sender - EntryNode$ communication establishment. The circuit is now established.

2.5.1.4 Onion Services

Tor is a system that allows users to communicate anonymously. However, in the normal mode of operation, this anonymity is only guaranteed for the sender being the receiver a known entity. Tor has a mechanism that allows both the sender and the receiver anonymity, the Onion Services [88] introduced in subsection 2.5.1. For this to happen, the Receiver chooses three Introduction Points (IPs), which are nodes, that will be its reception points

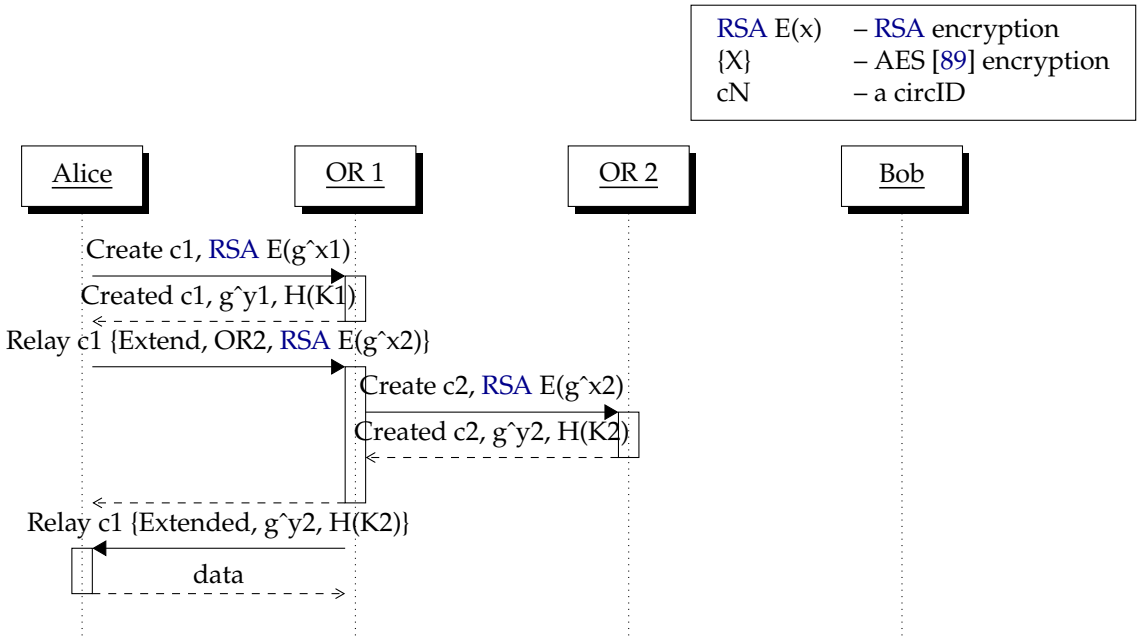

Figure 2.2: Sequence Diagram of the Tor Circuit Establishment Process

for traffic of entities wanting to contact it. The receiver establishes a circuit with each one of them, which guarantees its anonymity, even from his introduction points. It then signs a list that contains the addresses of the three IPs and publishes it on the directory servers so that future senders know its reception points when they want to contact them. The receiver is called an Onion Service.

When a sender wants to communicate with an Onion Service, the process begins with the sender requesting a list of IPs from a directory server. The sender verifies the integrity of this list by checking it against the public key of the Onion Service, which is embedded within the service's onion address. The address of an Onion Service has the following format `km3eflebhaa2gvtgywixa6fqlj4mdhvgem53trjtq55p72wmibc5xmid.onion` and is called an onion address. This address has 56 bytes; despite being used for contact purposes, it is also used to verify the authenticity of the Onion Service. Once the integrity of the IP list is confirmed, the sender proceeds to contact one of these IPs. The purpose of the IPs is to help the Onion Service establish a circuit with a RP that will also establish a circuit with the sender. Thereafter, this RP acts as an intermediary between them. The RP is crucial because it allows the sender to communicate with the Onion Service without either party knowing each other's actual network addresses. The RP also do not know the actual network addresses of both the sender and Onion Service considering they establish a circuit to the RP.

At this stage, the connection between the sender and the receiver is fully established, enabling anonymous communication between the two parties through the RP. In Figure 2.5.1.4, we can see the establishment of an Onion Service where a sender communicates

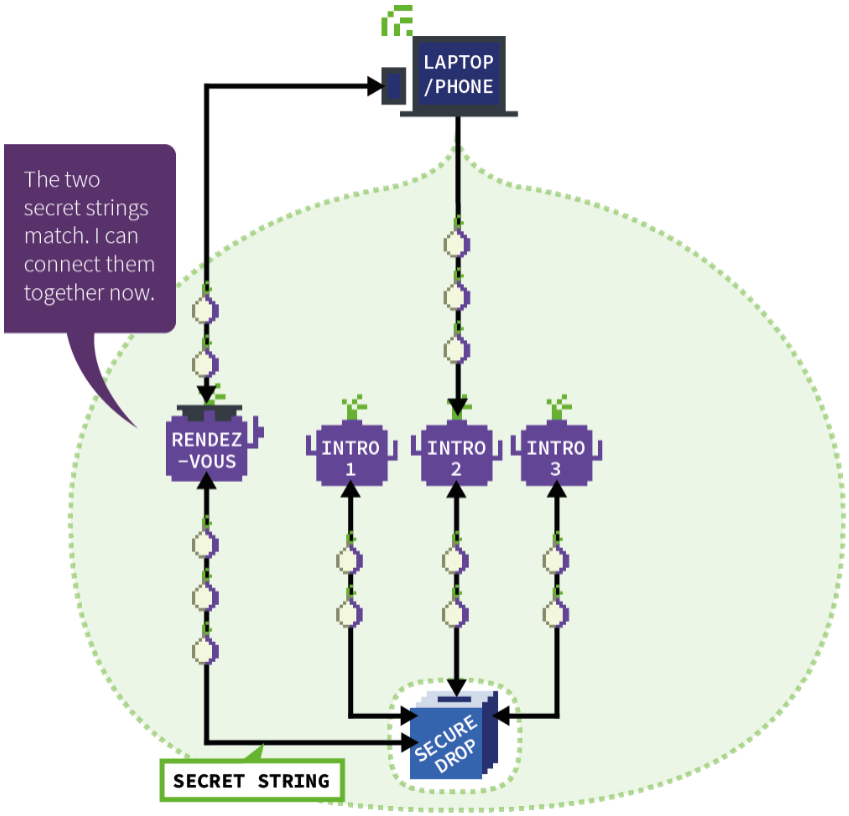

Figure 2.3: Onion Service Establishment [88]. Taken from Tor Project documentation.

with the Secure Drop Service Onion Service [87]. For these reasons, we chose Tor as the anonymity system for our prototype as this allows two entities to communicate anonymously with each other. In the next chapter, we discuss the related work, where we present the reasons why previous solutions don't solve the problem we intend to solve.

2.5.1.5 Attacks on Tor

Distributed systems, including anonymity networks like Tor, are susceptible to various types of attacks. These attacks can be broadly categorized into two main types: passive attacks and active attacks. Passive attacks on Tor typically aim to compromise the anonymity of users and services by collecting and analyzing traffic data. Depending on their focus, these attacks may target users, onion services, or both. Active attacks involve actively disrupting the system's normal operations. This section explores several attacks on Tor, detailing their objectives, mechanisms, and potential consequences.

Deanonymization of Users One of the primary goals of passive attacks is the deanonymization of users. This can be achieved through traffic analysis, where an attacker monitors the entry and exit nodes of the Tor network and correlates the traffic patterns to identify the users. Basyoni *et al.* [6] provide a comprehensive survey of traffic analysis attacks on

Tor, highlighting various methods that adversaries use to compromise user anonymity. Additionally, correlation attacks, such as those described by Johnson *et al.* [67], involve matching the timing and volume of traffic at different points in the network to trace a user's activity across the Tor network.

Deanonymization of Onion Services Onion services are also vulnerable to deanonymization attacks. Biryukov *et al.* [8] discuss methods for detecting, measuring, and deanonymizing Tor hidden services, revealing how attackers can identify and locate these services despite the protective measures Tor implements. More recent work by Oh *et al.* [85] has shown that advanced machine learning techniques can improve the effectiveness of flow correlation attacks, making it easier to deanonymize both users and hidden services.

Website Fingerprinting Attacks Another passive attack method is website fingerprinting, where an attacker attempts to deduce the websites a user visits by analyzing traffic patterns, even if the content itself is encrypted. Wang *et al.* [103] demonstrate that classifiers like the Multinomial Naïve-Bayes can be used to identify websites visited by Tor users with significant accuracy.

Attacks on Tor's Authentication Protocol Tor's authentication protocol is another potential target for passive attacks. Jansen *et al.* [56] discusses the security of Tor's authentication mechanism, pointing out vulnerabilities that could be exploited to undermine the network's anonymity guarantees.

Denial of Service (DoS) DoS attacks aim to turn the Tor network or specific nodes unavailable by overwhelming them with traffic. Specific DoS attacks like the CellFlood attack, as discussed by Barbera *et al.* [4], exploit vulnerabilities in Tor's circuit-building process, allowing attackers to disrupt the operation of Tor nodes with minimal resources. Sybil attacks are an example of a possible Distributed Denial of Service (DDoS) attack on distributed systems, like Tor. In this attack, an adversary creates multiple fake identities (Sybil nodes) to gain disproportionate influence over the network's operations. Douceur [44] discusses the feasibility of Sybil attacks in large-scale peer-to-peer networks and highlights the challenges in preventing such attacks without a centralized authority.

Censorship Attacks Censorship attacks involve blocking or limiting access to the Tor network, often by state actors. Ling *et al.* [71] explore techniques used to discover and block Tor bridges, which are essential for users in censored environments to access the network. Censorship attacks can severely restrict the availability of Tor, undermining its role as a tool for free and uncensored communication.

3

Related Work

In this chapter, we explore the body of work related to secure multicast communication. The existing literature in this field primarily spans two key areas: secure multicast protocols, and privacy-preserving communication systems, including mix networks and Tor. The following sections provide an overview of relevant contributions in these areas, introducing the core concepts of each approach and assessing their relevance and applicability to our use case.

3.1 Secure Multicast Communication

In this section, we will discuss the most relevant works in the field of secure multicast communication. We will discuss the most relevant proposals, their characteristics, and the problems they solve. We will also discuss the limitations of these solutions and why they are not suitable for our use case.

3.1.1 Centralized Systems

A part of the solutions in the field of secure multicast communication is based on a centralized approach. A centralized system is a system where all the control and decision-making power is concentrated in a single entity. Within this paradigm, a centralized Key Distribution Center (KDC) oversees a hierarchical arrangement of keys [59, 104]. These systems use a logical key hierarchy where a tree of keys is maintained. The tree consists of nodes, where the root node is the KDC, the leaf nodes represent the group members, and the intermediate nodes are logical nodes that help accomplish the system objectives. Each node receives a Key Encryption Key (KEK) by the KDC which is responsible for managing the key hierarchy. This means that the KDC knows the KEKs of all nodes in the tree. Each member of the group knows the KEKs of the nodes that participate in the path from the KDC to itself. In a balanced key tree with m members, each member must hold $O(\log(m))$ keys, which ensures efficient key management.

The other type of key used is the group secret key, which allows for decryption of the content of a specific group. When a group secret key is handed over to a member, it is

encrypted using the KDC's KEK and then multicast to all group members. If a member enters or leaves the system, it is necessary to update the KEKs again. The keys in the *Logical Key Hierarchy* ensure secure group communication by providing confidentiality through encryption, integrity verification, and authentication mechanisms. Associated with individual group members, these keys enable secure data transmission and help manage key distribution efficiently.

The centralized KDC plays a key role in updating and distributing keys, ensuring a trusted environment. The encryption keys safeguard against eavesdropping, and unauthorized modifications, and enable the secure sharing of group secret keys. These keys are essential for maintaining the confidentiality, integrity, and authenticity of communication within the group.

In Molva et al.'s work [78] we have the *Cipher Sequence (CS)* solution which is built on top of a multicast tree, where each non-leaf node in the tree is assigned an encryption function. When a message is sent from a non-leaf node, it is encrypted by the cryptographic function assigned to that specific node. As the message travels from the root of the tree to a member, it is encrypted by all nodes along the way. The encryptions performed by intermediate nodes form a sequence of cyphers. Each leaf node of the tree knows the members of the subgroup, and all members of a subgroup share the same reverse cypher sequence function. When a member joins or leaves a subgroup, the last internal node on the path closest to the subgroup members changes its encryption function. This dynamic change results in a new cypher sequence and a corresponding reverse function for that subgroup.

The work of Mukherjee et. al. [79] is the first paper to discuss the application of proxy encryption to secure multicast communications. This solution uses a similar network model to the previous solution. In contrast to the cypher sequence approach that assigns functions to nodes, this approach assigns cryptographic keys to nodes. A key control component associated with the group controller is introduced; this component is responsible for generating encryption and decryption keys for the sender, receiver and all intermediate transformation nodes - called proxies. When a new node joins the multicast tree, the group controller splits the decryption key of the newcomer's parent node into an encryption key and a decryption key. The encryption key is retained by the parent node for subsequent message transformation. The new decryption key is sent to the newcomer, allowing him to decrypt confidential messages.

Huang et al. [63] built upon previous work by utilizing the ElGamal encryption scheme [46]. This scheme allows for the composition and decomposition of keys. The operations are based on the principles of addition and subtraction. When keys are composed, they are summed, and when decomposed, they are subtracted. This means that if you know the added key, you can subtract it to retrieve the original key. This approach is valuable because the sender encrypts the message with its key, and as the message passes through each proxy on the sender-recipient path, each proxy modifies the encryption with its secret key. Ultimately, when the message reaches the recipient, it is encrypted with all the keys

of the nodes it passes through. The recipient can decrypt the message by applying its key, which needs to be the composition (sum) of all the keys of the nodes through which the message passed.

Despite all the benefits, the centralized nature of these systems introduces a single point of failure in the key management operations. Furthermore, due to the majority of the key management operations going through a central entity, this presents a scalability problem. Given the absence of the notion of anonymity, these solutions are also not suitable for our use cases.

3.1.2 Distributed Systems

Another part of the solutions in the field of secure multicast communication is based on a distributed approach. In a distributed system, members in one group cooperate to generate a shared secret key. One example of this approach is the Group Diffie-Hellman (GDH) where members from one group cooperate to generate a shared secret key using the *Diffie-Hellman* [41] key exchange protocol adapted to N parties [98].

The protocol operates on a cyclic group Cyclic Group (G) of order q with a generator g. Each participant in the group chooses a random value V. The GDH protocol starts with the first member generating a g_i number for the second member. Upon receiving the set of secret numbers from the *(j-1)-th* member, the *j-th* member increases the numbers in the set by adding its secret number g_j and generates a new set for the next member. After the n^{th} round, where n is the number of elements in the group, the final group key can be generated. The generated key is used for secure communication within the group.

The most significant limitation with this type of solution once again is scalability, making it not suitable for large groups. Furthermore, members are required to know everyone in the group, which is unrealistic in large-scale systems and it goes against the anonymity concept.

3.1.3 Decentralized Systems

Decentralized Systems were proposed to address the inherent limitations of centralized and distributed solutions. IOLUS [77] is one of these solutions; in *IOLUS*, the members of a multicast group are divided into several subgroups. The connectivity between these subgroups is managed by Group Security Agents (GSAs). Each subgroup is relatively independent of the others, allowing membership changes to be confined to the specific subgroup where they occur. During transmission of an encrypted message from one subgroup A to another B, the GSA decrypts the message with A's secret key and then encrypts it with B's secret key. This process guarantees the confidentiality of the message. Although the encryption/decryption process achieves the objective of confidentiality, GSAs must be trustworthy considering that he needs to know all the keys of its subgroup members and the key of the other GSAs. GSAs are a single point of failure for each

subgroup and beyond that we end up having the same scalability problem, only this time within each subgroup.

Another solution is the Dual Encryption Protocol (DEP) [43]. DEP divides a multicast group into several subgroups and is composed of three roles: the Subgroup Managers (SGMs), the Group Controllers (GCs) responsible for the Data Encryption Key (DEK) and the members of the subgroups. There are also three KEKs: KEK_1 which is shared between the GC and the SGM, KEK_2 shared between the SGM and the subgroup members and KEK_3 shared by the GC and the subgroup members (excluding the SGM). To deliver the DEK (which is the key that encrypts the data exchanged within the group) to the members of a subgroup, the GC uses a multi-step process. The DEK is first sent as $Enc_{KEK_1}(Enc_{KEK_3}(DEK))$ to the SGM. The SGM decrypts the message with KEK_1 and re-encrypts it with KEK_2. The new encrypted content, $Enc_{KEK_2}(Enc_{KEK_3}(DEK))$, is then sent to the subgroup members. Forward and backward secrecy depends on how often the DEK is updated. If the DEK is changed infrequently, newcomers or former members can easily decrypt old or current messages, compromising the security of the system. In case of being changed frequently, the system's performance decreases. Additionally, when a member joins or leaves a subgroup, the SGM changes the KEK_2 and sends it to all valid members. These solutions do not consider anonymity and therefore, they are not suitable for our use cases.

3.2 Tor: The Second-Generation Onion Router

In the previous chapter, we explained the more relevant aspects of the *Tor* network. However, we did not explain why this is not suitable for our use case. Tor is without any doubt the most well-known system for anonymous communication. It is a free and open-source software that enables anonymous communication by directing Internet traffic through a global network of relays (nodes). However, it is built exclusively for unicast communication and not multicast; multicast introduces additional and more complex challenges than unicast, such as the fact that the number of receivers can be large, perhaps thousands. To give an example, if a user wants to send a message to 20 users it would require 190 ($20 * 19/2$) connections for all users to be connected by applying the formula $n(n-1)/2$ which gives the number of diagonals and edges of a complete graph; making it not feasible for large groups.

3.2.1 SecureDrop

SecureDrop [87] is an open-source whistleblower submission system designed to facilitate secure communication between journalists and sources. Developed by the Freedom of the Press Foundation, SecureDrop provides a secure and anonymous way for whistleblowers to submit sensitive documents and information to journalists without fear of interception or exposure. The architecture of SecureDrop consists of the following components:

- **Submission Interface**: Sources access SecureDrop through the Tor network, which anonymizes their connection. They can submit documents and messages through a web-based interface.
- **Secure Server**: The submissions are stored on a secure server, only accessible to authorized journalists.
- **Journalist Workstation**: Journalists retrieve submissions from the secure server using a dedicated, secure workstation. This workstation is configured to prevent unauthorized access and to maintain the integrity of the retrieved information.
- **Encryption**: All submissions are encrypted using Pretty Good Privacy (PGP) [60] encryption. This ensures that even if the data is intercepted, it cannot be read without the corresponding decryption key.

To achieve the confidentiality and integrity of the communication, SecureDrop incorporates the following features:

- **Tor Network**: By using the Tor network, SecureDrop anonymizes the source's connection, making it impossible to trace the origin of the submission.
- **End-to-End Encryption**: Submissions are encrypted from the moment they are sent by the source until they are decrypted by the journalist. This prevents unauthorized access to the content during transmission and storage.
- **Regular Audits**: SecureDrop undergoes regular security audits to identify and mitigate potential vulnerabilities. This ensures that the system remains robust against evolving threats.

By providing a secure and anonymous submission platform, it enables sources to share sensitive information without fear of retaliation or exposure. This is particularly important in environments where whistleblowers face significant risks for disclosing information that is in the public interest. The drawback of SecureDrop is its design, directed for unicast communication. In our solution, we also use the Tor network as our anonymity system.

3.3 Mixnets

The main objective of a mix network [16] is to provide anonymity to the sender i.e. the identity of the originator of a message should be impossible or very difficult to discern.

Mix Nodes are the fundamental building blocks of Mixnets. These servers receive, shuffle, and re-encrypt messages before transmitting them along the communication chain. Each mix node is unaware of the identity of the sender (except the first node of the chain) and the receiver (except the final node of the chain). Each layer of the Mixnets has a series of mixed nodes that are connected to the next and previous layer mix nodes; we

can think of a Mixnet as a series of columns of squares parallel to each other, where each square is connected to the squares of the next and previous columns.

Mixnets employ strong encryption algorithms to protect data on each node. Public key cryptography, symmetric key cryptography, and cryptographic primitives such as hash functions are used to ensure the confidentiality and integrity of messages as they traverse the Mixnet. Shuffling mechanisms such as permutation networks [102] are implemented in mixed nodes. These shuffling algorithms reorder and re-encrypt messages, making it difficult to correlate input and output, thus preserving anonymity. It also uses padding schemes to standardize message lengths; the padding ensures that messages are uniform in size, preventing adversaries from inferring information based on the length of the messages thus increasing anonymity. They also incorporate traffic-splitting techniques [51] to distribute incoming messages over multiple paths; this diversification makes it harder for attackers to correlate messages based on their timing or volume.

A user sends an encrypted message to a *Mixnet*. The Mixnet routes the message through a sequence of mixed nodes; each node shuffles, re-encrypts and forwards its messages. Additionally, in each step, the message metadata is removed, preventing traceability. After traversing several *Mix Nodes*, the message reaches the last layer, where it is delivered to the intended receiver. The anonymity of the sender is preserved for all nodes apart from the first one, considering it receives the message directly from the sender.

3.3.1 The Loopix Anonymity System and The Nym Network: The Next Generation of Privacy Infrastructure

Loopix [86] is a low-latency anonymous communication system that provides strong privacy guarantees. However instead of being based on onion routing as Tor, it is based on Mixnets.

There are three types of traffic in *Loopix*, drop cover traffic which corresponds to the normal path through which two users communicate (in case they do not want to communicate, they send dummy messages to simulate communication and maintain the normal behaviour), the Client's Loop Cover traffic, and the Mix's Loop Cover traffic, the latter responsible for the name given to the system. These last two types of traffic detect the presence of malicious nodes in the system; a specific client or mix node by sending messages to itself can verify if the messages are being dropped or delayed by a malicious node. If they do not receive the messages on time, it means that there is a malicious node in the path that needs to be removed. This technique allows the system to protect itself from N-1 attacks [95], as it causes "dead ends"; even if the adversary follows the target message when it exits the node, the message ultimately arrives at a dummy node, resulting in the adversary expending additional resources without any corresponding benefit.

Loopix contains some mailboxes, that we call providers, that allow the user to have offline storage and that serve as intermediaries between the senders and receivers, and the

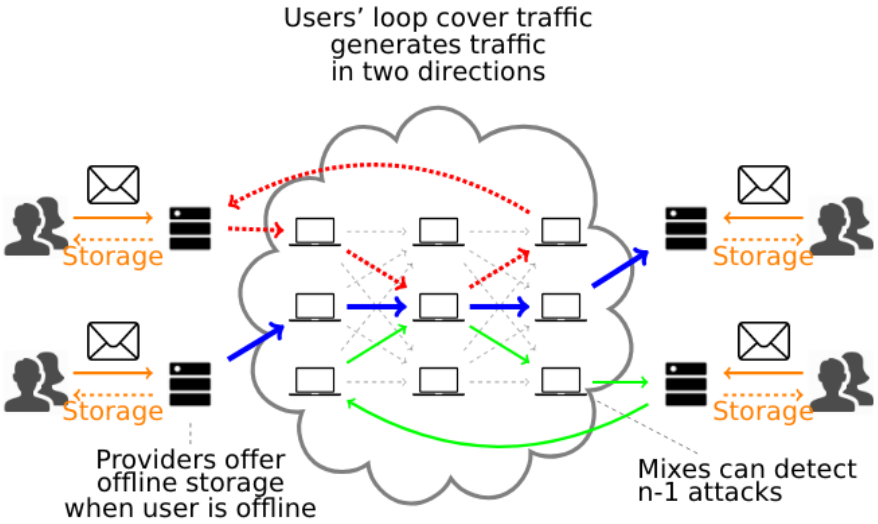

Figure 3.1: Loopix Architecture [86]. Taken from Loopix paper.

Mixnet; Figure 3.3.1 illustrate the organization of the system. A sender sends a message to its provider, which then forwards the message to the Mixnet. The Mixnet routes the message through a series of Mix Nodes, which shuffle, re-encrypt, and forward the message to the provider of the receiver. To avoid revealing any details about the messages sent/received by a user, the number of messages involved in an exchange with the Provider is constant. For example, if the user goes online to check for received messages and only receives one message, the provider generates nine dummy messages to obscure the actual communication and conceal the communication patterns.

This combination of real and cover traffic increases anonymity, making it difficult for adversaries to differentiate between genuine communication and cover activities. The major limitation of Loopix is the base assumption that all the traffic is emitted following a Poisson process which is not a valid assumption in the real world, at least for now. Additionally, it does not support multicast communication. The Nym network [40] is built on top of Loopix [86] and adds a cryptosystem on top of it, however, it is not relevant for our work and therefore, we will not discuss it here.

3.3.2 Karaoke: Distributed Private Messaging Immune to Passive Traffic Analysis and Stadium: A Distributed Metadata-Private Messaging System

Karaoke [70] is a privacy-preserving messaging system based on the principles of Mixnets that send messages to the users through the mailboxes previously mentioned, the dead-drops.

Karaoke is based on four entities, senders, receivers, mix nodes, and deaddrops. Deaddrops are servers where the senders send messages and from where the receivers

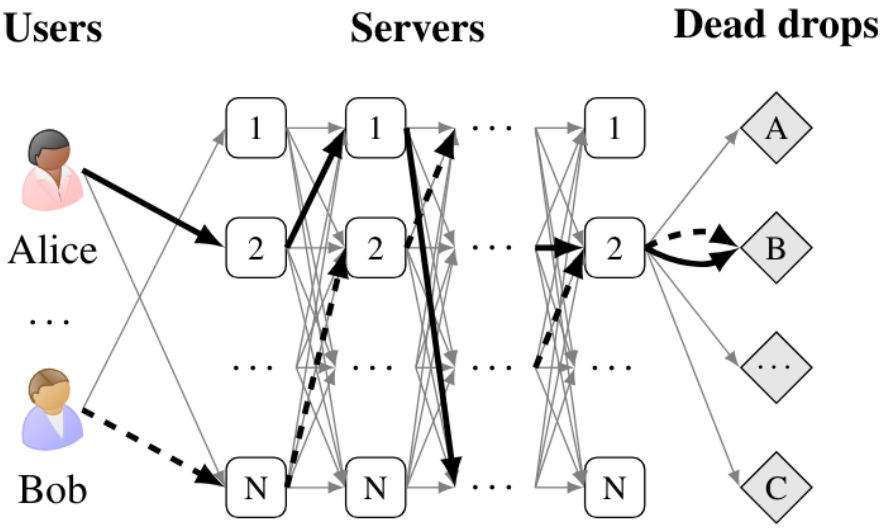

Figure 3.2: Loopix Architecture [86]. Taken from Karaoke paper.

retrieve them. The idea behind the deaddrops is to send messages to a server instead of directly to the receiver, working as a meeting point. A sender and a receiver agree on a deaddrop and use it to exchange messages; the sender sends the message to the deaddrop through the mix nodes and the receiver retrieves it. The senders achieve this by sending a publish message request to the deaddrop; the receivers achieve this by sending a retrieve message request to the deaddrop. In the case of a specific sender or receiver not communicating, it sends two dummy messages, one publish message request and one retrieve message request, to the deaddrop to simulate communication and maintain normal behaviour. This way it helps the system to maintain the traffic constant, even if the users are not communicating, which helps increase the anonymity of the system. The mix nodes themselves also generate dummy messages to detect malicious nodes in the system - called noise messages.

Karaoke uses Bloom Filters [9], a probabilistic data structure typically used to add elements to a set and test if an element is in a set, to deal with the noise messages. Each time a message passes from mix node to node server guarantees receipt of all noise messages by calculating a Bloom filter that covers all received messages. This filter is then shared with all other servers on the network. Subsequently, these servers cross-check whether the generated noise messages are present in the received Bloom filter.

Stadium [101] works in a very similar way to Karaoke. The only difference between the two alternatives is the shuffling method; Stadium uses a Hybrid Verifiable Shuffle [101] instead of the Bloom Filters used in Karaoke. However, this technique consumes many resources and adds significant computational overhead to the system. The Bloom filters in Karaoke minimize this computational burden, ensuring efficient and reliable scanning of noise messages without compromising security or performance. Karaoke also uses the Poisson technique previously described.

The major limitation of Karaoke is, similar to Loopix, the base assumption that all the traffic is emitted following a Poisson process which is not a valid assumption in the real world, at least for now. Additionally, it does not support multicast communication. In our system, we will incorporate the concept of deaddrops.

3.3.3 Groove: Flexible Metadata-Private Messaging

The most recent advancement in the domain of private messaging systems is Groove [5]. Groove is a privacy-preserving messaging system that is based on the principles of Mixnets. The most distinguishing feature of Groove is that allows multi-device messaging, meaning that a user can use multiple devices to communicate with the system. Groove uses the same path of nodes for communication between the same pair (Sender, Receiver); it accomplishes this through the use of a multidevice key that is shared among all devices belonging to a single user. When a sender wants to send a message, it generates paths based on an algorithm and this key, ensuring that all devices associated with the same sender use the same path for communication with the same receiver.

Similar to Loopix, to send a message, the sender sends the message to the provider that forwards it to the Mixnet. The Mixnet routes the message through a series of Mix Nodes until it reaches the provider of the receiver. The receiver retrieves the message from the provider. To avoid revealing any details about message volumes between a sender/receiver and a provider, the same technique of Loopix is applied; if the receiver goes online to check for received messages and only received one, the provider generates nine dummy messages to make the traffic uniform.

While in Karaoke, two dummy messages are sent to the deaddrop to simulate communication and maintain normal behaviour, in Groove a different approach is used; all sender providers send the same amount of messages at a time and maintain the same amount of circuits between them and receiver providers. This uniformity in an equal number of paths for each user, enhances anonymity and prevents any user from being disadvantaged by having less communication activity.

The major limitation of Groove is, similar to the previous mixnets, the base assumption that all the traffic is emitted following a Poisson process which is not a valid assumption in the real world, at least for now. Additionally, it does not support multicast communication. In our system, we will incorporate the concept of deaddrops.

3.4 Summary

In this Chapter, we presented the limitations of existing solutions for private and secure multicast communication and discussed some of their limitations. We analyze many different work categories related with ours, however, none of them showed to be capable of guaranteeing our requirements mentioned in Section 1.2. While private messaging systems don't provide adequate multicast capabilities, multicast systems don't provide

sufficient levels of privacy and anonymity. Now that we detailed all the fundamentals surrounding our topic and explained why each of the existing solutions does not fit our requirements, we propose in Chapter 4, a solution that addresses all the requirements mentioned in Section 1.2 and adapts to different use cases.

4

Anonycast

In this chapter, we propose Anonycast a multicast solution that provides security and privacy for publishers and receivers and can work on top of any anonymity system that allows the use of rendezvous points. We chose Tor for this purpose, considering that it is the most adopted anonymity system with these characteristics. Anonycast brings the mechanisms that are present in private unicast communication to multicast communication, namely, the ability to publish messages anonymously. Additionally, Anonycast adapts to different use cases that emerge in the real world. In the following section, we discuss the architecture of our solution as well as the components, participants and protocols that compose Anonycast.

4.1 Architecture

Figure 4.1 represents the global architecture of our solution. We have three main entities: Publishers, Receivers, and Group Owners. Publishers are the entities that want to publish messages to the group, Receivers are the entities that read the messages from the group, and the Group Owners are the entities responsible for a specific group. We only require a Group Owner in groups where the set of publishers and/or receivers is restricted. Also, note that we do not have restrictions on who these entities are; they must only comply with the system requirements.

Regarding components, our system contains two main components: deaddrops and a verifiable remote Randomness Generator [99]. The deaddrops are the components responsible for storing the messages and one of the elements responsible for verifying them; these deaddrops are Onion Services, ensuring anonymity for the publishers, receivers, and themselves. Besides their message storage functionality, message validation and more, deaddrops also inherit all the properties that come from being an Onion Service [88]. The verifiable remote Randomness Generator works as a source of randomness for all needed purposes. This architecture provides a robust and reliable communication platform that ensures compliance with all the requirements mentioned in section 1.2. In the following sections, we discuss the anonymity system and protocols that comprise the Anonycast

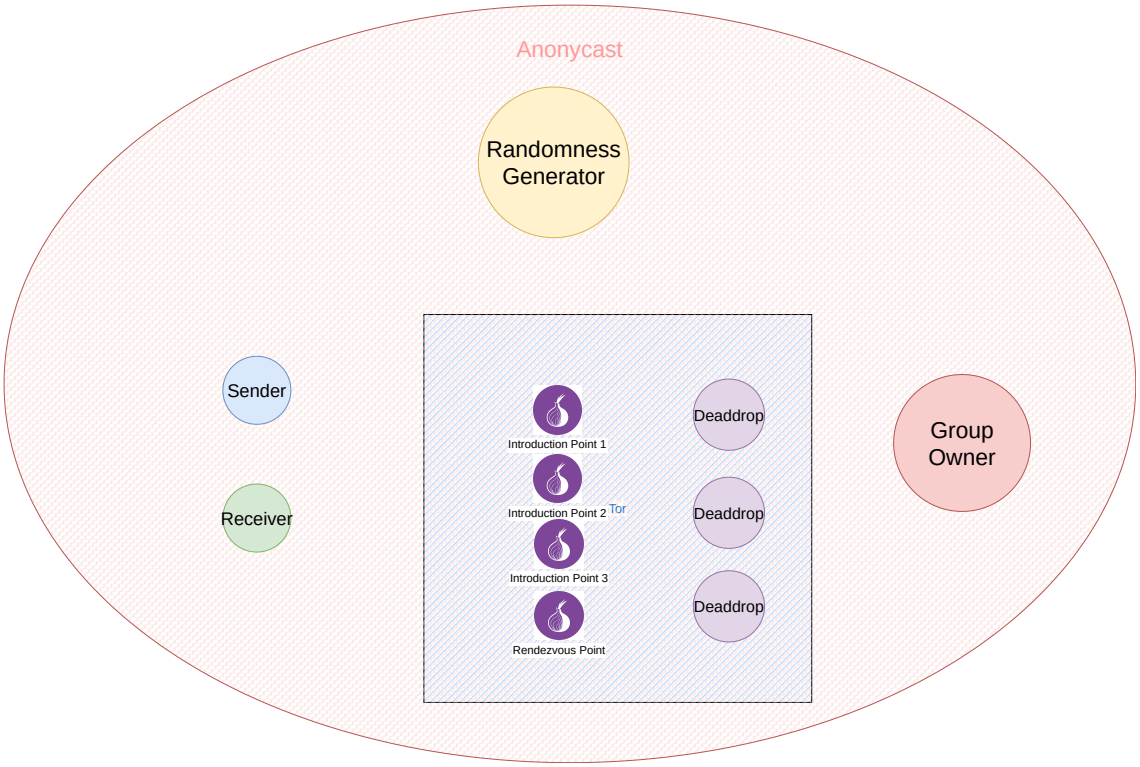

Figure 4.1: Anonycast Overall Architecture.

system and that allowed us to achieve our results mentioned in Chapter 5.

4.2 Anonymity system

Our solution is built on top of an anonymity system allowing the use of rendezvous points. These rendezvous points are used to allow the exchange of messages without the publisher and receiver knowing each other, in line with our privacy objectives. The prototype leverages Tor, a renowned anonymity system. However, as previously mentioned, our system is not attached to any specific anonymity solution and works with any system supporting rendezvous points. Decentralized control, Onion Routing integration, Censorship Resistance, Open Source implementation, Security, Anonymity, Privacy, Multi-Platform Support and Adaptability makes Tor a perfect fit for our goals.

4.3 Modes of Operation

Anonycast supports two types of modes of operation: Open mode and Restricted modes. In Open Mode, anyone can join the group and send messages to all participants. In Restricted modes, there are some restrictions depending on the concrete mode of operation. The three sub-modes associated with Restricted Mode are Publisher Restricted,

Receiver Restricted, and Publisher and Receiver Restricted (or Fully Restricted) modes. In Publisher Restricted groups, only specific users can generate a special signature required for publishing documents. Additionally, another property of this mode of operation is the indistinguishability to the receivers, who can not decipher which of the allowed publishers publish the message, preserving their anonymity; the only information revealed is that the message was sent by an allowed publisher. While Publisher Restricted focuses on signature methods, Receiver Restricted focuses on encryption methods, namely Broadcast encryption. In Receiver Restricted groups, only specific users have access to the key to decrypt the messages. The Group Owner is responsible for managing both the allowed publishers and receivers. In Publisher and Receiver Restricted groups, both publishers and receivers are restricted to a specific set of participants, which as in previous modes, are managed by the Group Owner; this is the most restrictive mode of the system. All of these modes allow our system to be used in different scenarios and adapt to the different needs that emerge in the day to day life.

4.4 Anonycast Protocols

In this section, we describe the protocols that compose Anonycast. These protocols are responsible for the communication between the different entities and components of the system, and the delivery of the messages.

4.4.1 Publish Message

In algorithm 3, we describe the process of publishing a message in Anonycast. Firstly, the publisher needs to solve the Cryptopuzzle, explained in Subsection 4.4.2. Secondly, the publisher attaches all the information necessary to validate the Cryptopuzzle solution to the message to be published. Next, the topic of the message and a message identifier (ID) are added by the publisher. The message identifier is a combination of the number of the round in which the cryptopuzzle was solved, the hash of the publisher's public key and the hash of the content. This is helpful for cases where the publisher may resend the same message for different reasons, such as guaranteeing the delivery of the message, and the deaddrops can identify the message as a duplicate; this is also helpful for retrieving purposes as we will see in Subsection 4.4.6.

Lastly, the publisher signs the message and publishes it to the deaddrops, which then validate the message. One of the validations performed by the deaddrops is the acceptance window validation. This is a predefined time window that specifies the maximum allowable time between solving the cryptopuzzle and the message reaching the deaddrops. The number of deaddrops where a publisher publishes its messages vary; the more deaddrops the publisher uses, the greater the guarantee that the publisher's message is successfully published. Similarly to the publish operation, the more deaddrops the

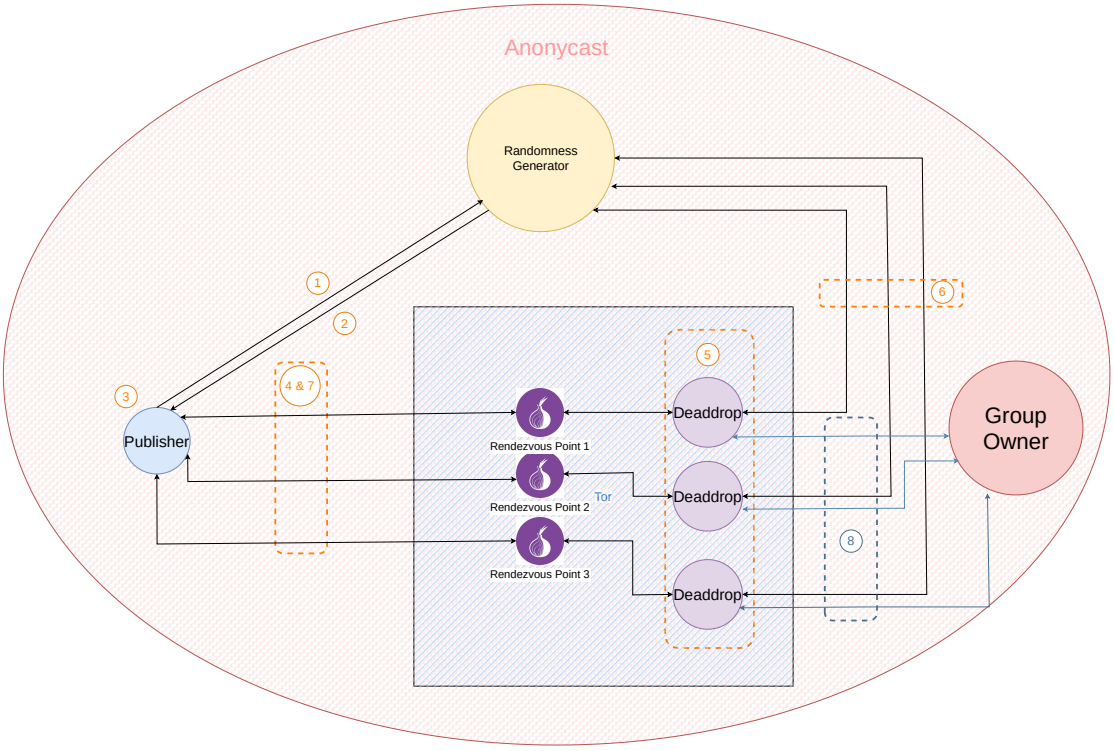

Figure 4.2: Publish Message Protocol.

receivers use to retrieve the messages, the greater the guarantee that they will successfully retrieve the message.

The main difference in this protocol between the different modes of operations is the signature method used and the existence or not of encryption. In Receiver Restricted Mode and Publisher and Receiver Restricted Mode, the sender encrypts the message using the Broadcast Encryption algorithm cited in line sixteen of the Algorithm 3. In Open Mode and in Publisher Restricted Mode, the message remains unencrypted. In figure 4.2 we illustrate the Publish Message Protocol. Note that the numbers in the figure correspond to the numbers of the following enumerated steps.

1. **Request Randomness**: The publisher requests the randomness details (round of randomness signed, randomness itself) from the Randomness Generator for Cryptopuzzle-solving purposes.

2. **Randomness generator response**: The Randomness Generator answers with the randomness details signed, allowing publishers to verify the authenticity of the randomness.

3. **Solve Cryptopuzzle**: The publisher initiates the Cryptopuzzle using the received randomness. After finding the solution, the publisher prepares the message to be sent, as previously described.

Algorithm 3 Publisher Publish Message Request

1: Let g be the generator of randomness.
2: Let c be the content.
3: Let d be the Cryptopuzzle difficulty.
4: Let k be the private key of the user sending the message (Note that in sender restricted mode, this key corresponds to the private key of the ring).
5: Let pk be the private key of the user sending the message (Note that in sender restricted mode, this key corresponds to the public key of the ring).
6: $NonceSolution = SolveCryptopuzzle(g, c)$
7: $ContentHash = Hash(content)$
8: $Message =$
9: (
10: $\{ContentHash, g.currentRound, Hash(pk)\}$,
11: $MessageSigned = Sign(Message, k)$
12: $NonceSolution$,
13: $MessageSigned$,
14: $ContentHash$,
15: $g.randomness_{Signed}$
16: Algorithm 2 for secure broadcasting.
17:)
18: **for** i:=0 **to** deaddrops.length **do**
19: $send(Message, deaddrops[i])$;
20: **end for**

4. **Send Message to deaddrops**: The publisher sends the message to the deaddrops.

5. **Verify Cryptopuzzle Solution**: The deaddrops verify the solution of the Cryptopuzzle. If the solution is correct, the deaddrops do the remaining validation steps (described in Subsection 4.4.5) and store the message; otherwise, they discard it.

6. **Exchange information about Randomness between deaddrop and Randomness Generator**: The deaddrops and the Randomness Generator exchange information about the current round of randomness. This is necessary for the validation of the Cryptopuzzle solution.

7. **Send Message informing the Publisher about the success of the Operation**: After the message storage, the deaddrops send a confirmation message back to the publisher. This informs the publisher that the operation was successful.

8. **Exchange information about allowed Publishers and Receivers between deaddrops and Group Owner**: The Group Owner sends the updated list of the allowed publishers and receivers according to a predefined refresh window. This happens so the deaddrops share this information with other publishers and receivers. Note that this step only happens in groups operating in Restricted Modes.

Algorithm 4 Solve Cryptopuzzle

Require: $v1$ (input byte array), $difficulty$ (difficulty level)
1: Initialize $nonce \leftarrow 0$
2: Initialize $solved \leftarrow$ False
3: **while** not $solved$ **do**
4: Initialize hasher with SHA256
5: Update hasher with $v1$
6: Update hasher with bytes of $nonce$
7: $result \leftarrow$ finalized hasher output
8: **if** all bits in $result$ up to $difficulty$ are 0 **then**
9: $solved \leftarrow$ True
10: **end if**
11: Increment $nonce$ by 1
12: **end while**
13: **return** $nonce$

4.4.2 Solve Cryptopuzzle

To solve the Cryptopuzzle, the publisher needs two pieces of information, the signature of the round of randomness and the randomness itself. The signature of the round of randomness guarantees the information's authenticity and that the solution was crafted within a certain time window. Additionally, the randomness itself guarantees that the puzzle is unique for each round of randomness; one factor contributing to this is the round of randomness influence on randomness, which is one of the elements involved in the randomness generation process; therefore, the randomness itself varies from round of randomness to round of randomness. These two pieces of information avoid spam and replay attacks, and allow a receiver to determine if the time of the solution exceeds the configured acceptance window. With this information, the publisher only needs to craft the final piece of the puzzle which is a number that when hashed with the round of randomness signed and the randomness itself, the result has a predefined number of leading zeros. The greater the number of zeros, the more difficult is to solve the puzzle. By changing the acceptance window and the Cryptopuzzle difficulty we allow the group owner to configure how much security, availability and performance his system has, making the system flexible and suitable to different use cases. In algorithm 4 we present the pseudocode for the Solve Cryptopuzzle protocol.

4.4.3 Sign Message

In Sender Restricted and Fully Restricted mode, the publisher uses a different signature method other than the traditional one to sign the messages, the Ring signatures. As described in Section 2.4.1.1, the Ring signature is a digital signature that allows a member of a group to sign a message on behalf of the group; this signing operation can be performed by any of the members of the group, being impossible to determine which member signed the message. In our system, the Ring signature is used to sign the message

on behalf of the group of publishers. Each publisher knows the public keys of the other publishers and its private key; with this information, the publisher signs the message without revealing their identity. The receivers, by knowing the list of public keys of the allowed publishers, can verify that the message was signed by one of the allowed publishers but cannot know which of the publishers signed the message. In Open Mode and Receiver Restricted modes, the publisher uses the traditional signature method, where the publisher signs the message with his private key and the receivers verify the signature with the public key of the publisher.

4.4.4 Update Allowed Publishers

A new entity emerges when Anonycast is operating in Restricted modes which is the Group Owner; this entity is responsible for updating the list of allowed publishers. The Group Owner generates or updates, in the case that it already exists, a set of public keys corresponding to the publishers allowed to publish messages in the group. The Group Owner signs the list and shares it with the deaddrops. The Group Owner only contacts the deaddrops; the deaddrops then propagate this information to the receivers. The message with the update is composed of the set of public keys of the allowed publishers signed by the Group Owner and the current round of randomness signed by the Randomness Generator. This last part is added because we can also configure, similar to the acceptance window, a validity for this list; this feature prevents deaddrops from disregarding updates and from keeping a deprecated list. It also allows the Group Owner to remove some publishers from the list if needed. The deaddrops, when receiving the update, verify the signature and update their list of allowed publishers.

When a receiver wants to retrieve messages, during the retrieve messages IDs request, the deaddrops share the allowed publishers update message to the receivers; the receivers do the same verification steps as the deaddrops upon receiving the message, trusting that at least one deaddrop will share the information but not trusting in the information itself. The deaddrops share this information to the receivers simultaneously with the delivery of the message IDs to the receivers. The receivers then verify for each of the messages retrieved if the message was sent by an allowed publisher. Note that this process only happens in groups operating in Sender Restricted and Fully Restricted modes, where the set of allowed publishers is restricted. In Open Mode and Receiver Restricted mode, the list of allowed publishers is not used and therefore this process is not needed.

4.4.5 Message Validation

The message validation process is done by the publishers, receivers and deaddrops. The deaddrops validate the messages directly sent by the publisher, by verifying the Cryptopuzzle and the signature of the message guaranteeing that the message complies with the mode of operation requirements. Despite deaddrops storing and validating the messages, Receivers trust neither the deaddrops nor the publishers; when a Receiver wants

to retrieve the messages from a topic, he needs to validate each message again. The Receiver does two validations, first, it validates the message itself, including the Cryptopuzzle and the signature of the message; secondly, it validates the allowed publishers' information shared by the deaddrops; this verification checks if the list is still valid. This ensures that deaddrops are not communicating old information about the allowed publishers. In algorithm 5 we can see the pseudocode for the Cryptopuzzle validation.

Algorithm 5 Verify Cryptopuzzle Solution

Require: $v1$ (input byte array), $nonce$ (solution nonce), $difficulty$ (difficulty level)
1: Initialize hasher with SHA256
2: Update hasher with $v1$
3: Update hasher with bytes of $nonce$
4: $result \leftarrow$ finalized hasher output
5: **if** all bits in $result$ up to $difficulty$ are 0 **then**
6: **return** True
7: **else**
8: **return** False
9: **end if**

4.4.6 Retrieve Messages

In algorithm 6, we describe the process of retrieving messages in Anonycast. Firstly, the receiver chooses' a topic he is interested in. After the user's selection, they contact the set of deaddrops related to that topic. The receiver requests the IDs of the messages published in the topic since a calculated point in time from each of the deaddrops. This point of time is given by the formula

$$Since = \max(\min(G.CurrentRound - AcceptanceWindow, \\ LastRoundSeen - AcceptanceWindow), 0),$$

where G is the generator of randomness, $CurrentRound$ is the current round of randomness, $AcceptanceWindow$ is the time window to accept messages and $LastRoundSeen$ is the round of the last message the receiver received in past conversations. The $LastRoundSeen$ is zero if it is the first retrieve on the topic by the receiver. This is needed to avoid retrieving all the messages every time the receiver wants to retrieve messages from a given topic and also to avoid missing messages that are valid according to the acceptance window. If our system has an acceptance window of four, and the current round of randomness is ten, the receiver receives messages since round six; simultaneously, we do not want receivers to retrieve messages they have already seen.

Algorithm 6 Receiver Retrieve Message Request

1: Let $acceptanceWindow$ be the time window to accept messages.
2: Let $lastRound$ be the round of the last message the user received.
3: Let g be the generator of randomness.
4: $Since$ = max(
5: min($g.currentRound - acceptanceWindow, lastRound - acceptanceWindow$),
6: 0)
7: **for** i:=0 **to** deaddrops.length **do**
8: $\quad MessageIds = MessageIds \cup RetrieveMessageIds(since)$
9: **end for**
10: **for** i:=0 **to** deaddrops.length **do**
11: $\quad Messages = Messages \cup RetrieveMessages(MessageIds)$
12: **end for**
13: **for** i:=0 **to** Messages.length **do**
14: $\quad verify(Messages[i])$
15: **end for**

Algorithm 7 Deaddrop Upon receiving a Retrieve Messages Request

1: Let c be the client requesting.
2: Let r be the request.
3: $Messages = GetMessages(r.ids)$
4: $send(c, Messages)$

Algorithm 8 Deaddrop Upon receiving a Retrieve Messages IDs Request

1: Let c be the client requesting.
2: Let $topic$ be the topic
3: Let $round$ be the round for the request.
4: $IDS = GetMessages(topic, round)$
5: $send(c, IDS)$

After deaddrops return the message IDs associated with the topic, the user needs to organize all the message IDs by deaddrop and filter which deaddrop to contact for each message, in case more than one deaddrop has the message; the messages are grouped into pairs of the form $(deaddrop, Messagestoberequested)$. In case a deaddrop can not be reached, the receiver distributes the messages on the list by the other deaddrops, if possible. Now, the receiver makes a second request to retrieve the messages from the corresponding deaddrops. After the receiver has all the messages, he just needs to apply the validation message process on all messages retrieved, explained in subsection 4.4.5 and, see their content. Note that the message IDs are retrieved in a separate process to avoid retrieving the same messages from different deaddrops, saving bandwidth and time. In Figure 4.3 we illustrate the Retrieve Message Protocol to illustrate how the algorithm applies to our previous architecture shown in diagram 4.1. Note that the numbers in the figure correspond to the numbers of the following enumerated steps.

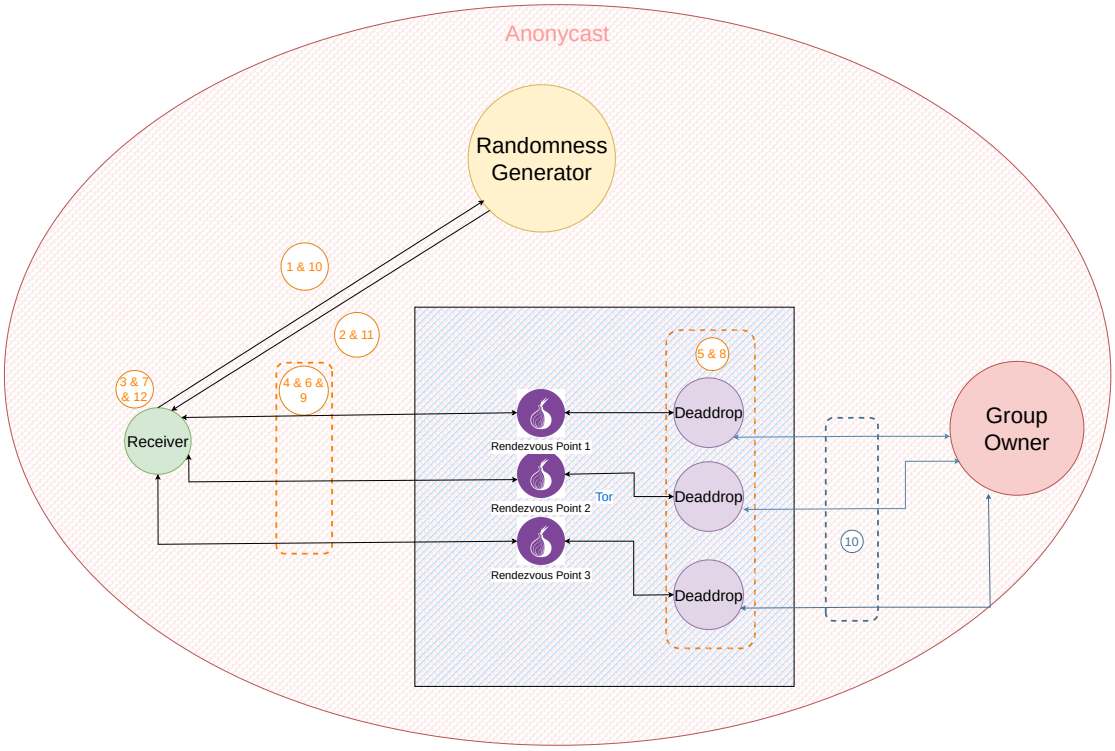

Figure 4.3: Retrieve Messages Protocol.

1. **Request Randomness**: The receiver requests the randomness details (round of randomness signed, randomness itself) from the Randomness Generator for Cryptopuzzle-solving purposes.

2. **Randomness generator response**: The Randomness Generator answers with the randomness details signed, allowing receivers to verify the authenticity of the randomness.

3. **Solve Cryptopuzzle**: The receiver initiates the Cryptopuzzle using the received randomness. After finding the solution, the receiver prepares the message IDs request to be sent, as previously described.

4. **Request IDs of the topic messages to deaddrops**: The receiver requests the message IDs of the messages from a given topic.

5. **Evaluate request**: The deaddrops evaluate the request and collect the message IDs of the topic messages.

6. **Send Message IDs to Receiver**: The deaddrops send the message IDs collected in the previous step to the receiver.

7. **Aggregate Message IDs and Ask for Messages**: The receiver aggregates the message IDs received from the deaddrops and sends a second request to deaddrops requesting the messages associated with the IDs.
8. **Deaddrop Filter Messages**: The deaddrops filter the messages based on the message IDs received.
9. **Send Messages to Receiver**: The deaddrops send the messages to the receiver.
10. **Receiver ask for Randomness information**: The receiver requests the current round of randomness from the Randomness Generator.
11. **Randomness generator response**: The Randomness Generator responds with the current round of randomness and its signature. The signature is used to verify the authenticity of the randomness.
12. **Validate Messages**: The receiver validates the messages using the round of randomness, the list of allowed publishers and receivers, if relevant in the mode of operation. If the messages are valid, the receiver stores them; otherwise, they discard them.
13. **Exchange information about allowed Publishers and Receivers between deaddrops and Group Owner**: The Group Owner sends the updated list of the allowed publishers and receivers according to the predefined refresh window. This happens so the deaddrops share this information with other publishers and receivers. Note that this step only happens in groups operating in Restricted Modes.

4.4.7 Update Allowed Receivers

As previously mentioned a new entity emerges when Anonycast is operating in Restricted modes which is the Group Owner; this entity is responsible for updating both the list of allowed publishers and the list of allowed receivers. The list of allowed receivers is a set of RSA public keys; we chose RSA due to its property of allowing signing a message with both the public and private key, a must property for the system. As happens with the list of publishers, the Group Owner signs the list and shares it with the deaddrops. The information shared is composed of a set of public keys signed by the Group Owner and the current round of randomness signed by the Randomness Generator for validity purposes, as explained in Subsection 4.4.4. The deaddrops verify that this information came from the Group Owner by verifying the signature and updating their list of allowed receivers.

When a publisher wants to share information, before the publish message request, it requests the deaddrops the latest allowed publishers update message received from the Group Owner, doing the same verification steps as the deaddrops when they receive it from the Group Owner. The list is used by the publishers in the Broadcast Encryption algorithm cited in line sixteen of algorithm 3, when the group is operating in Receiver Restricted Mode and Fully Restricted Mode.

4.4.8 Protocol Stability Amid Participant Variability

One challenge that we had to resolve is to allow receivers and publishers identify which deaddrops to use. The list of available deaddrops for each topic is predefined at the moment of the creation of the system and shared with the publishers and receivers. Currently, this list is passed as a parameter to the publishers and receivers, and they use it to send and retrieve messages. A private infrastructure mechanism to update the list of deaddrops is a possible future work described in section 6.1.

Another challenge that we addressed was Anonycast reaction when a publisher, receiver, or deaddrop leaves the protocol. When a publisher or receiver leaves the protocol the system is not affected, and they can always return. The only impact is that they are not able to publish or retrieve messages while they are absent. When a deaddrop leaves the protocol, the users are not able to publish or retrieve messages in this deaddrop. If the number of deaddrops that a publisher/receiver knows for a topic drops to zero, an updated list with the valid deaddrops needs to be shared with the publishers/receivers. If the total number of deaddrops for a given topic drops to zero, the publishers/receivers are not able to publish or retrieve messages on that topic, as there are no deaddrops to store the messages. The churn of deaddrops is a possible future work described in section 6.1. If the Group Owner leaves the protocol, the lists of allowed publishers and receivers are not updated during its absence. If the Group Owner returns before the list of allowed publishers and receivers surpasses its validity, the system continues to operate normally. If the Group Owner does not return, the system does not permit the publication or visualization of messages while operating in Restricted modes. In Open Mode, where the Group Owner does not exist, the system continues to operate normally; therefore, its functionality remains unaffected.

5

Implementation and Evaluation

In this chapter, we present the implementation and experimental evaluation of the Anonycast system from different perspectives in different environments. In the Implementation section, we present the structure of the project, the modules and the main components of the system. We also refer to some libraries used and scripts developed to run the system. In the Evaluation section, we present the results of the tests done in the system in the different environments, and we discuss the results obtained.

5.1 Implementation

The prototype of the Anonycast system is implemented in the Rust programming language. The project is structured modularly, with different modules for each system component. The main components of the system are the Group Owner, the Client and the deaddrop. The Group Owner is responsible for updating the latest version of both allowed publishers and allowed receivers lists. The Client which acts as publisher and then as receiver, is responsible for sending and receiving messages. The deaddrop is responsible for storing the messages and documents. In the following sections, we present the modules of the project and go into each one of them to explain what is its role and which tools and libraries were used to implement it. All the code described can be found in `https://github.com/rdromao27/anonycast`

Crypto Project We select the Rust programming language due to its performance, memory safety, and concurrency features. This project uses the following libraries: *bincode* for the efficient serialization and deserialization of Rust data structures, *aes-gcm* [19] to provide AES encryption with Galois/Counter Mode (GCM) for authenticated encryption, *curve25519-dalek* [22] for operations on the Curve25519 elliptic curve, ensuring modern, efficient, and secure elliptic curve cryptography, and *hex* [24] for encoding and decoding data in hexadecimal format. Additional libraries such as *nazgul* [26], *rand* [28], *ring* [31], *rsa* [32], *serde* [33], and *sha2* [34] are integrated to cover a broad spectrum of cryptographic operations, including random number generation, data serialization, and hashing. The

implementation of this cryptographic system necessitates the Rust compiler and Cargo, Rust's package manager and build system.

Drand Project We implemented the Drand project in the Rust programming language. This project uses the following libraries: the *drand-client-rs* [23] library serves as a Rust client for interacting with the randomness generator, offering functionalities such as verifying randomness. We utilize *Reqwest* [30] for making HTTP requests, with its JSON and blocking features enabled to facilitate easy JSON handling and synchronous operations. *Serde* [33], a serialization/deserialization framework, is used for data structure serialization. The *sha2* [34] library implements the SHA-256 hash function, essential for cryptographic operations. *Tokio* [35], an asynchronous runtime, is primarily used for its synchronization features. *Tracing* [36] provides structured logging, while *hex* [24] is used for encoding and decoding hexadecimal values. The *openssl* [27] library, a target-specific dependency, offers SSL and TLS protocols for secure network communications. We designed the project designed to be cross-platform, specifically targeting x86 and x64-unknown-linux-musl for Linux distributions.

Anonycast Project We implemented the anonycast project in the Rust programming language and it uses the following libraries: *Tokio* [35] is used for its asynchronous runtime capabilities, enabling efficient handling of non-blocking I/O operations crucial for network requests. *Clap* [21] simplifies command-line operations by parsing arguments in anonycast/src/cli. *Serde* [33] is crucial for serializing and deserializing data structures. *Rayon* [29] introduces data parallelism to improve performance by parallelizing tasks. *Anyhow* [20] aids in error handling. The development of Anonycast requires a Rust compiler and Cargo. The project is structured around key components such as the *Protocol*, which defines the communication protocol including message formats and procedures for secure and anonymous communication. A special allocator was still used to improve the performance of the system, the jemallocator [25].

5.1.1 Scripts Project

benchmark.py We develop an evaluation script for the Anonycast system in the Python programming language, leveraging its simplicity and the robust asynchronous capabilities provided by the *asyncio* [53] library, which is particularly advantageous for IO-bound tasks. This script incorporates several libraries, including *asyncio* for asynchronous programming, *argparse* [52] for parsing command-line arguments, *logging* [54] for configurable event logging, and a custom library named *lib.tor* designed for handling Tor-related operations. Its architecture is notably modular, featuring components such as CLI Argument Parsing for managing command-line inputs, Benchmark Functions that include various asynchronous tasks for different types of benchmarks like publish throughput, retrieve throughput, and latency, and Configuration Classes for encapsulating benchmark configurations within

data classes like PublishConfig, RetrieveConfig, and LatencyConfig. This modular design facilitates comprehensive testing capabilities across publish throughput, retrieve throughput, and latency under diverse conditions and simplifies the script's extension and maintenance. The script benchmarks the Anonycast system. During the development, we encountered challenges related to the efficient management of asynchronous operations and the parsing of command-line arguments for distinct benchmarks. We address these by utilizing the asyncio library for asynchronous task management and the argparse library for structuring the CLI, showcasing effective problem-solving strategies and reinforcing the script's utility in evaluating the Anonycast system's performance.

sync.sh The project utilizes Shell script due to its direct access to system commands and simplicity for writing automation scripts. It operates within a Unix-like environment, requiring rsync for file synchronization. It is designed to synchronize project files between local and remote hosts. It specifically excludes certain directories and files, such as build artefacts and .git, transferring only the necessary files to a remote host defined by the HOST environment variable.

graphics.iynb This jupyter notebook leverages the Python Standard Library, specifically utilizing the *os* and *json* modules for handling file operations and JSON data manipulation, respectively. This notebook's primary functionality includes reading JSON files from a designated directory, aggregating the extracted results, and preparing the data for visualization. The design of this module is intentionally kept simple and direct, focusing on efficient file operations and JSON parsing. One of the significant challenges encountered during the implementation was the efficient reading and parsing of multiple JSON files within a directory.

5.2 Evaluation

We analyse the Anonycast system from different perspectives in different environments. We evaluate the system in terms of scalability, performance, security and privacy. The scalability evaluation aims to assess the system's capacity to handle a large number of messages and clients. This capacity is tested through different message sizes and different numbers of messages retrieved in the Retrieve Messages operation. Regarding the performance evaluation, it focuses on the latency dynamics of the publish and retrieve operations across different modes, deaddrops, cryptopuzzle difficulties, and the number of allowed publishers and receivers. Lastly, we evaluate our system in terms of security and privacy and examine the system's capacity to address key security and privacy requirements mentioned in Chapter 1.

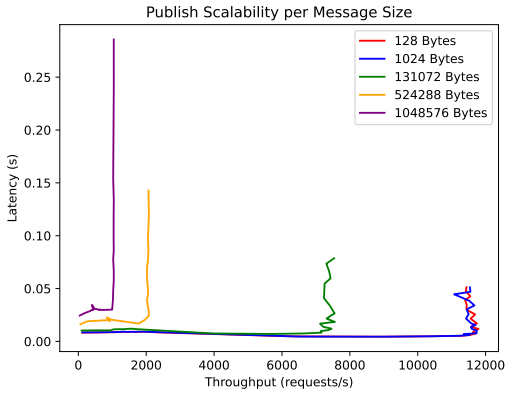

Figure 5.1: Throughput vs Latency of Publish Operation in different message sizes

5.2.1 Scalability

We evaluate our system in terms of scalability through 550 configurations of the system for both publish and retrieve operations. In these configurations, the number of messages sent and received, the size per message and the number of clients change. This test has the objective of verifying the system's performance when the number of messages, size of each message and clients increase, in terms of latency and throughput. The results of this test provided insight into how many messages a deaddrop can handle simultaneously and how easily it manages them while operating in Open Mode. This test aims to find the stress point of a deaddrop; the deaddrop does all the verifications as described in Chapter 4. It is not using the Tor network, so the latency is only related to the processing of the message and minor latency in the network; it measures the latency since the message leaves the sender until the reply is received by the sender; only the verifications by the deaddrop are considered. The machines used for this evaluation have the following specifications: CPUS AMD EPYC 7281, 16 cores, 32 threads, RAM 128 GiB DDR4 2666 MHz, $2 * 10$ Gbps Network and 1.8 TB HDD main disk.

By assessing the impact of message size on the publishing process in figure 5.1, the study evaluates message sizes from 128 bytes to 1,048,576 bytes. The results demonstrate a clear trend: as message size increases, latency also rises, something already expected. For smaller message sizes, throughput remains relatively high with minimal impact on latency. However, the main objective of this test is to identify the stress point of the system, the point from which the system starts to degrade its performance proportionally to the increase of the message size. For messages with 1MB, this point is around 1000 requests per second, where the latency is around 0.03 seconds; for messages with 500 KB, this point is around 2000 requests per second, where the latency is around 0.02 seconds; for messages with 128 KB, this point is around 7500 requests per second, where the latency is around 0.01; for messages with 1 KB and 128 bytes, this point is around 12000 requests per second, where the latency is around 0.01 seconds.

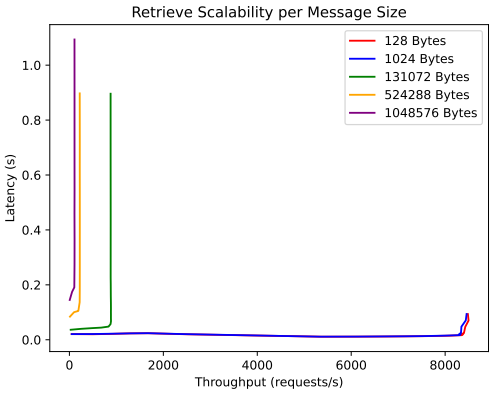

Figure 5.2: Throughput vs Latency of Retrieve Operation with different message sizes

By assessing the impact of message size on the retrieval process in figure 5.2, the study evaluates message sizes from 128 bytes to 1,048,576 bytes. The trend is similar to that observed in the publish scalability analysis, with larger message sizes leading to increased latency and reduced throughput. However, as in the previous test, the objective is to identify the stress point of the system. For messages with 1MB, this point is around 100 requests per second, where the latency is around 0.2 seconds; for messages with 500 KB, this point is around 150 requests per second, where the latency is around 0.2 seconds; for messages with 128 KB, this point is around 1000 requests per second, where the latency is around 0.08; for messages with 1 KB and 128 bytes, this point is around 8500 requests per second, where the latency is around 0.01 seconds. Note that these values include retrieves of 1, 10, 100, and 1000 messages at a time, meaning, some requests contain up to 1GB only in content data which justifies the numbers obtained. In figure 5.3 we do the same analysis but from the perspective of the number of messages retrieved, meaning, each line represents the retrieve of X messages with sizes varying from 128 bytes to 1MB. The trend is consistent with previous tests: as the number of messages retrieved increases, latency also increases while throughput decreases. Regarding the stress point of the system, for 1 message retrieved per request, the system can handle around 9000 requests per second; for 10 messages retrieved per request the system can handle around 8500 requests per second, for 100 messages retrieved per request, the system can handle around 5800 requests per second and for 1000 messages retrieved per request the system can handle around 300 messages per second, all with an average latency below 0.8 seconds.

5.2.2 Performance

In this section, we evaluate the Anonycast system in terms of performance. We analyse the performance of the main algorithms used in Anonycast, such as the Ring Signature, the RSA signature, the Cryptopuzzle, and the Broadcast Encryption, and we analyse the time required for both the publish message operation and retrieve messages operation to

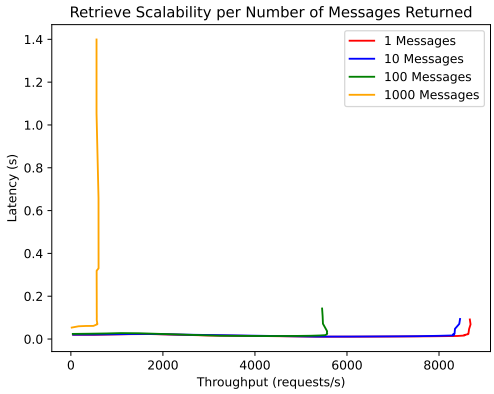

Figure 5.3: Throughput vs Latency of Retrieve Operation by number of messages retrieved

Size (bytes)	Time (s)
128	0.029124264
1024	0.030097788
4096	0.029376931
8192	0.029558019999999997
32768	0.030439215999999998
65536	0.031967059
131072	0.032250268
524288	0.039310591
1048576	0.048152831

Table 5.1: Average Signing Message Time by Message Size

complete. The machine used for this experiment was a HP Spectre x360 14-ef0004np (13.5" - Intel Core i7-1255U - RAM: 16 GB - 1 TB SSD PCIe - Intel Iris Xe Graphics).

5.2.2.1 Main Algorithms Performance

In this section, we evaluate the performance of the main algorithms used in the Anonycast system, such as the Ring Signature, the RSA signature, the Cryptopuzzle, and the Broadcast Encryption. We analyse the performance of these algorithms across different message sizes.

In Table 5.1, we analyse the performance of the RSA signature algorithm across different message sizes. The results show that the time required to sign a message increases with the size of the message. For messages with 128 bytes, the average time to sign a message varies from 0.029 to 0.048 seconds, corresponding to messages of size 128 bytes and 1MB, respectively.

In Figure 5.4, we analyse the performance of the Ring Signature algorithm across different message sizes and for different numbers of allowed publishers. As expected, as the number of allowed publishers increases, the time required to sign a message also increases, varying from almost 0 to 0.05 seconds. Similarly, the time required to sign a message increases with the size of the message. Another interesting point is that as the

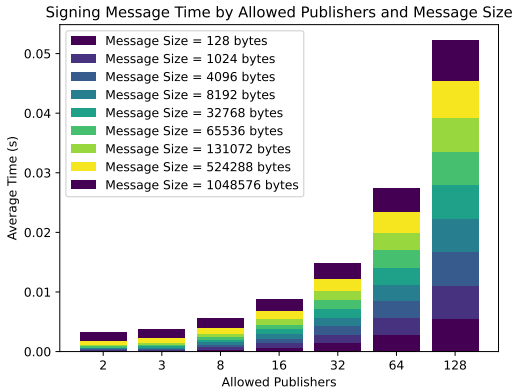

Figure 5.4: Average Signing Message Time by Number of Allowed Publishers and Message Size

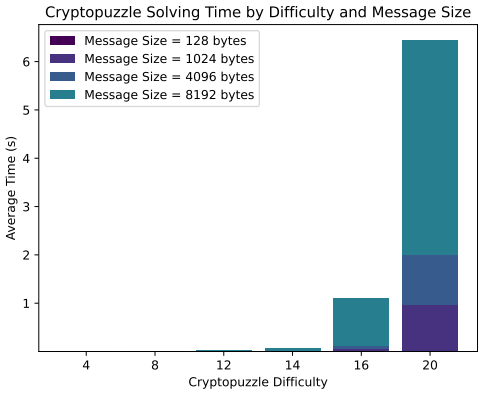

Figure 5.5: Average Cryptopuzzle Solving Time by Difficulty and Message Size

number of allowed publishers increases, the difference in time required to sign a message between different message sizes also increases.

In Figure 5.5, we analyse the performance of the Cryptopuzzle algorithm across different message sizes and for different difficulties. We can see that the time required to solve a cryptopuzzle increases with the difficulty of the cryptopuzzle and the size of the message. For messages with 128 bytes (which was the size used for the results in Subsubsection 5.2.2.2), the difference in the average time to solve the cryptopuzzle across the different difficulties is minimal, varying from almost 0 to 0.7 seconds. For messages with 8192 bytes, and cryptopuzzle difficulty 20, the average time to solve the cryptopuzzle is 6.5 seconds.

In Figure 5.6, we analyse the performance of the Broadcast Encryption algorithm across different numbers of allowed receivers and with only message size. In this case, we only use the message size of 32 bytes because is the size of the symmetric key used in the encryption of the message. As expected, this test shows that the time required to encrypt a message increases with the number of allowed receivers as more copies of the

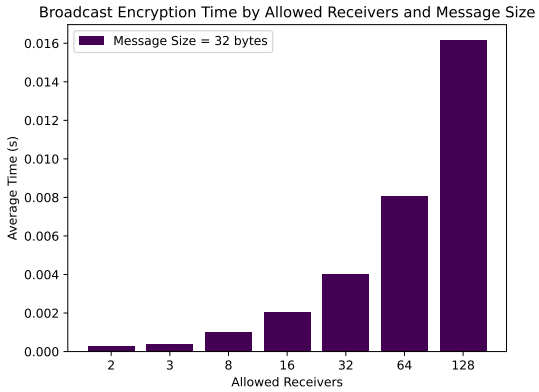

Figure 5.6: Average Broadcast Encryption Time by Number of Allowed Publishers and Message Size

symmetric key need to be created.

5.2.2.2 Publish and Retrieve Message Operation Performances

In this section, we evaluate the performance of the Anonycast system in terms of the time required to complete the publish message operation and the retrieve messages operation for one message with 128 bytes. For the publish message operation, we measure the time required to sign and encrypt the message, solve the cryptopuzzle, send the message to the deaddrops, and receive a confirmation of the success or failure of the operation. Note that the time required to encrypt the message is only considered in the Receiver Restricted modes, and the signature method changes from the Publisher Restricted modes to the Open Mode and Receiver Restricted modes. For the retrieve messages operation, we measure the time to solve the cryptopuzzle, request the message IDs from one topic, organize the message IDs, request the messages themselves, receive the messages, and validate them. The number of messages retrieved during this test was one. We measure the influence of the number of deaddrops, the mode of operation, and the number of allowed publishers and receivers in the latency of the operation as well correlate these results with the results obtained in Subsubsection 5.2.2.1.

Number of deaddrops In this paragraph, we analyse the impact of the number of deaddrops in the latency of both Publish and Retrieve operation. The objective of this test is to understand how the increase of the number of deaddrops affects the latency of both the Publish Message operation and Retrieve Message operation. In Tables 5.7 and 5.2, we verify that the number of deaddrops has a direct impact on the latency of the publish operation, as the increase in the number of deaddrops translates into more latency, ranging from 2.89 seconds for three deaddrops to 4.14 seconds for nine deaddrops. Despite the messages being published at the same time to all deaddrops, the confirmation that the message was successfully published is received from each deaddrop separately, meaning that the time

to conclude the operation is defined by the deaddrop that takes the longest to confirm the operation; we define a maximum limit for this time to guarantee that the operation ends in the presence of malicious deaddrops. Due to the nature of the Tor network, each deaddrop can be located in a different part of the world, meaning that the greater the number of deaddrops is, the greater the probability of a deaddrop being located in a distant Onion Service is, and therefore, the greater the probability of the operation taking longer. Another factor that increases the probability of a deaddrop being located in a distant Onion Service is the location of our tests, Portugal, which is far from the majority of the Tor relays, which are located in the United States and Central Europe. The latency of the Publish operation is given by the time to sign and encrypt the message (if applicable), solve the cryptopuzzle, send the message to the deaddrops, deaddrops validate the message and send a confirmation of the success or failure of the operation to the publisher. The total time that a publisher waits for deaddrops to validate the message and send a confirmation is given by the formula $MAX(latency_{deaddrop1}, latency_{deaddrop2}, ..., latency_{deaddropN})$.

Regarding the retrieve operation, we observe in Tables 5.8 and 5.3 that the number of deaddrops also has a direct impact on the latency of the operation, as the increase in the number of deaddrops translates into more latency, ranging from 4.03 seconds for three deaddrops until 5.12 seconds for nine deaddrops. The Retrieve message operation is even more affected by the number of deaddrops as there are more steps dependent on them. When a receiver requests the message IDs from the deaddrops, it waits for all deaddrops to organize the message IDs and decide which deaddrops to contact for which messages. As mentioned in Subsection 4.4.6 this happens to avoid asking the same messages to different deaddrops, which would be a waste of resources. This waiting time is the time of the deaddrop that takes the longest to collect the message IDs from a given topic. Another step that is affected by the number of deaddrops is the request for the messages themselves. The receiver requests the messages to the different deaddrops and waits for all of them to send the messages. Only after it validates all the messages. The time that a receiver waits for the messages to be sent is also the time of the deaddrop that takes the longest to send the messages associated with the topic. Thereafter, the receiver can validate the messages and the time of the retrieve operation is collected.

This helps us to understand that the Tor network is a crucial factor in the latency of the operation, as many steps of communication of both Publish and Retrieve operations are dependent on the deaddrops and therefore on the Tor network, considering they are Onion Services. The dynamism of the Tor network is much more significant than variations in parameters of the Anonycast system itself, in terms of latency.

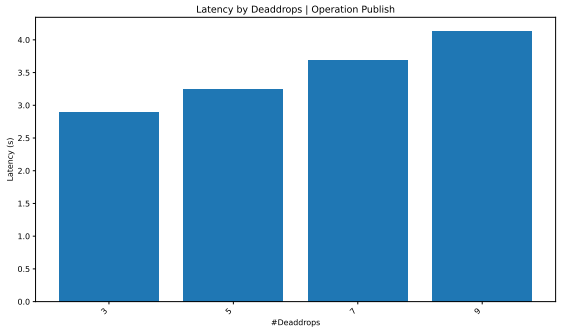

Figure 5.7: Latency of Publish Operation by number of deaddrops

#Deaddrops	Average Latency (s)
3	2.89130435955291
5	3.2514615109365077
7	3.6891648303395064
9	4.137455008471782

Table 5.2: Latency of Publish Operation by number of deaddrops

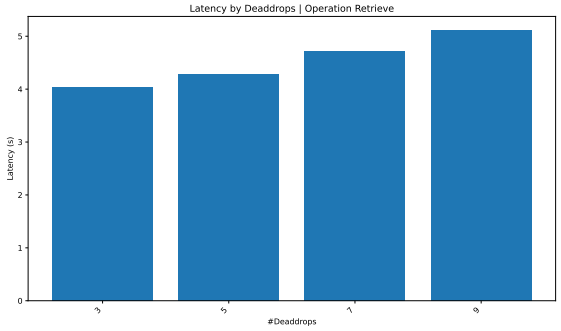

Figure 5.8: Latency of Retrieve Operation by number of deaddrops

#Deaddrops	Average Latency (s)
3	4.034793780592593
5	4.287767618833334
7	4.710993850685185
9	5.118584481443563

Table 5.3: Latency of Retrieve Operation by number of deaddrops

Mode of Operation In this paragraph, we analyse the impact of the mode of operation on the latency of the operation. The objective of this test is to understand how the increase of the restrictions in the system affects the latency of both Publish Message operation and Retrieve Message operation. In Figures 5.9 and 5.10, we can observe the trend of the latency of the operation in different modes; the results show that the restricted modes have a higher latency than the open mode, for both Publish and Retrieve Message operation. This is a natural result, as the more restrictions are imposed on the system, the more steps are required to complete the operation.

Another interesting point is that the difference in latency between the modes is more significant in the retrieve operation than in the publish operation. This matches the results obtained in paragraph 5.2.2.2 and it is justified by the higher number of steps involved in the retrieve operation. An additional conclusion that we take from this test is that the average latency of both Publish and Retrieve operations is higher in Receiver Restricted Mode than in Sender Restricted Mode. In Sender Restricted Mode, the only change to the Open Mode is the signature method, being RSA in Open Mode and Ring Signature in Sender Restricted Mode; the receivers of the message and the deaddrops take longer to verify the message compared to the Open Mode but the publisher does not need to any additional step despite having a list of receivers updated which does not influence the latency of the operation. In Receiver Restricted Mode, the publisher needs to encrypt the message with a symmetric key and create a copy of the symmetric key for each receiver. Additionally, the receivers need to find their copy of the symmetric key and decrypt the message. Note that the message is also signed in Receiver Restricted Mode, with RSA in this case. This justifies the higher latency in Receiver Restricted Mode than in Sender Restricted Mode. In Sender and Receiver restricted mode, the combination of both Ring Signature method and Broadcast Encryption method used, justifies the higher latency in this mode as includes the additional steps of both Sender Restricted and Receiver Restricted modes.

In Tables 5.4 and 5.5, we can see the specific latencies of the operation in different modes, the publish operation ranges from 1.87 seconds in Open Mode to 3.6 seconds in Sender and Receiver Restricted Mode, and the retrieve operation range from 3 seconds in Open Mode to 4.7 seconds in Sender and Receiver Restricted Mode. In Open Mode, the difference between the Publish and Retrieve operations is about 1.1 seconds, in Sender Restricted Mode the difference is about 0.9 seconds, in Receiver Restricted Mode the difference is about 0.9 seconds and in Sender and Receiver Restricted Mode, the difference is about 1.1 seconds.

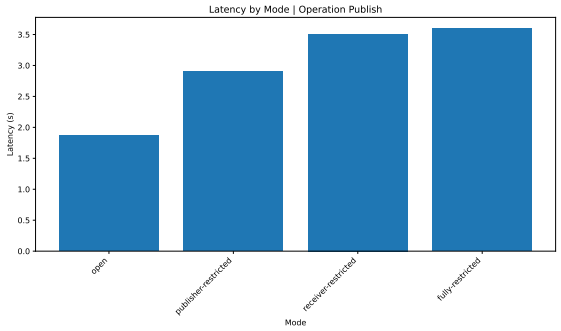

Figure 5.9: Latency of Publish Operation in different modes

Mode	Average Latency (s)
open	1.8651429442083334
publisher-restricted	2.902950752925926
receiver-restricted	3.5102789033697914
fully-restricted	3.596932213182581

Table 5.4: Latency of Publish Operation in different modes

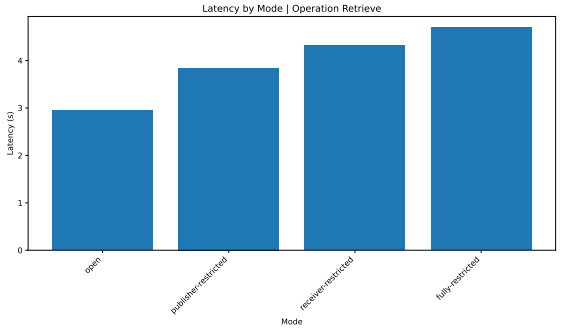

Figure 5.10: Latency of Retrieve Operation in different modes

Mode	Average Latency (s)
open	2.9513449083055554
publisher-restricted	3.8479577099976847
receiver-restricted	4.3127403982239585
fully-restricted	4.694899717039641

Table 5.5: Latency of Retrieve Operation in different modes

Number of Allowed Senders and Receivers In this paragraph, we understand how the combined impact of the number of allowed publishers and receivers in the latency of the operation and which of them has more impact. In figures 5.11 and 5.12 we observe the trend of the combined impact of both number of allowed publishers and receivers in the latency of the operation. By analysing these figures, we conclude that the increase in the number of allowed publishers and receivers tends to increase the latency of the operation, as anticipated in previous tests. In Subsubsection 5.2.2.1 we observed that the time required to sign a message in Sender Restricted modes would increase with the number of allowed publishers; we also observed that the time required to encrypt a message in Receiver Restricted modes would increase with the number of allowed receivers.

Regarding the most impactful we can see that the latency of the operation is more affected by the number of allowed receivers than the number of allowed publishers, matching the results obtained in the previous test. Despite the broadcast encryption algorithm being more scalable than the Ring Signature algorithm used, according to our results in Subsubsection 5.2.2.1, the influence of the number of allowed receivers is higher on the time required to complete the steps dependent on the number of allowed receivers than, the influence of the number of allowed publishers on the time required to complete the steps dependent on the number of allowed publishers. The specific latencies can be seen in the tables 5.6 and 5.7, ranging from 2.5 to 4.7 in the publish operation and from 3.4 to 6.1 in the retrieve operation.

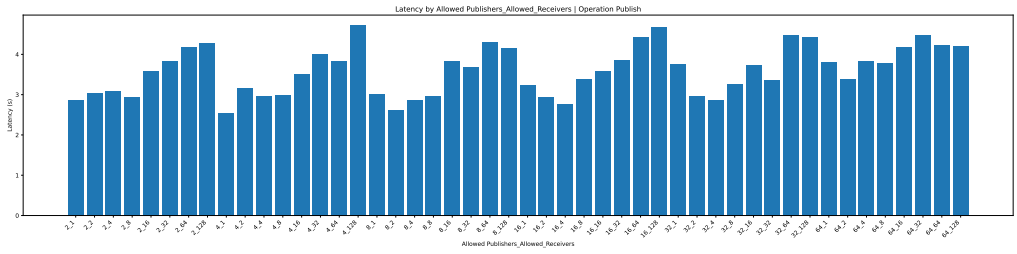

Figure 5.11: Latency of Publish Operation by number of allowed publishers and receivers in Sender and Receiver Restricted Mode

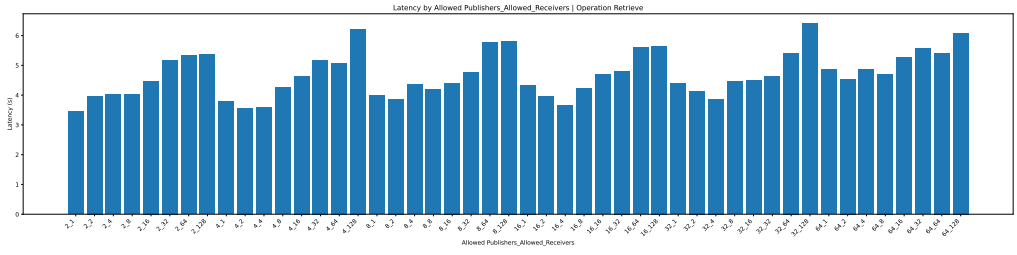

Figure 5.12: Latency of Retrieve Operation by number of allowed publishers and receivers in Sender and Receiver Restricted Mode

Allowed Publishers_Allowed Receivers	Average Latency (s)	Allowed Publishers_Allowed Receivers	Average Latency (s)
2_1	2.8652611484583335	16_1	3.2362181635416665
2_2	3.029491099833333	16_2	2.933155228222222
2_4	3.09235441295833	16_4	2.773400282736111
2_8	2.931228529	16_8	3.381847704777778
2_16	3.592859126541667	16_16	3.587083589583333
2_32	3.833669467194444	16_32	3.859835172361111
2_64	4.193848322402778	16_64	4.429844479652778
2_128	4.279201200569444	16_128	4.667124821277778
4_1	2.535261851125	32_1	3.7602092735833335
4_2	3.1758514363194443	32_2	2.976414159138889
4_4	2.9762849658611112	32_4	2.870407526930556
4_8	2.9849576446944446	32_8	3.265302700125
4_16	3.510426922027778	32_16	3.740616784861111
4_32	4.0194082588611115	32_32	3.3535692936111112
4_64	3.842129980375	32_64	4.475919621430555
4_128	4.738120251972222	32_128	4.431093764847222
8_1	3.0059436895416667	64_1	3.8076306659166668
8_2	2.6081913409444444	64_2	3.3926195196111113
8_4	2.856600978680555	64_4	3.822436349638888
8_8	2.971574992541666	64_8	3.7842652453333336
8_16	3.824699270763889	64_16	4.177350893458334
8_32	3.690342943333333	64_32	4.476301566861111
8_64	4.318004158041667	64_64	4.224940003527778
8_128	4.150357357263888	64_128	4.197090072430556

Table 5.6: Latency of Publish Operation by number of allowed publishers and receivers in Sender and Receiver Restricted Mode

Allowed Publishers_Allowed Receivers	Average Latency (s)	Allowed Publishers_Allowed Receivers	Average Latency (s)
2_1	3.458650260930556	16_1	4.343352967083333
2_2	3.974839211083333	16_2	3.951599310638888
2_4	4.012426579638885	16_4	3.657195479444443
2_8	4.026114601291667	16_8	4.246071749458333
2_16	4.473795144569444	16_16	4.693123538208334
2_32	5.174920291819444	16_32	4.809458979944444
2_64	5.336250671861111	16_64	5.596784282138889
2_128	5.368647524527777	16_128	5.653419441861111
4_1	3.7890929314305555	32_1	4.395506346888889
4_2	3.5748416446944447	32_2	4.1356952160416665
4_4	3.591498105458333	32_4	3.872943178222222
4_8	4.250253339375	32_8	4.466493378638889
4_16	4.647660199833333	32_16	4.496657452222222
4_32	5.182969693708333	32_32	4.622750993597222
4_64	5.079214532763889	32_64	5.418835588083333
4_128	6.21426948048611	32_128	6.411554258125
8_1	3.987203259811113	64_1	4.855460095097222
8_2	3.87503870325	64_2	4.540866615319445
8_4	4.353317937777778	64_4	4.855035666902777
8_8	4.190853679583336	64_8	4.705480508416667
8_16	4.399847330916667	64_16	5.257940231597222
8_32	4.758342071458333	64_32	5.567497544263889
8_64	5.760123460694445	64_64	5.422285396277778
8_128	5.817046753402778	64_128	6.081960791638888

Table 5.7: Latency of Retrieve Operation by number of allowed publishers and receivers in Sender and Receiver Restricted Mode

Summary The performance evaluation of the Anonycast system reveals several insights into its operational efficiency and scalability. By analyzing the performance of key algorithms such as the Ring Signature, RSA signature, Cryptopuzzle, and Broadcast Encryption, we have established an understanding of how these components behave under varying conditions. Our findings indicate that the time required for cryptographic operations, including signing and encryption, increases with the size of the message and the number of allowed publishers and receivers. This is evident in the Ring Signature and Broadcast Encryption algorithms, where the computational overhead grows with the number of participants.

The evaluation of the publish and retrieve message operations shows the impact of system parameters on latency. The number of deaddrops, mode of operation, and the number of allowed publishers and receivers all contribute to the overall latency of these operations. The latency of both Publish and Retrieve operations is primarily influenced by the deaddrop that takes the longest to confirm the operation, highlighting the variability introduced by the Tor network's global distribution.

Our analysis also reveals that the mode of operation significantly affects latency,

with restricted modes exhibiting higher latencies due to additional cryptographic steps. The Receiver Restricted Mode, in particular, incurs higher latency due to the need for symmetric key encryption and decryption. The combined impact of the number of allowed publishers and receivers shows that while both factors increase latency, the number of allowed receivers has a bigger impact, due to the the overhead of managing multiple symmetric keys in the Broadcast Encryption algorithm.

In conclusion, the Anonycast system demonstrates performance and scalability, capable of handling several messages with relatively low latency. The primary source of latency is attributed to the Tor network, which accounts for up to ninety-eight per cent of the total operation time while providing us with several important security and privacy features. This dependency on the Tor network suggests that future improvements in Tor's performance could directly enhance the efficiency of the Anonycast system. Our findings provide a foundation for further optimization and adaptation of the system, ensuring its continued relevance and effectiveness as advancements in related research areas progress. The insights gained from this evaluation not only validate the current design and implementation of Anonycast but also offer guidance for future enhancements aimed at reducing latency and improving overall system performance.

5.2.3 Security and Privacy

In this section, we evaluate the Anonycast system in terms of security and privacy. We analyse the system's ability to address the requirements outlined in section 1.2 and the Threat Model of Anonycast.

5.2.3.1 Threat Model

We consider an adversary that can control up to $N - 1$ deaddrops, where N is the total number of deaddrops assigned to a topic. The adversary can also control all publishers and receivers in the system. The adversary can also control a fraction of the network and eavesdrop on a fraction of the messages sent in the network. The adversary can also perform a DDoS attack on the network. We assume the adversary is computationally bounded such that they cannot break the well-known cryptographic assumptions.

5.2.3.2 Security and Privacy Requirements

Linkability Anonycast **MUST NOT** reveal the IP of any publisher, receiver, or deaddrop to any other publisher, receiver or deaddrop during the communication process (requirement two). We achieve this through the use of Tor Onion Services. As referenced in Subsection 2.5.1.4, Tor Onion Services provides a way to host services in a way that both the server and the client remain anonymous. Given that the deaddrops are only available through Tor Onion Services, we ensure that the IP of the deaddrop is not revealed to any publisher or receiver. By ensuring that the communication between publishers, receivers,

and deaddrops occurs over Tor, we effectively mitigate the risk of linkability by the adversary. This is because, even if the adversary can observe traffic, the layered encryption and routing through multiple nodes within the Tor network obscure the source, destination, and content of the messages. Consequently, this prevents the adversary from linking any two parties in the communication or determining the content of their communication. This mitigation strategy is essential for preserving the confidentiality and integrity of the messages against the described adversary, thereby upholding the system's privacy and security objectives as previously outlined.

Authentication A publisher or receiver **MUST** be able to ensure that it is communicating with the intended deaddrop(s) (requirement six). The publisher or receiver has a list of the Onion Addresses of the deaddrops that store information about the topic(s) they are interested in. These onion addresses are public keys which are used for deaddrop authentication. A receiver of a message **MUST** be able to ensure that the message was sent by an allowed publisher when operating in Publisher Restricted Mode or Publisher and Receiver Restricted Mode. We achieve this through the use of ring signatures that allow us to verify that a message was sent by one of the allowed publishers. The adversary's ability to control a significant portion of the system's infrastructure, including the potential for monitoring all network traffic, underscores the importance of ensuring that communications between publishers, receivers, and deaddrops are authenticated securely. The use of Onion Addresses as public keys for deaddrop authentication provides a layer of verification that the publisher or receiver is communicating with the intended deaddrop. This method, coupled with the use of ring signatures for verifying the authenticity of messages in Publisher Restricted modes, forms a defence against the adversary's potential to impersonate or inject messages within the network. This approach ensures that, even in the face of an adversary with significant control over the network and the ability to monitor traffic, the authenticity of communications within the system is maintained.

Integrity A publisher, receiver or deaddrop **MUST** verify all messages to ensure they have not been tampered with, maintaining message integrity (requirement three). This is achieved through Hashing. The content of the message is hashed and the hash is signed by the publisher; the receiver verifies the signature and the hash of the message to ensure that the message was not tampered with. Even in the case of an adversary that can eavesdrop on the network, the adversary cannot tamper with the message as the message contains its hash signed by a key that the adversary does not own and therefore, guarantees integrity. This integrity verification mechanism is crucial in the context of our Threat Model, where the adversary possesses the capability to eavesdrop on all network traffic and potentially control a significant portion of the system's infrastructure. Despite these capabilities, the adversary's ability to alter messages is effectively nullified by the measures in place. The use of digital signatures ensures that any tampering with the message content would invalidate the signature, as the hash of the modified message would not match the signed

hash. Moreover, this approach not only protects against external threats but also against insider threats where a compromised publisher or receiver might attempt to alter the message content. By mandating that all parties in the communication chain must verify the integrity of the messages, we create a trustless system where trust is placed not in the entities themselves but in the cryptographic protocols that underpin the system's security. This aligns with the overarching goal of our Threat Model to maintain the system's integrity, confidentiality, and availability, even in the face of a highly capable and motivated adversary. This helps us to achieve the requirement *Both receivers and deaddrops **MUST** validate the messages independently of previous validations* (requirement eleven).

Availability Anonycast **SHOULD** be resistant to attacks aimed at disrupting the system's availability (requirement four). This is achieved through the resolution of a cryptopuzzle. A user **MUST** solve a cryptopuzzle to send a message (requirement five). Moreover, the resolution time of the cryptopuzzle can be adjusted by the group owner and therefore, can be as hard as needed. Besides this, Anonycast depends on the DDoS protection from the underlying anonymity system to help with more complex attacks. The inclusion of cryptopuzzles as a mechanism to ensure availability directly addresses the capabilities of the adversary outlined in our Threat Model, particularly their ability to conduct DDoS attacks and potentially control a significant portion of the system's infrastructure. By requiring the resolution of a cryptopuzzle before a message can be sent, Anonycast has the power to control the rate at which messages can be introduced into the system, by changing the cryptopuzzle difficulty, thereby mitigating the risk of flooding attacks that could overwhelm the system's resources. This measure ensures that, even in the face of a determined adversary seeking to disrupt service, the system remains resilient and available to legitimate users. Coupled with the DDoS protection mechanisms inherent in the underlying anonymity system, such as distributed network architecture and traffic obfuscation, Anonycast's approach to ensuring availability is well-suited to counter the threats posed by the adversary described in our Threat Model.

Confidentiality Anonycast **MUST** ensure, when operating in Receiver Restricted or Sender and Receiver Restricted Mode (requirement nine), that the messages are only readable by the intended receiver(s). This is achieved through the use of broadcast encryption, where the message is encrypted with a symmetric key and the symmetric key is encrypted with the public key of the receiver(s), and therefore, only accessible by the intended receiver(s). The implementation of broadcast encryption as a means to ensure confidentiality directly addresses the capabilities of the adversary as outlined in our Threat Model. Given the adversary's ability to monitor all network traffic and potentially control a significant portion of the system's infrastructure, the encryption of messages using a symmetric key, which is then encrypted with the public key of the intended receiver(s), ensures that even if the adversary can intercept the messages, they cannot decrypt and read the contents without access to the private key of the receiver. This

method of double encryption provides a robust defence against the threat of unauthorized access and eavesdropping, safeguarding the privacy of the communication against a highly capable adversary. Furthermore, this approach to confidentiality is particularly effective in the context of our Threat Model, where the adversary's extensive capabilities might allow them to bypass simpler security measures. By requiring that the adversary would need to compromise the private key of the intended receiver(s) to decrypt the message, Anonycast significantly raises the bar for successful attacks on the system's confidentiality. This aligns with the overarching goal of our Threat Model to protect against both passive and active threats to the system, ensuring that the confidentiality of messages is maintained even in the face of sophisticated and motivated adversaries. The property of padding the messages by the Tor network also helps to achieve the confidentiality requirement.

6

Conclusion and Future Work

In this chapter, we provide a detailed summary of the findings and contributions of this work, followed by a discussion on potential future enhancements and expansions of the Anonycast system to accommodate additional use cases.

6.1 Future Work

Regarding future work, we have several ideas that can be implemented to improve the Anonycast system. One possible expansion would be to add the PBFT protocol [14] to fit other use cases, similar to the different modes we have. This would guarantee that an operation is accepted by a majority or discarded. This would require $3N - 1$ deaddrops for each topic where N is the number of possible byzantine deaddrops we would want to tolerate. A different use case than the ones used for this work. Another possible improvement would be to create a private public key infrastructure to manage the keys of the system and also the deaddrops list of each topic. A voting system to kick out members (publisher, receiver and/or deaddrops) who are not well-behaved could be also easily added to our system.

6.1.1 Adapt Tor Onion Services

Another possible solution to the problem we addressed with Anonycast is adapting Tor to allow Multicast communication. One possible solution would be when building the circuits, the sender would specify if the connection is with a group of people (services) or if it is with a single person (service). If it is with a group of people, the last onion relay in the circuit would be responsible for sending the message to all the receivers. This would require a change in the initial setup of the Tor protocol. This adaptation of the Tor Onion Services would become a crucial step to ensure the privacy and anonymity of the users even in Multicast communication. In Figure 6.1.1, you can see a diagram of the Publish Message Protocol running in Open Mode after the adaptation of the Tor Onion Services. By adding an option in Tor that allows us to build circuits where the last Tor Relay or the Rendezvous point is the same, we would enable users to publish messages to multiple

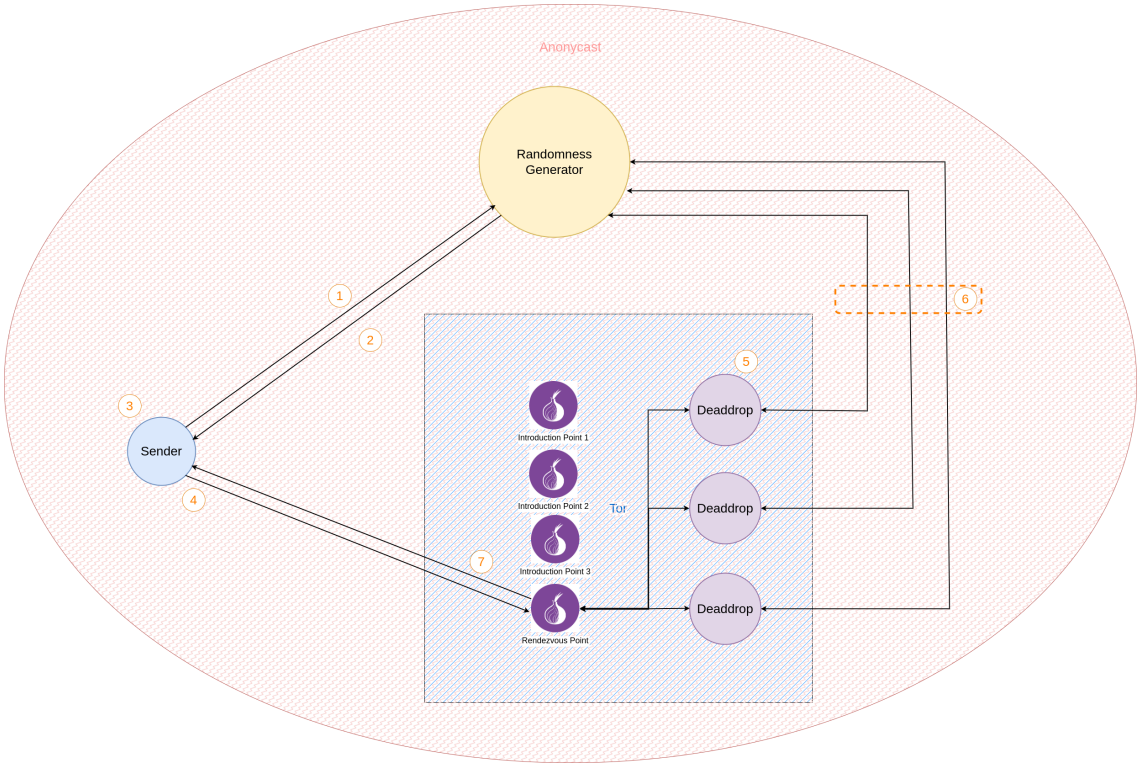

Figure 6.1: Publish Message Protocol running in Open Mode after Tor adaptation.

Onion Services without the need to build multiple Tor circuits. The biggest overhead would be the last node of the circuit and the possibility of compromising Tor's privacy and security objectives. We hope that this work can be a starting point for future research in the area of private multicast systems.

6.2 Conclusion

In this work, we have presented Anonycast, a system that allows users to publish messages and retrieve messages anonymously. We have presented and tested a modular architecture that complies with our requirements described in section 1.2, performing with low latency and high throughput, similar to other similar systems used in Unicast. We wanted our system to be helpful in the future, something capable of solving the problems of today and staying updated in the future. We achieved this by making sure that Anonycast was not dependent on any specific elements or technologies. Anonycast can use any anonymity system allowing Rendezvous Points. This flexibility also happens with all the elements and technologies used in the system, including the Randomness Generator, the Ring Signature algorithm, the Broadcast encryption algorithm and the cryptopuzzle algorithm. This flexibility is crucial for, allowing Anonycast to progress with future developments in the area of Distributed Systems. Besides this, due to its modularity, it can integrate other

systems and technologies that want to add to their system the requirements of privacy, security, performance and scalability that Anonycast delivers.

Bibliography

[1] Houghton Mifflin Harcourt, 2011 (cit. on p. 8).

[2] T. Ballardie, P. Francis, and J. Crowcroft. "Core based trees (CBT)". In: *SIGCOMM Comput. Commun. Rev.* 23.4 (1993-10), pp. 85–95. ISSN: 0146-4833. DOI: 10.1145/167954.166246. URL: https://doi.org/10.1145/167954.166246 (cit. on p. 5).

[3] A. Banks, R. Gupta, and E. Briggs. *MQTT Version 3.1.1*. Tech. rep. OASIS Standard, 2014. URL: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (cit. on p. 5).

[4] M. V. Barbera et al. "CellFlood: Attacking Tor Onion Routers on the Cheap". In: *Computer Security – ESORICS 2013*. Ed. by J. Crampton, S. Jajodia, and K. Mayes. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 664–681. ISBN: 978-3-642-40203-6 (cit. on p. 19).

[5] L. Barman et al. "Groove: Flexible Metadata-Private Messaging". In: *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, 2022-07, pp. 735–750. ISBN: 978-1-939133-28-1. URL: https://www.usenix.org/conference/osdi22/presentation/barman (cit. on pp. 2, 28).

[6] L. Basyoni et al. "Traffic Analysis Attacks on Tor: A Survey". In: *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*. 2020, pp. 183–188. DOI: 10.1109/ICIoT48696.2020.9089497 (cit. on p. 18).

[7] A. Bikos et al. "Random Number Generators: Principles and Applications". In: *Cryptography* 7 (2023-10), p. 54. DOI: 10.3390/cryptography7040054 (cit. on p. 6).

[8] A. Biryukov, I. Pustogarov, and R.-P. Weinmann. "Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013 (cit. on p. 19).

[9] B. H. Bloom. "Space/time trade-offs in hash coding with allowable errors". In: *Commun. ACM* 13.7 (1970-07), pp. 422–426. ISSN: 0001-0782. DOI: 10.1145/362686.362692. URL: https://doi.org/10.1145/362686.362692 (cit. on p. 27).

[10] D. Boneh and M. Franklin. "Identity-Based Encryption from the Weil Pairing". In: *Advances in Cryptology — CRYPTO 2001*. Ed. by J. Kilian. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 213–229. ISBN: 978-3-540-44647-7 (cit. on p. 13).

[11] D. Boneh, C. Gentry, and B. Waters. "Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys". In: *Advances in Cryptology – CRYPTO 2005*. Ed. by V. Shoup. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 258–275. ISBN: 978-3-540-31870-5 (cit. on p. 13).

[12] S. O. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. RFC 2119. 1997-03. DOI: 10.17487/RFC2119. URL: https://www.rfc-editor.org/info/rfc2119 (cit. on pp. 2, 3).

[13] J. Brendel et al. "The Provable Security of Ed25519: Theory and Practice". In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021, pp. 1659–1676. DOI: 10.1109/SP40001.2021.00042 (cit. on p. 10).

[14] M. Castro and B. Liskov. "Practical Byzantine Fault Tolerance". In: *OSDI* (1999-03) (cit. on p. 60).

[15] A. Chator, M. Green, and P. R. Tiwari. "SoK: Privacy-Preserving Signatures". In: *IACR Cryptol. ePrint Arch.* 2023 (2023), p. 1039. URL: https://api.semanticscholar.org/CorpusID:259336889 (cit. on p. 11).

[16] D. L. Chaum. "Untraceable electronic mail, return addresses, and digital pseudonyms". In: *Commun. ACM* 24.2 (1981-02), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/358549.358563. URL: https://doi.org/10.1145/358549.358563 (cit. on pp. 1, 24).

[17] Y.-P. Chiu, C.-L. Lei, and C.-Y. Huang. "Secure Multicast Using Proxy Encryption". In: *Information and Communications Security*. Ed. by S. Qing et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 280–290. ISBN: 978-3-540-32099-9 (cit. on p. 2).

[18] Y.-h. Chu et al. "A case for end system multicast". In: *IEEE Journal on Selected Areas in Communications* 20.8 (2002), pp. 1456–1471. DOI: 10.1109/JSAC.2002.803066 (cit. on p. 5).

[19] T. aes-gcm Contributors. *aes-gcm: AES-GCM encryption and decryption*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/aes-gcm (cit. on p. 42).

[20] T. anyhow Contributors. *anyhow: Simplified error handling*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/anyhow (cit. on p. 43).

[21] T. clap Contributors. *clap: Command-line argument parsing*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/clap (cit. on p. 43).

[22] T. curve25519-dalek Contributors. *curve25519-dalek: Curve25519 elliptic curve operations*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/curve25519-dalek (cit. on p. 42).

[23] T. drand-client-rs Contributors. *drand-client-rs: Rust client for drand randomness beacon*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/drand-client-rs (cit. on p. 43).

[24] T. hex Contributors. *hex: Hexadecimal encoding and decoding*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/hex (cit. on pp. 42, 43).

[25] T. jemallocator Contributors. *jemallocator: Jemalloc allocator*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/jemallocator (cit. on p. 43).

[26] T. nazgul Contributors. *nazgul: (specific details needed)*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/nazgul (cit. on p. 42).

[27] T. openssl Contributors. *openssl: SSL and TLS protocols*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/openssl (cit. on p. 43).

[28] T. rand Contributors. *rand: Random number generation*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/rand (cit. on p. 42).

[29] T. rayon Contributors. *rayon: Data parallelism library*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/rayon (cit. on p. 43).

[30] T. reqwest Contributors. *reqwest: HTTP client with JSON and blocking features*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/reqwest (cit. on p. 43).

[31] T. ring Contributors. *ring: Cryptographic operations*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/ring (cit. on p. 42).

[32] T. rsa Contributors. *rsa: RSA encryption and decryption*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/rsa (cit. on p. 42).

[33] T. serde Contributors. *serde: Serialization framework*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/serde (cit. on pp. 42, 43).

[34] T. sha2 Contributors. *sha2: SHA-2 hashing*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/sha2 (cit. on pp. 42, 43).

[35] T. tokio Contributors. *tokio: Asynchronous runtime*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/tokio (cit. on p. 43).

[36] T. tracing Contributors. *tracing: Structured logging*. Accessed: 2024-08-22. 2024. URL: https://crates.io/crates/tracing (cit. on p. 43).

[37] G. Danezis, R. Dingledine, and N. Mathewson. "Mixminion: design of a type III anonymous remailer protocol". In: *2003 Symposium on Security and Privacy, 2003.* 2003, pp. 2–15. DOI: 10.1109/SECPRI.2003.1199323 (cit. on p. 1).

[38] S. E. Deering and D. R. Cheriton. "Multicast routing in datagram internetworks and extended LANs". In: *ACM Trans. Comput. Syst.* 8.2 (1990-05), pp. 85–110. ISSN: 0734-2071. DOI: 10.1145/78952.78953. URL: https://doi.org/10.1145/78952.78953 (cit. on p. 5).

[39] C. Delerablée. "Identity-Based Broadcast Encryption with Constant Size Ciphertexts and Private Keys". In: *Advances in Cryptology – ASIACRYPT 2007*. Ed. by K. Kurosawa. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 200–215. ISBN: 978-3-540-76900-2 (cit. on p. 13).

[40] C. Diaz, H. Halpin, and A. Kiayias. "The Nym Network". In: (2021) (cit. on pp. 2, 26).

[41] W. Diffie and M. E. Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976-11), pp. 644–654 (cit. on pp. 16, 22).

[42] R. Dingledine, N. Mathewson, and P. Syverson. "Tor: The Second-Generation Onion Router". In: *13th USENIX Security Symposium (USENIX Security 04)*. San Diego, CA: USENIX Association, 2004-08. URL: https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router (cit. on pp. 1, 2, 14, 15).

[43] L. Dondeti, S. Mukherjee, and A. Samal. "A dual encryption protocol for scalable secure multicasting". In: *Proceedings IEEE International Symposium on Computers and Communications (Cat. No.PR00250)*. 1999, pp. 2–8. DOI: 10.1109/ISCC.1999.780748 (cit. on pp. 2, 23).

[44] J. R. Douceur. "The Sybil Attack". In: *Peer-to-Peer Systems*. Ed. by P. Druschel, F. Kaashoek, and A. Rowstron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 251–260. ISBN: 978-3-540-45748-0 (cit. on p. 19).

[45] M. Edman and B. Yener. "On anonymity in an electronic society". In: *ACM Computing Surveys* 42 (2009-12), pp. 1–35. DOI: 10.1145/1592451.1592456 (cit. on p. 1).

[46] T. Elgamal. "A public key cryptosystem and a signature scheme based on discrete logarithms". In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472. DOI: 10.1109/TIT.1985.1057074 (cit. on p. 21).

[47] L. of Entropy. *Drand: Distributed Randomness Beacon*. https://drand.love/. Accessed: 2024-08-29. 2024 (cit. on p. 7).

[48] P. Eugster and R. Guerraoui. "Probabilistic multicast". In: *Proceedings International Conference on Dependable Systems and Networks*. 2002, pp. 313–322. DOI: 10.1109/DSN.2002.1028915 (cit. on p. 2).

[49] P. T. Eugster et al. "The many faces of publish/subscribe". In: *ACM Comput. Surv.* 35.2 (2003-06), pp. 114–131. ISSN: 0360-0300. DOI: 10.1145/857076.857078. URL: https://doi.org/10.1145/857076.857078 (cit. on p. 5).

[50] B. Fenner and M. Handley. *Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification (Revised)*. Tech. rep. RFC 4601. Internet Engineering Task Force (IETF), 2006. URL: https://www.rfc-editor.org/rfc/rfc4601 (cit. on p. 5).

[51] H.-W. Ferng and C.-C. Peng. "Traffic splitting in a network: split traffic models and applications". In: *Computer Communications* 27.12 (2004). Advances in Computer Communications, pp. 1152–1165. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2004.02.018. URL: https://www.sciencedirect.com/science/article/pii/S0140366404001094 (cit. on p. 25).

[52] P. S. Foundation. *argparse: Command-line parsing*. Accessed: 2024-08-22. 2024. URL: https://docs.python.org/3/library/argparse.html (cit. on p. 43).

[53] P. S. Foundation. *asyncio: Asynchronous I/O*. Accessed: 2024-08-22. 2024. URL: https://docs.python.org/3/library/asyncio.html (cit. on p. 43).

[54] P. S. Foundation. *logging: Configurable event logging*. Accessed: 2024-08-22. 2024. URL: https://docs.python.org/3/library/logging.html (cit. on p. 43).

[55] "Broadcast Encryption". In: *Encyclopedia of Multimedia*. Ed. by B. Furht. Boston, MA: Springer US, 2006, pp. 45–46. ISBN: 978-0-387-30038-2. DOI: 10.1007/0-387-30038-4_15. URL: https://doi.org/10.1007/0-387-30038-4_15 (cit. on p. 12).

[56] I. Goldberg. "On the Security of the Tor Authentication Protocol". In: vol. 4258. 2006-06, pp. 316–331. ISBN: 978-3-540-68790-0. DOI: 10.1007/11957454_18 (cit. on p. 19).

[57] V. Goyal et al. "Attribute-based encryption for fine-grained access control of encrypted data". In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS '06. Alexandria, Virginia, USA: Association for Computing Machinery, 2006, pp. 89–98. ISBN: 1595935185. DOI: 10.1145/1180405.1180418. URL: https://doi.org/10.1145/1180405.1180418 (cit. on p. 13).

[58] C. Gulcu and G. Tsudik. "Mixing E-mail with Babel". In: *Proceedings of Internet Society Symposium on Network and Distributed Systems Security*. 1996, pp. 2–16. DOI: 10.1109/NDSS.1996.492350 (cit. on p. 1).

[59] E. J. Harder and D. M. Wallner. *Key Management for Multicast: Issues and Architectures*. RFC 2627. 1999-06. DOI: 10.17487/RFC2627. URL: https://www.rfc-editor.org/info/rfc2627 (cit. on p. 20).

[60] C. Heinrich. "Pretty Good Privacy (PGP)". In: *Encyclopedia of Cryptography and Security*. Ed. by H. C. A. van Tilborg and S. Jajodia. Boston, MA: Springer US, 2011, pp. 955–958. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_215. URL: https://doi.org/10.1007/978-1-4419-5906-5_215 (cit. on p. 24).

[61] H. W. Holbrook and B. Cain. *Source-Specific Multicast for IP*. Tech. rep. RFC 4607. Internet Engineering Task Force (IETF), 2006. URL: https://www.rfc-editor.org/rfc/rfc4607 (cit. on p. 5).

[62] M. Howard et al. *Computer security: Principles and practice*. Pearson Education, 2012 (cit. on pp. 8, 10).

[63] C.-Y. Huang et al. "Secure multicast in dynamic environments". In: *Computer Networks* 51.10 (2007), pp. 2805–2817. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2006.11.027. URL: https://www.sciencedirect.com/science/article/pii/S1389128606003677 (cit. on pp. 2, 21).

[64] D. Hugenroth, M. Kleppmann, and A. R. Beresford. "Rollercoaster: An Efficient Group-Multicast Scheme for Mix Networks". In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021-08, pp. 3433–3450. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/hugenroth (cit. on p. 2).

[65] K. Jenkins, K. Hopkinson, and K. Birman. "A gossip protocol for subgroup multicast". In: *Proceedings 21st International Conference on Distributed Computing Systems Workshops*. 2001, pp. 25–30. DOI: 10.1109/CDCS.2001.918682 (cit. on p. 2).

[66] L. Ji and M. Corson. "A lightweight adaptive multicast algorithm". In: *IEEE GLOBECOM 1998 (Cat. NO. 98CH36250)*. Vol. 2. 1998, 1036–1042 vol.2. DOI: 10.1109/GLOCOM.1998.776885 (cit. on p. 2).

[67] A. Johnson et al. "Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013 (cit. on p. 19).

[68] D. E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. USA: Addison-Wesley Longman Publishing Co., Inc., 1997. ISBN: 0201896842 (cit. on p. 7).

[69] J. Kreps, N. Narkhede, and J. Rao. *Apache Kafka: A Distributed Streaming Platform*. Tech. rep. Confluent White Paper, 2015. URL: https://kafka.apache.org/documentation/ (cit. on p. 5).

[70] D. Lazar, Y. Gilad, and N. Zeldovich. "Karaoke: Distributed Private Messaging Immune to Passive Traffic Analysis". In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, 2018-10, pp. 711–725. ISBN: 978-1-939133-08-3. URL: https://www.usenix.org/conference/osdi18/presentation/lazar (cit. on pp. 2, 26).

[71] Z. Ling et al. "Tor Bridge Discovery: Extensive Analysis and Large-scale Empirical Evaluation". In: *IEEE Transactions on Parallel and Distributed Systems* 26.7 (2015), pp. 1887–1899. DOI: 10.1109/TPDS.2013.249 (cit. on p. 19).

[72] J. K. Liu, V. K. Wei, and D. S. Wong. "Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups". In: *Information Security and Privacy*. Ed. by H. Wang, J. Pieprzyk, and V. Varadharajan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 325–335. ISBN: 978-3-540-27800-9 (cit. on p. 11).

[73] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/main/template.pdf (cit. on p. i).

[74] M. Matsumoto and T. Nishimura. "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator". In: *ACM Trans. Model. Comput. Simul.* 8.1 (1998-01), pp. 3–30. ISSN: 1049-3301. DOI: 10.1145/272991.272995. URL: https://doi.org/10.1145/272991.272995 (cit. on p. 7).

[75] J. McLachlan et al. "Scalable Onion Routing with Torsk". In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. CCS '09. Chicago, Illinois, USA: Association for Computing Machinery, 2009, pp. 590–599. ISBN: 9781605588940. DOI: 10.1145/1653662.1653733. URL: https://doi.org/10.1145/1653662.1653733 (cit. on p. 2).

[76] A. Menezes. "An Introduction to Pairing-Based Cryptography". In: 2005. URL: https://api.semanticscholar.org/CorpusID:17544631 (cit. on p. 13).

[77] S. Mittra. "Iolus: A Framework for Scalable Secure Multicasting". In: *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '97. Cannes, France: Association for Computing Machinery, 1997, pp. 277–288. ISBN: 089791905X. DOI: 10.1145/263105.263179. URL: https://doi.org/10.1145/263105.263179 (cit. on p. 22).

[78] R. Molva and A. Pannetrat. "Scalable Multicast Security with Dynamic Recipient Groups". In: *ACM Trans. Inf. Syst. Secur.* 3.3 (2000-08), pp. 136–160. ISSN: 1094-9224. DOI: 10.1145/357830.357834. URL: https://doi.org/10.1145/357830.357834 (cit. on pp. 2, 21).

[79] R. Mukherjee and J. W. Atwood. "Proxy encryptions for secure multicast key management". In: *28th Annual IEEE International Conference on Local Computer Networks, 2003. LCN '03. Proceedings.* (2003), pp. 377–384. URL: https://api.semanticscholar.org/CorpusID:12874324 (cit. on pp. 2, 21).

[80] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Accessed: 2024-06-07. 2008. URL: https://bitcoin.org/bitcoin.pdf (cit. on p. 8).

[81] D. Naor, M. Naor, and J. Lotspiech. "Revocation and Tracing Schemes for Stateless Receivers". In: *Advances in Cryptology — CRYPTO 2001*. Ed. by J. Kilian. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 41–62. ISBN: 978-3-540-44647-7 (cit. on p. 12).

[82] National Institute of Standards and Technology. *Digital Signature Standard (DSS)*. Tech. rep. FIPS PUB 186-4. U.S. Department of Commerce, 2013. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf (cit. on p. 10).

[83] National Institute of Standards and Technology. *Secure Hash Standard (SHS)*. Federal Information Processing Standards Publication FIPS PUB 180-2. U.S. Department of Commerce, 2002-08. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-2.pdf (cit. on p. 8).

[84] S. Noether and A. Mackenzie. "Ring Confidential Transactions". In: *Ledger* 1 (2016), pp. 1–18. URL: https://api.semanticscholar.org/CorpusID:892598 (cit. on p. 11).

[85] S. E. Oh et al. "DeepCoFFEA: Improved Flow Correlation Attacks on Tor via Metric Learning and Amplification". In: *2022 IEEE Symposium on Security and Privacy (SP)*. 2022, pp. 1915–1932. DOI: 10.1109/SP46214.2022.9833801 (cit. on p. 19).

[86] A. M. Piotrowska et al. "The Loopix Anonymity System". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017-08, pp. 1199–1216. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/piotrowska (cit. on pp. 2, 25–27).

[87] T. F. of the Press Foundation. *SecureDrop: An open-source whistleblower submission system*. Accessed: 2024-08-21. n.d. URL: https://freedom.press/securedrop (cit. on pp. 18, 23).

[88] T. Project. *Onion Services Overview*. Accessed: 2024-08-21. n.d. URL: https://community.torproject.org/onion-services/overview/ (cit. on pp. 15, 16, 18, 30).

[89] F. I. P. S. Publication. *Advanced Encryption Standard (AES)*. Tech. rep. 197. National Institute of Standards and Technology, 2001. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf (cit. on p. 17).

[90] D. Raposo et al. "MACHETE: Multi-path communication for security". In: *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*. 2016, pp. 60–67. DOI: 10.1109/NCA.2016.7778594 (cit. on p. 2).

[91] M. Reed, P. Syverson, and D. Goldschlag. "Anonymous connections and onion routing". In: *IEEE Journal on Selected Areas in Communications* 16.4 (1998), pp. 482–494. DOI: 10.1109/49.668972 (cit. on p. 1).

[92] R. L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Commun. ACM* 21.2 (1978-02), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: https://doi.org/10.1145/359340.359342 (cit. on p. 10).

[93] R. L. Rivest, A. Shamir, and Y. Tauman. "How to Leak a Secret". In: *Advances in Cryptology — ASIACRYPT 2001*. Ed. by C. Boyd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 552–565. ISBN: 978-3-540-45682-7 (cit. on p. 11).

[94] SerHack and ErCiccione. *Zero to Monero: Second Edition*. 2021. URL: https://www.getmonero.org/library/Zero-to-Monero-2-0-0.pdf (cit. on p. 10).

[95] A. Serjantov, R. Dingledine, and P. Syverson. "From a Trickle to a Flood: Active Attacks on Several Mix Types". In: *Information Hiding*. Ed. by F. A. P. Petitcolas. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 36–52. ISBN: 978-3-540-36415-3 (cit. on p. 25).

[96] C. Shields and B. N. Levine. "A Protocol for Anonymous Communication over the Internet". In: *Proceedings of the 7th ACM Conference on Computer and Communications Security*. CCS '00. Athens, Greece: Association for Computing Machinery, 2000, pp. 33–42. ISBN: 1581132034. DOI: 10.1145/352600.352607. URL: https://doi.org/10.1145/352600.352607 (cit. on p. 2).

[97] N. I. of Standards and T. (NIST). *Secure Hash Standard (SHS)*. Tech. rep. FIPS PUB 180-4. Includes SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512. Federal Information Processing Standards Publication, 2015-08. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (cit. on p. 8).

[98] M. Steiner, G. Tsudik, and M. Waidner. "Diffie-Hellman Key Distribution Extended to Group Communication". In: *Proceedings of the 3rd ACM Conference on Computer and Communications Security*. CCS '96. New Delhi, India: Association for Computing Machinery, 1996, pp. 31–37. ISBN: 0897918290. DOI: 10.1145/238168.238182. URL: https://doi.org/10.1145/238168.238182 (cit. on p. 22).

[99] E. Syta et al. "Scalable Bias-Resistant Distributed Randomness". In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 444–460. DOI: 10.1109/SP.2017.45 (cit. on p. 30).

[100] *TaRDIS: Trustworthy and Resilient Decentralised Intelligence for Edge Systems*. https://cordis.europa.eu/project/id/101093006. Visited on 2024-02-05 (cit. on p. 3).

[101] N. Tyagi et al. "Stadium: A Distributed Metadata-Private Messaging System". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. SOSP '17. Shanghai, China: Association for Computing Machinery, 2017, pp. 423–440. ISBN: 9781450350853. DOI: 10.1145/3132747.3132783. URL: https://doi.org/10.1145/3132747.3132783 (cit. on pp. 2, 27).

[102] A. Waksman. "A Permutation Network". In: *J. ACM* 15.1 (1968-01), pp. 159–163. ISSN: 0004-5411. DOI: 10.1145/321439.321449. URL: https://doi.org/10.1145/321439.321449 (cit. on p. 25).

[103] T. Wang et al. "Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier". In: *Proceedings of the 2014 IEEE Symposium on Security and Privacy*. 2014 (cit. on p. 19).

[104] C. K. Wong, M. Gouda, and S. Lam. "Secure group communications using key graphs". In: *IEEE/ACM Transactions on Networking* 8.1 (2000), pp. 16–30. DOI: 10.1109/90.836475 (cit. on p. 20).