universität
uulm

Institut für Verteilte Systeme, Universität Ulm

Broadcast Privacy for Blockchains

Dissertation zur Erlangung des Doktorgrades Dr. rer. nat. der Fakultät für Ingenieurwissenschaften, Informatik und Psychologie der Universität Ulm.

DAVID JOACHIM MÖDINGER
aus Schwäbisch-Gmünd, Deutschland, 2021

Amtierender Dekan: Prof. Dr.-Ing. Maurits Ortmanns

Gutachter: Prof. Dr.-Ing. Franz J. Hauck,
Institute of Distributed Systems,
Ulm University, Germany

Gutachter: Prof. Dr. Max Mühlhäuser,
Telecooperation Lab,
TU Darmstadt, Germany

Tag der Promotion: 12.07.2021

Abstract

Public blockchains have reached high popularity among technically inclined people, laypersons and researchers alike. Similarly, privacy has gained much attention in the same circles. This attention and high sensitivity of information transmitted in blockchains, lead more and more blockchain-based systems, especially cryptocurrencies, to provide privacy for their users. Popular approaches include ring signatures or zero-knowledge proofs to achieve unlinkable payments within the blockchain.

However, these systems solely examine privacy by considering the blockchain and its embedded transactions. The underlying peer-to-peer network of a public blockchain is rarely considered. This leaves the dissemination of transactions open for privacy attacks, as the IP address of the originator of a transaction can be mapped to their real-world identity.

In this thesis, we look into the important privacy aspects of broadcasting blockchain transactions. We collect and analyse data of a large blockchain network and construct a privacy-preserving latency estimator. Building on the insights gained from the analysis, we combine dining-cryptographers networks with a secret sharing technique and layer a flood-and-prune broadcast on top, to provide enforced k-anonymity to network participants.

To increase the flexibility of this approach, we extend two established privacy protocols. First, we extend a dining-cryptographers based group messaging protocol to transmit arbitrary length messages. Further, we optimize the protocol for common cases, to improve its performance for various environments, especially blockchain transaction dissemination. As a performant intermediate privacy layer, we transform adaptive diffusion from a contact graph protocol to a computer network protocol. We achieve this by changing the underlying network assumptions and the attacker model. We derive optimal forwarding probabilities based on a statistical network model of unstructured peer-to-peer networks. These two sub-protocols are combined in an intertwined layering approach to create 3P3, a flexible privacy-preserving broadcast protocol.

Lastly, to manage the groups required for 3P3 and other proposed and common protocols, we propose Pixy. Pixy is a privacy increasing group creation scheme, allowing for filtering and testing of group participants to establish trust. The system allows for smaller group sizes while maintaining privacy guarantees of previous systems, or better privacy for same-sized groups.

The software, concepts, data and models in this thesis help researchers and developers of privacy preserving network protocols. Developers can use 3P3, tuning its parameters to the needs of their network. Researchers can build on the data, concepts and models to create novel schemes and generalizations of our insights. This improves privacy for all users of modern and future networks.

Zusammenfassung

In den letzten Jahren hat das interesse an Datenschutz und Blockchain Systemen deutlich zugenommen. Sowohl Lauen, technisch interessierte als auch Forscher haben sich mit beiden Themen und ihren Anküpfungspunkten ausführlich befasst. Entwickler von Blockchain Systemen haben verschiedene Ansätze, etwa Ringsignaturen oder Zero-Knowledge-Beweise, genutzt um das Finanzgeheimnis in Blockchain Systemen zu stärken.

Die gewählten Ansätze beschränken sich in ihrer Funktionsweise jedoch auf die in der Blockchain hinterlegten Informationen. Das Peer-to-Peer-Netzwerk, das einer öffentlichen Blockchain zugrunde liegend, wird selten berücksichtigt. Das Netzwerk ist jedoch anfällig für Angriffe, welche den Datenschutz aushebeln. So ausgespähte IP-Adressen erlauben sogar Rückschlüsse auf die wahre Identität des Nutzers.

Diese Arbeit beschäftigt sich mit Aspekte des Datenschutzes bei der Verbreitung von Transaktionen durch Broadcasts. Wir sammeln und analysieren Daten in einem großen Blockchain Netzwerk und konstruieren einen geheimen und unauffälligen Latenz-Schätzer. Zudem kombinieren wir Dining-Cryptographers Netzwerke mit Shamir's Secret Sharing, für einen stärkeren Anonymitätsschutz.

Um die Flexibilität dieses Ansatzes zu verbessern, erweitern wir zwei bereits etablierte Protokolle. Wir verändern und optimieren ein modernes Protokoll zur Gruppenkommunikation, welches auf einem Dining-Cryptographers Netzwerk basiert. Dies erlaubt uns beliebig lange Nachrichten zu versenden, sowie die nötige Leistung von Teilnehmern substantiell zu verringern.

Wir transformieren Adaptive Diffusion von in ein Netzwerkprotokoll. Um dies zu erreichen, passen wir die Netzwerkannahmen und das Angreifermodell des Protokolls an. Hierfür entwickeln wir ein statistisches Netzwerkmodell von Peer-to-Peer-Netzwerken. Diese Veränderungen erlauben den Einsatz von Adaptive Diffusion als Zwischenschicht zur Erhöhung der Datensicherheit einzusetzen und dabei die Leistungseinbußen gering zu halten.

Zur Erstellung der für 3P3 nötigen Gruppen, entwickelten wir Pixy,w elches die Vertrauensbasis zwischen Teilnehmern durch Filter und Tests während der Gruppenerstellung verbessert. Dies ermöglicht kleinere, und damit effizientere, Gruppen unter Beibehaltung der Datenschutzgarantien früherer Systeme.

Die entwickelten Konzepte, Software, Daten und Modelle in dieser Arbeit helfen Forschern und Entwicklern von Netzwerkprotokollen den Datenschutz für Netzwerkteilnehmer zu verbessern. Software Entwickler können 3P3 verwenden und dessen Parameter auf die Bedürfnisse ihres Netzwerks abstimmen. Forscher können auf den Daten, Konzepten und Modellen aufbauen, um neuartige Schemata und Verallgemeinerungen unserer Erkenntnisse zu erstellen.

Publications

Some earlier versions of the material presented in this thesis have previously been published in the following publications:

Reviewed

[1] H. Kopp, D. Mödinger, F. J. Hauck, F. Kargl, and C. Bösch. "Design of a Privacy-Preserving Decentralized File Storage with Financial Incentives". In: *Proceedings of IEEE Security & Privacy on the Blockchain (IEEE S&B) (affiliated with EUROCRYPT 2017)*. IEEE, 2017.

[2] D. Mödinger, J. Dispan, and F. J. Hauck. "Shared-Dining: Broadcasting Secret Shares using Dining-Cryptographer Groups". In: *Accepted at 21st International Conference on Distributed Applications and Interoperable Systems (DAIS)*. 2021.

[3] D. Mödinger, N. Fröhlich, and F. J. Hauck. "Pixy: A Privacy-Increasing Group Creation Scheme". In: *5th International Conference on Network Security (ICNS)*. 2020.

[4] D. Mödinger and F. J. Hauck. "3P3: Strong Flexible Privacy for Broadcasts". In: *4th International Workshop on Cyberspace Security (IWCSS 2020)*. 2020.

[5] D. Mödinger, H. Kopp, F. Kargl, and F. J. Hauck. "A Flexible Network Approach to Privacy of Blockchain Transactions". In: *Proc. of the 38th IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, 2018.

[6] D. Mödinger, J.-H. Lorenz, and F. J. Hauck. "n-Adaptive Diffusion for k-Growing Networks". In: *Submitted to Plos One*. 2021.

[7] D. Mödinger, J.-H. Lorenz, R. W. van der Heijden, and F. J. Hauck. "Unobtrusive monitoring: Statistical dissemination latency estimation in Bitcoin's peer-to-peer network". In: *PLOS ONE* 15.12 (Dec. 2020), pp. 1–21.

Unreviewed

[8] D. Mödinger and F. J. Hauck. *Bitcoin Network Transaction Inv Data with Java Timestamp and Originator Id*. `https://doi.org/10.5281/zenodo.2547396`. Jan. 2019.

[9] D. Mödinger, A. Heß, and F. J. Hauck. "Arbitrary Length k-Anonymous DC Communication". In: (2021). arXiv: `2103.17091 [cs.NI]`.

Contents

I Preliminaries 1

1 Introduction 3

2 Background 7

3 Privacy Protocols 15

II 3P3: Strong Flexible Privacy 25

4 Overview 27

5 Phase I: Arbitrary Length k-Anonymity 31

6 Phase II: η-Adaptive Diffusion 39

7 Security and Privacy 59

8 Performance 65

III Privacy Extensions 73

9 Overview 75

10 Unobtrusive Monitoring 77

11 Threshold Cryptography for k-Anonymous Broadcasts 95

12 Pixy: Privacy-Increasing Group Creation 107

IV End 117

13 Conclusion and Outlook 119

Part I

Preliminaries

Chapter 1

Introduction

Cryptocurrencies and blockchains have been popular for years, not only in research and industry but also for the general population. This phenomenon went as far as many people of varying technical proficiency investing their money in cryptocurrencies hosted by blockchains.

Many blockchains contained sensitive data of their users in unprotected and unprotectable forms. Such data contains information on purchasing behavior, credit balances, and how the money was acquired [77, 95]. Many new cryptographic applications came to life to protect this sensitive information. Approaches to achieve unlinkable payments include the use of ring signatures [1, 86, 87] and zero-knowledge proofs [20, 79]. Even already existing blockchain systems, such as Bitcoin were augmented with privacy-enhancing mechanisms [26, 97].

However, these systems focus solely on privacy on the persisted blockchain, as every user can acquire and inspect this data. This publicly available data has been intensively investigated even for strong privacy-protected blockchains such as zcash [63].

Some researchers investigated the dissemination of transactions within the peer-to-peer network [21, 22, 67]. They recognized a lack of network privacy for blockchain systems, leading to the development of protocols to protect the network-layer participants. Dandelion [24] tackles some of the issues on the network layer for Bitcoin. Unfortunately Dandelion is weak against certain attacks. Monero [87] adopted a variant of the onion routing protocol I2P [34] called Kovri. Kovri provides strong privacy but is unsuited for a general broadcast abstraction.

Thesis Structure and Contributions

To fill the gap left by these systems, we propose new approaches around broadcasting messages in peer-to-peer networks. To clearly outline the problem, we give an in-depth description of the background of this thesis in Chapter 2. Building on this background, Chapter 3 gives an overview over various privacy preserving protocols, including dining-cryptographers networks and Dandelion.

The second part of this thesis details 3P3, a three-phase privacy-preserving broadcast protocol. Chapter 4 describes the design and core principles of 3P3, which are flexible parameters for various environments, a strong base protocol, based on a dining-cryptographers network, and an efficient topological privacy phase for enhanced protections against common attacks. 3P3 allows network developers to tweak the network parameters based on their performance and privacy requirements. The following chapters provide details on its design, privacy and performance.

Chapter 5 provides and discusses the necessary changes to a modern k-anonymous dining-cryptographers network to allow for arbitrary-length messages, at the cost of an additional transmission. To make arbitrary length messages possible, we introduce a message-length establishment and a privacy-preserving dissemination of validation seeds. These seeds are required to secure the transmission of the actual message. The proposed changes make the protocol suitable for a larger number of environments, such as the dynamic environment of blockchain broadcasts. Furthermore, various optimizations increase the performance of the protocol, which we show through a proof-of-concept implementation.

Chapter 6 provides a transformation of the contact-graph privacy-preserving broadcasting protocol adaptive diffusion to a network protocol. Adaptive diffusion uses attacker and network models unsuitable for computer networks, requiring changes to provided information during a protocol run, as well as different network assumptions. We generalize the network model from a regular tree to a random graph. We derive optimal forwarding probabilities for the resulting protocol, based on the distance distribution of nodes in the network. Further, we show when the optimal result can not be reached and provide an algorithm to compute the closest reachable probabilities. Lastly, we analyze the distribution of shortest paths in a simplified network modeled after the networks used by many public blockchain systems and provide a short overview over alternative distance distributions in common peer-to-peer networks. These distributions enable efficient forwarding probabilities for maximum privacy in the resulting protocol.

Chapter 7 provides the security and privacy analysis of 3P3. We show that 3P3 will successfully disseminate a message throughout the network, as long as the honest nodes form a connected sub-graph and enough honest nodes are available for a dining-cryptographers group. We analyze the privacy of 3P3 by considering different attacker models. First, we consider a global passive attacker, which can observe messages sent between nodes. We consider colluding nodes within the dining-cryptographers group as a second possible attacker. Lastly, we discuss the increased benefits of adaptive diffusion while combating a honest-but-curious attacker, which are common in public networks.

Chapter 8, on the other hand, provides and extensive performance analysis for 3P3. The chapter contains a theoretical discussion for the expected bandwidth consumption, a simulated comparison with statistical privacy-preserving broadcasting protocols and a performance evaluation of the proof-of-concept implementation of 3P3. We show that 3P3 performs well compared to statistical privacy measures proposed for blockchains in a simulated environment, with expected latencies for a full network broadcast around one second. Further, we show that the most performance-critical parts of 3P3, the dining-cryptographer network, performs well in a proof-of-concept implementation. The evaluation is performed using a reproducible setup based on container deployments with limited link capacities.

The third part of this thesis discusses additional contributions in the space of privacy-preserving broadcast technologies. Chapter 9 gives a short overview of the structure.

Chapter 10 details the collection of data within the Bitcoin network. We collected transaction dissemination data from multiple locations around the world spread out over weeks, to ensure time and space independence of any analytic results. This data provides a basis for evaluation of protocols and modeling of Bitcoin network behavior. The analytic results give a frame of reference for the performance results of 3P3, showing a possible reduction of 80% in dissemination times by using 3P3. We generalize this model of Bitcoins network behavior and build an estimator of future latencies based on local observations with little data from the network. The model captures the trend of the noisy real-world data well, but is limited to Bitcoins unique implementation and unable to reliably capture short time misbehavior or anomalies due to its abstraction.

In Chapter 11, we propose a novel protocol fusing dining-cryptographers networks and

Shamir's secret sharing technique. This system aligns the incentives of the source of a broadcast and participating nodes, by requiring at least $k - 1$ participants to broadcast their message shares before any participant can read the disseminated message. The fusion enforces k-anonymity on the scale of the whole network at the cost of reduced security guarantees within the group. The system provides an alternative to punitive misbehavior correction, as it is used in 3P3, and can be used as an alternative protocol.

The protocols using dining-cryptographers groups require groups on the network layer. Chapter 12 describes Pixy, a privacy increasing group creation scheme. The expected privacy guarantees of a group creation scheme are based on the peer-selection process. Pixy improves on a previous random selection by requiring a two-phase verification. The first phase is based on pre-established trust information, either from globally known sources, e.g., based on IP addresses, or previous protocol runs. A second phase incorporates resource testing of participants, to harden the system against single entities pretending to be multiple participants.

Chapter 2

Background

In this chapter, we introduce the notation, scenario and required background information. The background encompasses blockchain systems, probability distributions and the notion of privacy.

2.1 Notation

Table 2.1 provides an overview of less common notations used throughout this thesis.

Term	Meaning
$x =\sim \mathcal{U}(a, b)$	Choose a random real number uniformly with $x \in [a, b]$.
$i =\sim \mathcal{U}\{a, b\}$	Choose a random integer uniformly with $a \leq i \leq b$.
$E =\sim \mathcal{U}_n\{S\}$	Select n distinct elements from S uniformly at random.
$\mathbb{G} = (E, V)$	Graph $\mathbb{G}$ consisting of a set of vertices V and edges E.
G	All participants of a group of network participants.
$\mathcal{N}$	The set of neighbors chosen in our modified diffusion.
$\eta = \lvert\mathcal{N}\rvert$	Size of $\mathcal{N}$.
$\mathcal{H}(x)$	Function to compute the hash of element x.
$\text{process}(m)$	Call application logic to process the given message m.
$C_r(x)$	Commitment on value x, blinded by random factor r.
$\{X\}_i$	Value X encrypted under the public key of i.

Table 2.1: Notation used in this thesis.

2.2 Scenario

Throughout this thesis, we work with unstructured peer-to-peer networks. An unstructured peer-to-peer network does not determine the connections of its participants in an identification space, i.e., participants are free to connect to any other participant.

Nodes participating in the network usually use a gossip protocol to disseminate known participants. Each node keeps a list of known participants, either from direct contact, or

addresses they received via the gossip protocol. If they require a new connection, e.g., because a connection drops, they can connect to an arbitrary participant from the list.

The network provides a list of publicly known participants of the network, either through fixed addresses or a naming service. These nodes are the entry points of new participants. A new participant connects to an entry node, receives a list of network participants and closes the connection, to prevent overload of entry nodes. The list is then used to create new network connections to fully participate in the peer-to-peer network.

The maintained number of connections is often fixed in the client, i.e., a node only creates 8 outgoing connections. However, nodes accept additional incoming connections, leading to older nodes having more connections than new nodes.

Network Assumptions

The security functionality of the protocols we propose in this thesis requires the fulfillment of some assumptions about the underlying peer-to-peer network. For the remaining chapters, we assume that the underlying network has no multiple edges or loops and that the network remains connected, even after removing all malicious or faulty nodes. Otherwise, the network could be partitioned by malicious nodes, preventing participants of one partition from communicating via the network with any participant of the other partition. No network protocol would be able to ensure delivery to all nodes under these circumstances. Known properties of connectedness [58, Ch. 4] can consider a given rate of attackers for network creation, ensuring networks employing a strategy for creation will stay connected, even though a given fraction of nodes won't forward messages. Therefore our assumption is warranted and not a strong restriction on the underlying network.

Channels between nodes must be authenticated, encrypted and integrity protected. Such a channel can be realized using modern cryptographic schemes, e.g., via mTLS. Further, we require a public-private key pair for encryption with shared public keys among the group as well as an agreed-upon ordering within any established groups, e.g., ordered by public keys.

Unless noted differently, we assume the network underlying our protocol takes care of basic network functions and management operations, e.g., re-establishing interrupted connections, group join and leave operations, authentication and encryption. These operations can be cleanly separated from our protocols and can be provided by existing peer-to-peer networking implementations.

2.3 Blockchains

Bitcoin [84] is the first implementation of a so-called blockchain: A distributed data structure of time-stamped transactions between an indeterminate amount of users. To identify the users of the blockchain protocol, Bitcoin uses asymmetric cryptographic keys. Identities are denoted by public keys, and possession of the secret key is proven by cryptographic signatures.

The core elements of a blockchain are blocks and transactions. A transaction is an asset transfer. It can have multiple inputs and outputs. An input is a proof of ownership of a previous output, usually a signature proving the possession of the key used for the output. Inputs can only be used once for a transaction, if two transactions exist referencing the same output, only one can be valid. To decide which transactions are valid, the blocks and mining process are used.

A block is an accumulation of transactions, which also contains a hash of the previous block, forming a chain through these references. The blocks represent the consensus of the

system on which transactions are valid. To form a distributed consensus, a mechanism called proof-of-work is used in Bitcoin and most other blockchains [51]. This proof-of-work forces the participants that build a block to spend an amount of resources proportional to the available resources in the system. The required resources usually lead to a constant average rate of block creation While competing blocks, i.e., blocks containing similar but not identical sets of transaction, can be created, participants will continue the longest chain. The contribution of a majority to the longest chain forms a probabilistic consensus on all valid transactions, i.e., the probability of a block being overruled by a competing faster growing chain shrinks exponentially based on the difference in length of the competing chains.

Permission-less blockchains allow everyone to participate. New blocks and transactions need to be transmitted to all participants within such a permission-less blockchain. As, in principle, everyone can participate, the dissemination protocol is especially important.

2.4 Commitments

The protocols presented in this thesis make use of commitments. A commitment $C_r(x)$ over some data x is a unique, irreversible identifier. Given $C_r(x)$ and r, any user can validate that $C_r(x)$ was created using x.

Commitments possess two core properties: They are hiding and binding on the data x. Hiding ensures that the committed data x can not be extracted, only given the commitment $C_r(x)$. Binding, on the other hand, prevents a committed party from finding values x', r', such that $C_r(x) = C_{r'}(x')$, revealing a different value than they committed to.

We make use of the well known Pedersen commitments [90] over an elliptic curve, which fulfill this property efficiently. Given two points G, H on the elliptic curve, a data item x and a random value r the commitment is given by

$$C_r(x) = xG + rH.$$

Pedersen commitments are homomorphic regarding addition. Given two commitments $C_{r_1}(x_1)$ and $C_{r_2}(x_2)$, it holds that:

$$C_{r_1}(x_1) \oplus C_{r_2}(x_2) = C_{r_1+r_2}(x_1 + x_2).$$

Intuitively, if data is combined, the commitments on this data can be combined as well, enabling some of the security protocols within this thesis. For our implementations, we make use of the standard elliptic curve Secp256k1[1], therefore, commitments are 32 bytes in size. Note that we only commit to 31 bytes of data, due to the curve modulus.

2.5 Probability Distributions

For this thesis, we revisit some statistical fundamentals in the form of probability distributions, as we use them to model network latency distributions and the number of neightbors of network participants in this thesis. Probability distributions can be separated into two categories: discrete and continuous distributions [61, 88]. A discrete probability distribution is one that only takes a countable set of values. Continuous distributions, on the other hand, have a support (points where the probability density function is not zero) of uncountable size, e.g., the real numbers of zero to infinity. In quite a few cases, continuous distributions might be more useful to model a discrete problem. Using a continuous distribution requires transforming the result back into the discrete space.

[1] See `https://en.bitcoin.it/wiki/Secp256k1`.

Normal Distribution

One of the most famous representatives of the continuous distributions is the so-called normal distribution, designated by $\mathcal{N}(\mu, \sigma^2)$, where μ represents the expected value and σ^2 the variance of the distribution. The parameters are also known as scale and shape in different contexts. The probability distribution is defined by its probability density function (PDF)

$$\mathrm{PDF}_{\mathcal{N}}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}.$$

The cumulative distribution function (CDF), the integration of the PDF, of the normal distribution is often written as

$$\mathrm{CDF}_{\mathcal{N}}(x) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right] = \int_{-\infty}^{x} \mathrm{PDF}_{\mathcal{N}}(t)dt$$

where erf represents the error function defined by

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

The support, i.e., non zero values of the PDF, of the normal distribution is $(-\inf, +\inf)$, i.e., all of the real numbers $\mathbb{R}$. As this can be problematic for some applications, there exists the truncated normal distribution, limiting the distribution to some interval $[a, b] \subset \mathbb{R}$. This requires re-normalization of the result, i.e., ensuring that $\int_{-\inf}^{+\inf} \mathrm{PDF}(x)dx = 1$, as the integral of the PDF of the normal distribution over $[a, b]$ is smaller than 1.

Lognormal Distribution

The lognormal distribution is a transformation of the normal distribution: The logarithm of the random variable is normally distributed. In other words, given a normally distributed random variable X, then e^X follows a lognormal distribution. Therefore, the lognormal distribution has a support of $(0, +\inf)$.

The parameters of a lognormal distribution are usually given as μ and σ of the underlying normal distribution. Sometimes a third parameter is used, γ, sometimes also called location, which represents a shift of the distribution and results in the following probability density function:

$$\mathrm{PDF}_{\mathrm{lnN}}(x) = \frac{1}{(x-\gamma)\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{\ln(x-\gamma)-\mu}{\sigma})^2}.$$

Parameter estimation of such a shifted lognormal distribution is more complex [57] than estimating parameters for a two-parameter distribution. We will address this in Section 10.6.

Weibull Distribution

Lastly, we would like to mention the Weibull distribution. The Weibull distribution is part of the extreme value distributions family, modeling maxima or minima, with a support of $[0, +\inf)$. As such, this distribution arises when combining various other random variables using extreme-value functions, i.e., the minimum of multiple random variables.

The Weibull distribution has two parameters, the scale λ and shape k. For further details on the Weibull distribution, we refer to established standard works on distributions [88]. For completeness, the probability density functions is given by

$$\mathrm{PDF}_{\mathrm{WB}}(x) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-\left(\frac{x}{\lambda}\right)^k}.$$

Distribution Analysis and Percentiles

After modeling a problem using a distribution and determining the parameters of the distribution, we are interested in conclusions drawn from this model. Within this thesis, one application of distributions is a model of dissemination latency, i.e., the distribution models the fraction of participants in a network receiving a broadcast message at any given time. The cumulative distribution function represents the fraction of the network that has received the message so far. Answering the question of "How long will it take for the message to reach 50% of the network" involves inverting the cumulative distribution function. If the inversion is limited to the equidistant parts of 0 to 100, these are called percentiles: Let $p \in [0, 1]$, the percentile $100p$ is then calculated by solving $\mathrm{CDF}_{\mathrm{lnN}}(x) = p$ for x.

For the normal and lognormal distribution, the cumulative distribution functions are not analytically solvable. We require an efficient way to produce analysis results for a lognormal distribution in later chapters of this thesis, fortunately it can be efficiently approximated. A simple way to approximate the $\mathrm{CDF}_{\mathrm{lnN}}$ of the lognormal distribution is given by [11, p. 7.1.27], with an error $|\epsilon(x)| \leq 5 \times 10^{-4}$, as:

$$\begin{aligned}
\mathrm{CDF}_{\mathrm{lnN}}(x) &= \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{\ln(x) - \mu}{\sigma}\right)\right] \\
&\approx 1 - \frac{1}{2 \times (1 + \alpha_1 s + \alpha_2 s^2 + \alpha_3 s^3 + \alpha_4 s^4)^4}, \\
s &= \frac{\ln(x) - \mu}{\sigma}.
\end{aligned}$$

Where the values for α_i are constants given by $\alpha_1 = -0.278393$, $\alpha_2 = 0.230398$, $\alpha_3 = 0.000972$ and $\alpha_4 = 0.078108$. This accuracy is sufficient for the application within this thesis. For other applications there are methods with higher accuracy, i.e., lower error, available, at the cost of more mathematical operations.

2.6 Privacy, Unlinkability and k-Anonymity

Privacy is the notion of being able to selectively release information. In the context of this thesis, this is mainly accomplished through unlinkability [91]. Unlinkability in our given scenario prevents an answer to the following question: Given a set of network participants and a message that was disseminated by them, which participant created the message? In other words, the message and its creator can not be linked.

Anonymity Set

There are various established measures for privacy. In this thesis, we will focus on the notion of an anonymity set. Given the previous task of identifying the original sender of a message, the anonymity set is the smallest set of network participants, which can not be excluded as the potential sender.

In other words, all possible senders together form the anonymity set for a given message. A participant is said to be anonymous in regards to a given anonymity set, as they can not be identified within this set.

k-Anonymity

While the maximum possible anonymity set in a network protocol is that of all participants - as long as participation can be detected - it can not be efficiently realized. However, for many applications it is sufficient to guarantee a much smaller size to provide reasonable privacy. If the size of the anonymity set is guaranteed to have at least k participants, we call the protocol k-anonymous [12]. For a suitable value of k, e.g., $k > 10$ [80], a participant is anonymous for most practical purposes.

2.7 Problem: Flood-and-Prune Privacy

The most basic way to broadcast a message throughout a network is a flood-and-prune approach: When a message is encountered for the first time, share it with all neighbors, otherwise drop the message. A flood-and-prune broadcast is easy to implement, resilient against attacks and has low dissemination latency. Unfortunately, the protocol does not provide any privacy guarantees.

The lack of privacy guarantees can be exploited by various kinds of attackers. One attack exploits the symmetry in propagation of message dissemination: A message is forwarded and similar speeds in all directions. See Figure 2.1 for an example.

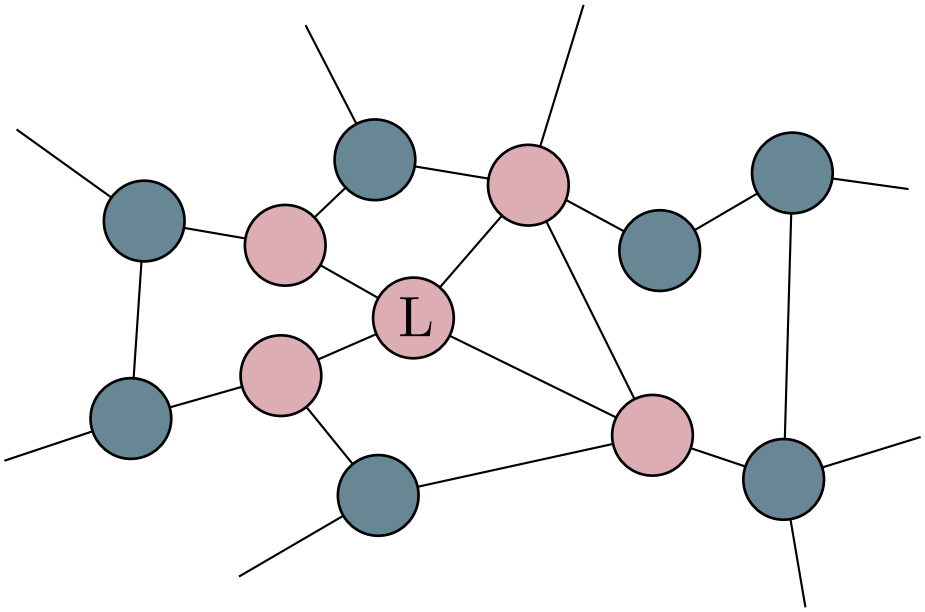


Figure 2.1: A broadcast in progress. Light red nodes already received the broadcast, while dark blue nodes did not. The likely originator is marked (L). [5] © 2018 IEEE

Attackers can determine the likely origin by distributing unobtrusive nodes throughout the network and recording the arrival time of the received messages [21]. The likely origin can be reconstructed by the distance from the measurement nodes, i.e., the node that is at the center of the sub-graph of nodes that already received the message, also known as the Jordan center. This attack works even for so called unreachable nodes [76], i.e., nodes that refuse incoming connections, which are harder to deanonymize.

2.8 Summary

In this chapter we introduced the primary application domain of this thesis, blockchains, as well as models for the network's underlying public blockchains. We revisited probability distributions, with a focus on the normal and lognormal distribution. Lastly, we introduced the measure of privacy, k-anonymity, to quantify unlinkability between a sender and a message. With these tools and basics, we can now dive deeper into the matter of privacy preserving network protocols.

Chapter 3

Privacy Protocols

In this chapter, we discuss existing systems to provide privacy for blockchain transactions on the network layer. The introduced systems provide a basis to develop novel protocols conforming to our expectations and provide valuable comparisons for performance and privacy properties. We loosely separate the protocols into topological, or statistical, mechanisms, which provide efficient dissemination but weak privacy guarantees and cryptographic systems that can withstand stronger attackers but are less efficient.

First, we introduce basic dining-cryptographers networks, the k-anonymous message transmission protocol and adaptive diffusion, as we extend these protocols throughout the thesis. Further, we give an overview over alternative privacy protocols, which we use as comparison for the results obtains in this thesis.

3.1 Dining-Cryptographers Networks

The base protocol of the dining-cryptographers (DC) network by Chaum [32] allows for unconditional sender untraceability for sending a single bit. This protocol has been extended in various ways to send longer messages. One possibility is the sum-modulo-q-based implementation, given by Algorithm 1. In this more general form, the protocol allows a single participant to share a message in an unlinkable way. The protocol requires secure channels to communicate and an upper limit for message lengths ℓ, all shorter messages will be padded to reach this length.

Given a fixed-length message, e.g., by a modulus for modular arithmetic, all participants g_i decide on their message m_i to share. If participants do not intend to send a message, they will prepare $m_i = 0$. Only one participant is allowed to share a message $m_i \neq 0$ for a successful transmission, otherwise the resulting message will be unreadable, i.e., a combination of all transmitted messages. Each participant g_i creates k random shares $s_{i,j}$ from its message m_i, so that $m_i = \sum_{j=1}^{k} s_{i,j}$ and shares $s_{i,j}$ with participant g_j, whereas $s_{i,i}$ remains with g_i. All participants combine every share they receive and share the combination with all other participants. Therefore, all participants will receive an aggregate of all shares produced by every other participant and can calculate the aggregation of all messages, which corresponds to $\sum_i m_i$.

Algorithm 1 Dining cryptographers protocol using addition modulo q.

Input: Message m_{self} of length ℓ

Environment: Participants g_i with $i \in \{1 \dots k\}$, including the executing node g_{self}, message length ℓ

$X = \begin{cases} m_{\text{self}} & \text{if } m_{\text{self}} \text{ not empty} \\ 0 & \text{else} \end{cases}$

for $i \in \{1 \dots k\}$ **do**

$s_{\text{self},i} = \begin{cases} \sim \mathcal{U}\{0, 2^\ell - 1\} & \text{if } i < k \\ X - \sum_{j=1}^{k-1} s_{\text{self},j} & \text{if } i = k \end{cases}$

end for

Send $s_{\text{self},i}$ to $g_i : i \neq \text{self}$

Wait until g_{self} received all $s_{j,\text{self}}$

$S_{\text{self}} = \sum_{j=1}^{k} s_{j,\text{self}}$

{Note: This sum includes $s_{\text{self},\text{self}}$ which was not sent}

Broadcast S_{self}

Wait until g_{self} received all S_j

return $\hat{X} = \sum_j S_j$

This simple implementation is information-theoretically secure against identifying the originator of a message, but it is not secure against interruption. The security is based on the combination of random bitstrings, as all messages are transmitted via all other participants. To identify the origin, all other participants must combine their shares received from the suspected origin. A participant can use a random message to create a collision with other messages, preventing other participants from reading the message, whereas the participant itself can actually retrieve the sent message. Further, with a message complexity of $\mathcal{O}(n^2)$ a dining-cryptographers network does not scale well to high numbers of participants.

There are various ways to implement dining-cryptographers networks. Applying the XOR operation instead of addition mod q or different structures leads to a similar algorithm with the same guarantees. Algorithm 2 shows a variant of the original algorithm [32], as we use it in Chapter 11. Both versions are helpful for modeling different types of extensions, e.g., when combining with messages homomorphic under XOR, the XOR variant is more useful.

Algorithm 2 Dining-Cryptographers Protocol using XOR, the resulting message $m_{out} = \bigoplus_{k=1\dots n} m_k$ is the same across all participants. [2]

Input: Message m_{self} of length ℓ

Environment: Participants g_i with $i \in \{1 \dots k\}$, including the executing node g_{self}

Establish random secrets $s_{\text{self},i}$ of length ℓ with each member $g_i, i \neq \text{self}$

$M_{\text{self}} = m_{\text{self}} \oplus \bigoplus_{i=1\dots n, i\neq \text{self}} s_{\text{self},i}$

Send M_{self} to $g_i \; \forall i \in \{1 \dots n\} \setminus \{\text{self}\}$

Receive M_i from $g_i \; \forall i \in \{1 \dots n\} \setminus \{\text{self}\}$

$m_{out} = \bigoplus_{i=1\dots n} M_i = \bigoplus_{i=1\dots n} \left(m_i \oplus \bigoplus_{j=1\dots n, j\neq i} s_{i,j} \right)$

return m_{out}

3.2 k-Anonymous Transmission Protocol

As noted before, DC networks do not scale well with the number of participants and can be prevented from making progress by malicious participants. To create a practical application despite these weaknesses, von Ahn et al. [12] realized a sender- and receiver-k-anonymous protocol for message transmission. Note that this is no longer a broadcast protocol. Instead it relies on assigning participants to a number of disjoint groups with only k members each. To achieve the sender-k-anonymity, the message is first shared anonymously within a source group G_s, using a dining-cryptographers variant. Afterwards, all participants of this group G_s send the message to all participants of the target group G_t.

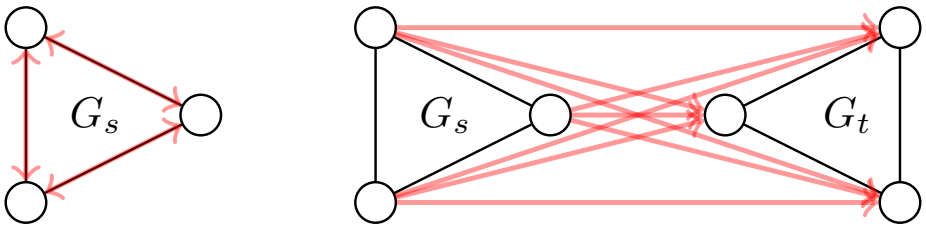


Figure 3.1: Visualization of the two parts of the protocol by von Ahn et al., sharing within the group and the final transmission between the source group G_s and target group G_t. [9]

As the message does not need to reach all participants, the application to groups of sizes much smaller than all participants n comes naturally. The protocol also builds upon other improvements [52], especially to identify cheaters through commitment schemes. Thus, this new protocol deals with two severe weaknesses, even though it changed the use-case.

In the protocol by von Ahn et al. the dining-cryptographers phase shares an array of $2k$ slots, where each slot can consist of a message of fixed size and a non-zero group identifier, identifying the receiver group. Each slot sums up to a total, fixed length of ℓ. To present the protocol, we will split the protocol into multiple phases, based on the presentation in the original paper [12]. These phases are the commitment phase, the sharing phase, the broadcast phase, result computation and, lastly, the transmission phase.

In the commitment phase (see Algorithm 3) participants prepare their message in a random slot and set all other slots to zero. Further, they create commitments on all messages and broadcast these to the group. The commitments allow for a blame protocol to identify malicious actors after detecting misbehavior: if more than half of all slots are occupied, the one or multiple perpetrators can be uncovered. If a sender detects a collision, i.e., their message is not correctly represented in the result, without less than half of all slots being occupied, it is deemed an accident instead of malicious action. Note that all arithmetic is performed over $\mathbb{Z}_q$ or, for commitments, the respective commitment structure.

Algorithm 3 Commitment phase of the k-anonymous message transmission protocol.

Input: Message m_{self}, target group G_t
Environment: Group $G_{\text{self}} = \{g_i : i \in \{1 \dots k\}\}$, including the executing node g_{self}, slot length ℓ

Slot$=\sim \mathcal{U}\{0, 2k-1\}$

$$X[i] = \begin{cases} (m_{\text{self}}, G_t) & \text{if } i = \text{Slot} \wedge G_t \neq 0 \\ (0,0) & \text{else} \end{cases}$$

for $t \in \{1 \dots 2k\}$ and $i \in \{1 \dots k\}$ **do**

$$s_{\text{self},i}[t] = \begin{cases} \sim \mathcal{U}\{0, 2^\ell - 1\} & \text{if } i < k \\ X[t] - \sum_{j=1}^{k-1} s_{\text{self},j} & \text{if } i = k \end{cases}$$

$r_{\text{self},i}[t] =\sim \mathcal{U}\{0, 2^\ell - 1\}$

Compute $\hat{\mathcal{C}}_{\text{self},i}[t] = C_{r_{\text{self},i}[t]}(s_{\text{self},i}[t])$

end for

Broadcast $\{\hat{\mathcal{C}}_{\text{self},i}[t] : i \in \{1 \dots k\}, t \in \{1 \dots 2k\}\}$

This results in a protocol with more overhead compared to basic dining-cryptographers networks but provides fairness and robustness in malicious environments. It also allows for the transfer of multiple messages within one round. Algorithm 4 is called the sharing round, where all computed information is shared. Algorithm 5, the local broadcast round, is the final group internal communication and allows all participants to reconstruct the messages with Algorithm 6. These Algorithms 3 to 6 are just the extension of the second half of Algorithm 1 by the validation and dissemination of commitments. Finally, in Algorithm 7 each participant transmits the received message to every participant in the receiving group.

Algorithm 4 Sharing phase of the k-anonymous message transmission protocol.

Input: $\forall i \in \{1 \dots k\} : s_{\text{self},i}, r_{\text{self},i}$ of Algorithm 3
Environment: Group $G_{\text{self}} = \{g_i : i \in \{1 \dots k\}\}$, including the executing node g_{self}, commitments $\hat{\mathcal{C}}_{i,j}[t]$

for $g_i \in G \setminus \{g_{\text{self}}\}$ **do**

Send $\{(r_{\text{self},i}[t], s_{\text{self},i}[t]) : t \in \{1 \dots 2k\}\}$ to g_i

Receiving node g_i validates that $C_{r_{\text{self},i}[t]}(s_{\text{self},i}[t]) = \hat{\mathcal{C}}_{\text{self},i}[t]$

end for

Algorithm 5 Broadcast within the group of the k-anonymous message transmission protocol.

Input: $\forall j \in \{1 \dots k\} : (r_{j,\text{self}}[t], s_{j,\text{self}}[t])$ transmitted by others in Algorithm 4
Environment: Group $G_{\text{self}} = \{g_i : i \in \{1 \dots k\}\}$, including the executing node g_{self}, commitments $\hat{\mathcal{C}}_{i,j}[t]$

Wait until g_{self} received all pairs $(r_{j,\text{self}}[t], s_{j,\text{self}}[t])$

$R_{\text{self}}[t] = \sum_j r_{j,\text{self}}[t]$

$S_{\text{self}}[t] = \sum_j s_{j,\text{self}}[t]$

Broadcast $\{(R_{\text{self}}[t], S_{\text{self}}[t]) : t \in \{1 \dots 2k\}\}$

Everyone checks $C_{R_{\text{self}}[t]}(S_{\text{self}}[t]) = \oplus_j \hat{\mathcal{C}}_{j,\text{self}}[t]$

Algorithm 6 Result computation of the k-anonymous message transmission protocol.

Input: $\forall j \in \{1 \dots k\} : (R_j[t], S_j[t])$ broadcasted by others in Algorithm 5
Environment: Group $G_{\text{self}} = \{g_i : i \in \{1 \dots k\}\}$, including the executing node g_{self}, commitments $\hat{\mathcal{C}}_{i,j}[t]$
Wait until g_{self} received all pairs $(R_j[t], s_j[t])$
$R[t] = \sum_j R_j[t]$
$X[t] = \sum_j S_j[t]$
Everyone checks $C_{R[t]}(X[t]) = \oplus_{i,j} \hat{\mathcal{C}}_{i,j}[t]$
return X

Algorithm 7 Transmission phase of the k-anonymous message transmission protocol.

Input: X, result from Algorithm 6
Environment: The executing node g_{self}, known groups of network participants $G_i = \{g_{j,i} : j \in \{1 \dots k\}\}$
for $(m_i, G_i) \in X$ **do**
  if $G_i \neq 0$ **then**
    Send m_i to all members of G_i
  end if
end for

The protocol by von Ahn et al. provides a notion of fairness: during honest execution, each participant has a probability of $p > \frac{1}{2}$ for successful transmission. The protocol was described by von Ahn et al. with an extension to ensure everyone used at most one slot, i.e., the protocol run was fair, using a zero-knowledge proof. This extension is not shown in the previous algorithms. The proof is triggered when more than half of all slots are used and works in the following way: A suspected participant, the prover within the protocol, creates a permutation of all slots and a second set of commitments. Other participants, the verifier within the zero-knowledge protocol, chose to either have the prover open all but one of the permuted commitments, i.e., showing that at most one commitment was not zero. Alternatively, the verifier may choose to have the prover reveal the permutation and prove that they can open each corresponding commitments to the same value.

Lastly, von Ahn et al. provide an additional modification, which was not included in the previous description. To handle selective non-participation, e.g., refusing communication with a single participant, all nodes must share any message they send with all other participants as well. The shared message parts are encrypted for the intended recipient, to prevent information leakage. If a participant claims another refuses to communicate with them, any other participant can provide the encrypted backup of the required message. This vastly increases the overhead of the protocol, but is relevant for environments with highly expected malicious behavior.

3.3 Adaptive Diffusion

Adaptive diffusion [46] is a protocol developed to address the privacy impact of the symmetry of flood-and-prune broadcasts. It breaks the symmetry present in regular flood-and-prune broadcasts by creating a virtual, or fake, source of the message, identified by a virtual source token. The owner of the virtual source token spreads the message in such a way that they are the Jordan center of the graph of nodes that received the message so far is the

virtual source, not the true source of the message. This prevents the previously outlined deanonymization attack on flood-and-prune message broadcasting.

The virtual source can not stay with the true source. To move the virtual source away, the owner of the virtual source token forwards the token probabilistically. The goal is to equalize the probability of any node, that already received the message, to be the true source. In other words, if n nodes received the message, the probability of any node v_i having received the message being the true source v should be approximately $P[v_i = v] \approx \frac{1}{n}$.

The forwarding probabilities are dependent on the underlying network assumptions. Adaptive diffusion uses a d-regular infinite tree as its basic network model, i.e., each node has exactly d neighbors and there are infinite participants within the network. Further, message transmission happens only at discrete time steps. These assumptions are called contact network, while the message and message transmission is also called infection. The probability of forwarding depends on the number of previous forwards h, the current number of steps so far t and the degree of the underlying tree-network d. Using these parameters, the probability of forwarding the virtual source token is derived by Fanti et al. as

$$p_d(t,h) = \begin{cases} \frac{t-2h+2}{t+2} & \text{if } d = 2, \\ \frac{(d-1)^{\frac{t}{2}-h+1}-1}{(d-1)^{\frac{t}{2}+1}-1)} & \text{if } d > 2. \end{cases} \tag{3.1}$$

The dissemination of the message is performed in steps, where the virtual source node acts according to a simple scheme: In any odd timestep, forward the message to all neighbors. In an even timestep, either forward or keep the virtual source token, depending on the derived probability. Any other node will forward the message, if they have received it at least once before — informally, they behave opposite to a flood-and-prune broadcast. The algorithm realizing this protocol is given by Algorithm 8 and Algorithm 9 as it was presented originally [46]. A node that received the message is considered infected in this presentation.

Although this approach is designed for cycle-free networks, Fanti et al. show initial results claiming [46] it works well even for general networks. For a network protocol, adaptive diffusion provides a few challenges. A suitably powerful attacker can subvert the protocol by connecting to a large number of nodes, as a node informs all neighbors of new messages, reducing the privacy guarantees to distance one. Further, later messages deliver the hop count h of current forwards, eliminating all nodes of a distance not equal to h. Lastly, the discrete-time model needs to be transformed into a continuous-time model of real-world computing systems.

3.4 Further Privacy Protocols

This section shortly introduces various protocols that are used as comparative results or have a significant mind-share of developers of privacy preserving protocols and should, therefore, be addressed.

Dissent

The anonymity system **Dissent** [36, 122] utilizes a dining-cryptographers network and provides similar properties to the k-anonymous message transmission protocol. Dissent uses a small number of core servers as anonymity providers and an anonymous announcement phase per round, where every participant announces the length of message they want to transmit. This allows for variable-sized messages. The announcement phase uses a secure group

Algorithm 8 Adaptive Diffusion as described by Fanti et al. [46].

Require: Contact network $G = (V, E)$, source v', time T, degree d
Ensure: Set of infected nodes V_T
  $V_0 = \{v'\}, h = 0, v_0 = v'$
  v' selects one of its neighbors u at random
  $V_1 = V_0 \cup \{u\}, h = 1, v_1 = u$
  let $N(u)$ represent u's neighbors
  $V_2 = V_1 \cup N(u) \setminus \{v'\}, v_2 = v_1$
  $t = 3$
  for $t \leq T$ **do**
    v_{t-1} selects a random Variable $X \sim U(0, 1)$
    if $X \leq p_d(t-1, h)$ **then**
      for $v \in N(v_{t-1})$ **do**
        Infection Message(G, v_{t-1}, v, V_t)
      end for
    else
      v_{t-1} randomly selects $u \in N(v_{t-1}) \setminus \{v_{t-2}\}$
      $h = h + 1$
      $v_t = u$
      for $v \in N(v_t) \setminus \{v_{t-1}\}$ **do**
        Infection Message(G, v_t, v, V_t)
        if $t + 1 > T$ **then**
          break
        end if
        Infection Message(G, v_t, v, V_t)
      end for
    end if
    $t = t + 2$
  end for

Algorithm 9 Infection Message algorithm as described by Fanti et al. [46].

Require: Contact network $G = (V, E)$, source v', time T, degree d
Ensure: Set of infected nodes V_T
  if $v \in V_t$ **then**
    for $w \in N(v) \setminus \{u\}$ **do**
      Infection Message(G, v, w, V_t)
    end for
  else
    $V_t = T_{t-2} \cup \{v\}$
  end if

shuffle for all: All nodes encrypt their announcement with layers for all participants according to a fixed permutation of the users per round. Each node then in turn shuffles all values and removes their respective layer of encryption. After every node performed such a shuffle, all nodes can trust the shuffle, since they participated, and only one honest shuffle is necessary to hide the originator. The last participant publishes the list of message lengths. With this information, they perform a DC-round to transmit the actual data. The announcement round causes a startup phase [36] scaling linearly in the number of group members and becoming noticeably slow, e.g., 30 seconds, for group sizes of 8 to 12. This latency might not be acceptable in real-world blockchain applications.

k-Anonymous Overlay Protocol

Wang et al. [115] propose a peer-to-peer transmission protocol, implementing the same functionality as the k-anonymous message transmission protocol by von Ahn et al., i.e., point-to-point communication.

The network is partitioned in a number of rings, similar to RAC [82]. Within each ring, nodes can communicate anonymously with any other node. This is achieved by a combination of layered encryption and shuffling batches of messages, similar to Dissents shuffle. During each forwarding, every node must insert a message to prevent deanonymization of participants.

If a node wants to send a message, they forward the ring identifier and message anonymously to another node in their own ring. The receiving node forwards the message to a node in the target ring, unless the target is the local ring. The receiving node broadcasts the message within the target ring, so that the recipient will receive the message.

The protocol claims to require a byzantine-secure setup phase to prevent malicious insertions in the ring configurations. Further, the message transmission does not provide arbitrary-length messages, as their payload slots are fixed in size.

Dandelion

Dandelion [24] proposes a two-phase protocol for statistical spreading. Phase 1 spreads the message along a line graph, i.e., an approximation of a Hamilton path within the network. Each node along the line graph has a fixed probability of starting the second phase. Phase 2 uses a flood-and-prune broadcast to ensure delivery to all nodes. For an attacker to detect the originator with reasonable certainty, he needs to be the first node receiving the message from the originator. With known topology, many attackers could further improve estimates without being the first recipient. A visualization of the approach is shown in Figure 3.2.

Dandelion++ [47] is an improvement over the initial version. The iteration covers enhanced defenses against graph learning and graph construction attacks, as well as intersection, selective non-participation, and partial deployment attacks.

Gossip with Muting

Bellet et al. [19] investigate a simple gossip protocol with a mute parameter s over a complete graph, i.e., every node is connected to any other node. A node disseminating a message has a probability of $1 - s$ of stopping to disseminate after each interaction with a neighbor. With $s = 0$ a node forwards a message exactly once, while $s = 1$ leads to nodes forwarding the message via all available connections — essentially a flood-and-prune broadcast. Bellet et al. show that gossip with a mute value of s has a differential privacy guarantee of $\delta =$

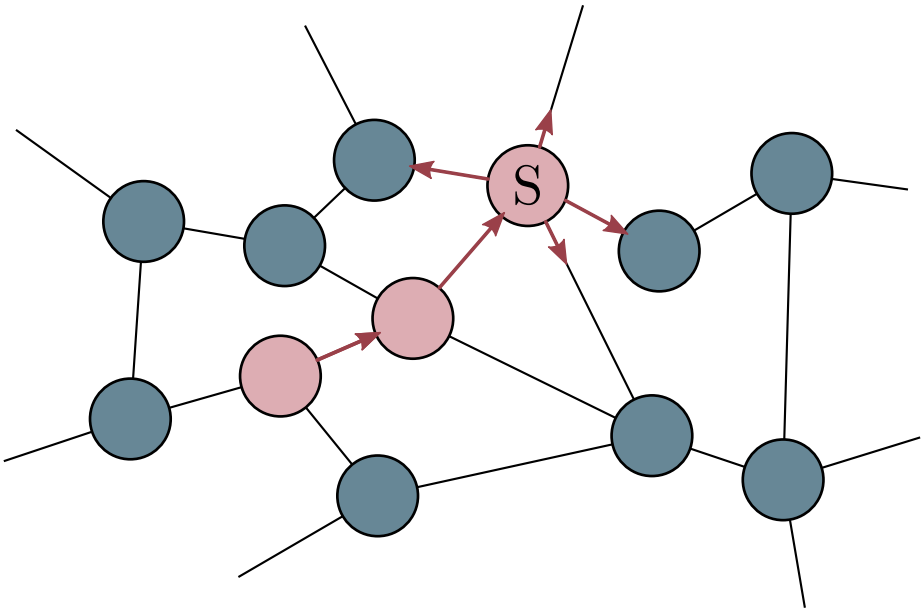


Figure 3.2: Example of Dandelion dissemination: The red nodes have received the message along a line. The last node (S) spreads the message in a regular broadcast manner. [5]

$s + (1 - s)\beta$, where β is the fraction of attackers in the network, resulting in the lower bound of differential privacy equal to β.

Tor

The prominent anonymous communication system Tor [40] is usually one of the first approaches when trying to achieve privacy on the network layer. Tor only supports a direct connection between a pair of nodes and does not provide an abstraction layer for broadcasts.

While it would be possible to tunnel all connections through Tor, it is not well suited to implement a broadcast mechanism for blockchains. Participants might be forced to connect without Tor [21] or participants connect without Tor voluntarily, reducing privacy guarantees for all participants. Tor can also be used in addition to the presented systems for a defense in depth approach. For these reasons, we will not discuss Tor further in this thesis.

Herd, RAC and Cover Traffic

Networks for different applications, such as **Herd** [69] for voice over IP or **RAC** [82], use other building blocks. These building blocks include mix nodes, trust zones, multiple rings with onion encryption and cover traffic. While these could be used for designing a privacy-preserving broadcast mechanism, they create different problems.

Mix nodes lead to increased load for central infrastructure, due to their need to process all traffic. Cover traffic creates continuous load, which is a problem for rare network utilization, such as transaction transmissions and limits other uses as the rate is dependent on the use case. Small DC-nets might scale the number of rounds depending on usage, scaling of cover traffic is much more restricted. If a node increases or reduces their bandwidth consumption, this

change in behavior can be attributed to their personal change in usage and is not reduced to a change in the behavior of members of a group. This leaks the information of data usage and in the context of blockchain systems can be correlated to the arrival times of new transactions, undermining the privacy-preserving aspect.

Part II

3P3: Strong Flexible Privacy

Chapter 4

Overview

This chapter is based on two previous publications at ICDCS and IWCSS [4, 5] (with permission, © 2018 IEEE).

[4] D. Mödinger and F. J. Hauck. "3P3: Strong Flexible Privacy for Broadcasts". In: *4th International Workshop on Cyberspace Security (IWCSS 2020)*. 2020.

[5] D. Mödinger, H. Kopp, F. Kargl, and F. J. Hauck. "A Flexible Network Approach to Privacy of Blockchain Transactions". In: *Proc. of the 38th IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, 2018.

In the background chapter we introduced various building blocks and protocols to disseminate messages in a network. One of them, Chaum's dining-cryptographers networks [32], provides strong guarantees and has been used by various state-of-the-art protocols. Although dining-cryptographers networks provide very strong privacy, they are inefficient for larger networks.

In this part of the thesis, we introduce a novel protocol: 3P3. 3P3 combines dining-cryptographers networks with a statistical dissemination protocol to provide a flexible privacy-preserving dissemination tool to peer-to-peer network developers.

4.1 Structure

3P3, the three phase privacy preserving protocol, consists, as the name suggests, of three phases. Figure 4.1 illustrates the behavior of the three phases, which are:

1. A dining-cryptographers network.
2. A variant of adaptive diffusion.
3. A flood-and-prune broadcast.

Each phase fulfills its own purpose. The dining-cryptographers network provides a strong base privacy, resistant against powerful attacker models, including a global passive observer. The adaptive diffusion phase increases the privacy guarantees for cheap and common attacks, often observed in peer-to-peer networks. These attacks include observer or measurement nodes with many connections throughout the network. Lastly, the flood-and-prune broadcast ensures delivery to all participants in an effective manner, once sufficient privacy thresholds are ensured by the previous protocol layers.

The following sections give a high level view of the functionality of each phase.

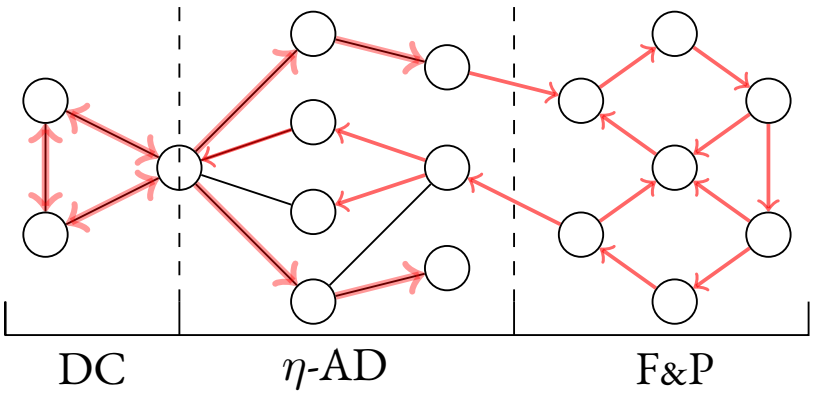


Figure 4.1: Overview of the three phases of 3P3. A DC network, our modified η-diffusion (η-AD) and the final flood-and-prune (F&P) phase. [4]

4.2 Phase I: Dining-Cryptographers Network

The first phase uses a dining-cryptographers network to disseminate the broadcast message in a small group. Using dining-cryptographers networks in such a way allows for better performance but preserving fairly strong privacy guarantees.

In the first phase, we perform a preliminary round to establish all desired message lengths, similar to Dissent [122], using the k-anonymous transmission protocol by von Ahn et al. [12], see Chapter 3. The actual messages are sent in a separate DC round.

For a fixed group size of k participants, we prepare $2k$ slots for message lengths. This number is to ensure a probability of delivery greater than $\frac{1}{2}$ even when all participants send a message. All slots consist of a random identifier r, a message length ℓ and a set of k values K_i, encrypted to each participant i. The random identifiers r are required for a node to identify the slot of their message in the presence of multiple identical lengths. When a participant i wants to disseminate a message m_i, they chose a slot and an identifier at random. Once this information is shared through the DC network, all nodes prepare a round with message length $2\sum_i \ell_i$, i.e., the sum of all provided lengths.

Any node that submitted a length $\ell \neq 0$ can find their designated position by their position in the message slot. All other message parts are set to zero. The commitment aggregation to zero for these message parts needs to use the provided random value from round one. Using these prepared values, allows the author to blame any node that does not send a zero message in the author's slot. The messages are summarized in Table 4.1.

If all lengths are zero, the second round can be skipped. Valid lengths should be restricted to prevent a denial-of-service attack, e.g., 16 bits for the use case of blockchains. A global passive observer may notice a group only performing the first stage, concluding that nothing has been sent this round. But the broadcast could be traced back to the group anyways by a global passive observer anyways, so no privacy is lost.

To assign responsibility to start the next round without communication, the node which has the node identifier closest to the random identifier should start the next round. Nodes sort the node identifiers and split the available space of values for random identifiers uniformly to prevent an uneven workload.

The original protocol by von Ahn et al. does not permit arbitrarily long messages and provides significant opportunities for optimizations. These are discussed in Chapter 6.

Round	Transmitted Message
1.	$[(r_1, \ell_1, K_1), \ldots, (r_{2k}, \ell_{2k}, K_{2k})]$
2.	$[m_1, \ldots, m_i]$

Table 4.1: Phase 1 messages transmitting messages m_1 through m_i.

4.3 Phase II: Adaptive Diffusion

The second phase uses adaptive diffusion [46], which consists of two parts, the spreading mechanism and the virtual source subprotocol. The virtual source should forward messages so that all nodes of a given distance from them either all received the message or all did not receive the message.

If a node, which is not the virtual source, receives a message for the first time, they remember the sender to prevent unwanted extension of the spreading sub-graph over cross-connections. Then, they select η neighbors. Whenever they receive the same message again from the same sender, they will forward the message to the selected neighbors. As adaptive diffusion was originally constructed for contact networks, we transformed the protocol slightly as a network protocol. The virtual source token is forwarded with probability p_t based on the expected average degree of the network, and the current time step t.

To initiate the protocol, the initiating node v chooses a random neighbor. Then v sends the message m and the virtual source token transmission message $(v, t = 1, r = \mathcal{H}(m))$, tied to the message through the hash of the message $\mathcal{H}(m)$, prompting the execution of the virtual source algorithm. Every virtual source should monitor the network for the progress of the protocol. A timeout will trigger re-transmission to a different participant, as the previously selected might refuse cooperation or be unreachable. The timeout will extend on messages received through the protocol but only stop when receiving a flood-and-prune message.

Chapter 6 goes into detail for phase II of 3P3. Further, the original proposal by Fanti et al. [46] is not suitable for direct use in a peer-to-peer-network. The required changes are discussed in Chapter 6 as well.

4.4 Phase III: Flood-and-Prune Broadcast

Lastly, the last virtual source initiates a flood-and-prune broadcast to ensure delivery to all nodes. When a node receives a message for the first time, it forwards the message to all neighbors, excluding the node which sent the message. This phase of the protocol will not be discussed in detail, as it is realized through a standard and well known algorithm.

Chapter 5

Phase I: Arbitrary Length k-Anonymity

This chapter is based on an unreviewed publication [9].

[9] D. Mödinger, A. Heß, and F. J. Hauck. "Arbitrary Length k-Anonymous DC Communication". In: (2021). arXiv: 2103.17091 [cs.NI].

In the previous chapter we presented 3P3, a multi-layer protocol for broadcasts. This proposal builds on top of a dining-cryptographers network. The protocol by von Ahn et al. [12] constructs a modern implementation of such a dining-cryptographers network. Unfortunately, the protocol was neither designed for arbitrary length messages nor as a broadcast protocol, so it cannot fill the required role directly. To address these shortcomings and make the protocol applicable for a general broadcast use-case, we:

- Extend the k-anonymous message transmission protocol by von Ahn et al. [12] to arbitrary length messages.
- Optimize the extended protocol for performance by reducing the overhead for the most common scenarios.
- Provide and evaluate an implementation of these protocols.

This chapter is structured in the following way: In Section 5.1, we describe our extension of the protocol by von Ahn et al. to handle messages of arbitrary length. The privacy and security properties of this new protocol are discussed in Section 5.2. Section 5.3 introduces our approach to improve the performance of the protocol.

5.1 Arbitrary Length Messages Protocol

Overview

Our protocol consists of two consecutive rounds, each based on the dining-cryptographers protocol. The initial round determines whether there are senders that want to disseminate a message of a certain length. During the final round these messages are distributed.

As a first step during the initial round, participants with a message m first determine the length ℓ of their message in Bytes. They further prepare a random round identifier r and

a set K, composed of k random values of globally fixed length. The latter are required to ensure fairness during the actual message dissemination process in the final round, as we will point out later. If a participant has no message to disseminate, they set all values to zero. Finally, they share the set of values (r, ℓ, K) according to the protocol proposed by von Ahn et al. [12], which we introduced in Chapter 3. To achieve this, each participant with a tuple of non-zero values chooses a random slot in a pre-prepared vector with $2k$ slots, where they place their values. In case of four participants, this vector has the following form.

$$[(r_1, \ell_1, K_1), (r_2, \ell_2, K_2), (r_3, \ell_3, K_3), (r_4, \ell_4, K_4),$$
$$(r_5, \ell_5, K_5), (r_6, \ell_6, K_6), (r_7, \ell_7, K_7), (r_8, \ell_8, K_8)]$$

The protocol then merges the vectors of all participants, using a secure multiparty computation to ensure that the contents of the individual vectors are hidden. This is done by using the protocol of von Ahn et al.

During the final round, the actual messages are disseminated, similar to the approach implemented by Dissent [122]. Every participant prepares a second dining-cryptographers round for a message length equal to the sum of the message lengths announced during the initial round $\ell = \sum_{i=1}^{k} \ell_i$. A participant g_i with a message length greater than zero determines their offset in the final message based on their random identifier r_i and their announced length ℓ_i. Afterwards they fill their reserved section with their message m. The remaining portions of the message have to be set to zero.

The resulting message is then shared using a dining-cryptographers variant. The final inter-group transmission of von Ahn et al.'s protocol has not been adopted by our protocol since we only assume a single group at this point.

Initial Round

The initial round of our protocol is used to share lengths of messages and prepare verifiable randomness for the final round. The preparation protocol executed is given by Algorithm 10. The group broadcast following this preparation is identical to Algorithms 4 to 6 of the protocol by von Ahn et al. [12].

For a fixed group size of k participants, we prepare $2k$ slots for message lengths. All slots consist of a random identifier r_i, a message length ℓ_i in Bytes and a set of k random values $\{\{K_{i,j}\}_j : j \in \{1 \dots k\}\}$, encrypted to each participant g_j. When a participant g_i wants to disseminate a message m_i they chose a slot and a non-zero identifier r_i at random.

The random identifiers r_{self} are required for a node to identify the slot of their message in the final round. The length is not enough, as there might be multiple identical lengths. After a successful execution of the initial round, each participant possesses a list of the form:

$$[(r_1, \ell_1, \{K_{1,1}\}_1 \dots \{K_{k,1}\}_k), \dots, (r_{2k}, \ell_{2k}, \{K_{1,2k}\}_1 \dots \{K_{k,2k}\}_k)] .$$

Final Round

After the initial round has been successfully completed, i.e., the above list has been shared, all participants progress to the final round. If the previous list contains at least one $\ell_i \neq 0$, everyone prepares a new compound message of length $\sum_i \ell_i$, the sum of all provided lengths. Otherwise, the final round can be omitted and a new initial round is executed after a configurable time interval. The preparation is given by Algorithm 11, while the group broadcast is again identical to Algorithms 4 to 6, but based on the output of Algorithm 11.

Algorithm 10 Preparation phase of the first DC round.

Input: Message m

Environment: Group $G_{\text{self}} = \{g_i : i \in \{1 \dots k\}\}$, including the executing node g_{self}, public keys of group participants, slot length ℓ in byte

$r =\sim \mathcal{U}\{0, 2^{16} - 1\}$

Slot$=\sim \mathcal{U}\{0, 2k - 1\}$

$K = \{K_i = \{\sim \mathcal{U}\{0, 2^{256} - 1\}\}_i : i \in \{1 \dots k\}\}$

$$X[i] = \begin{cases} (r, \text{length}(m), K) & \text{if } i = \text{Slot} \\ (0, 0, 0) & \text{else} \end{cases}$$

for $t \in \{1 \dots 2k\}$ and $i \in \{1 \dots k\}$ **do**

$$s_{\text{self},i}[t] = \begin{cases} \sim \mathcal{U}\{0, 2^{8\ell} - 1\} & \text{if } i < k \\ X[t] - \sum_{j=1}^{k-1} s_{\text{self},j} & \text{if } i = k \end{cases}$$

$r_{\text{self},i}[t] =\sim \mathcal{U}\{0, 2^{8\ell} - 1\}$

Compute $\hat{\mathcal{C}}_{\text{self},i}[t] = C_{r_{\text{self},i}[t]}(s_{\text{self},i}[t])$

end for

Broadcast $\{\hat{\mathcal{C}}_{\text{self},i}[t] : i \in \{1 \dots k\}, t \in \{1 \dots 2k\}\}$

Any node that submitted a length $\ell \neq 0$ has reserved ℓ bytes in the next message, where the order is determined by the resulting layout of the initial round. Assume there is a non-zero length ℓ_j in slot j. The start of this message inside of the single compound message transmitted in the final round can simply be computed by the length of all previous messages mentioned in slots i with $i < j$, formally message j starts at $1 + \sum_{i=1}^{j-1} \ell_i$. All other parts not used by the message are set to zero.

Assume an example network with 4 participants. Then there will be 8 slots in the initial round. Let us further assume that the transmitted lengths are 2, 5 and 4, in the order of appearance in these slots, e.g. $(2, 0, 0, 5, 4, 0, 0, 0)$. The participant having sent length 4, will then put its message from the eleventh to the fourteenth byte, as seen in Figure 5.1. All other bytes are set to zero.

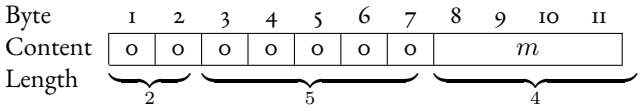


Figure 5.1: Example message allocation in the final round. The first row gives the index of the bytes, the second row the occupation using the message lengths as an indicator and the last row shows the actual usage of the bytes. [9]

For every message slot j not designated for node i, each nodes needs to decrypt the seed value $\{K_{i,j}\}_i$ addressed to them. This seed must be used to deterministically create the randomness for all commitments for this slot j. The creator of this slot can later validate that the commitments are commitments to zero. As all values should be zero, this has no implications for the privacy of the node creating the commitments.

After a successful transmission, all participants receive the same $[m_1, \dots, m_i]$:

Algorithm 11 Preparation phase of the final round. Note that slots are variable in length and number, so they are only included, i.e., appended to a variable length list, when appropriate. For the same reason, commitments need to be generated for appropriate parts of correct length of the message, i.e., here 31 Bytes based on the base curve of the commitment scheme.

Input: Message m, r and ℓ from Algorithm 10, X from previous round
Environment: Group $G_{\text{self}} = \{g_i : i \in \{1 \dots k\}\}$, including the executing node g_{self}, executing node g_{self}
for $t \in \{1 \dots 2k\}$ **do**
  $(r[t], \ell[t], K[t]) = X[t]$
  if $\ell[t] = 0$ **then**
    continue
  end if
  $$\hat{Y} = \begin{cases} m & \text{if } r[t] = r \wedge \ell[t] = \ell \\ \underbrace{0 \dots 0}_{\ell[t] \text{ many}} & \text{else} \end{cases}$$
  for $i \in \{1 \dots k\}$ and part $\in \{1 \dots \lceil \frac{\ell[t]}{31} \rceil\}$ **do**
    $$\hat{s} = \begin{cases} \sim \mathcal{U}\{0, 2^{8\times 31} - 1\} & \text{if } i < k \\ \hat{Y}_{\text{part}} - \sum_{j=1}^{k-1} s_{\text{self},j} & \text{if } i = k \end{cases}$$
    {Note: $\hat{Y}_i$ references the i-th block of 31 Bytes of $\hat{Y}$}
    $\hat{r} = \text{PRNG}_{\text{K[t][self]}}(\sim \mathcal{U}\{0, 2^{8\times 31} - 1\})$
    $s_{\text{self},i}.\text{append}(\hat{s})$
    $r_{\text{self},i}.\text{append}(\hat{r})$
    $\hat{\mathcal{C}}_{\text{self},i}.\text{append}\,(C_{\hat{r}}(\hat{s}))$
  end for
end for
Broadcast $\{\hat{\mathcal{C}}_{\text{self},i} : i \in \{1 \dots k\}\}$

$$\begin{aligned} Y_{result} &\overset{Alg.\ 6}{=} \sum_j S_j \overset{Alg.\ 5}{=} \sum_j \sum_h s_{h,j} \\ &= \sum_h \sum_j s_{h,j} \overset{(*)}{=} \sum_h \left(s_{h,k} + \sum_{j \in \{1 \dots k-1\}} s_{k,j} \right) \\ &\overset{Alg.\ 11}{=} \sum_h \left((\hat{Y}[h] - \sum_{j \in \{1 \dots k-1\}} s_{k,j}) + \sum_{j \in \{1 \dots k-1\}} s_{k,j} \right) \\ &= \sum_h \hat{Y}[h] = \sum_h [0, \dots, 0, m_h, 0, \dots, 0] = [m_1, \dots, m_i]. \end{aligned}$$

Where $(*)$ just removed one entry from the sum.

Once the sender of a message notices a collision, i.e., their own message is damaged, they verify that all commitments of participants are correct. If a commitment of participant i for slot j does not correctly validate to zero, the sender creates a blame message of the form $(i, K_i, j, \text{round-offset})$. Here, the round offset identifies the instance of the protocol where the violation occurred, e.g., 1 for the previous instance. This message is inserted in the next round, allowing other participants to validate the blame without revealing the sender of the

message.

5.2 Privacy and Security

In this section, we discuss the privacy and security of our protocol. Here, privacy concerns itself with how to identify the originator of a given message. Security, on the other hand, covers the correctness and robustness of the protocol, i.e., if the protocol can be prevented from sharing the message.

Initial Round

Since the initial round is based on the k-anonymous message-transmission protocol proposed by von Ahn et al. [12], it exhibits the same correctness, robustness, fairness, and anonymity and is secure in the discrete logarithm model. Here, robustness means that either all honest participants that have a message to disseminate will eventually successfully transmit their message, or an attacker is exposed. To prevent a denial-of-service attack in the final round, the message lengths should be restricted, e.g., 2^{16} Bytes, when used in blockchain applications. Otherwise, an attacker could announce arbitrary message lengths, which would drastically slow-down the execution of the final round.

Final Round: Security

During the final round, the random values r for the commitments are generated using a pseudo random number generator seeded with the K_i distributed during the initial round. Therefore, the legitimate message sender can validate the commitments as zero commitments in case a collision occurs, since they can deterministically recompute all values r based on the K_i they sent, allowing them to check the commitment to zero.

If a commitment does not reveal to be zero, the legitimate message sender will inject a blame message in a later protocol instance. Other participants can verify the seed and validate that the commitment is not zero. If the accusation is true, they exclude the attacker. An honest participant can not be blamed without breaking the security assumption of the underlying commitment scheme. Therefore, either all messages are successfully transmitted or an attacker is identified and excluded.

The protocol is capable of dealing with non-cooperation, similar to the approach proposed by von Ahn et al. [12], by sharing encrypted instances of all $\{(s_{\text{self},j}, r_{self,j})\}_j$ pairs with every participant, allowing the reconstruction of the contents even during attacks using selective non-cooperation. This creates considerable additional load, so forming of a completely new group might be more economical.

Given the previous conclusions, the final round provides robustness properties similar to those of the initial round: either the transmission succeeds, or an attacker is excluded. Therefore, the protocol will eventually succeed, for a finite number of attackers and sufficiently many honest participants.

Final Round: Privacy

By construction, as it is using the same dissemination mechanism as the initial round, the protocol fulfills the privacy requirements of dining-cryptographers protocols. Nonetheless, we would like to highlight the privacy properties of important situations and how they come to be.

Revelations over recent years have shown that service providers and intelligence agencies across the world collect and analyze information, coming reasonably close to a global passive attacker. If such an observer could collect all slices $s_{i,j}$ of a participant, they could recompute the original message sent by the participant. This is prevented by authenticated encrypted channels between participants.

In order to prevent traffic correlation, DC networks require all participants to send the same amount of data during each round, to ensure that the senders cannot be identified based on the amount of generated network traffic. While a global passive attacker can detect the communicating DC groups and the broadcast message, they can not identify the originator within the group [12, 32]. Our protocol provides k-anonymity against this type of attacker, with $k = |\text{group}|$.

Therefore, to improve the detection of DC group participants, an attacker has to be part of the group. For β attackers within the group, DC-network based communication trivially provides $(k - \beta)$-anonymity: removing the β known keys s of attackers from the key-graph results in a remaining graph of size $k - \beta$, cf. Chaum [32].

The anonymity guarantees, i.e., the expected number of attackers β, depend on the group formation mechanism [12]. Current strategies of a random selection of participants with an assumed attacker probability p require group sizes of $\frac{2k}{1-p}$ to provide k-anonymity with high probability [12]. These bounds depend on the attacker probability distribution, which could be improved through external trust information during the group formation.

5.3 Performance Optimization

In this section, we discuss a multitude of optimizations built into the protocol, to improve its performance.

Small Optimizations

The protocol allows for various smaller optimizations to improve performance.

Precomputation: To reduce startup latency, commitments can be prepared before a protocol instance is started or during the downtime of a previous instance. The random blinding factors for the commitments of the initial round can be pre-generated. Since each participant has to generate at least $2k - 1$ zero commitments for the empty slots, these commitments can be precomputed as well.

Deferred validation: While the validation of commitments is important to identify misbehaving nodes, it is sufficient to validate the commitments only when malicious behavior is observed. To achieve this, messages and commitments should be stored on disk for later validation. Additionally, hash-based message codes augment the sent elements, i.e., the (r, ℓ, K) tuples in the initial round and messages in the final round, to detect misbehavior more reliably. A sender can always identify previously unnoticed misbehavior and trigger validation.

Direct transmission: If the message m a participant wants to send is short, i.e., shorter than the elements of the length message $|m| < |(r, \ell, K)|$, the message can replace the length identifier and be sent directly. To allow this behavior, as well as blame messages, some special values for r can be reserved to indicate a direct transmission.

Size of the Random Identifier: To keep the size of random identifiers small, we do a quick evaluation based on the birthday paradox formula. It provides a rough boundary, given a collision probability p and group size k:

$$\text{sizeof}(r) \geq \log_2 \left(\frac{1}{1 - (1-p)^{\frac{2}{k(k-1)}}} \right).$$

For a collision probability of 1%, we can tolerate 36 participants using 16 bits per random identifier, as collisions should already be rare due to requiring the same length. Most applications should function well with these values.

Overhead Reduction in Normal Operation

The proposed protocol has massive overhead to secure its operation in a malicious environment, e.g., commitments, seeds for commitments and more. This overhead can be severely reduced in an environment where maliciousness is the exception. For example, in normal operation, there is likely no attack on the robustness of the system. If a possible attack is detected, e.g., multiple collisions or garbage results in a round, the protocol can still switch to a secure mode. In its secure mode, the protocol is used as described in Section 5.1.

The unsecured variant removes several aspects. No commitments are created or added to the protocol. Therefore, no commitment seeds need to be generated and transmitted in the initial round. This results in a message of the form $(r_1, \ell_1), \ldots, (r_{2k}, \ell_{2k})$ which can be encoded as $2k$ integers of 4 or 8 bytes. It's important to note that this removal of commitments does not impact the privacy of the dining-cryptographers construction.

This construction results in a more complex protocol state machine, which is shown in Figure 5.2. The group init phase represents the (re)formation of the group, possibly expelling identified attackers. Once a group key G_{PC} is established, the protocol starts with unsecured rounds. Once a likely attack is detected, i.e., more slots are occupied than there are participants or there are many collisions, the protocol will switch to a secure variation.

If the attacker correctly participates in the secured round, this will impose the aforementioned overhead without any attacker being revealed. Depending on the scenario, instead of permanently operating in the secured mode, it may be advisable to create a group with fresh participants.

5.4 Conclusion

In this chapter, we provided an extension of von Ahn et al.'s protocol to arbitrary length messages. This allows its application for a sender-k-anonymous broadcast protocol for arbitrary messages. Further, we optimized the protocol to run faster in its secure mode as well as in a less secure mode for common use-cases, without compromising the privacy of the participants. With these changes, the resulting protocol is suitable to use in 3P3 as an implementation of the first phase, which provides a strong base privacy.

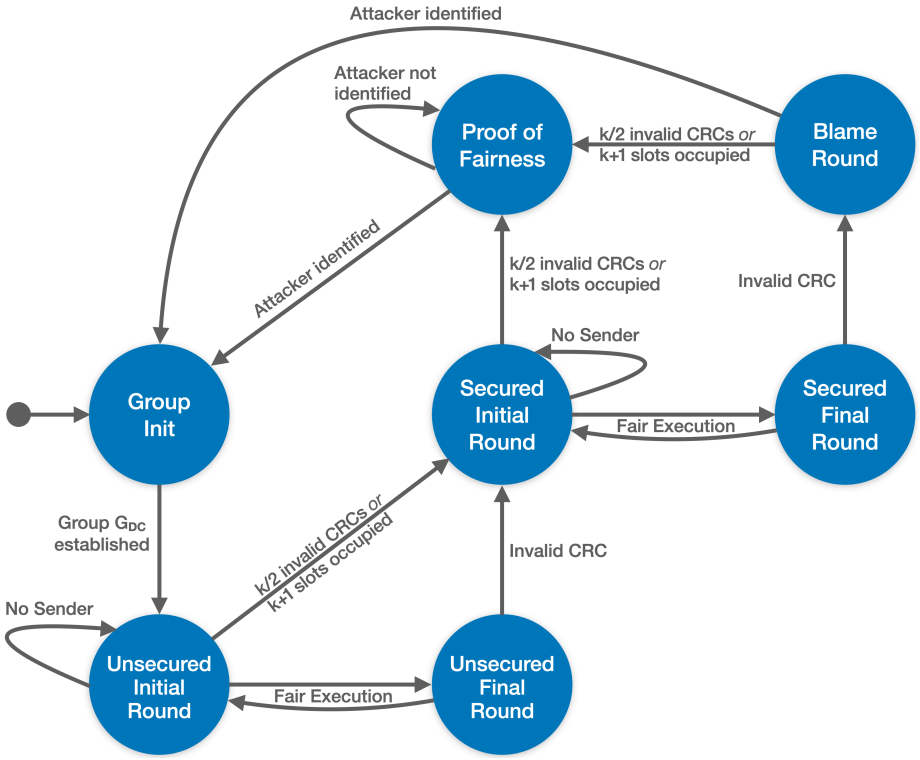


Figure 5.2: State machine of the optimized protocol. [9]

Chapter 6

Phase II: η-Adaptive Diffusion

This chapter is based on a paper published at Plos One [6].

[6] D. Mödinger, J.-H. Lorenz, and F. J. Hauck. "n-Adaptive Diffusion for k-Growing Networks". In: *Submitted to Plos One*. 2021.

In Chapter 4 we presented the design of 3P3. 3P3 requires a statistical privacy mechanism for its Phase II, to provide augmented privacy guarantees. The notion of statistical privacy was introduced in Chapter 2. Adaptive diffusion [46] is one of the representatives of this class of protocols. Unfortunately, the attacker model and network model of adaptive diffusion are not suitable for real-world computer networks, so it can not be used directly in 3P3. To make adaptive diffusion into a feasible network protocol, in this chapter, we:

- derive optimal forwarding probabilities for adaptive diffusion, based on the abstract distribution of shortest paths in the underlying network,
- model the distribution of shortest paths in k-growing graphs,
- provide an estimator for the distribution of shortest paths based on the number of participants n and edges per node k, and
- change the protocol so that it can withstand a more realistic attacker model.

This chapter is structured in the following way: Section 6.1 gives a short description of graph models for networks used in this chapter. Section 6.2 gives an overview of the resulting transformation of adaptive diffusion. In Sections 6.3 and 6.4 we discuss the details of the required changes to the protocol. Section 6.3 considers the changes in privacy and network assumptions of adaptive diffusion for the transformation to a network protocol. In Section 6.4, we investigate distributions to determine a concrete implementation of the probabilities involved with the protocol. To achieve this, we derive an estimation of the shortest paths for networks following a k-growing model. Section 6.5 discusses the privacy properties of the resulting protocol.

6.1 Graph Models for Networks

Within this chapter we perform some theoretical and practical analysis. For this analysis we require a model for peer-to-peer networks. As this chapter builds on adaptive diffusion, we reiterate the model used by Fanti et al. and describe the model we use to replace this original model.

Infinite Regular Trees

Adaptive diffusion [46] uses infinite d-regular trees as their base network model. Such a graph has no cycles, i.e., between any two nodes exists exactly one path. Further, each node is connected to exactly d neighbors via d edges and, therefore, has in- and out-degree d. As it is infinite, there is no number n limiting the number of nodes and no maximum distance, often called a graph diameter, within the graph. For a more in-depth introduction to graphs, we recommend Jackson's Social and Economic Networks [58].

Scenario

In this chapter, we discuss the privacy of broadcasts within an unstructured peer-to-peer network. For some applications, e.g., broadcasts of financial transactions in a blockchain network, the sender of a broadcast message has an interest in not being revealed. This is, despite the main goal being everyone receiving their message. The goal is, therefore, to hide the originator of such a message.

The default solution to broadcasting a message in an unstructured network is a flood-and-prune broadcast. Here, the sender sends the message to all its neighbors. A node that has not received the message yet will send it to all of its neighbors. The node excludes the link over which it received the message. Broadcasting, in this way, produces a highly symmetrical dissemination pattern, leading to possible identification attacks.

Assume there are nodes, which are collaborating to identify the originator of such a message, as shown in Figure 6.1. Those nodes might be distributed throughout the network and can learn the topology of the network over time. These nodes can reliably estimate the identity of the sender of the message by determining the graph center, or Jordan center, of the nodes that already received the message. The Jordan center of the graph is the node, which has the smallest distance to all affected nodes. Given a graph G with a set of vertices V and edges E, as well as a distance function d, the Jordan center can be defined as:

$$\text{center}(G_{V,E}) := \text{argmin}_{u \in V} \max(d(u, v) : v \in V).$$

In our scenario, the affected nodes are those that already received the message.

While varying network latencies might distort the result, the set of likely originators is small. Lastly, an attacker might also create connections to all nodes, always receiving the message as a neighbor of the true source.

Networks and Graphs

Peer-to-peer networks can be constructed in various ways [44]. One of the broadest distinctions of peer-to-peer networks is between structured and unstructured networks. Structured peer-to-peer networks tightly control their overlay, while unstructured peer-to-peer networks have peers join the network on loose rules. Unstructured networks often use broadcasts, often called flooding in peer-to-peer contexts, across the overlay [44].

One set of rules to create such a network has a new peer connect to nodes selected randomly from a list of known participants. This list can be initially retrieved by publicly known participants or via a gossip protocol while part of the network. The number of created connections maintained by new nodes is often fixed in the client source code. Examples of this construction are the network of Bitcoin [45] and classic Gnutella [93].

To model this behavior via graphs, we use an establishing algorithm. Let us try to establish a network of n nodes, or vertices, by successively adding nodes. Each node establishes k edges, or connections, to previously existing nodes. No node establishes loops, i.e., edges

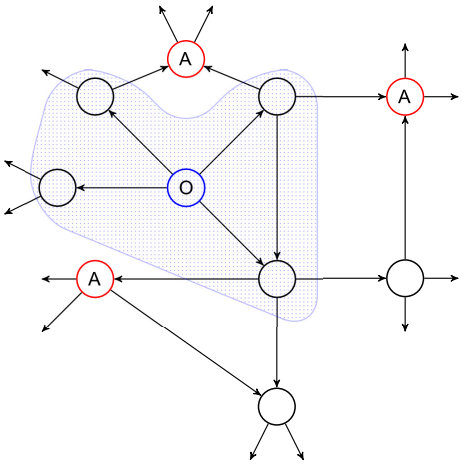


Figure 6.1: Motivating example: The originator of a message (O, blue) disseminates a message along the connections (arrows). The area highlights all nodes that received the message so far. Attackers (A, red) distributed throughout the network will receive the message in the next step and can reconstruct the originating node. [6]

with itself or multiple edges. The result is, therefore, a simple graph. In this thesis, we call this a k-growing graph with n nodes.

The previous design ensures a connected component of all network participants. Further, the design is resistant to churn, the act of peers joining and leaving the network, which is not reflected in the model. Churn within a model is a complicated parameters [104], as churn rates may differ for nodes dependent on their network participation, i.e., long-running nodes are often less likely to leave the network.

The original adaptive diffusion uses an infinite d-regular tree. Such a graph has no cycles and is connected, i.e., between any two nodes exists exactly one path. Further, each node has a degree of d, i.e., each node is connected to exactly d neighbors via d edges. As it is infinite, there is no number n limiting the number of nodes and no maximum distance within the graph, often called a graph diameter. For a more in-depth introduction to graphs, c.f. Jackson's Social and Economic Networks [58].

Attacker Model

Throughout this chapter, we consider attacks on the privacy of network participants. This attack is performed by a colluding fraction of participants of the network, which act in a semi-honest or honest-but-curious manner. Attackers follow the protocol but will attempt to infer the identity of the originator of a given message, i.e., which node created it. This model is used in similar settings [19], as it focuses on information leaks within the protocol.

6.2 k-Growing η-Adaptive Diffusion

The general idea is still that of adaptive diffusion: The virtual source should forward messages so that it is the Jordan center of the sub-graph created from all nodes that received the message. In detail, we apply some modifications to the protocol.

First, we limit the spread, i.e., the number of neighbors involved in the dissemination, to η many neighbors. This change reflects in the message handling sub-protocol Algorithm 12, as nodes need to select a limited set of neighbors for a given protocol run, compare Line 2. We store the selected neighbors $\mathcal{N}_m$ across multiple runs of the protocol, but for different messages m, the neighbors are selected again.

Further, arbitrary networks may have multiple paths between nodes, so a node may be selected as a neighbor for this protocol run, by multiple nodes. To prevent asymmetric spread, a node must only react on messages received via a single path. To enforce this, we store the first node we interact with given a message m as the predecessor_m, see Line 3.

Algorithm 12 η-adaptive diffusion message handling algorithm. [6]

Input: Message m
Environment: Message sender v, Self v_{self}
1: **if** $\text{predecessor}_m = \varnothing$ **then**
2: $\mathcal{N}_m =$randomly select η neighbors out of $N(v_{\text{self}}) \setminus \{v\}$
3: $\text{predecessor}_m = v$
4: **else**
5: **if** $\text{predecessor}_m = v$ **then**
6: send m to all $\mathcal{N}_m$
7: **end if**
8: **end if**

The virtual source sub-protocol Algorithm 13 requires further changes. The true source uses the message $(v, t = 1, r = \mathcal{H}(m))$ to initiate the protocol, which we call the virtual source token. $\mathcal{H}(m)$ is a suitable hash function to identify the current message efficiently. The hop counter h has been dropped, as it leaks the distance to the true source to possible attackers.

On receiving the virtual source token, e.g., via Line 12, the recipient balances the spread of the message, so they are the center of the spread graph. This is achieved by triggering the message handling algorithm on all neighbors, not including the node that sent the initiation message. This process is covered by lines 1 through 6.

The later part of the algorithm either forwards the message to all selected neighbors, see Line 16, which were selected in the message handling algorithm, Algorithm 12. Alternatively, the virtual source token is forwarded to a new virtual source with probability p_t, c.f., Line 10. The probability p_t can be computed based on the distance distribution within the network f, i.e., $f(i)$ gives the expected number of nodes in distance i of any node. The exact computation is quite involved; compare Section 6.3 for details.

After a suitable threshold is reached for the privacy of the originator, i.e., the set of potential originators is large enough, the protocol switched to a flood-and-prune broadcast. This will ensure delivery to all participants and increase efficiency. At this point, privacy would barely improve by continuing adaptive diffusion. Expected privacy has reached its maximum once the full network is part of the set of potential originators.

To preserve the privacy of participants, nodes must monitor network latencies, as they have to artificially slow down the protocol when keeping a virtual source token. Further, every virtual source node must monitor the network for the progress of the protocol. A time-out will trigger retransmission to a different participant, as the previously selected is considered as refusing cooperation or unreachable. The time-out will extend when a message related to the current protocol instance is received, i.e., it concerns the message m. The time-out will only stop when receiving a flood-and-prune message relating to the same message

Algorithm 13 η-Adaptive Diffusion virtual source handling algorithm.

Input: Previous virtual source v_p, message identifier $\mathcal{H}(m)$, current timestep t
Environment: Neighbors $\mathcal{N}_m$ with $|\mathcal{N}_m| = \eta + 1$, depth d

for $v \in \mathcal{N}_m \setminus \{v_p\}$ **do**
  Send m to v
  if $t + 1 \leq d$ **and** $t > 1$ **then**
    Send m to v
  end if
end for
while $t \leq d$ **do**
  $t = t + 1$
  $x =\sim \mathcal{U}(0, 1)$
  if $x \leq p_t$ **then**
    $v_{\text{next}} =\sim \mathcal{U}\{\mathcal{N}_m \setminus \{v_p\}\}$
    Send $(v_{\text{self}}, t, \mathcal{H}(m))$ to v_{next}, to call Algorithm 13
    break
  else
    Wait for $\approx$1 expected network latency
    for $v \in \mathcal{N}_m$ **do**
      Send m to v
    end for
  end if
end while

m.

6.3 Privacy for General Networks

Challenges

Some generalizations arise when considering general computer networks instead of infinite tree graphs. General networks may have cycles, i.e., multiple paths between participants, and a non-regular distance distribution. To extend the model of adaptive diffusion to these circumstances, we replace the calculations based on properties of a tree with a more general distribution function f, i.e., there are $f(i)$ nodes with distance i.

To prevent an attacker from learning additional information about the originator, we have to modify some aspects of the protocol. First, we have to remove the h used in the protocol, as an attacker can infer the exact distance to the originator. As other participants may not know the distance distribution of the originator and to keep the protocol general, we will use a homogeneous distribution, i.e., all nodes use the same distribution f to compute their probabilities.

First, we will analyze the ideal situation for virtual source passing. Based on the results in an ideal setting, we show the minimal required modifications for non-ideal settings.

Ideal Virtual Source Passing Probabilities

As there is no fixed topology to analyze, we need to model the process of passing the virtual source token in a more abstract way. To model the process, we use a time inhomogeneous

Markov chain, i.e., the probabilities involved may change based on the time t. For a network of diameter $\varnothing$, the chain has $\varnothing + 1$ states $0, 1, \ldots, \varnothing$. Each state represents the current distance of the virtual source from the true source.

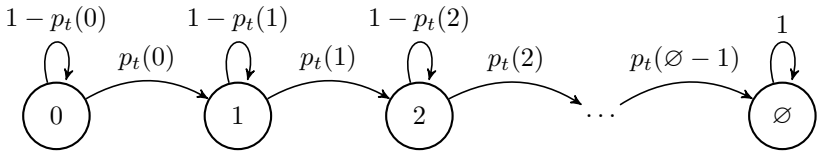


Figure 6.2: Time inhomogeneous Markov chain of passing the virtual source token. [6]

A node of distance h to the true source should pass the virtual source token to a more distant node with probability $p_t(h)$. Alternatively, the node keeps the distance the same with probability $1 - p_t(h)$. The Markov chain with these properties is visualized in Figure 6.2.

As noted before, a participant may not know its actual distance to the true source h, so the probabilities $p_t(h)$ may not depend on the distance to the true source.

At time t, let the i-th row of the vector $P_t \in [0, 1]^t$ describe the probability of the virtual source token being with a node of distance i from the true source. We have $P_1 = (1)$, as the true source has distance 0 to itself and has the token initially. Further, let $M_t \in [0, 1]^{t+1\times t}$ be the stochastic column matrix describing the transition from the t-th to the $(t+1)$-st step, i.e., $P_{t+1} = M_t P_t = M_t M_{t-1} \ldots M_1 P_1$. Based on our Markov model, the matrix M_t has the form:

$$M_t = \begin{pmatrix} 1 - p_t(0) & 0 & 0 & \cdots & 0 \\ p_t(0) & 1 - p_t(1) & 0 & & \vdots \\ 0 & p_t(1) & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & 0 \\ \vdots & & & \ddots & 1 - p_t(t-1) \\ 0 & \cdots & \cdots & 0 & p_t(t-1) \end{pmatrix}.$$

To solve for probabilities $p_t(h)$, we define our goal state: the probabilities for all any reachable node should be

$$\frac{1}{\#\text{reachable nodes in step } t} = \frac{1}{\sum_{s=0}^{t-1} f(t)}.$$

Using this, we can describe the probability of a node of distance i having the token at step t as

$$\mathfrak{f}_t(i) = \frac{f(i)}{\sum_{s=0}^{t-1} f(s)}.$$

Using this, we can write the goal of equal probability as

$$P_t = \begin{pmatrix} \mathfrak{f}_t(0) \\ \mathfrak{f}_t(1) \\ \vdots \\ \mathfrak{f}_t(t-1) \end{pmatrix} = \frac{1}{\sum_{s=0}^{t-1} f(s)} \begin{pmatrix} f(0) \\ f(1) \\ \vdots \\ f(t-1) \end{pmatrix} \stackrel{!}{=} M_t M_{t-1} \ldots M_1 P_1.$$

Unfortunately, the number of restrictions does not necessarily allow for a single solution, perfectly fulfilling our goal. We can compute a possible solution p_t based on the last row of our transition equation.

$$\begin{aligned} M_{t-1}P_{t-1} &= \begin{pmatrix} 1-p_{t-1}(0) & & \\ p_{t-1}(0) & \ddots & \\ & \ddots & 1-p_{t-1}(t-2) \\ & & p_{t-1}(t-2) \end{pmatrix} \begin{pmatrix} \mathfrak{f}_{t-1}(0) \\ \mathfrak{f}_{t-1}(1) \\ \vdots \\ \mathfrak{f}_{t-1}(t-2) \end{pmatrix} \\ &= \begin{pmatrix} \mathfrak{f}_t(0) \\ \mathfrak{f}_t(1) \\ \vdots \\ \mathfrak{f}_t(t-1) \end{pmatrix} = P_t \end{aligned}$$

Extracting the entries involved in computing the first entry of the target vector, we can compute $p_t(0)$. This result provides a base case for recursively computing solutions for all rows.

$$\begin{aligned} & (1-p_{t-1}(0))\mathfrak{f}_{t-1}(0) = \mathfrak{f}_t(0) \\ \Leftrightarrow \quad & 1-p_{t-1}(0) = \frac{\mathfrak{f}_t(0)}{\mathfrak{f}_{t-1}(0)} \\ \Leftrightarrow \quad & p_{t-1}(0) = 1 - \frac{\mathfrak{f}_t(0)}{\mathfrak{f}_{t-1}(0)} \\ \Leftrightarrow \quad & p_t(0) = 1 - \frac{\mathfrak{f}_{t+1}(0)}{\mathfrak{f}_t(0)} \end{aligned}$$

Using this base case, we can extract all entities involved in the $(i+1)$-st line and find a solution for $p_t(i)$, based on results established in previous lines.

$$\begin{aligned} & \mathfrak{f}_t(i) = p_{t-1}(i-1)\mathfrak{f}_{t-1}(i-1) + (1-p_{t-1}(i))\mathfrak{f}_{t-1}(i) \\ \Leftrightarrow \quad & (1-p_{t-1}(i))\mathfrak{f}_{t-1}(i) = \mathfrak{f}_t(i) - p_{t-1}(i-1)\mathfrak{f}_{t-1}(i-1) \\ \Leftrightarrow \quad & p_{t-1}(i) = 1 - \frac{\mathfrak{f}_t(i) - p_{t-1}(i-1)\mathfrak{f}_{t-1}(i-1)}{\mathfrak{f}_{t-1}(i)} \\ \Leftrightarrow \quad & p_t(i) = 1 - \frac{\mathfrak{f}_{t+1}(i) - p_t(i-1)\mathfrak{f}_t(i-1)}{\mathfrak{f}_t(i)} \end{aligned}$$

By induction we arrive at a cleaner iterative result, removing all mentions of p_t in the calculation.

$$\begin{aligned} & \underbrace{\mathfrak{f}_t(i)p_t(i)}_{a_i} = \mathfrak{f}_t(i) - \mathfrak{f}_{t+1}(i) + \underbrace{p_t(i-1)\mathfrak{f}_t(i-1)}_{a_{i-1}} \\ \Leftrightarrow \quad & \mathfrak{f}_t(i)p_t(i) = \sum_{j=0}^{i} (\mathfrak{f}_t(j) - \mathfrak{f}_{t+1}(j)) \\ \Leftrightarrow \quad & p_t(i) = \frac{\sum_{j=0}^{i} (\mathfrak{f}_t(j) - \mathfrak{f}_{t+1}(j))}{\mathfrak{f}_t(i)} \end{aligned}$$

As the actual distance of a node from the origin is unknown, we have to determine a single probability. As the distribution over h is known — it is our desired state $\mathfrak{f}_t$ — we can combine these with the precomputed probabilities per distance. This achieves a single forwarding probability:

$$p_t = \sum_{h=0}^{t-1} \mathfrak{f}_t(h) p_t(h).$$

A node that did not forward the token could recompute the forwarding probability using its expected distance from the previous round to achieve better hiding.

Non-Ideal Virtual Source Passing

The ideal solution only holds if and only if the next state P_t is reachable from P_{t-1} by a single increase or stay. The condition can be formalized with the following requirements, derived from the solution:

$$0 \leq \frac{\mathfrak{f}_t(0)}{\mathfrak{f}_{t-1}(0)} \leq 1 \tag{6.1}$$

$$0 \leq \frac{\sum_{j=0}^{i} (\mathfrak{f}_t(j) - \mathfrak{f}_{t+1}(j))}{\mathfrak{f}_t(i)} \leq 1 \tag{6.2}$$

Equation (6.1) is always true by construction, as $f(i) > 0$ and

$$\frac{\mathfrak{f}_t(0)}{\mathfrak{f}_{t-1}(0)} = \frac{\frac{f(0)}{\sum_{s=0}^{t-1} f(s)}}{\frac{f(0)}{\sum_{s=0}^{t-2} f(s)}} = \frac{\sum_{s=0}^{t-2} f(s)}{\sum_{s=0}^{t-1} f(s)} \leq 1.$$

Equation (6.2) intuitively describes that the probability of a node in distance j possessing the token cannot exceed the probability of a node of the same distance possessing the token in the previous time step in addition to the total change in lower distances.

If this condition is violated, we need to compensate in the distribution or probabilities. Either way, the resulting distribution will be non-optimal regarding its hiding capabilities. To minimize the deviation, we determine the final desired state of the protocol, after t steps, with $t \leq \varnothing$. We then compute a new $P'_i, \forall i \leq t$ as

$$P'_i = \begin{pmatrix} \mathfrak{f}'_i(0) \\ \vdots \\ \mathfrak{f}'_i(i-1) \end{pmatrix}$$

Here, $\mathfrak{f}'$ is derived from $\mathfrak{f}$ as:

$$\mathfrak{f}'_t(i) = \begin{cases} \mathfrak{f}_t(i) & \text{if } t \text{ is max desired state} \\ \mathfrak{f}_t(i) + \max(\chi_{t,i}, \delta_{t,i}) & \text{otherwise} \end{cases}$$

$$\chi_{t,i} = \sum_{j=i+1}^{t} \mathfrak{f}_t(j) - \mathfrak{f}'_t(j)$$

$$\delta_{t,i} = \mathfrak{f}'_{t+1}(t) - \mathfrak{f}_t(i) + \sum_{ji+1}^{t-1} \left(\mathfrak{f}'_{t+1}(j) - \mathfrak{f}'_t(j)\right)$$

The value $\delta_{t,i}$ represents the difference required to fulfil Equation (6.2). On the other hand, $\chi_{t,i}$ represents all changes made to later entries, i.e., propagating the changes made through δ. Note that Equation (6.2) is equivalent to the following.

$$\begin{aligned} 0 \leq \quad & \frac{\sum_{j=0}^{i}(\mathfrak{f}_t(j) - \mathfrak{f}_{t+1}(j))}{\mathfrak{f}_t(i)} \leq \quad 1 \\ 0 \leq \quad & \sum_{j=0}^{i}(\mathfrak{f}_t(j) - \mathfrak{f}_{t+1}(j)) \leq \quad \mathfrak{f}_t(i) \end{aligned}$$

Applying this equation to our goal state $\mathfrak{f}'$ we find the generation of $\mathfrak{f}'$ through the following changes:

$$\begin{aligned} \mathfrak{f}'_t(i) \geq & \sum_{j=0}^{i}(\mathfrak{f}'_t(j) - \mathfrak{f}'_{t+1}(j)) \\ = & \sum_{j=0}^{i}\mathfrak{f}'_t(j) - \sum_{j=0}^{i}\mathfrak{f}'_{t+1}(j) \\ \overset{\sum_{j=0}^{k-1}\mathfrak{f}_k(j)=1}{=} & 1 - \sum_{j=i+1}^{t-1}\mathfrak{f}'_t(j) - (1 - \sum_{j=i+1}^{t}\mathfrak{f}'_{t+1}(j)) \\ = & \sum_{j=i+1}^{t-1}\mathfrak{f}'_t(j) + \sum_{j=i+1}^{t}\mathfrak{f}'_{t+1}(j) \\ = & \sum_{j=i+1}^{t-1}\mathfrak{f}'_t(j) + \sum_{j=i+1}^{t-1}\mathfrak{f}'_{t+1}(j) + \mathfrak{f}'_{t+1}(t) \\ = & \sum_{j=i+1}^{t-1}\left(\mathfrak{f}'_t(j) + \mathfrak{f}'_{t+1}(j)\right) + \mathfrak{f}'_{t+1}(t). \end{aligned}$$

This leaves us with only known values, allowing us to compute the minimum difference required, i.e., $\delta_{t,i}$. We showed that if it is possible to achieve an optimal result, probabilities derived from $\mathfrak{f}'$ are optimal. If such a result is not possible, probabilities derived from $\mathfrak{f}'$ will yield a result with minimum deviation for intermediate steps.

Continuous Time

All previous discussions are in discrete time, i.e., the time t is in steps, especially natural numbers. A network protocol must operate in some form of continuous-time or at discrete timesteps small enough to be considered continuous for practical purposes. Fortunately, network protocols lend themselves for a simple conversion technique: network latency.

If there was no delay between messages, a token transfer to another node could be observed by all participants of the protocol so far. To prevent this observation, a node must insert an artificial latency when not forwarding the message. The latency should be drawn from a distribution indistinguishable from real network latencies. Therefore, a node must observe the latencies of its connections.

Spread reduction

One remaining privacy problem of adaptive diffusion is the selection of all neighbors for dissemination. If an attacker is a neighbor of the first recipient of the virtual source token, they will notice the broadcast as soon as possible without being the first virtual source recipient. An attacker can force this situation by creating connections to all participants in the network. Even with many unobtrusive attackers distributed throughout the network, the chance of selection is high.

To reduce privacy leaks, we introduce the parameter η. Participants only select η neighbors to participate in the protocol instead of all neighbors. This reduces the chance of selecting at least one attacker.

Limiting the set of participating neighbors prevents full coverage of the network through adaptive diffusion alone. Therefore, an additional flood-and-prune phase is necessary to ensure delivery to all network participants. Lastly, lower values of η increase the required time to reach larger parts of the network, e.g., 21 nodes are reached after three spread rounds with $\eta = 2$, while $\eta = 5$ reaches 30 nodes in two spread rounds.

Limitations

Unfortunately, a node can not reliably decide which edge increase or decrease the distance to the true source, as the source is unknown or an edge with the desired probability is not available. For every node, keeping the token will keep the distance the same. As a heuristic for early nodes, returning the token along the path it was received likely reduces the distance by one, while forwarding it to another node likely increases the distance. Knowledge about the neighbors of neighbors can increase the accuracy of this heuristic.

For networks observed in real-world peer-to-peer-networks, the small-world property likely holds [58, 116]: the shortest distance between any two nodes is likely below or equal to 6. After six steps, the candidate pool for the true source is most of the network. Therefore, performing the analysis is sufficient for the early steps of the protocol.

Due to the lack of information stemming from the privacy requirements, the distribution of nodes holding the virtual source is distorted at every step, making the result less accurate. Alternative approximations based on the distribution may perform better empirically. One improvement may be a better approximation by nodes holding the virtual source token. They can infer their distance to the true source to be at most t the moment they receive the token. Therefore, they have no reason to use the probabilities as if they were at distance $t + 1$ should they keep the token.

Lastly, the result of the previous section is based on a distribution of the shortest paths within the networks. This distribution is not generally known for most graph types and could not be empirically determined by a participant without knowledge of the topology.

6.4 Distribution Model

We analyzed expected k-growing network topologies, which are similar to real-world peer-to-peer network growth, for their distance distributions. This relieves the final limitation, knowledge about a concrete distribution. The result allows a node to compute p_t based on the number of expected edges per node and the number of nodes in the network.

Distributions for Alternative Models

Fronczak et al. [49] derive an exact solution for random Erdös-Rényi graphs, i.e., random graphs where all edges are equally likely. They especially consider Erdös-Rényi graphs with two hidden variables h_i and h_j. Let $\gamma \approx 0.5772$ be Euler's constant, the resulting average degree distribution is given by

$$l = \frac{-2(\ln h) + \ln N + \ln(h^2) - \gamma}{\ln N + \ln(h^2) - \ln \beta} + \frac{1}{2}.$$

Loguinov et al. [72] investigate structured peer-to-peer networks. They provide a succinct overview over shortest path results for Chord and CAN networks. Chord shortest paths are binomial distributed, which tends to a normal distribution for larger values, while CAN becomes normal as well, for increasing CAN dimensions. Lastly, they propose an architecture using de Bruijn graphs, which have no closed-form for their distribution of shortest paths but give an exponential approximation.

Roos et al. [96] derive a model for Kademlia like systems. Kademlia is a structured peer-to-peer network with routing based on b-bit long identifier spaces. They consider a network of order n, where routing considers α close nodes to reach β nodes close to the target, based on a k-bucket routing system. They model the hop count distribution of a given system via a Markov chain. They derive a space complexity of $\mathcal{O}(\frac{b^{2\alpha}}{(\alpha!)^2})$ and computation complexity of $\mathcal{O}(nb^{\alpha(\beta+2)})$ for their resulting model. As α, β and k are usually constant for a given deployment, it is manageable for network participants but not optimal. For details about the fairly complex model, refer to the original publication [96].

The results provided by these works are applicable if the network conforms to the proposed structure. Unfortunately, they do not map well to the proposed model, lack an approach for parameter inference and are complex to use.

Methodology

We determine suitable distributions and their parameters by creating and analyzing random graphs. To create the graphs, we use igraph[1], while the analysis is done using scipy [62].

We chose igraph's graph establishment function, which takes a number of nodes n and a number of edges per node k. The method creates a random graph by sequentially adding nodes. Each node creates k edges to already existing nodes. This scheme leads to a connected graph, where older nodes have a higher number of connections, while new nodes have at least k connections.

We chose this scheme as it is similar to the schemes used in peer-to-peer networks. A new node connects to publicly known nodes and asks for a set of participants. The new node then chooses some number of nodes to connect to. This model is a simplification, as it ignores churn, i.e., nodes leaving and joining the network again, but it is a close fit for real-world applications.

To reproduce the steps and results of this chapter, we provide a repository of our data and scripts under the MIT open source license[2], including an interactive notebook for experimentation.

[1] `https://igraph.org/`

[2] `https://github.com/vs-uulm/eta-adaptive`

Models for Distance Distribution

To model the observed behavior, we chose various discrete and continuous distributions. As discrete candidates, we looked at Poisson, Planck, Binomial and Geometric distributions. For continuous distributions, we considered the normal, log-normal, truncated normal and Weibull distribution. We evaluated the fit of continuous distributions by the overall shape, as the data is discrete.

The Weibull distribution was chosen as a candidate for the extreme value distributions, as shortest paths are calculated as minimums over paths. The normal distribution was chosen due to the central limit theorem, i.e., the normal distribution as the limit of independent samplings. The log-normal and truncated normal distributions were selected as a candidate as its support can be limited to positive values — a sensible limitation for path lengths.

We estimated parameter fits for all distributions from many generated graphs. The truncated normal was mostly indistinguishable from the produced normal distribution. Similarly, the log-normal distribution was transformed to mimic the normal fit closely. Therefore, we removed the truncated and log-normal distribution as candidates to not over-complicate the model. Representative examples for continuous fits are shown in Figure 6.3.

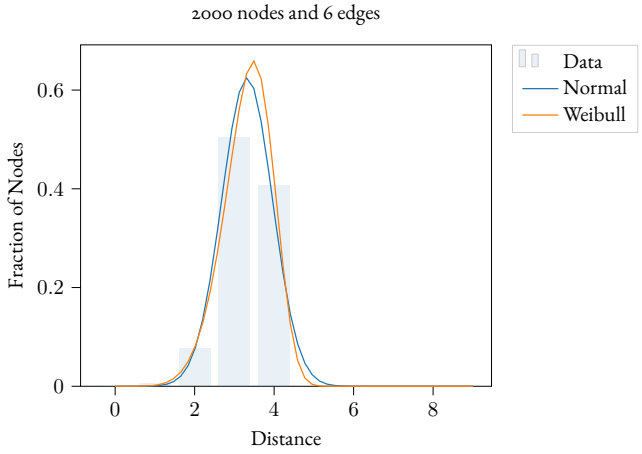


Figure 6.3: Fitted continuous distributions from an example dataset, which was created using 2000 nodes and 6 edges per node. A lognormal and truncated normal fit were plotted identically to the normal distribution and were, therefore, omitted. [6]

Similarly, the results for the discrete distributions was not a good fit. Only the Poisson distribution produced a convincing fit for any graph but limited to graphs with $k = 1$. We did not test other discrete distributions as often no efficient maximum likelihood estimators exist or are implemented. Representative examples for discrete fits are shown in Figure 6.4.

Finally, the results for the normal distribution produced good point-wise fits for graphs with $k > 1$. The normal distribution fits were especially accurate for the core section of the distribution, which is also consistent with findings of normally distributed path lengths in other peer-to-peer networks [72]. The most significant deviation from the data could be observed for the low end of the distribution: for distance 0 or 1. Fortunately, these values can be fitted manually based on the parameters n, k, as the mass at a distance of zero should be $\frac{1}{n}$ and the expected mass at the distance of one should be $\frac{k(2n-k-1)}{n^2}$, i.e., the average degree.

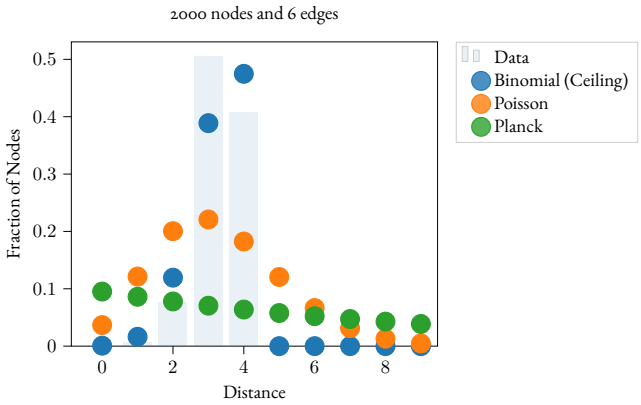


Figure 6.4: Fitted discrete distributions from an example dataset, which was created using 2000 nodes and 6 edges. Only the binomial estimation using the ceiling operator to discretize the parameters shows any resemblance to the desired data. [6]

Discretization

To apply the normal distribution to our given problem, the resulting distribution needs to be discretized, i.e., turned from a continuous distribution in a discrete one. The main goal is to keep the properties of a probability distribution, i.e., the sum of all points not equal to zero needs to add up to 1.

A valid discretization can be constructed based on the cumulative distribution function (CDF) over intervals, capturing the full support of the distribution, e.g., $f(x) = \mathrm{CDF}(x)$

As we fit the distribution based on the points of data, the natural discretization can be achieved by point-wise evaluation and re-normalization of the result. Let PDF be the probability density function, then a new probability mass function with evaluation points $0, 1, \ldots, t$ (the discrete equivalent to a PDF) is given by $x \in \{0, 1, \ldots, t\}$

$$f_t(x) = \frac{\mathrm{PDF}(x)}{\sum_{s=0}^{t} \mathrm{PDF}(s)}.$$

This approach can easily accompany special values at certain points. Therefore, the full discretization of our normal distribution shall be:

$$\begin{aligned} f(0) &= \frac{1}{n} \\ f(1) &= \frac{k(2n-k-1)}{n^2} \\ f(x) &= \frac{\mathrm{PDF}(x)}{f(0) + f(1) + \sum_{s=2}^{t} \mathrm{PDF}(s)}. \end{aligned}$$

The maximum point t should be chosen in such a way, that the remaining error $1 - \mathrm{CDF}(t) \leq \epsilon$ is small enough for the given purpose.

Model Parameter Estimator

The previous section concluded that the distribution of shortest paths could be modeled using a discretized normal distribution. Building upon this conclusion, we are further inter-

ested in the parameters μ and σ^2 of a normal distribution $\mathcal{N}(\mu, \sigma^2)$. The parameters of the normal distribution should only depend on the parameters of our network, the number of nodes n and number of edges k. We are interested in functions M, S, with a small error err such that

$$\mu = M(n,k) + \text{err}$$
$$\sigma = S(n,k) + \text{err}\,.$$

These are statistical estimators. To determine these, we fitted a large number of randomly generated graphs and stored the resulting values for μ and σ. The determined functions M, S are approximated using the functional equations:

$$M(n,k) \approx \frac{\alpha \ln(\beta n)}{e^{\gamma k}} + \delta \ln(\eta n) + \frac{\zeta}{e^{\gamma k}} + \epsilon,$$
$$S(n,k) \approx \mathfrak{a} \ln(\mathfrak{b} n) + \frac{\mathfrak{c}}{e^{\mathfrak{d} k}} + \mathfrak{e}.$$

Here, the greek and fracture constants α to ϵ and $\mathfrak{a}$ through $\mathfrak{e}$ are determined by least-square fitting of the function to the acquired data. Through a fit of experimental data, we reached the following approximate functions:

$$M(n,k) \approx \frac{0.595 \log(2.135n)}{\exp(0.314k)} + 0.341 \log(1.626n) + \frac{0.241}{\exp(0.314k)} - 0.224,$$
$$S(n,k) \approx 0.0345 \log(0.925n) + \frac{1.222}{\exp(0.301k)} + 0.189.$$

Landscape

We used the fitted parameters for random graphs to determine the behavior of the parameters. The ranges of the parameters depend on the size of the network n and the number of connections created in each step k.

By splitting the dimensions based on n and k, an initial estimation is possible. The dimension dependent on k shows a behavior proportional to $\frac{1}{e^k}$. The dimension dependent on n, on the other hand, shows a behavior proportional to $\log(n)$, a square root behavior could be excluded as fitted parameters easily overestimated the data. A random selection of the dimensional analysis is shown in Figure 6.5.

The estimation of values for σ show much more pronounced residues in the form of a saw-tooth function. The forms can be recognized from the similarly shaped but smaller residues of the μ estimations. The values of σ show to be within 0.3 to 0.7, even for large numbers of nodes n, e.g., $n = 1\,000\,000$. Further, the values seem to jump rapidly and slowly descend, forming a saw-tooth pattern, which is hard to predict accurately. The pattern arises as additional nodes in the network are more likely to create shortcuts than to increase path length until the overall network diameter increases by one, steeply widening the distribution - and therefore increasing the variance, i.e., σ. The likelihood of such an increase follows its own probability distribution, which we did not determine for this thesis.

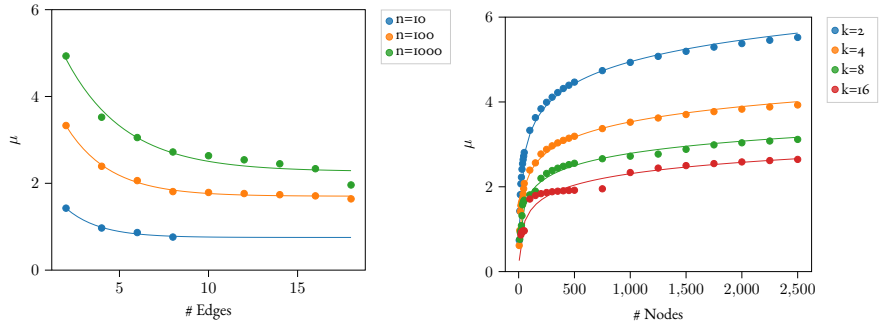


Figure 6.5: Datapoints for various graph sizes split by number of edges per node k and number of nodes n. μ estimate fitted using $\frac{1}{e^k}$ for the number of edges and fitted using $\log(n)$ for the number of nodes dimension. [6]

Estimator Models

Based on our one dimensional evaluation, we want to construct a two dimensional estimator model $M(n,k)$. Model candidates are based on possible combinations of our one dimensional approximations, i.e., the partial derivatives are the derivatives of our observations:

$$\frac{\partial M}{\partial n} \approx \frac{d}{dn}\log(n),$$
$$\frac{\partial M}{\partial k} \approx \frac{d}{dk}\frac{1}{e^k}.$$

The constructed models have various constants, denoted by greek lower case symbols. These constants are not shared between the models but were independently fitted. The models are denoted by

$$M_1(n,k) = \alpha\ln(\beta n) + \frac{\gamma}{e^{\delta k}} + \epsilon,$$
$$M_2(n,k) = \frac{\alpha\ln(\beta n)}{e^{\gamma k}} + \delta\ln(\eta n) + \frac{\zeta}{e^{\gamma k}} + \epsilon,$$
$$M_3(n,k) = \frac{\alpha\ln(\beta n)}{e^{\gamma k}} + \epsilon,$$
$$M_4(n,k) = \frac{\alpha\ln(\beta n)}{e^{\gamma k}} - \frac{\alpha\delta k}{e^{\gamma k}} + \frac{\zeta}{e^{\eta k}} + \frac{\theta\ln(\iota n)}{(\kappa n)^{\lambda n}} + \nu\ln(\xi n).$$

We fit the parameters of the model based on our first dataset. The residuals of the fit parameters, i.e., the difference between the true value and estimation, shows a saw-tooth form. This arises as additional nodes likely create shortcuts until the overall diameter of the network grows.

To evaluate the performance besides this observed error, we created a new independent dataset. We measured the difference between the calculated values by our model and the actual fitted parameters. This difference corresponds to the bias of the estimator, which is simply referred to as bias. Figure 6.6 shows the distribution of the bias of our models for μ.

For the μ estimation, model M_2 and M_4 perform the best. Model M_2 requires less parameters than M_4, i.e., it is simpler, therefore we prefer model M_2.

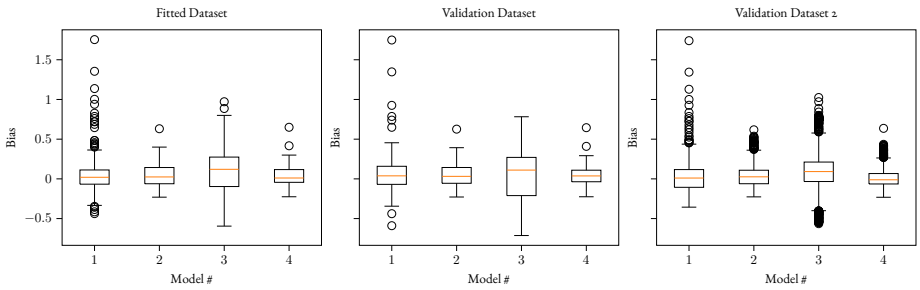


Figure 6.6: Boxplot of the bias distribution of the μ-estimator, using the four models, compared to the measured value. [6]

For the estimation of σ, none of the models performs exceptionally well. In general, the observed sigma values are small and within the range 0.3-0.7, even for graphs using one million nodes. As no model performed exceptionally well, but also not exceptionally bad, we stuck with the simplest model, i.e., model 1. Figure 6.7 provides a discretization of an example prediction for a dataset based on 2000 nodes and 6 edges.

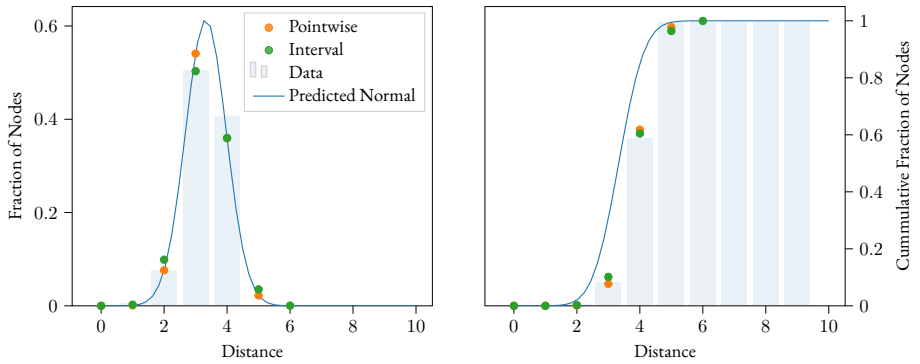


Figure 6.7: Discretization of an example dataset using 2000 nodes and 6 edges. The predicted normal distribution is based on parameters estimated using model 2 for μ and model 1 for σ, with a point-wise discretization and an interval discretization, using midpoint intervals. [6]

We can now use these values to compute concrete p_t for a given network of n nodes using a k-growing approach from Section 6.3.

6.5 Privacy Discussion

In this section, we investigate the privacy impact of the protocol. The main challenge for quantification is the arbitrary topology and topology abstractions.

Model

Given a network of size n, we have a set of attackers A of size $|A|$ participating in the network; therefore, $n - |A|$ is the number of all fully honest nodes. The value $\beta = \frac{|A|}{n}$ represents the fraction of attackers within the network, which is the probability of selecting an attacker

when selecting a node from the network uniformly at random. Lastly, any node has an expected number of connections $c > 1$ to other nodes.

To locate a node in a flood-and-prune broadcast, an attacker node registers an incoming message through one of its c connections. Given the knowledge of the topology, the attacker can separate the network in c many sets of nodes, where any node is in the same set, if a broadcast from this node would reach the attacker via this connection. These sets are not necessarily disjunct, as there may be multiple shortest paths between a node and the attacker. To find a lower bound on the privacy effects, we assume the sets are disjunct, as this improves the position of the attackers.

The expected size of such a set for a single attacker node will be $\frac{n-|A|}{c}$, as an attacker would not consider colluding attackers. For this model, we consider only untargeted attacks, where attackers try to deanonymize any sending node, not a single particular node. In this case, the ideal set of network nodes any attacker can distinguish based on a message is $\frac{n-|A|}{c}$. Further, ideal sets for attackers minimize the size of multiple observing attackers to the size of $\frac{n-|A|}{c^{|A|}}$.

Attackers can fully deanonymize a sending node once this size is below one. It follows that attackers can correctly determine the location of a sender in this ideal setting, once more than $\log_c(n - |A|)$ attackers have received the message, as it holds, that

$$\frac{n - |A|}{c^{|A|}} \leq 1 \Leftrightarrow |A| \geq \log_c(n - |A|).$$

Spreading Sub-Protocol

Given the required number of attackers, we are interested in when this number is likely reached during the spreading protocol. Nodes that select neighbors may select nodes that already participate in the protocol, so the tree sub-graph resulting from participating nodes is not complete.

If all nodes select their neighbors uniformly at random from the full set of network participants, this becomes a form of the coupon collectors problem in packs [102]. The coupon collectors problem using packs describes the problem of receiving at least one of each coupon when receiving coupons in packs of a given size. Here, the size of the packs is given by η, and each network node represents a coupon.

Given a subset A of all coupons, the random variable $Z_\ell(A)$ is the number of drawings necessary to obtain at least ℓ elements of A [102]. The expected value of reaching the required number of attackers $Z_{\log_c(n-|A|)}(A)$ can be calculated using the results of Stadje:

$$E(Z_{\log_c(n-|A|)}(A)) = \binom{n}{c} \sum_{j=0}^{\lceil \log_c(n-|A|)\rceil - 1} \frac{(-1)^{\lceil \log_c(n-|A|)\rceil - j + 1}}{\binom{n}{c} - \binom{n-|A|+j}{c}} \binom{|A|}{j} \binom{|A| - j - 1}{|A| - \lceil \log_c(n - |A|)\rceil}. \tag{6.3}$$

The number of drawings corresponds to the number of nodes participating in the protocol before reaching the required number of attackers. To calculate a lower bound of the depth of the η-adaptive diffusion dissemination tree at this point, we invert the calculation of the number of nodes in a complete tree.

$$Z_{\log_c(n-|A|)}(A) \geq 1 + (\eta + 1) \sum_{i=0}^{t-2} \eta^i \tag{6.4}$$

$$\Leftrightarrow t \geq \log_\eta \left(1 - \frac{(Z_{\log_c(n-|A|)}(A))(1-\eta)}{\eta + 1} \right) + 1 \tag{6.5}$$

Table 6.1 provides an overview of the evaluation of this function when 5% of the network is colluding. It shows that for low values of η a significant depth can be reached, as four steps in a network with an average degree of $c = 8$ results in a candidate pool for the true source for approximately $c^4 = 8^4 = 4096$. The likelihood of any of the candidate nodes being the true source depends on the distribution $\mathfrak{f}$ of Section 6.3.

Table 6.1: Expected tree depth for an attacker fraction of $\beta = 0.05$ before deanonymization of the virtual source.

η \ n	100	1000	10000
3	4.3	3.9	7.8
5	2.5	2.7	2.7
10	1.6	1.7	1.9

Virtual Source Sub-Protocol

A node received additional information when chosen as the virtual source. An attacker is chosen as the virtual source with probability

$$P[\text{virtualsource} \in A] = \frac{|A|}{n-1}.$$

In this formulation, a successful selection occurs when the virtual source is an attacker. This is a simple Bernoulli trial, so the geometric distribution gives the expected number of trials until a success occurs as

$$E(P[\text{virtualsource} \in A]) = \frac{n-1}{|A|} \approx \frac{1}{\beta}.$$

For a reasonable network size ($n > 100$) this value stays above 6 for fractions of attackers below $\beta = 0.166$. This result is expected, as the process of virtual source selection mimics the privacy-optimal process of the work by Bellet et al. [19] with a muting parameter of $s = 0$ and earlier similar results.

6.6 Future Work

There are various smaller improvements possible to increase the privacy or efficiency of the protocol. First, instead of switching to a flood-and-prune from the last virtual source node, the protocol could instead trigger the flood-and-prune broadcast from all leaf nodes. The last message transmission message, see Algorithm 12, would instruct leaf nodes to start a flood-and-prune process. This would reduce leaks of information during the flood-and-prune protocol and improve the efficiency of the protocol.

To improve resistance against linkable broadcasts and to hinder an attacker in the first step, the current timestep t may be randomized on initiating. This would also require the originator to use a spreading message first, as the initial transmission would otherwise be special, as a node should only receive the virtual source after receiving a spreading message.

The protocol has little guards against non-participation attacks and communication failures, which could be mitigated through retransmissions and time-outs. While allowing the protocol to complete, these would still reduce the efficiency of the protocol in selective non-participation attacks. As those do not prevent every connected node from receiving the message and do not diminish the privacy results, we did not tackle these in this paper.

Lastly, a more extensive privacy analysis would benefit the protocol. Due to the lack of topology information, the privacy analysis is limited in its applicability. A more accurate result could be achieved for specific topologies or considering distributions over topologies.

6.7 Conclusion

In this chapter, we transformed the adaptive diffusion protocol [46] into a protocol for peer-to-peer networks. To achieve this, we remodeled the virtual source passing probabilities in a more general way, based on the distance distribution of the underlying network and improved the attacker model by removing information from protocol messages. Further, we provide a privacy-friendly solution to solve these equations, while smoothing out otherwise unachievable states.

We analyzed expected k-growing network topologies, which behave similar to real-world peer-to-peer network growth, for their distance distributions. The analysis showed the distances in the networks to be approximately normally distributed. Lastly, we performed a parameter analysis of the resulting normal distributions, showing that μ and σ of the normal distribution can be approximated by a combination of logarithmic and inverse exponentials.

The provided results allow for a concrete instantiation of this revised η-adaptive diffusion to be used in 3P3 as a phase II protocol.

Chapter 7

Security and Privacy

This chapter presents results of previous publications [4, 5] (with permission, © 2018 IEEE).

[4] D. Mödinger and F. J. Hauck. "3P3: Strong Flexible Privacy for Broadcasts". In: *4th International Workshop on Cyberspace Security (IWCSS 2020)*. 2020.

The previous chapters introduced 3P3, a strong privacy protocol for broadcasts. While we discussed some privacy and security implications of the design in the chapters detailing each phase, the overall security and privacy discussion of 3P3 remains.

In this chapter, we first discuss the security and robustness of 3P3 in Section 7.1. Further, we analyze the privacy aspects of 3P3 in Section 7.2.

7.1 Security: Functionality under Attack

For the protocol to be considered secure and correct, all non-malicious nodes should receive a disseminated message. It is sufficient to show that the last phase will always be reached and that it fulfills the desired requirements. Therefore, after introducing the considered attacker model, we work backwards from the last phase.

Attacker Model

The base model for the network, cf. Section 2.2, especially includes a connected graph after removing all malicious nodes from the network. Malicious nodes are interested in preventing a message from being broadcast. They are considered successful if any honest nodes do not receive the message, despite all honest nodes forming a connected graph.

Attackers are computationally limited, they are especially not able to break cryptographic primitives, such as decrypting messaged without the proper key. As channels and transactions use strong cryptographic primitives, the restriction results in authenticated and secure channels and messages. Attackers act as participants of the network, not as internet service providers, hardware vendors or other outside entities. We do not consider vulnerabilities of implementations or general systems security. Intuitively, we limit the attackers to exploit weaknesses in the protocol by sending, forging or inserting messaged or by refusing participation.

Phase III: Flood-and-Prune Broadcast

When removing malicious nodes, the network remains a connected graph, based on the network requirements. Therefore, there exists a path between any node and the initiator of the flood-and-prune broadcast. The communication can be reduced to any two neighboring nodes, with one receiving the message. The node will forward it to the neighbor, propagating along all available paths. This process is unaffected by non-participation or message injection. Therefore, we only require reaching the flood-and-prune stage and having an honest initiator to reach all nodes.

Phase II: Diffusion

We restrict the discussion to the virtual source sub-protocol. Reaching Phase 3 only hinges on this sub-protocol, so this restriction is warranted.

Let v_c be the current virtual source node and v_p the previous virtual source node. We can separate two cases. Either, v_c is fully uncooperative, i.e., sends no correct message. Alternatively, they are partly uncooperative by only sending correct messages back to v_p. Other cases, such as randomly sending correct and incorrect messages, can be treated as the partially uncooperative case.

Fully uncooperative: v_p will not detect any correct messages. Therefore, the timeout of v_p will trigger, marking v_c as failed and continue the protocol itself. As a connected graph is available, even after removing all malicious or failed nodes, the protocol can continue eventually.

Partially uncooperative: v_p will not be able to distinguish this case from a fully functional run of the protocol. This situation is the case for all previous virtual source nodes. Either, the attacker switches to a flood-and-prune broadcast, but only forwarding it towards v_p or the attacker will send infinite diffusion messages. In the case of the flood-and-prune broadcast, the protocol is successful, due to the connected network. To combat infinite diffusion messages, v_p determines a maximum bound of messages they should observe, based on the state of the timestep counter s they received. A malicious node can therefore not send endless messages towards v_p to stall progress towards Phase 3.

Based on the observation that both cases are dealt with, the last phase will be initiated if there is any available honest virtual source. Therefore, to reach all nodes, we only require reaching the diffusion stage and having an honest initiator of the diffusion stage.

Phase I: Dining-Cryptographers Network

The first instance of the DC network corresponds to the protocol by von Ahn et al. [12] with $m = (r, \ell, K)$. Therefore, round one exhibits the same correctness, robustness, fairness and anonymity as the von Ahn protocol, which is secure as long as computing the discrete logarithm is considered hard. Whereby robustness means either the protocol succeeds, or an attacker is exposed, so the protocol will eventually succeed for finite attackers.

On successful transmission of the second instance, all participants will receive the same list of messages $[m_1, \ldots, m_i]$, as shown previously. The commitments of round two are created using a PRNG, whose seed is K_i from the previous instance. If a collision occurs, the legitimate message sender can validate the commitments as zero commitments, as they know $r = K_i$ and the commitment. If a commitment does not reveal to be zero, the legitimate message sender will inject a blame message in the next protocol instance. The blame replaces the message identifiers r, ℓ, K and contains the seed, the blamed participant as well as a round

identifier and message identifier. Other participants can verify the seed and that the commitment is not zero, and exclude the attacker. An honest participant can not be blamed without breaking the security assumption of the underlying commitments.

The case of non-cooperation by participants can be handled similar to von Ahn et al. [12] by sharing encrypted instances of all $\{(s_{\text{self},j}, r_{self,j})\}_j$ pairs with every participant, allowing for reconstruction of the contents with selective non-cooperation. This creates considerable additional load, so forming of a completely new group might be more economical.

As all sub-protocols are robust, we conclude 3P3 to either succeed or remove an attacker and therefore to be robust in its entirety.

Further Security Concerns

Besides the basic function of the protocol, we would like to note some security concerns. We do not consider confidentiality, as we are dealing with a broadcast protocol. Similarly, to keep messages smaller by default, we have no additional integrity checks, especially as these are often included in application protocols. Any application that requires integrity or confidentiality needs to provide application layer solutions.

Lastly, a node holding the virtual source token can decide to start the last phase of the protocol early. Nodes, beside previous virtual source nodes, can not distinguish this situation from a valid phase switch, but the privacy impact and incentives for this attack are low.

7.2 Privacy

For the sake of privacy discussion, we will use the notion of k-anonymity. Hereby k is the number of participants indistinguishable from the true originator, which optimally would be $k = |\text{participants}| - |\text{attackers}|$.

Independence of DC Network and Diffusion

Messages sent after completion of phase I, especially any messages of phase II and III, must not provide further insights into the group to keep the privacy guarantees of the DC network. The second phase is started by transmission of the message m and the elements $(v, s = 1, r = \mathcal{H}(m))$. The element $s = 1$ identify that a new diffusion run is started but are independent of any contents, i.e., they are fixed for all possible runs. As v identifies the sender, v must be part of the originating DC network. Within the network, v was chosen by the originator, indirectly through a random identifier. This random identifier is not transmitted further, so no clues about group participants can be inferred. Therefore, the DC network can keep its privacy guarantees intact.

Global Passive Attacker

Revelations over recent years have shown that service providers and intelligence agencies across the world are collecting and analyzing information, coming reasonably close to a global passive attacker. If such an observer could collect all slices $s_{i,j}$ of a participant, they could recompute the original message sent by the participant. This is prevented by authenticated encrypted channels between participants.

To prevent traffic correlation, DC networks require all participants to send the same amount of data, not just the sender. While a global passive attacker can detect the commu-

nicating DC groups and the broadcast message, they can not identify the originator within the group [12, 32]. Against this attacker, phase one provides $k = |\text{group}|$-anonymity.

Dining-Cryptographers Network Insider

To improve the detection of DC group participants, an attacker has to be part of the group. For β attackers within the group, DC-based communication trivially provides $(k - \beta)$-anonymity. The anonymity guarantees depend on the group formation mechanism [12]. Current strategies of a random selection of participants with an assumed attacker probability p require group sizes of $\frac{2k}{1-p}$ for k-anonymity with high probability [12]. These bounds depend on the attacker probability distribution. External trust information during the group formation could improve the probability distribution.

Honest but Curious Botnet-like Attacker

The previous sections covered attack vectors against the DC network, i.e., an attacker on the outside and attackers on the inside. Therefore, we will focus on the privacy provided within the diffusion phase, as an improvement over the guarantees by the DC network.

An honest but curious attack, where nodes follow the protocol honestly but try to violate the privacy of the protocol, could be performed by a single computer or a deployed botnet. The attackers connect to as many participants as possible and collect every message they observe. This behavior can be seen often, e.g., by research groups [76] or information crawlers.

The original analysis of Fanti et al. [46] for adaptive diffusion gives a worst-case privacy estimate, as our modifications only limit the exposure to possible attackers. Based on this, we provide the argument for improved privacy over the DC network.

If an attacker were to receive the virtual source token, they could easily compute a candidate pool of d^s candidates, with s the current depth of the diffusion phase and d the expected degree of the graph. For most sensible configurations, it holds for $s \geq 0$ that $d^s > \text{groupsize}$ improving the privacy of previous steps. Attackers with a penetration $p = \frac{|\text{attackers}|}{n}$ of the network, will receive approximately $p \cdot |\text{broadcasts}|$ of all first virtual source messages. This group increases further for participants of the spreading protocol, which never receive a virtual source message.

Consider two models: An attacker who creates a single connection to all nodes and distributed attackers with a network penetration of $p = \frac{|\text{attackers}|}{n}$ but normal behavior in respect to number of connections created. Let X be the random variable modeling the number of attackers chosen for a run of the spread sub-protocol. The probability $P(X \geq 1)$ of having at least one attacker included and the expected number of connections to attackers per node $\mathbb{E}(X)$ are $\frac{\eta}{d}$ for a single attacker. For the distributed attack $P(X \geq 1) = 1-(1-p)^\eta$ and $\mathbb{E}(X)\eta p$. At step t the expected amount of attackers hit are

$$\mathbb{E}_t(X) = \mathbb{E}(X) \sum_{i=1}^{t-1} \eta^i = \mathbb{E}(X)(\frac{\eta^t - 1}{\eta - 1} - 1).$$

Lastly, the forwarding probabilities are determined so that the deviation from perfect hiding is optimal, see Section 6.2.

7.3 Conclusion

In this chapter, we established the robustness and privacy of 3P3. First, we showed that 3P3 will succeed if the flood-and-prune phase of 3P3 is reached and that the phase is reached as long as the adaptive diffusion phase is reached. As the first phase of 3P3 is based on the protocol by von Ahn et al., we showed that it fulfills the same robustness criteria, i.e., it either makes progress or removes an attacker, and will then reach the adaptive diffusion stage. These results combined, show that 3P3 in total is robust in the same sense.

In the second part of the chapter, we analyzed the privacy properties regarding various attacker models. The DC construction provides k-anonymity against global passive attackers and anonymity in the number of non cooperating participants within the group. The adaptive diffusion stage extends this guarantee in an honest but curious setting, which is commonly seen through probing nodes or unobtrusive botnet attacks.

These results prove the ability of 3P3 to function in real-world networks and provide the expected privacy guarantees.

Chapter 8

Performance

This chapter presents results of previous reviews publications at IWCSS [4] (with permission, © 2018 IEEE) and ArXiv [9].

[4] D. Mödinger and F. J. Hauck. "3P3: Strong Flexible Privacy for Broadcasts". In: *4th International Workshop on Cyberspace Security (IWCSS 2020)*. 2020.

[9] D. Mödinger, A. Heß, and F. J. Hauck. "Arbitrary Length k-Anonymous DC Communication". In: (2021). arXiv: 2103.17091 [cs.NI].

This chapter is dedicated to the performance analysis of 3P3. The performance analysis is split into three parts. First, we take a look at the expected bandwidth consumption from a theoretic perspective by calculating the expected size of messages. Secondly, we evaluate the proof-of-concept implementation with a focus on phase I, due to the size of networks involved. Lastly, we apply a simulation to evaluate large networks running 3P3.

8.1 Bandwidth Consumption

The most performance-critical part of our system is phase one. Due to the nature of DC networks, we expect the group phase to mostly scale with bandwidth with little influence by latency.

Given a commitment and randomness of 32 Bytes size, e.g., using Pedersen commitments [90], we can commit on 31 Bytes at a time. This results in $2 \times 2k(4 + 32k)$ bytes for the DC round and $2k \times 32$ bytes for commitments during the initial round. The final round requires $2 \times |m| \times (k-1)$ bytes for the DC round and $2 \times 32(k-1) \times \lceil |m| \frac{32}{31} \rceil$ bytes of commitments.

Assuming a group size of 15, as used on the blockchain level in Monero, and a message length of 1024 Bytes, the magnitude of common Bitcoin transactions, we determined the bandwidth consumption of phase one. A participant will send $\approx$ 1.34MiB of data and receive $\approx$ 18.83MiB for a full round. To only compute the initial round of phase one, which is independent of any message size, a participant sends $\approx$ 434KiB and receives $\approx$ 6.09MiB. While these numbers can be handled quite easily by modern network devices, as they are on the scale of modern websites, it is easy to see that it does not scale well beyond sizes of around 30 to 40 privacy-group participants. These sizes are sensible privacy settings, comparable to settings used in other strong systems [122].

8.2 Proof-of-Concept Implementation

Methodology

To evaluate the performance of our protocol we build a prototype implementation. The implementation was deployed via docker containers on a single machine with 32 physical cores and 2 threads per core. We repeated experiments 100 times to improve confidence in the results. All software and experiments are available as open-source software, including an explanation for reproduction, on GitHub[1]. We used the optimized secure variation with prepared commitments and no validation during successful rounds for the evaluation of the secure version.

We introduced an artificial network latency of 100 ms and used traffic control to limit bandwidth per interface to 50Mbit/s to achieve more realistic results. The values are estimates based on common usage [54]. We used a fixed message size of 512 B as an approximation of Bitcoin transaction sizes. We performed an experiment with multiple message senders, which is equivalent to longer messages.

A central container was used to manage the connections, setup and logging of all other deployments. To run more nodes on the single machine, we used an approximation of cryptographic operations. We performed a real-time benchmark of the relevant cryptographic operations on the machine, resulting in 0.0109 ms for point addition and 0.6698 ms for scalar multiplication. Overall this results in a 1.35 ms commitment over the elliptic curve secp256k1, greatly simplifying the setup.

Number of Participants

For the first comparison between the unsecured and secured protocol versions, we looked at the scaling behavior based on the number of participating nodes. For evaluation purposes, the nodes use their node id as the slot index to prevent collisions during the first round. We scale the system from 8 to 24 participants in increments of 2 for both the secured and unsecured version. We fixed the number of threads for the secured version to 4 and the number of senders to 4, which results in a message size of 2KB. This experiment is codified in `nodes.sh` in our code repository.

The results of the experiment for scaling by the number of participants are visualized in Figure 8.1. Please note that the visualization uses a logarithmically scaled y-axis. While the secured version quickly increases in the time taken per instance, the unsecured version performs at nearly constant speeds. Further, the unsecured version takes around 0.5s per protocol instance.

For large numbers of participants, we would expect the unsecured version to scale similarly to the secured version. Though, for the numbers at hand, the bandwidth and computation complexity are not dominant over the transmission latency. For a similar setup as in the previous discussion, i.e., at most a 5% utilization, we can sustain four parallel messages every 10s or $\frac{2\,\mathrm{KiB}}{10\,\mathrm{s}} = 204.8\,\mathrm{B\,s^{-1}}$. With full utilization we reach $\frac{2\,\mathrm{KiB}}{0.5\,\mathrm{s}} = 4\,\mathrm{KiB\,s^{-1}}$. This speed is sufficient for most text-based applications.

Number of Senders and Message Size

For the second comparison, we looked at the scaling based on the number of senders within a protocol run. This is equivalent to the message size, as we fixed the slot number per node

[1] `https://github.com/vs-uulm/3p3-evaluation`

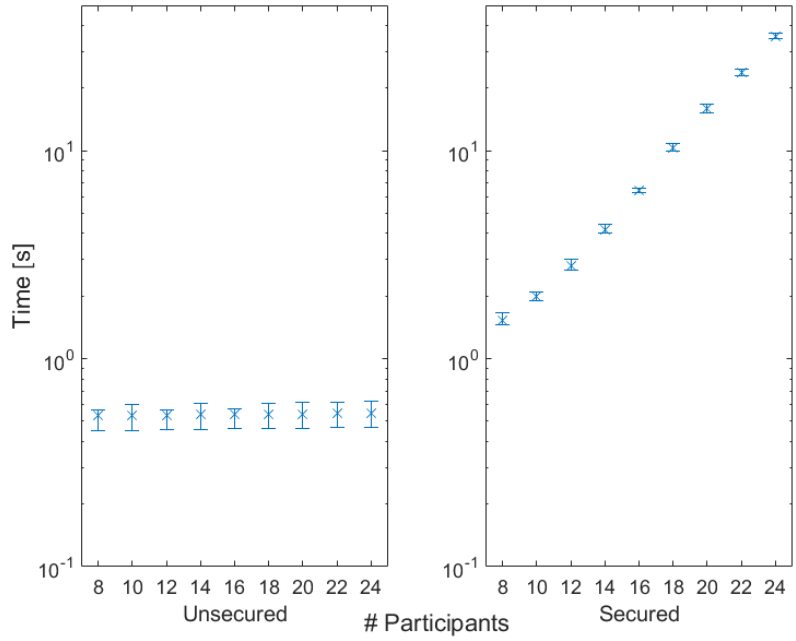


Figure 8.1: Comparison of minimum, median and maximum runtime for various numbers of participants. Note that the vertical axis is in log-scale. [9]

to prevent collisions. We did not perform this experiment without fixed slot numbers, as the expected number of collisions and repetitions can be calculated directly.

We fixed the number of participating nodes to 20 with 4 threads for the secured variant of the protocol. The number of senders s was changed from 1 to 20 in increments of 1, resulting in message sizes of $s \times 512\,\mathrm{B}$. This experiment can be executed through `messages.sh`.

The visualization of the results can be found in Figure 8.2. Again, we can see the performance of the unsecured variant barely budge under the generated load, while the secured variant scales as expected. In the optimized unsecured variant, the message size does not increase the runtime of the system noticeably.

We can combine this with our result from the previous experiment to improve our previous bound to $\frac{10\,\mathrm{KiB}}{0.5\,\mathrm{s}} = 20\,\mathrm{KiB\,s^{-1}}$. As the experimental results did not consider larger messages, this provides no upper bound on the transmission efficiency. For n participants, the available bandwidth b of a node will provide an upper limit of $\frac{1}{2n}b$ to the transmission efficiency of the system, as the message has to be transmitted $2n$ times for a successful transmission.

Comparison

An interesting performance comparison for the introduced protocol can be found in Dissent. While Dissent [36] fulfills a different purpose than our protocol, it provides similar levels of privacy. Performance-wise, the core mechanism of dissent requires multiple seconds, up to minutes, for a message round. The updated version [122] shows transmission times from 0.5 to 10 seconds for their message exchange process. These results use realistic latency distributions through a Planetlab setup, which provides similar latency to our estimated 100ms [53].

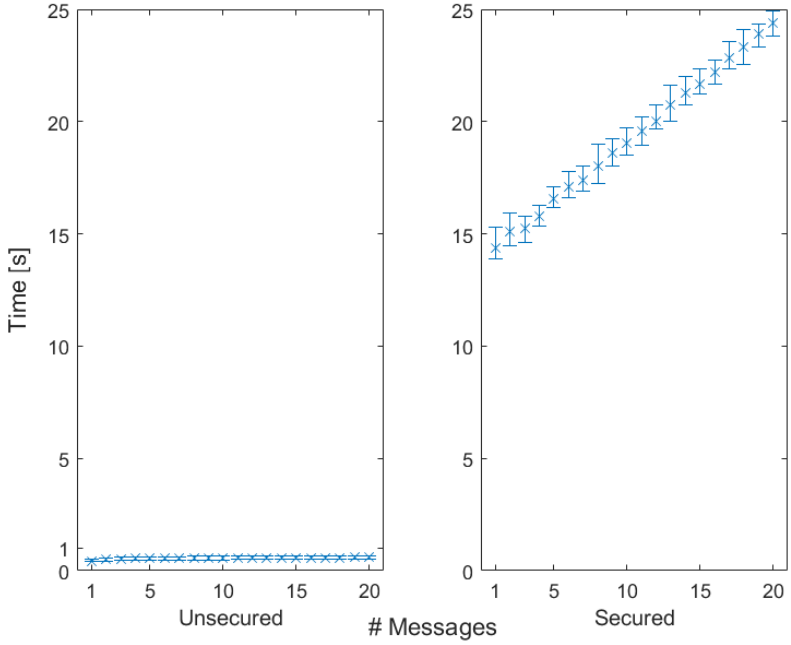


Figure 8.2: Comparison of minimum, median and maximum runtime for various numbers of messages. Overall message length is equivalent to 512 times the number of messages. [9]

The proposal by Wang et al. [115] provides lower latency, 50-100ms, in a 3-hop setup with only 4 to 6 group participants. Assuming a latency comparable to our setup, the communication latency should increase to at least 330ms, based on the results presented. Due to the protocol structure, a much higher latency is to be expected.

Further, we compare the performance of our protocol to the protocol proposed by von Ahn et al. [12]. Since there is no source code available, accompanying their publication, we implemented our own version that only marginally deviates from their specification. Instead of utilizing Pedersen Commitments over large prime fields, we utilize Elliptic Curve Pedersen Commitments similar to our own protocol. We executed the experiments with a varying number of participants and three different message sizes. Figure 8.3 visualizes the performance of the non-optimized version of our protocol with k and $\frac{k}{2}$ senders and the performance of our implementation of the von Ahn et al. protocol. While our protocol is slower for 512B messages, it approaches the performance of the von Ahn et al. protocol with 1024B messages, and noticeably surpasses its performance with 2048B messages. Additionally, it can be observed that while our protocol scales with the number of senders, the protocol proposed by von Ahn et al. can create significant overhead if the number of parallel senders is low, because its runtime is only affected by the number of participants and the message size used.

Lastly, the strategy employed by Bitcoin (see Chapter 10) leads to an average dissemination latency of five seconds to reach 50% of the network. 3P3 can improve this time by 80% while reaching the full network.

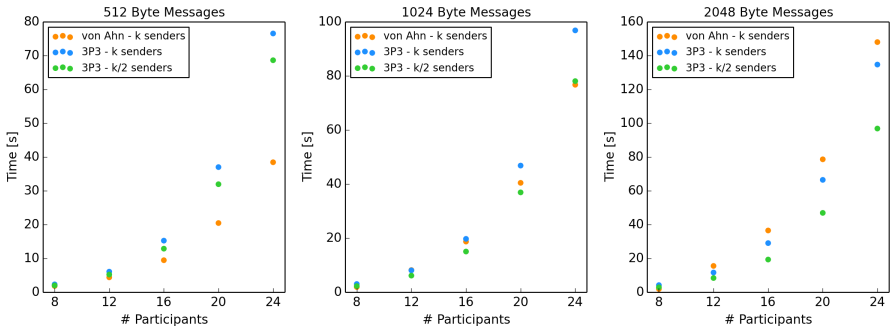


Figure 8.3: Comparison of our protocol and the von Ahn et al. protocol with different message sizes. Median runtime of several protocol runs. [9]

8.3 Simulation

Simulation Methodology

To evaluate our algorithm further, we built a discrete event simulation[2] for random networks. We implemented a skeleton of 3P3, where all message types and flows exist, but processing takes only simulated time, e.g., commitments take 0.5ms[3]. We implemented adaptive diffusion, flood-and-prune and a simplified Dandelion broadcast for comparison. Communication delay between nodes is based on a normal distribution $\mathcal{N}(\mu = 80, \sigma = 15)$ clamped between 20ms and 200ms. The values are estimates from well-connected clients of Hoiland et al. [54].

To generate a network, nodes create connections sequentially, until they have c connections. This construction results in c connections for most participants, but more for early participants.

We varied the parameters of the simulation by the number of participating nodes (100 to 10000, using steps of 1, 2.5 and 5 times 10^x), created connections between nodes (8 to 20, in increments of 2), the depth of the diffusion phase (up to 8) and the spread of the diffusion (up to 8). We repeat all parameter combinations for 50 runs. A simulation run creates a random network and initiates a single protocol run until no messages are left.

Network Size

An overview of the results for scaling based on network size is shown in Figure 8.4, using the median, 99th and 99.9th percentile. This representation allows for analysis of the expected and worst-case performance of the protocols. Please note the logarithmic spacing of the x-axis.

The flood-and-prune entry provides a baseline for evaluation. The results show that the anonymity phases mostly dominate 3P3. Overall the performance of our system is reasonable, while noticeably slower than the baseline. We did not include results for the 99th percentile of Dandelion, which has, due to its random nature, fairly large outliers.

[2] Available here `https://github.com/vs-uulm/netsim2`.

[3] Average computation time of a commitment on our simulation hardware.

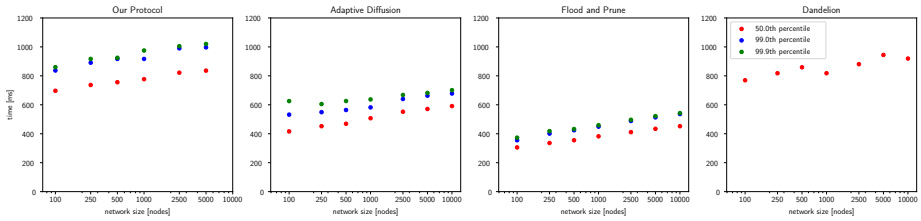


Figure 8.4: The median, 99th percentile and 99.9th percentile of full protocol instance latency, i.e. until no more messages related to that protocol instance remain. Dandelion only shows the median, as the 99th percentile is far off the top of the chart due to its random phase switch. Network sizes are scaled logarithmically. [4]

Number of Connections and Diffusion Depth

We chose the number of connections between nodes to stay above the $\log(n)$ connectedness boundary [58, Ch. 4]. The amount of connections is relevant to keep the preconditions intact as a fully connected network, even in the presence of malicious nodes. Minimum connections per node had a minuscule impact on the overall performance.

Similar to the number of connections of nodes, the diffusion depth had little impact on overall results. The trend in runtime for longer diffusion runs was noticeably upwards, but negligible compared to other factors.

Diffusion Spread

We expected the spread limitation of adaptive diffusion η to have some impact on the number of nodes reached over time, but not on total runtime. The visual analysis of Figure 8.5 indicates a difference in the base distribution for reaching 50% and 75% of the network. The results for 99% are inconclusive, which is expected. The means m_η and confidence intervals for $\eta = 2$ and $\eta = 8$, which are $m_2 = 640.23\text{ms} \pm 0.34\text{ms}$ and $m_8 = 615.60\text{ms} \pm 0.27\text{ms}$., support this interpretation, as they are strictly non-overlapping. This holds for all intermediate values of η as well. Overall the cost is small compared to the privacy gain of reducing η.

8.4 Conclusion

In this chapter, we evaluated the performance of 3P3 and its parts using a simulation and proof-of-concept implementation of 3P3. We calculated the expected bandwidth consumption and validated the results with experiments using limited bandwidth links. All experiments show practical applicability of 3P3 in real-world scenarios with real-world bandwidth limitations.

The optimized group stage of 3P3 takes only $0.5 \pm 0.1s$ in high latency environments, i.e., 100ms per link. The results are consistent with our simulation results, demonstrating that 3P3 performs well in larger networks with better performance than Dandelion and only half to one-third of the speed of a flood-and-prune broadcast. The performance is sufficient for many text based protocols and especially for small payloads, which are common in blockchain systems.

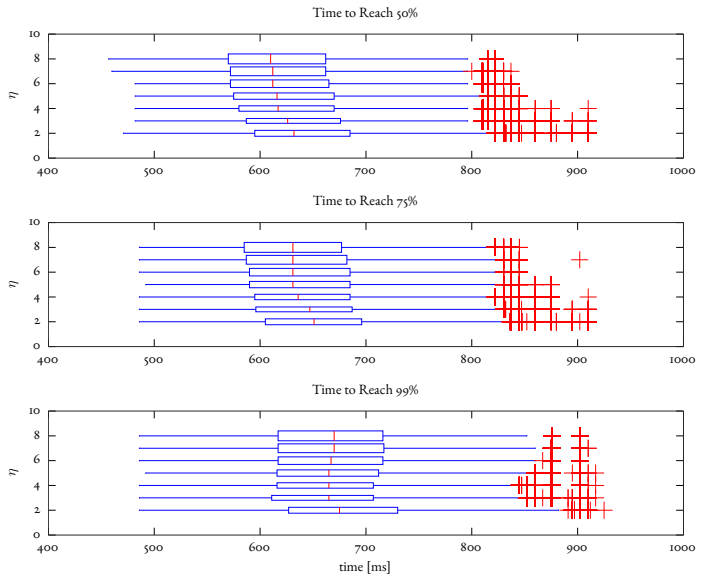


Figure 8.5: Behavior of the protocol based on η. [4]

Part III

Privacy Extensions

Chapter 9

Overview

In the previous part of this thesis we introduced 3P3, a privacy-preserving protocol for broadcasts. 3P3 fulfills the important role of providing strong and flexible privacy during message dissemination. This role is not the only privacy sensitive operation surrounding broadcasts in peer-to-peer networks. Measurements, enhancements for dining-cryptographers groups, group creation and network organization have significant privacy impact as well.

In this part of the thesis, we discuss additional privacy enhancements around 3P3 and broadcasts. This chapter provides a quick overview over the topics introduced in the following chapters: unobtrusive monitoring of Bitcoin, the fusion of threshold cryptography and dining-cryptographers networks, as well as Pixy, a privacy increasing group creation scheme.

9.1 Unobtrusive Latency Monitoring

Applications and network health and optimizations often depend on network metrics. For Bitcoin, the most widely known blockchain system, a few large scale operations are known to collect information on the network: bitnodes and two research groups [81, 85]. These metrics are collected using thousands of connections, i.e., one connection to any reachable node in the network. While these external resources are sufficient to compare the performance of 3P3 against real-world data, they are not geographically diverse. Further, nodes interested in network metrics have a hard time replicating this approach and are easily detectable by other nodes. This detection might lead to reduced cooperation from other nodes or malicious entities attempting to feign or modify collected metrics.

In Chapter 10 we discuss a scheme to approximate broadcast dissemination latency metrics with only few connections. This allows for unobtrusive monitoring, i.e., undetectable by other participants. We achieve this by modeling the latency of the Bitcoin network using a log-normal distribution. The distribution is initially determined by observing the network over geographically diverse locations and multiple points in time, to ensure stability of the results over time and space. The resulting dataset is made available as validation and comparison data for protocols such as 3P3. Later, we validate the results of our novel scheme over small sub-samples of the collected dataset.

Lastly, we use an approach borrowed from Kalman filters, sampling and error models to reduce noise in the few measuring signals. The chapter provides an evaluation of the results, showing real-world applicability over the collected datasets.

9.2 Dining and Threshold Cryptographers

Various protocols, including 3P3, use dining-cryptographers groups [32] for privacy. DC groups provide strong privacy, but the lack of efficiency makes a layering approach lucrative for most applications. Non-cooperating participants can lead the layers built on top of the dining-cryptographers group to fail. To incentivize participants in these layers, instead of only detecting their misbehavior. We propose a novel combination of dining-cryptographers groups and threshold cryptography. The novel protocol can be used as a alternative to phase I of 3P3.

Chapter 11 details the system combining dining-cryptographers groups with Shamir's secret sharing. In this scheme, each group participant only recovers a share of a message. Only through cooperation can participants recover the actual message. This cooperation can be achieved by broadcasting the shares. Only when the threshold set by the scheme is crossed, can all participants decrypt the message.

This scheme allows for enforced k-anonymity using the threshold cryptography properties while maintaining privacy through the dining-cryptographers construction.

9.3 Privacy-Increasing Group Creation

There are various protocols applying dining-cryptographers groups, i.e., groups of restricted size compared to the full network. Examples for such networks are the k-anonymous message transmission protocol [12] or the previously introduced 3P3. The protocols rely on the integrity of the underlying groups.

The protocol by von Ahn et al. assumes random selection of participants and constructs an expected group size of $\frac{2k}{1-\beta}$ for an anonymity level of k and a fraction of attackers β. Groups created using this scheme reach significant sizes quickly, degrading performance of the protocol.

To improve on this bound, by raising the trust in participants, we propose Pixy [3]. Chapter 12 details the construction of Pixy, a novel privacy-increasing group creation scheme. Pixy classifies and applies suitable detection mechanisms in a two stage protocol to detect impostors and less trustworthy nodes. This allows for smaller, and therefore more efficient, groups while maintaining privacy guarantees of the dissemination protocols.

Chapter 10

Unobtrusive Monitoring

This chapter is based on a previous publication at Plos One [7] and data published at Zenodo [8].

[8] D. Mödinger and F. J. Hauck. *Bitcoin Network Transaction Inv Data with Java Timestamp and Originator Id*. `https://doi.org/10.5281/zenodo.2547396`. Jan. 2019.

[7] D. Mödinger, J.-H. Lorenz, R. W. van der Heijden, and F. J. Hauck. "Unobtrusive monitoring: Statistical dissemination latency estimation in Bitcoin's peer-to-peer network". In: *PLOS ONE* 15.12 (Dec. 2020), pp. 1–21.

In previous chapters we introduced Bitcoin [84] and its underlying peer-to-peer network. Bitcoin, and other blockchain systems, provide an abstraction layer for applications to build upon. Many of these applications, such as an automated teller machine (ATM) [121], file storage [1] or, in general, marketplaces [105], rely on low latencies. They provide user feedback for expected duration of an operation or reduce their operational risk, e.g., by estimating how long it would take to notice a double-spend transaction. The dissemination data of a real world blockchain system has further uses for advanced privacy protocols as well. The collected data can be used to model behavior of network participants for evaluation of network protocols.

Live monitoring data on the dissemination times in Bitcoin is available by third parties in large scale measurements, e.g., by Bitnodes[1] and research [81, 85]. While an interested user could use the data produced by these third parties, this would introduce a, possibly unwanted, dependency on these parties. Users would also need to trust these parties and their provided data to be correct, reliable and up to date. Participants could also apply the measurement techniques directly, but these require a large amount of resources, as connections to the majority of network participants are required. This approach is thus infeasible for typical network participants. Furthermore, this approach is rather conspicuous and does not scale to a large number of users.

In this chapter, we enable live measurements of transaction-dissemination latencies in Bitcoin in an unobtrusive fashion. This approach is accessible to typical network participants, using only eight connections, which is the minimum number of connections in Bitcoin. This chapter reaches this goal by:

[1] `https://github.com/ayeowch/Bitnodes`

- Providing a dataset of Bitcoin network transmission data, collected over various places in time spread over the world.
- Providing a dissemination-latency model using a lognormal distribution and discussing alternative models.
- Providing an approach to adapt the parameters of such a lognormal distribution to new observations, e.g. changes in the network, with an unknown shift parameter.
- Providing a tool estimating the parameters of a lognormal distribution modeling latencies if transaction-dissemination with only eight connections, verifying our previous contributions.

While the implementation relies on behavior specific to Bitcoin, the general approach is not as limited. The isolation and estimation of dissemination latencies can be applied to various broadcast networks and mechanisms, e.g., peer-to-peer queries.

The structure of this chapter is as follows: Section 10.1 discusses existing network-latency measurement strategies, while Section 10.2 discusses the relevant background of this chapter and the general circumstances of measuring within Bitcoin. In Section 10.3 we provide an overview of our network monitoring solution, which uses few connections. Sections 10.4 to 10.7 focus on aspects of the network monitoring and its evaluation. Section 10.4 gives details on the data collection and the resulting data sets. Section 10.5 focuses on the interpretation and modeling of the collected data. In Section 10.6 we describe the process to deduce similar results with much fewer connections by a Bayesian mechanism. Lastly, in Section 10.7 we show the experimental evaluation results of the Bayesian mechanism based on the collected data. An visualization of the aspects of this chapter is given by Figure 10.1.

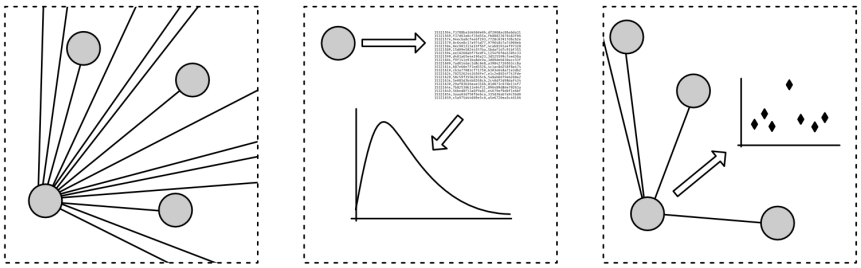


Figure 10.1: Visualization of the aspects of this chapter: Data collection, modeling and monitoring. [7]

10.1 Related Work

Monitoring network properties, such as latency, is widely applied in a multitude of different network types. In this section, we discuss different approaches to latency measurements in similar network environments.

Internet-Protocol Latency Measurements

Measurements in general internet-protocol (IP) networks are a common denominator of network measurements. Although, most IP-level measurements target single-path latencies instead of dissemination latencies.

Yu et al. [124] for example, use an active measurement approach in Software-Defined Networking. They instruct the network devices to route specific control packets through the monitored path. Then they send timestamped packets through the established route. The measured time difference is used to estimate the latency of the targeted path. The estimation is required as there is noise introduced by fluctuations in network behavior and latency introduced through traveling from and to the in and out routers of the target path. To compute the latency estimates from their measured timestamps they use an estimation distribution, an approach applied on a larger scale in Section 10.5. The active approach they use still produces strain on a large network and does not scale to the distributed approach required for measuring a broadcast. Breitbart et al. [28] use a similar approach.

Others [98, 117] use passive monitoring. This requires sharing of collected information to compute a global view on the information, which poses some challenges in a distributed system. While this creates reliable data, as long as participants are honest, it also creates additional traffic for information sharing. Shaer et al. [98] use a passive approach to measure latencies and other network properties in IP multicast environments. They build on the separation of data collection and processing for high-speed analytics. The approaches also require control, or at least dependence and trust, over many network participants to produce reliable results. We attempt to minimize these, due to the negative effects of dependencies and the trust model of Bitcoin. To reduce network overhead we infer the desired information from regular traffic instead of active sharing of measurements.

Peer-to-Peer Latencies

Bitcoin is built on top of a peer-to-peer network, therefore measuring techniques used in peer-to-peer networks, in general, might apply. Classical peer-to-peer networks are built on the idea of sharing and, more importantly, finding information. They are not built to spread information to all participants, as transactions in Bitcoin. Instead, they are built to locate information in a distributed fashion. Locating of information is accomplished through search queries, implemented by flooding techniques or random walks. Therefore, most measurements of classical peer-to-peer networks focus on hop count and search depth of flooding queries instead of dissemination latency [68, 109]. Others, such as Saroiu et al. [103], focused on peer properties. They only measure pairwise roundtrip latencies between a measurement station and each peer, which are not indicative of in-network latencies over multiple hops and many paths.

Butnaru et al. [30] and Almeida et al. [16] proposed testing and benchmarking frameworks for peer-to-peer networks. These frameworks actively create queries to the network and measure response times. As some of the classical networks, e. g. Gnutella, implement search queries by flooding the network, measurements of the response time for queries is collected by these frameworks. Query response times correlate to dissemination times for rare lookups, but it is imprecise and not considered in these publications.

Active probes are not suitable for the use case of monitoring Bitcoin transaction latencies, as valid transactions can create high cost per probe. They also would require large amounts of connections, similar to the fully passive approaches used by third parties in Bitcoin.

Bitcoin Monitoring

The Bitcoin network has been monitored for different goals. Various research and private projects [21, 22, 67, 81, 85] measure and discuss network properties of Bitcoin. While they actively build connections to participants, they measure the desired traits in a fully passive way: the monitoring software crawls the network for possible clients using the gossip protocol of the network. Then it connects to all found addresses and collects various statistics provided through the network protocol, including user agent, protocol version numbers and more. The software keeps the connection open and logs all received messages with their respective timestamps. This approach allows them to perform accurate measurements throughout the network. As they track actual traffic instead of probe and control messages, the results are reliable representations of the actual behaviour of nodes.

We apply this approach in Section 10.4 to collect comparable datasets. We deviate from this for our live monitoring by an abstraction of the desired metric, i.e., latency, and severely reducing the required connections for reliable results.

10.2 Background

In this section we discuss the special properties of the Bitcoin network.

Bitcoin Network

The underlying network of Bitcoin and, in general, of permission-less blockchains is an unstructured peer-to-peer network. The reference implementation of a client requires a participant to create at least eight connections. Blocks and transactions are broadcast throughout the network [39] by forwarding them through all existing connections. Connected nodes in turn forward to their neighbors.

In principle, this is a flood-and-prune broadcast: New transactions and blocks are advertised through an inventory message. An inventory message contains identifiers for these new transactions and blocks. A client can then request the actual block or transaction, with a so-called `getdata` message.

To hide the topology of Bitcoin and hide the originator of a block or transaction, the reference implementation does not instantly propagate new information. The Bitcoin-core software creates exponentially distributed values, which are used as waiting times until the next inventory message is sent.[2] This results in a Poisson point process with an average rate dependent on the expected value of the exponential distribution.

To calculate the waiting times, with an expected average delay of a ms and a minimum of 0.5, Bitcoin uses the formula:

$$\ln(1 - \frac{\text{rand}(0, 2^{48})}{2^{48}})(-10^6)a + 0.5.$$

The recent version `0.20.0` uses a value resulting in an average of 5 seconds as default. To prevent unacceptable long waiting times, Bitcoin caps the generated values at 7 times the average, i.e., 35 seconds. According to the sources, the privacy consideration of outbound connections are different from inbound connections and outbound connections have therefore half the delay.

[2] Cf. net_processing.cpp:4140 and net.cpp:2852 of the Bitcoin sources on GitHub, on commit `ea595d39f7e782f53cd9403260c6c1d759d4a61a`.

There are alternative implementations of Bitcoin[3] which do not have to follow this implementation. Further, there are proposals for different privacy approaches[5, 47] which are not implemented yet and are therefore not considered during this paper. If they were to be implemented, the strategies of this paper would need adaption and reevaluation.

Time Measurements in Bitcoin

Given some source node s and some target node t and a measurement node m connected to both. The measurement node receives timestamps T_s, T_t by node s and t. The difference $T_t - T_s =: M_{s,t}$ is a measurement of some property of the network connecting these two nodes.

First, let us denote the latency of n successive connections as the term $\ell(n)$). Secondly, the random variable modeling the slowdown of the connection between nodes i, j shall be named $X_{i,j}$. $M_{s,t}$ is then the minimum time taken through the network from s to t through all possible paths between them. Considering the measuring connection slowdown and latency, the result is:

$$M_{s,t} = \min(\text{path}_{s,t}) + X_{t,m} - X_{s,m} + \epsilon(2, \ell).$$

Let path_n be a possible path of length n between nodes. Further, X and X_k denote the exponentially distributed random variables between two nodes within the Bitcoin network, without the addition of 0.5. X_k is either Exponentially distributed with $\text{Exp}(\lambda)$ or $\text{Exp}(\frac{\lambda}{2})$.

$$\begin{aligned}
\text{path}_1 &= \frac{1}{2} + X + \ell(1) \\
\text{path}_n &= \sum_{k=1}^{n} \left(\frac{1}{2} + X_k + \ell(1) \right) = \frac{n}{2} + \sum_{k=1}^{n} X_k + \ell(n) \\
&= \frac{n}{2} + \sum_{k=1}^{n-m} \underbrace{X_k}_{\sim \text{Exp}(\lambda)} + \sum_{k=1}^{m} \underbrace{X_k}_{\sim \text{Exp}(\frac{\lambda}{2})} + \ell(n) \\
&= \frac{n}{2} + \underbrace{\sum_{k=1}^{n-m} X_k}_{\sim \text{Erlang}(n-m, \lambda)} + \underbrace{\sum_{k=1}^{m} X_k}_{\sim \text{Erlang}(m, \frac{\lambda}{2})} + \ell(n)
\end{aligned}$$

So a measurement $M_{s,t}$ is a sum of two Erlang, or Gamma, distributions, with some noise linear in the number of participants. Further simplifications of the description of a single path will complicate the notation, as the sum of two Erlang distributions with different scale has no named or well-researched form. Therefore, there is no well-understood model of a minimum of such a sum either. Lastly, with an expected value of 5 seconds, the slowdowns and the $\frac{n}{2}$ term dominate all usual models for latencies of connections and therefore $\ell(n+2)$.

Given these circumstances, we are interested in the expected time required to reach a given fraction of the network.

[3] Cf. Bitnodes [123] list of user agents.

10.3 Unobtrusive Live Monitoring

We introduce a tool[4] to monitor expected transaction dissemination times in the Bitcoin network. The tool requires only eight connections to produce reliable estimates of the given network behavior.

Functionality

Our tool produces an estimate of parameters μ and σ for a lognormal distribution. This distribution represents the current dissemination latencies for transactions in the network and can be used to compute the time required to reach a desired fraction of the network. A discussion on the chosen lognormal distribution and other possible models can be found in Section 10.5.

Our tool uses initial values for μ and σ as a Bayesian prior and a default of 8 connections. However, these values can be configured by the user. Estimates are generated based on transactions: Data points are collected for each transaction. In principle, there will be one measurement per transaction and connection, showing when a transaction was broadcast by a certain connected neighbor. The monitoring will ignore any data points above the given connection number, due to the mechanism used to adapt the estimations (cf. Section 10.6).

Technically, the tool consumes text-based input from the standard input. Each line represents one measurement: A timestamp, a node identifier and a transaction identifier. Both identifiers are SHA-256 encoded hashes, but they are treated as arbitrary strings. An example input is given in Listing 10.1.

Listing 10.1: Example of a single input line for the monitoring tool.

```
1547779473468,
6cf1100aaccec75da23995512fc7c7a5b6e25224f5903af011e78691c03d0455,
a73578820a41aa6180621bcd90af1997c88794b33d8db2f004ee37c3e09b10ec
```

Interpretation Output

The estimates for lognormal parameters by the monitoring can be used to calculate interesting properties of the transaction dissemination. The cumulative probability of the distribution $\mathrm{CDF}(t)$, therefore, represents the fraction of the network that was likely reached by a given broadcast before a given time t.

Given $\mu = 8.5$ and $\sigma = 1$ as a result, the time to reach 75% of the network can be determined as the 75th percentile of the distribution. For the given values, this would be reached at $t \approx 9500$ms or 9.5 seconds.

Constraints

A client or library needs to be modified to produce logs in the required format for our tool. One such modification is used in Section 10.4 using bitcoinj. This library is freely available, and necessary modifications are provided in our code repository. Our modified version does not participate in further distributing received transactions to produce quicker and more accurate results.

As this behavior can be detected and may be suspicious to other participants, a modified client or library could select its measurement times and otherwise behave normally. We recommend relying on the behavior of the exponential distribution in Bitcoin dissemination,

[4] Available online at `https://github.com/vs-uulm/btcmon`

and measure transactions during long pauses created by high values drawn from the distribution. Measurements of transactions produced by the client itself can be used all the time, by sending it to only a single neighbor.

The tool is tuned on data collected from the Bitcoin network. If it is used on a different network, the assumptions and modeling, e.g., if the lognormal distribution is applicable, need to be revisited. These assumptions and modeling are described in the following sections.

10.4 Data Collection

In a preliminary step, we collected data on the amount and dissemination of transactions in the Bitcoin network [8]. This data is required to model the behavior of interest (cf. Section 10.5), i.e., the transaction dissemination latencies, and to evaluate the newly developed estimation tool (cf. Section 10.7). Note that recreation of this step is not necessary to run the resulting software, but only preparatory to create and validate models.

Related Work

Several projects have measured network effects of Bitcoin. We do not consider work that observed the Bitcoin network to deanonymize clients participating in the network [21, 22, 67]. While they are monitoring the network, their results have a different goal.

Bitnodes provides public live data about the Bitcoin network through an application programming interface (API) and web interface. Bitnodes uses the discovery mechanism of the Bitcoin peer-to-peer network to find new peers and connects to them. Information provided by Bitnodes includes version numbers of clients and protocols used, nodes distribution over countries, node counts and more. According to the website, the servers connect from a German data-center.

Coinscope [81] and Neudecker et al. [85] analyzed and measured the Bitcoin network to infer its topology. Coinscope is available as standalone modular software and provides large scale monitoring capacity. The software attempts to connect to any reachable node in the Bitcoin network, similar to Bitnodes. However, the topology inference techniques rely on outdated behavior of the Bitcoin core client.

The DSN research group of the Karlsruher Institute of Technology produces similar live monitoring information[5] as the one from Bitnodes. The group provides information including churn, versions of protocols and clients, node counts, propagation times and more. The information is provided as graphs and tab separated raw datasets. The nodes used to collect this information are located in Germany, and as noted by them, results may vary depending on location.

To validate the data collected by different groups and analyze it further, we collected and provide our own data set. These datasets were taken at worldwide locations and at different but similar points in time.

Collecting Methodology

To connect to the network, we modified the library bitcoinj[6] in version 0.14.7 to add logging capabilities, without modifying any core behavior. Our modifications on bitcoinj-core are:

[5] `https://dsn.tm.kit.edu/bitcoin/`

[6] `https://github.com/bitcoinj/bitcoinj/releases/tag/v0.14.7`

a. Generation of a runtime key as a hash of random bytes, to apply a keyed hash to the identities of network participants.

b. Creation of a logging file.

c. For new inventory messages of transactions, the library logs information as a comma-separated line.

The information logged is structured in the following way:

1. Current timestamp of the java virtual machine (JVM) in milliseconds,

2. a keyed hash of the sender identity,

3. the hash of the transaction.

The generation of the runtime key (a.) and keying of sender identities (2.) is added to provide privacy to the participants of the network so that the data can be published. The anonymization of network participants is done during data collection. No person had access to personal information.

With this modified library, we implemented an application that uses a connection limit of 5000 and does not broadcast received transactions, as to not influence the behavior intended to measure. This application was run on the local university servers, as well as on Microsoft Azure virtual computers in three regions: Eastern Unites States, South-East Asia and Southern Great Britain. The collection was run for about ten hours each and was rerun on the same virtual machines on multiple dates.

Reproduction

The modifications for bitcoinj and the application code are available on GitHub[7]. Further, we provide a precompiled version of the modified library for ease of reproduction. The collection should be reproducible as long as the network will accept the version of the protocol used by the library.

To collect data using these modifications, download the respective files and switch to the `collection/application` sub-directory. Start data collection within a docker container using the commands of Listing 10.2, which are also documented in the repository.

Listing 10.2: Example of a single input line for the monitoring tool.

```
docker run -it --rm -v $(PWD):/usr/src/btccol \
        -w /usr/src/btccol --rm openjdk:8 /bin/bash
javac -cp ./bitcoinj-core-0.14.7-bundled.jar ./research/*.java
java -cp ".:./bitcoinj-core-0.14.7-bundled.jar" research/Main
```

Collected data will be written to a file of the form: "crawler-dd.mm.yyyy hh.mm.ss.csv" where date and time shortages are replaced by the current date and time. The participation in the network in this form is not prohibited by any terms of use.

[7] `https://github.com/vs-uulm/CoinView`, collection sub-folder

Info

We collected nine datasets over multiple dates and locations[8] [8], each between 200–670 million individual points of data. Collections from Microsoft Azure provide much fewer data points, as the virtual machines could not create as many connections as the local university server. A description of all datasets can be found in Table 10.1. All collected datasets are available online[9] as compressed archives.

Server position	Nr.	Date	Data points	Note
Ulm, Germany	1	2019-01-17	570 million	
Azure US East	1	2019-01-24	207 million	
Azure South-East Asia	1		207 million	
Azure Great Britain South	1		205 million	
Ulm, Germany	2	2019-02-01	202 million	
Ulm, Germany	3	2019-02-06	669 million	
Azure US East	2		204 million	Same
Azure South-East Asia	2		203 million	identity
Azure Great Britain South	2		204 million	key

Table 10.1: All collected datasets of the Bitcoin network.

10.5 Modeling

To reduce the number of connections needed, we abstract from the collected data to a statistical model. The model represents the frequencies of measured dissemination latencies. This allows us to compute values of interest, such as expected time to reach 90% of the network.

Methodology

First, to determine the dissemination time for each transaction, we split the dataset by transaction. To simplify the process, assume that the first logged occurrence of a transaction was produced by the originator of the transaction. This assumption is reasonable on average, as the collected data is from a large fraction of the network: Either the originator or a node very close to the originator is present in the data. As a consequence of this assumption, the data is normalized for each transaction by subtracting the timestamp of the assumed originator, i.e., the first entry for the transaction in the log.

The resulting time series for each transaction was then analyzed for fitting distributions by visual analysis. SciPy [62] fits reasonable distributions and produces a visual representation of the data and created fits.

Other Tested Models

We explored several possible distributions before establishing the lognormal distribution as the most fitting model. The tested distributions include a power-law dependency, an expo-

[8] https://doi.org/10.5281/zenodo.2547396
[9] DOI 10.5281/zenodo.2547396

nential, gamma or generalized Pareto distribution. Those distributions are suitable due to their usage in network modeling and relationships to the Poisson distribution created by the privacy mechanism of Bitcoin.

A power law might be applicable for later percentiles of the datasets. Figure 10.2 shows one evaluation of a possible power law. A linear segment at the end can imply a power-law dependency. Some of the datasets show a stronger linear end, while most show less of a linear end, implying a power law is not a suitable description of the data.

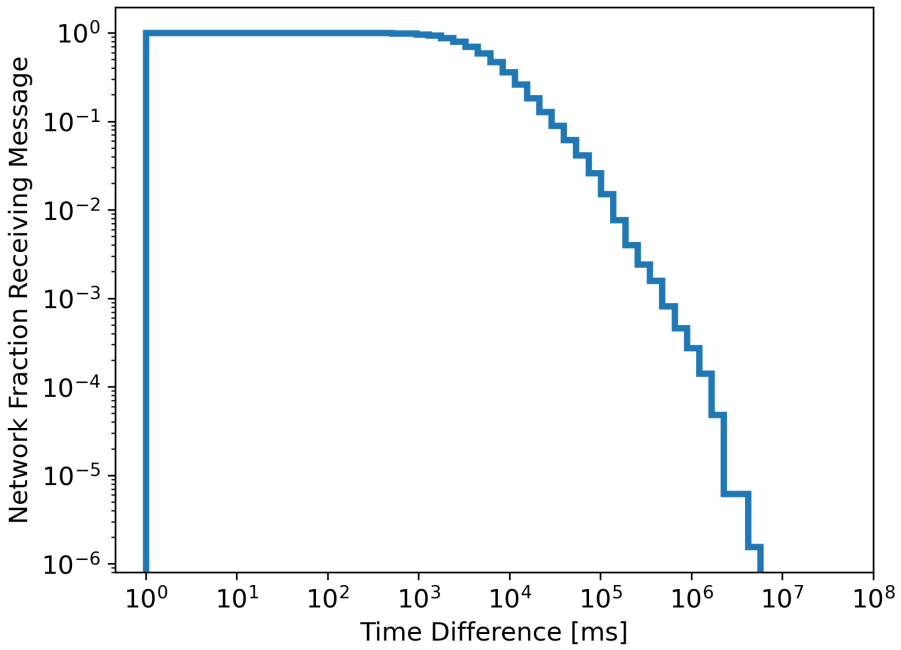


Figure 10.2: Density of the data of the February Southern Great Britain dataset in logarithmic scale, a power law dependency should show as a linear dependency in the later part of the visualization. [7]

The gamma and exponential distributions are suitable due to their relation to network modeling and the Bitcoin protocol. Both, and their similar related distributions, do not seem to be a good fit for most of the data. Figure 10.3 shows an example in the form of probability density functions.

Lastly, the generalized Pareto distribution was chosen due to their competition with the lognormal distribution. Similar to the description in [88], we found the generalized Pareto distribution to describe the extremities better than the lognormal distribution. In contrast, the lognormal distribution is a better fit to describe the main part of the data. If the interest lies more on the tail of the distribution, the generalized Pareto fit will give more accurate results. The generalized Pareto is also included in Figure 10.3, showing a similar fit as the lognormal distribution.

Lognormal Model

While all analyzed models produced some outliers, the lognormal distribution described a huge chunk of the data well. Figure 10.4 and Figure 10.5 provide normal probability plots of

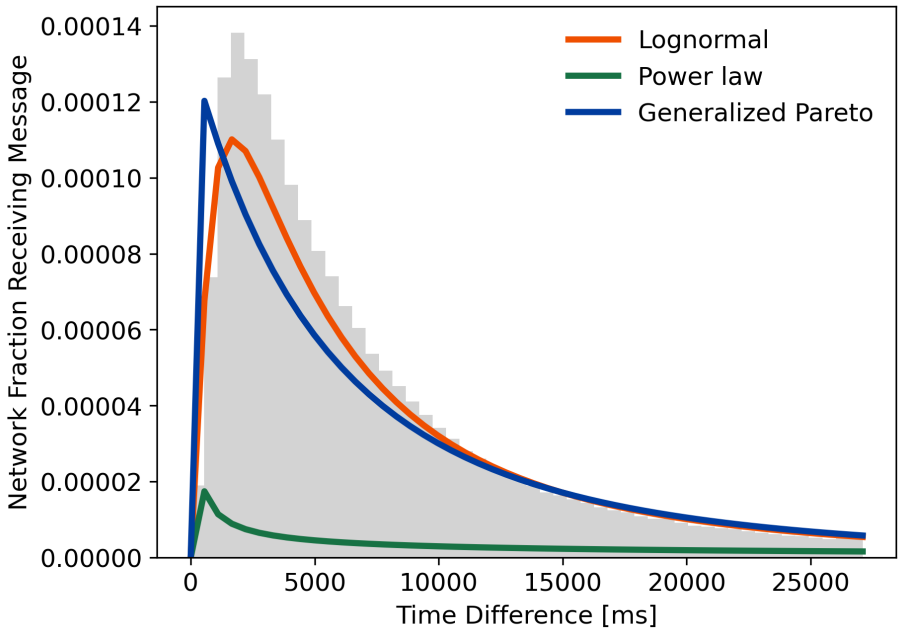


Figure 10.3: Evaluations of different distributions for the February Southern Great Britain dataset. Compares probability density functions and the histogram. [7]

the (base 10) logarithm of several datasets. A normal probability plot shows a linear dependency for normally distributed data, so it should show a linear dependency of the logarithm of the data, for a lognormal distribution. Both, and in general all generated normal probability plots, show a strong deviation of a linear trend in the first and last percentiles. The 95th to 99th percentiles show a medium deviation from the linear trend.

Figure 10.4 shows datasets collected at the same time. As suggested by the DSN Bitcoin monitoring [85], results vary by location, but the variation is small for most parts of the data.

Figure 10.5 shows datasets produced from the same place at different times. Together with the description in Section 10.2 this strengthens the belief that the model is stable over time.

One of the nine datasets, the Germany 1 dataset, shows more outliers. These outliers are better explained by a mixed Gaussian distribution over the logarithm of the data. We did not further explore this to reduce the risk of overfitting and due to the lognormal model providing good results of most transactions within this dataset.

10.6 Live Adaption of Parameter Estimates

The monitoring should be live, which means it should update its output over time to reflect the current state of the network. Further, the monitoring should not depend on thousands of connections, nor on the collection of huge amounts of data before processing. To solve these problems, we first evaluate schemes to estimate the parameters of a lognormal model containing an unknown shift. A simulation of all approaches, implemented based on the C++ standard library lognormal random distribution, helps to evaluate the quality of results. Lastly, to improve simulation results, a noise reduction and error compensation scheme is

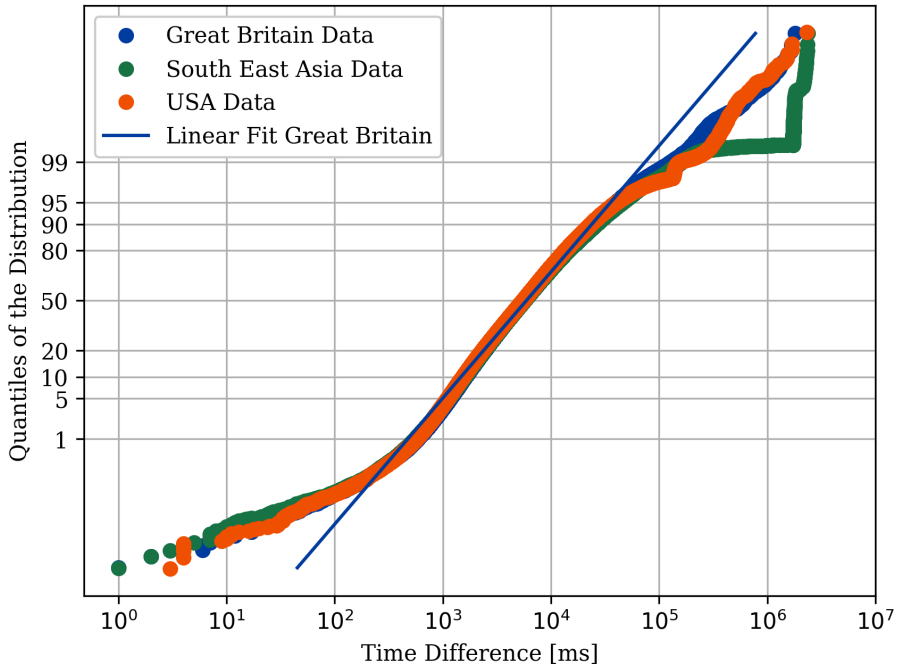


Figure 10.4: Lognormal distribution plot of data collected from different Microsoft Azure zones on the 6th of February. Data is reasonably lognormal distributed if it is linear, indicated by a linear fit for the data collected from Great Britain. [7]

applied.

Parameter Inference

As only a low number of connections is desired, we can no longer assume the originator of a transaction is captured in the data. The result is a three-parameter lognormal distribution, with an unknown parameter γ, which represents the true origination time of the transaction. The data points of a given transaction are considered a measurement of this unknown γ shifted distribution.

We apply an analytical method based on methods of Iwase and Kanefuji [57] as well as a sampling-based method to deduce the three parameters. Note that the variance and skewness of the measurement $m = \{m_1, \ldots, m_n\}$ are calculated as:

$$\operatorname{skew}(m) = \sum_{i=1}^{n} \frac{(m_i - \operatorname{mean}(m))^3}{n-1},$$
$$\operatorname{var}(m) = \sum_{i=1}^{n} \frac{(m_i - \operatorname{mean}(m))^2}{n}.$$

This also allows us to calculate σ directly.

The analytical method, using the moments skewness and variance result in the following

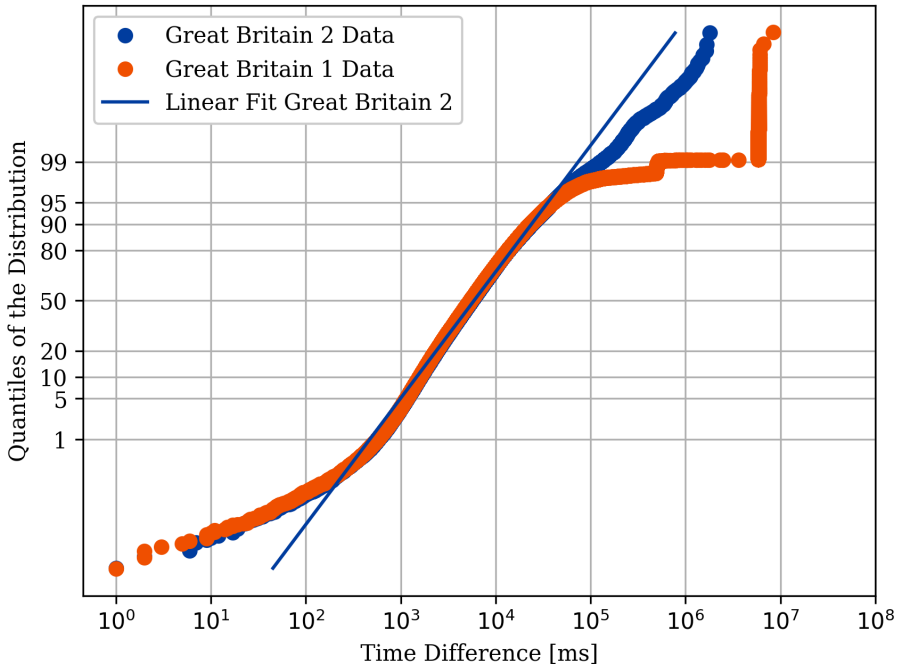


Figure 10.5: Lognormal distribution plot of data collected from the Southern Great Britain Microsoft Azure zone on different dates. Data is reasonably lognormal distributed if it is linear, indicated by the same fit as in Figure 10.4. [7]

(with $c = \text{skew}(m)$):

$$t = \left(\frac{2 + c^2 + \sqrt{4 \cdot c^2 + c^4}}{2} \right)^{\frac{1}{3}}$$

$$\sigma = \left| \sqrt{\log \frac{t+1}{t-1}} \right|$$

$$\mu = \frac{\log(\frac{\text{var}(m)}{\exp(\sigma^2)-1}) - \sigma^2}{2}$$

$$\gamma = \text{mean}(m) - \exp(\mu + (\frac{\sigma^2}{2}))$$

The sampling-based approach avoids directly calculating γ, as only the parameters μ and σ are of interest. Even further, given a previous estimate, only the difference of the estimated and unknown μ are of interest, which can be calculated using Algorithm 14.

We simulated both approaches using a hidden lognormal distribution to generate measurements for the transactions. This can be considered an ideal environment for the algorithm, as the samples obey the distribution without systematic outliers. During the simulation, we noticed both presented approaches produce an error dependent on the parameters of the hidden distribution, and there is a significant amount of noise, due to the low amount of samples used. After some warmup steps, we compared the absolute error as well as the variance of the error. The sampling-based approach produced a mean absolute error of ≈ 0.13

Algorithm 14 Algorithm to compute the difference of an estimated μ value and a measurement, without regard for a possible γ shift of the measurement.

Input: List of measured timestamps m, estimated distribution e, number of connections c, number of rounds r
Output: Difference of estimated and unknown μ

```
m ← {m_i − min(m) : i ∈ {1 ... c}}
means ← ∅
for 1 to r do
    s ←Draw c samples from e
    s ← {s_i − min(s) : i ∈ {1 ... c}}
    means ← means ∪{mean(s)}
end for
return mean(means) − mean(m)
```

with a variance of ≈ 0.02. The formula-based approach produced a mean absolute error of ≈ 17.65 with a variance of ≈ 0.11. We focused on the sampling-based approach, as the error and variance of the error is substantially lower.

Bayesian Approach

To compensate for the noise of a small number of samples, we apply a Bayesian approach inspired by Kalman filters [119]. Conceptually, the estimates are improved with each measurement, based on the difference between the measurement and the estimate.

Our a priori lognormal estimation is denoted by μ_e, σ_e. Let m be a measurement of a transaction with eight participants, then the differences of measured and estimated lognormal parameters are:

$$d_\mu = \text{Algorithm14}(m, \text{lnN}(\mu_e, \sigma_e), 10, 100),$$
$$d_\sigma = \sqrt{\text{var}(m)} - \sigma_e.$$

Given this difference, the estimates are updated based on a fraction of the difference, as there are huge errors in measurements, due to the low number of connections. The update process is rather simple:

$$\mu_{e+1} = \mu_e + \frac{d_\mu}{c_1},$$
$$\sigma_{e+1} = \sigma_e + \frac{d_\sigma}{c_2}.$$

For first evaluations $c_1 = 20$ and $c_2 = 2000$ were chosen. Further analysis of the error could improve the speed of convergence, but sufficiently converged values for μ were reached after the expected 25–30 steps.

Using this adaption mechanism, we detected an additional error between estimations and measurements, even in an ideal-world simulation, using lognormal distributions instead of datasets approximating lognormal distributions.

Error Compensation

To zero in on the error, we created further simulation experiments. The experiment performed the Bayesian adaption using the sampling-based difference algorithm in Algorithm 14.

We restricted the experiment to the modification of μ, i.e., the estimated σ was fixed to the σ of the hidden distribution. This setup allows us to evaluate the convergence of μ via the adaption, as more and more measurements are captured.

The resulting error is shown in Figure 10.6 dependent on the μ and σ values of the hidden distribution. The dependence on μ is negligible, while the dependence on σ is super linear.

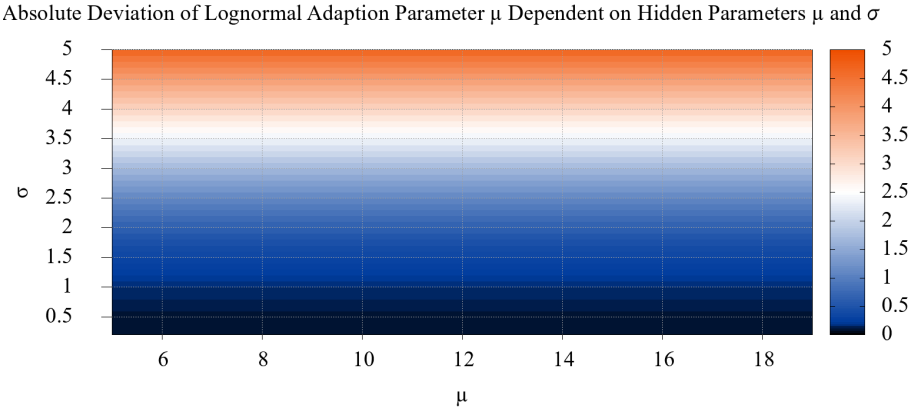


Figure 10.6: Heat map of the distance measured between a hidden distribution and our adapted estimates in a simulation. The x axis shows the dependence of the distance to the μ parameter, while the y axis shows the dependence on the σ parameter. The dependence on μ is negligible compared to σ. [7]

Restricting the analysis to one dimension, i.e. σ, a simple quadratic fit to the data provides further insights. The results are shown in Figure 10.7. While this simple error correction mechanism produced good results, we recommend a differently fitted error correction, should the system be applied to networks with huge deviations, outside the highlighted area in the figure.

The fit produced the following error correction function:

$$error(\sigma) = -0.207898 \cdot \sigma^2 + 0.083586 \cdot \sigma - 0.032573.$$

As a result, we added the error correction to the return statement of Algorithm 14.

10.7 Evaluation

This section focuses on the evaluation of the full scheme of monitoring Bitcoin.

Methodology

We use the datasets collected in Section 10.4 to evaluate the estimations provided by our tool. To reduce evaluation load and remove connection warmup artefacts, we focused on the last 1 million lines of each log for most log. We used our largest logs for a long term evaluation by only removing the first million lines.

We split each log prepared in this way by participant, i.e., each part contains all log entries received from one network participant. We create one thousand new logs by selecting eight

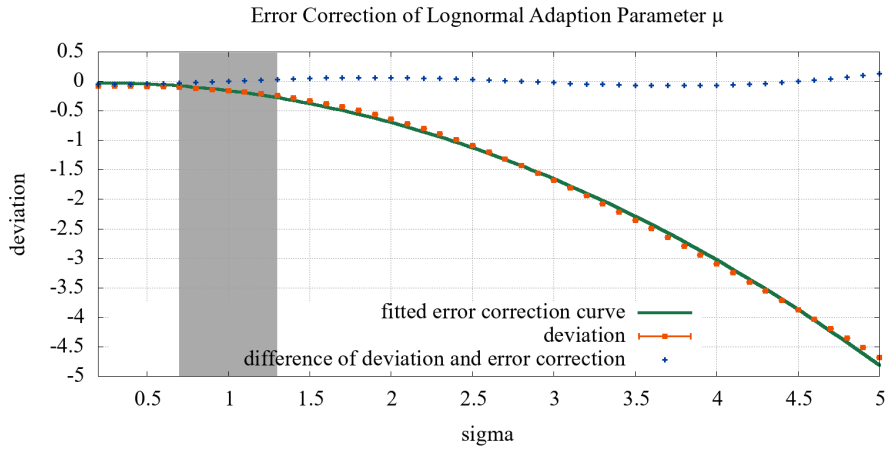


Figure 10.7: Simulation results to show the deviation between a hidden distribution and our adapted estimates dependent on σ, similar to Figure 10.6, as well as a quadratic fit to correct for the deviation and the difference between the fit and data, i.e., the corrected deviation. [7]

participants at random and merging the logs in chronological order. Each new log represents a log of a virtual node, having connections to only the selected participants.

We ran the monitoring tool on each new log, collecting all estimates over time. As the estimation tool uses initial parameters which require some time to converge, we prepared alternate versions of the results, where the first 30 steps, the warmup phase, has been removed. This was important to have a realistic estimate on the deviation of the results, as some logs might start much later into the original monitoring time, and create a bigger spread by logging values close to the initial values.

The results of all runs were then aggregated into a single result log for each original dataset. The aggregation creates bins, by time, to collect data. We then calculate the average and standard deviation of all collected data points in each bin.

Per dataset, we calculated the ground truth by splitting the logs by transaction, similar to the method in Section 10.5. Each transaction was then normalized and used to fit a lognormal distribution using SciPy. The fitted parameters were stored with the timestamp of the last contributing log entry, to replicate the process of assigning a time from the estimation. For the ground truth, we did not apply any averaging or further aggregation.

All software used for the evaluation is available in our code repositories[10].

Results

Figure 10.8 shows the results of the evaluation of the Great Britain 2 dataset, including warmup steps for the estimation of μ. The adaption of μ shows a fairly large spread, which can be explained by the original data: The dissemination of transactions is inherently noisy, but the estimates capture the bulk of the data.

Using the estimated parameters $\mu \approx 8.5$ and $\sigma \approx 1.1$ to estimate network behavior leads to the following latency estimates: The time to reach 50% of all network participants is approximately $e^{8.5} \approx 5000$ms, while reaching 90% would take ≈ 20100ms.

[10] `https://github.com/vs-uulm/CoinView` and `https://github.com/vs-uulm/btcmon`

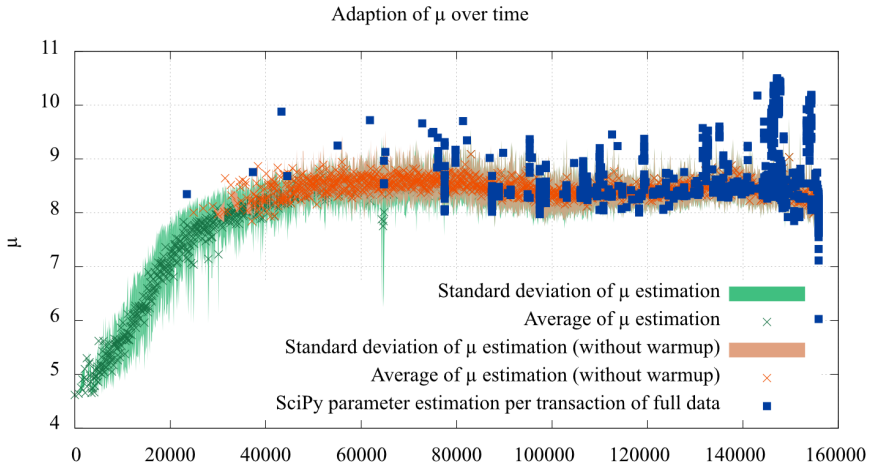


Figure 10.8: Evaluation of the μ parameter estimation using the Great Britain 2 dataset. The estimates are based on eight randomly selected connections, while the SciPy estimation had access to the full data. [7]

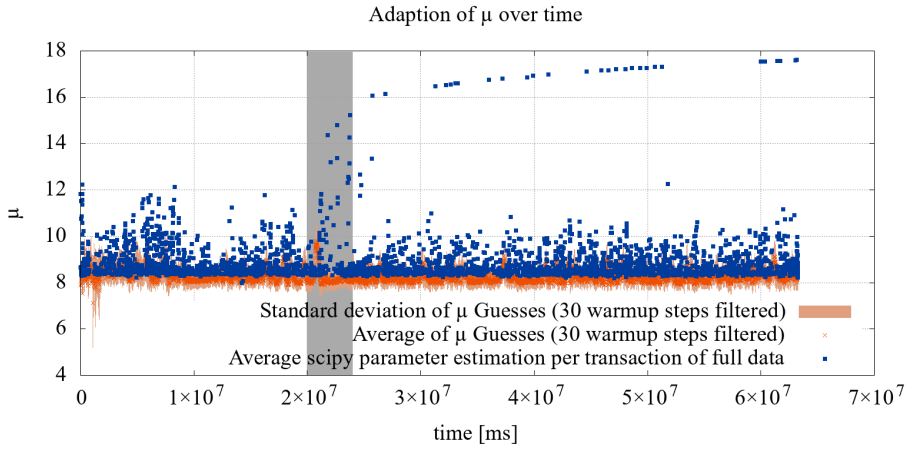


Figure 10.9: Evaluation of the μ parameter estimation using the Germany 3 dataset. The estimates are based on eight randomly selected connections, while the SciPy estimation had access to the full data. [7]

The long term evaluation using the Germany 3 dataset is shown in Figure 10.9 for μ and Figure 10.10 for σ. The estimation shows an underestimation of the real-world data for μ but can capture strong deviations as in the highlighted area. Sparse, long-lasting deviations cannot be detected, though. This is expected behavior, as the estimate attempts to capture overall network performance.

The estimation for σ overestimates the overall network behavior during this long term test. The reason for this seems to be the conflict of estimation of single transactions versus

the overall network behavior. The results are sufficiently accurate to use for computations, though. The highlighted area relates to the highlight in the μ adaption, creating a huge spread in the overall data, which can not be captured well by individual transactions.

Overall, the evaluation shows results sufficient for computation of dissemination times in the network. Unfortunately, the real-world data is severely noisy, but the μ estimates capture the bulk of the data well.

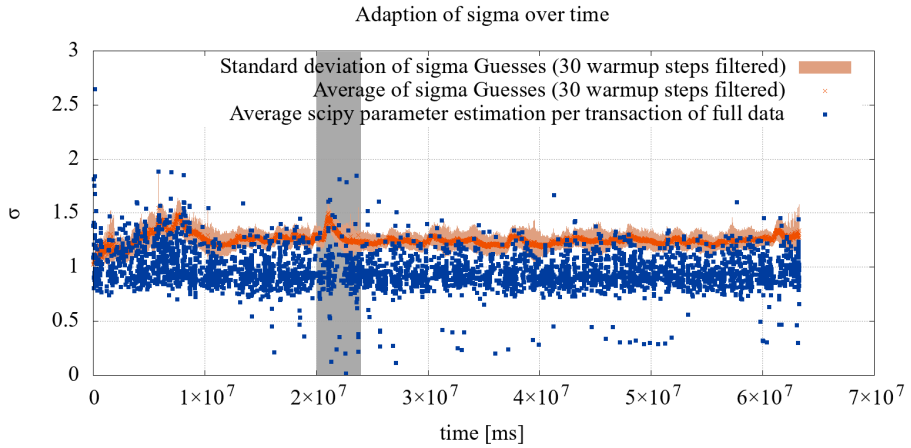


Figure 10.10: Evaluation of the σ parameter estimation using the Germany 3 dataset. The estimates are based on eight randomly selected connections, while the SciPy estimation had access to the full data. [7]

10.8 Conclusion

In this chapter, we showed that overall dissemination times of transactions in the Bitcoin network can be estimated using the minimum number of connections required by the Bitcoin reference client, i.e., eight connections. Low latency blockchain applications, such as ATMs and file storage applications, profit from such monitoring capabilities for an improved user experience and estimation of double-spend risk.

The monitoring solution is realized by modeling the dissemination times using a lognormal distribution. Such a distribution describes 98% of the collected transaction data well. We provide a proof-of-concept implementation[11] of our monitoring as well as all collected datasets[12] and methodology tools[13].

The noise created by using a very small number of connections is reduced by a Bayesian scheme to adapt the estimates over several measurements. We also provide a mechanism to determine the difference between the measurement and estimate, circumventing the unknown shift of the real distribution. While the concrete modeling and implementation rely on effects present in Bitcoin, variants are possible for similar networks, and the methodology can be applied to different distributions and models.

The results of the provided tool show good adaption to inherently noisy real-world data independent of geographic location and stable over time.

[11] `https://github.com/vs-uulm/btcmon`
[12] DOI 10.5281/zenodo.2547396
[13] `https://github.com/vs-uulm/CoinView`

Chapter 11

Threshold Cryptography for k-Anonymous Broadcasts

This chapter is based on a paper accepted, but not yet published, at DAIS[1] *[2].*

[2] D. Mödinger, J. Dispan, and F. J. Hauck. "Shared-Dining: Broadcasting Secret Shares using Dining-Cryptographer Groups". In: *Accepted at 21st International Conference on Distributed Applications and Interoperable Systems (DAIS)*. 2021.

In the background chapter we introduced various building blocks and protocols to disseminate messages in a network. Chaum's dining-cryptographers groups [32] provide strong guarantees and have been used by various state-of-the-art protocols such as Dissent [36, 122] and the k-anonymous message transmission protocol [12].

Although DC networks provided very strong privacy, to use them efficiently for broadcast communication requires additional protocols layered on top of the DC network, e.g., a flood-and-prune broadcast, as used in 3P3. This creates additional risks, as non-cooperating participants in the layered protocol might force the true originator to step up and jeopardize their anonymity. Ideally, a system incentivizes nodes to participate instead of only punishing misbehaving nodes.

In this chapter, we propose a system combining dining-cryptographers groups and (n, k)-Shamir's secret sharing. Our system prevents identification of the originator in the presence of up to $k - 1$ attackers in the DC group, for a given security parameter $k < n$ with a DC group size of n. Broadcasting the shares requires at least k participants, leading to enforced k-anonymity during the broadcast. This chapter is accompanied by a proof-of-concept implementation and it's evaluation.

This chapter is structured in the following way: In Section 11.1, we introduce relevant notation and Shamir's secret sharing technique. We propose our k-resistant solution to broadcast messages using a DC-protocol and Shamir's secret sharing in Section 11.2. We provide a proof of the security and privacy of our scheme in Section 11.3, while an evaluation of the performance of our scheme can be found in Section 11.4. Lastly, in Section 11.5, we discuss possible applications of our scheme.

[1] DAIS 2021 - 21st International Conference on Distributed Applications and Interoperable Systems

11.1 Background

In this section, we detail the specific background of this chapter. This includes Shamir's secret sharing technique, as well as polynomial properties and goals of network participants.

Shamir's Secret Sharing

Shamir's secret sharing [99] splits a message into n shares so that k with $1 \leq k \leq n$ shares are required to reconstruct the original message. This is often called a (n, k) threshold scheme.

Any polynomial $f = \sum_{i=0}^{k-1} a_i x^i, a_{k-1} \neq 0$ of degree $k-1$ is unambiguously defined by any k points $(x_i, f(x_i))$ and can be reconstructed from them [50]. Given n pairwise distinct points of f, we can denote the set as:

$$\{(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_n, f(x_n)) : \forall i \neq j, x_i \neq x_j\}.$$

The original polynomial can be recovered from any subset of points of size k using Lagrange interpolation. The recovered polynomial is independent of the chosen points [50] and is computed by:

$$f(x) = \sum_{i=1}^{k} f(x_i)\mathcal{L}_i(x),$$
$$\mathcal{L}_i(x) = \prod_{j=1, j\neq i}^{k} \frac{x - x_j}{x_i - x_j}.$$

These approaches work over the reals as well as over fields $\mathbb{Z}_p$, making all operations over integers modulo p. For security reasons, a suitably large prime p is required, which depends on the use case and expected computing power of an attacker.

Given a message $m \in \mathbb{Z}_p$ we now want to construct a polynomial $f \in \mathbb{Z}_p[x]$, the polynomial space over the given integers. The boundary conditions for f are:

$$\deg(f) = k - 1,$$
$$f(0) = m.$$

A polynomial can be constructed easily by choosing integers $r_1, \ldots, r_{k-1} \in \mathbb{Z}_p \backslash \{0\}$ randomly and computing

$$f(x) = m + \sum_{i=1}^{k-1} r_i x^i.$$

It is easy to see that $f(0) = m$, as all other coefficients will be eliminated and it holds that the degree of f is $k-1$. The required n secret shares can then be computed as

$$s_i = (i, f(i)), i \in \{1, \ldots, n\}.$$

A Galois field $\mathrm{GF}(2^n)$ of suitable size is used to implement Shamir's secret sharing efficiently, usually $\mathrm{GF}(2^8)$. A notable property of these fields is that the addition of elements is equivalent to bitwise xor of their binary representation.

Goals of Participants

Our honest peers' goal is to broadcast a message within the network while maintaining sender anonymity, i.e., at least $k-1$ other nodes should be indistinguishable from them as the originator, where k depends on the parameters chosen in the system. Honest nodes will strictly follow the protocol, as their goal is to broadcast messages correctly.

The primary goal of the attacker is to identify the participant sending the message. Attackers follow the semi-honest model, i.e., they follow the protocol, with a small modification: They are allowed to refuse cooperation in the flood-and-prune broadcasting phase. They will combine all knowledge they can acquire throughout the protocol, e.g., all messages they receive. Attackers cannot manipulate the network, compromise other nodes, and solve computationally-infeasible problems such as encryption schemes. The privacy section details additional measures and their applicability with malicious attackers.

11.2 Secret-Sharing Dining-Cryptographers Protocol

Consider a group of size n, where one participant wants to transmit a message. Instead of transmitting the same message m to all participants, we split m into n secrets $s_1, \ldots, s_n$, one for each participant, using a (n, k)-Shamir's secret-sharing technique. Each secret is transmitted simultaneously during a modified dining-cryptographers round, resulting in each participant ending up with a single share of the message. After receiving their secret, every participant starts a flood-and-prune broadcast to transmit their share. Once a participant accumulates k shares, they can reconstruct the message m. The values of k and p required for the secret-sharing are system parameters, i.e., they are known beforehand and stay the same in the whole system.

Our protocol consists of three phases, which are shown in Figure 11.1. In the first phase, named Split, a given message m is split into n secret shares. To split the message, we chose $k-1$ random numbers $r_1, \ldots, r_{k.1}$. To create a random polynomial f which evaluates as $f(0) = m$, we use $f(x) = m + \sum_{i=1}^{k-1} r_i x^i$. Lastly we compute the secret shares $s_i = (i, f(i) \mod p)$ for all $i \in [1, n]$.

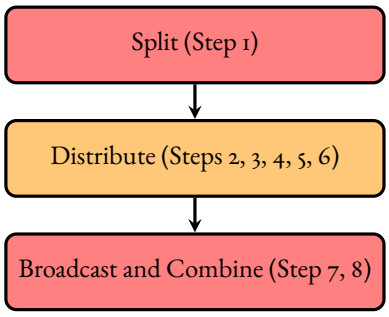


Figure 11.1: The three phases of the protocol and their corresponding steps explained in Algorithm 15. A message gets split up into n shares, which are then distributed to group members via a dining-cryptographers broadcast, one share for each. Any k members can then cooperate and recover the original message, corresponding to the split phase. [2]

In the following phase (the distribution phase), each of the n participants of the network then receives a unique secret s_i. Note that the DC protocol as described previously

(cf. Chapter 3) can only be used to make anonymous broadcasts, but cannot send individual messages to certain participants anonymously. We modify the protocol in such a way that this becomes possible. The modified DC protocol version is shown in Algorithm 15, note that a node that does not intend to send anything, still proceeds with $m_{\text{self}} = 0$. The key modification compared to the original DC protocol as described by Chaum [32] (shown in Algorithm 2) is that Step 4 no longer makes a broadcast, but transmits individual messages to other participants. This reduces the privacy guarantees of the dining cryptographers protocol to the security property of the secret sharing technique, this is further discussed in section 11.3.

Algorithm 15 Modified DC protocol using a secret sharing scheme to distribute individual message shares to all participants and broadcasting said shares to reconstruct the original message. [2]

Input: Message m_{self} of length ℓ
Environment: Group $G_{\text{self}} = \{g_i : i \in \{1 \dots k\}\}$, including the executing node g_{self}

1: Split m_{self} into n parts $m_{\text{self},1}, \dots, m_{\text{self},n}$ using the secret-sharing scheme
2: Establish random secrets $s_{\text{self},i}$ of length ℓ with each member $g_i, i \neq \text{self}$
3: $M_{\text{self},i} = m_{\text{self},i} \oplus \bigoplus_{j=1\dots n, j\neq\text{self}} s_{\text{self},j}\ \forall i \in \{1 \dots n\}$
4: Send $M_{\text{self},i}$ to $g_i\ \forall i \in \{1 \dots n\} \setminus \{\text{self}\}$
5: Receive $M_{i,\text{self}}$ from $g_i\ \forall i \in \{1 \dots n\} \setminus \{\text{self}\}$
6: $m_{\text{self},out} = \bigoplus_{i=1\dots n, j\neq\text{self}} M_{i,\text{self}}$
7: Broadcast $m_{\text{self},out}$
8: Reconstruct m_{out} after receiving $k-1$ other shares

The output of the distributed xor function that participant g_h computes is no longer $m_{out} = \bigoplus_{i=1\dots n} m_i$ but rather $m_{h,out} = \bigoplus_{i=1\dots n} m_{i,h}$. Each member must now broadcast the message m_{h_out}. If at least k participants broadcast their message, every recipient can decode the original message. If $k-2$ or fewer participants broadcast the message, no one can decode the message. When exactly $k-1$ participants broadcast, only non-broadcasting participants of the group can decode the message, as they possess the last share required to decrypt the message themselves. Verifying the correctness of the result is omitted for the simplicity of the presentation. It would require application-level integrity protection, i.e., there needs to be a way to ensure a message is valid for the application using the protocol.

Correctness

For the protocol's correctness, we assume all participants execute the DC protocol correctly, no errors occurred, and everyone used a (n, k)-Shamir's secret sharing technique. In a first step, we show that participants can reconstruct the sum of all Shamir's secret sharing points from the messages received in the DC protocol. From this, we reconstruct the original message $m_i \neq 0$ in a second step, given a successful sharing round.

Recovering the Sum of All Shared Points

The i-th participant receives the $n-1$ messages

$$M_{1,i} \dots M_{i-1,i} M_{i+1,i} \dots M_{n,i}.$$

Further, they create the message $M_{i,i}$ themselves. Each message has the form

$$M_{h,i} = m_{h,i} \oplus \bigoplus_{j \in \{1 \dots n\} \setminus \{h\}} s_{h,j}.$$

Therefore, the combination through xor of all receives messages is

$$\begin{aligned}
\bigoplus_{h \in \{1 \dots n\}} M_{h,i} &= \bigoplus_{h \in \{1 \dots n\}} \left(m_{h,i} \oplus \bigoplus_{j \in \{1 \dots n\} \setminus \{h\}} s_{h,j} \right) \\
&= \left(\bigoplus_{h \in \{1 \dots n\}} m_{h,i} \right) \oplus \left(\underbrace{\bigoplus_{h \in \{1 \dots n\}} \bigoplus_{j \in \{1 \dots n\} \setminus \{h\}} s_{h,j}}_{=0, \text{ as } s_{h,j} \oplus s_{j,h} = 0} \right) \\
&= \bigoplus_{h \in \{1 \dots n\}} m_{h,i}.
\end{aligned}$$

As $m_{h,i}$ was created through the Shamir's secret sharing protocol, they have the form $m_{h,i} = p_h(i)$. Here p_h is the polynomial created by participant h to split their message. The polynomial is created over the Galois field $\mathrm{GF}(2^8)$, a field with characteristic 2. In fields of characteristic 2, xor and addition are equivalent. Therefore, it holds that

$$\bigoplus_{h \in \{1 \dots n\}} m_{h,i} = \bigoplus_{h \in \{1 \dots n\}} p_h(i) \overset{\text{over GF}(2^q)}{=} \sum_{h \in \{1 \dots n\}} p_h(i).$$

Reconstruction of the Shared Message

In this second step, we show that receiving k distinct results allows us to reconstruct the original message input of the protocol. We assume that the flooding mechanism, or any appropriate sharing protocol, correctly distributed k shares to all participants. Without loss of generality, we assume a participant received the first k messages

$$\sum_{h \in \{1 \dots n\}} p_h(1), \dots, \sum_{h \in \{1 \dots n\}} p_h(k).$$

We saw in the section on Lagrange interpolation, that polynomial interpolation is uniquely possible with k evaluation points $p(1), \dots, p(k)$ for a polynomial p of degree $\deg(p) = k - 1$. We interpret our received messages as points of a polynomial $p_\sum$:

$$p_\sum(i) := \sum_{h \in \{1 \dots n\}} p_h(i).$$

Polynomial interpolation is unique with the given degree restrictions and polynomial addition can not increase the degree of the resulting polynomial. It holds, therefore, that

$$p_\sum = \sum_{h \in \{1 \dots n\}} p_h.$$

Evaluation and addition is commutative for polynomials, i.e., $(f + g)(x) = f(x) + g(x)$. Lastly, assume the messages are encoded at evaluation position s.

$$p_{\sum}(s) = \left(\sum_{h \in \{1 \dots n\}} p_h \right)(s) = \left(\sum_{h \in \{1 \dots n\}} \underbrace{p_h(s)}_{=m_i} \right)$$

If at most one message $m_i \neq 0$ exists, the reconstruction of the message is successful. Otherwise, the sum of all non-zero messages is restored.

11.3 Security and Privacy Evaluation

For this evaluation, we assume a group size of n participants using a secure (n, k)-secret sharing scheme. We restrict ourselves to the group communication, as the flood-and-prune broadcast has no interesting privacy or security properties. We will first establish the goal of our privacy analysis and analyze two attacker models, semi-honest participants and outside observers, regarding this goal.

Goal

Let $M_i = (M_{i,1}, \dots, M_{i,n})$ be the vector of messages created by node i in a system with n participants, and Setup the creation of groups and distribution of keys and parameters. Let f be the function combining such a vector into the intended message, i.e., the combination algorithm of the secret sharing scheme. Within the formalization, we denote the previously presented Algorithm 15 as Alg15, which is used to create all messages $M_{i,j}$. Let the probability that $k-1$ attackers A successfully identify a node ℓ sending a message be denoted by

$$P\left[f(M_\ell) \neq 0 \middle| \begin{array}{c} pp \leftarrow \text{Setup}(\lambda, k, f) \\ M_i := M_{i,j}, i, j \in \{1 \dots n\} \leftarrow \text{Alg15}(pp) \\ \ell \in \{1, k+1, \dots, n\} \leftarrow A(pp, M_{i,j}, j \in \{2 \dots k\}) \end{array} \right].$$

We call our scheme $(n, k-1)$ secure if this probability is only negligibly different from selecting a participant out of the $n-k+1$ non attackers at random, i.e.,

$$\left| P - \frac{1}{n-k+1} \right| < \text{negl}(\lambda).$$

Informally, this definition is true when $k-1$ colluding nodes can not identify the originator of the message within the set of $n - |\text{attackers}|$ non-colluding nodes. But once k nodes cooperate, no guarantees are made.

Semi-Honest Model

To show our scheme fulfills the previous definition, let there be $k-1$ colluding attackers present in the group, which follow the semi-honest model. Assume, without loss of generality as the nodes can be renumbered, that the victim has index 1 and the attackers' index 2 through k.

These colluding participants can collect $k-1$ messages $M_{i,j}$ of the form $M_{i,j} = m_{i,j} \oplus \bigoplus_{h \in \{1 \dots n\}} s_{i,h}$ by any participant i and the honest reconstruction of $p_{\sum}$, which provides the transmitted message m and the sum of all point evaluations. To identify the originator, the attackers need to compute any $m_{1,j}$ of the victim or, equivalently, their aggregate key

$\bigoplus_j s_{1,j}$. The original proof of Chaum holds for directly reconstructing $\bigoplus_j s_{1,j}$, so we will focus on $m_{1,j}$. Note that $m_{1,j} = p_1(j)$ is equivalent, where the polynomial p_1 has degree $\deg(p_i) = k - 1$ and the form

$$p_i(x) = \sum_{\ell=1}^{k} a_\ell x^{\ell-1}.$$

Given $k - 1$ messages $M_{1,2} \dots M_{1,k+1}$ and $i \neq j$ we can see that it holds that

$$\begin{aligned}
M_{1,i} \oplus M_{1,j} &= \left(m_{1,i} \oplus \bigoplus_{h\in\{1\dots n\}} s_{1,h} \right) \oplus \left(m_{1,j} \oplus \bigoplus_{h\in\{1\dots n\}} s_{1,h} \right) \\
&= m_{1,i} \oplus m_{1,j} \oplus \left(\bigoplus_{h\in\{1\dots n\}} s_{1,h} \oplus \bigoplus_{h\in\{1\dots n\}} s_{1,h} \right) \\
&= m_{1,i} \oplus m_{1,j} \oplus \bigoplus_{h\in\{1\dots n\}} \left(\underbrace{s_{1,h} \oplus s_{1,h}}_{=0} \right) \\
&= m_{1,i} \oplus m_{1,j}
\end{aligned}$$

As xor and addition are equivalent over base fields of characteristic 2, which we use, and that $m_{i,j} = p_i(j)$, we can see that

$$m_{1,i} \oplus m_{1,j} = p_1(i) + p_1(j).$$

Note that this only holds for even combinations, i.e., we can not create $p_1(2) + p_1(3) + p_1(4)$. All combinations with an even number of parts can be constructed as a linear combination of combinations of two parts. Therefore, using this equation, we can create only $k - 2$ linearly independent equations

$$[p_1] = \begin{cases} \sum_{i=1}^{k} 2a_i 2^{i-1} 3^{i-1} & = & p_1(2) + p_1(3) \\ \vdots & & \vdots \\ \sum_{i=1}^{k} 2a_i (k-1)^{i-1} k^{i-1} & = & p_1(k-1) + p_1(k) \end{cases}$$

The attackers can reconstruct the transmitted message $m = p_{\sum}(0)$ by following the protocol normally. Removing all attacker polynomials $p_2 \dots p_k$ leaves

$$p_{\sum} - \sum_{j=2}^{k} p_j = p_1 + \sum_{j=k+1}^{n} p_j =: p_{\text{remains}}.$$

Using this and applying the strategy to compute $[p_1]$ on all non-colluding participants allows the attackers to create the following matrix

$$\begin{bmatrix} [p_1] & [0] & \cdots & [0] & S_1 \\ [0] & [p_{k+1}] & & [0] & S_{k+1} \\ \vdots & & \ddots & & \vdots \\ [0] & [0] & & [p_n] & S_n \\ 1\dots1 & 1\dots1 & \dots & 1\dots1 & p_{\text{remains}} \end{bmatrix}$$

All entries $[p_i]$ represent the previous equation systems with their respective solution vectors $S_i = (p_i(2) + p_i(3), \ldots, p_i(k-1) + p_i(k))$ generated from the messages $M_{i,j}$. Each block $[p_i]$ and $[0]$ have $k-2$ rows, while the final row models p_{remains}, where all coefficients are present exactly once. All further derivations of p_{remains} would not be linearly independent equations. There is no further relation between the remaining polynomials $p_1, p_{k+1}, \ldots, p_n$, as all are chosen independently at random.

Solving the equations for a single participant leaves us with $k-2+1$ rows ($[p_i]$ and p_{remains}) and k indeterminants $a_1, \ldots, a_k$ and therefore k columns. The full matrix has $(n-k+1) \times (k-2)+1$ rows and $k \times (n-k+1)+1$ columns. According to the Rouché–Capelli theorem there is a unique solution for a system of equations $Ax = b$ iff $rank(A) = rank(A|b)$. Therefore, our system of equations leads to ambiguous reconstruction, as there are infinitely many solutions. Further, it would break the security assumption of the base secret sharing protocol.

If a message can be verified after decryption, an exhaustive search for solutions is possible. The underlying field size corresponds to λ in our previous definition as it determines the cost for an exhaustive search. Absent any notes identifying correct solutions, all solutions to the system of equations are equally valid and likely. Therefore, any of the $n-k+1$ possible victims might be the sender with equal probability $P[f(M_\ell) \neq 0] = \frac{1}{n-k+1}$.

Outside Observers

Outside observers cannot determine the origin of a broadcast as long as secure channels are used, as all participants have to send data of the same size for each transmission. Similar to classical DC networks, no guarantees can be retained when the channels are no longer secure.

Modern DC Malicious Mitigation

While attackers act semi-honest in the previous evaluation, modern dining-cryptographers protocols apply various mechanisms to deal with collisions, fairness, and robustness issues of the protocol [12, 52].

To increase fairness the protocol can apply $2n$ slots, where every participant may use at most one slot at a time, which they chose randomly. Participants create commitments on each secret share they create, to prevent cheating. Each commitment is broadcast to the whole group. When more than n slots are occupied, a zero-knowledge proof allows every honest participant to show their innocence. As long as every participant uses at most one slot, any participant has a fair chance of at least $\frac{1}{2}$ to transmit their message.

Lastly, the most problematic case, selective non-participation, can be combated by preemptively sharing all secrets in encrypted form with the group. If any node claims another refuses to send their messages to them, any other node can take over by forwarding the encrypted shares.

These techniques can be applied to our proposed protocol to make it resistant to malicious participants. Slots can be easily introduced by applying the secret splitting per slot, instead of the full message. Commitments can be created in the same form as by von Ahn et al. [12]: each slot provides its own commitments. The zero-knowledge proof of fairness by von Ahn et al. can be easily extended as well: The opening of commitments is combined with a reconstruction of the secret shares into the actual message. The resulting message has to be zero.

11.4 Performance Evaluation

This section shows the performance results for our scheme and the methodology used to acquire those results.

Methodology

We implemented a prototype simulation that can simulate both the original DC protocol and our modified version. The simulation is available online[2] and written in Java. We use built-in synchronization utilities to model the communication and synchronization of participants. For threshold cryptography, we used the open-source library shamir[3] in version 0.7.0. The Shamir library uses a Galois field GF(2^8) as a base field. The library provides two methods, split and join, of combined complexity of $\mathcal{O}(\ell \cdot (n + k^2))$, where ℓ is the length of the message and n, k are the parameters of Shamir's secret sharing.

We ran this implementation 10 to 30 times for each combination of parameters. We aggregated the measured throughput and computed the average and standard deviation.

Network latency is simulated, but we set it to 0 to prevent influence on the measured variable when not specified. To mitigate our results' distortion due to runtime optimization attempts by the Java virtual machine, we ran a warm-up phase before each test. In this warm-up phase, 100 runs were performed that are not included in our results.

We compared the modified DC protocol, denoted as Broadcast, to Chaum's original version's performance, denoted as DC phase in graphs. We investigated several core issues:

- The size ℓ of the transmitted message,
- the scaling behavior of the protocol, i.e., increasing n,
- the performance impact of variable k values,
- the influence of network latency.

For the performance evaluation, we consider a simple collaboration protocol in place of the broadcast to reduce simulation effort. Participants collaborate with at least $k - 1$ other members to recover the original message m. Algorithm 16 provides a cooperation scheme which, when executed correctly by each network member, produces the minimum number $n(k - 1)$ of transmitted messages.

Algorithm 16 Combine protocol to emulate broadcast, reconstructing the original message from cooperating with other participants. [2]

Input: Message part $m_i := \bigoplus_j m_{j,i}$, number of required message shares k
Environment: Group $G_{\text{self}} = \{g_i : i \in \{1 \dots n\}\}$, including the executing node g_{self}
1: Send m_i to g_j $\forall j \in \{i + a \mod (n + 1) \mid a \in \mathbb{N}, 1 \leq a \leq k - 1\}$
2: Receive m_r from $g_r \forall r \in \{x \mid \exists a, 1 \leq a \leq k - 1 : x + a \mod (n + 1) = i\}$
3: **return** m_{out} from the $k - 1$ received messages and m_i.

We opted not to evaluate a full flooding approach, as this would shift the focus from the modifications we performed. Additionally, the performance characteristics of flooding approaches are well known.

[2] See `https://github.com/vs-uulm/thc-in-dc-simulation`
[3] `https://github.com/codahale/shamir`

Message Size ℓ

Both the original DC protocol and our modified protocol transmit a message of the fixed length ℓ each round. We want to keep ℓ as close as possible to the actual length of the information we want to send.

Messages longer than ℓ can be split into multiple messages, increasing overhead and, therefore, decreasing throughput. If the information is shorter than ℓ, it can be padded with 0-bytes to make it size ℓ, leading to the transmission of more data than necessary, producing overhead as well.

We show the results of this overhead in Figure 11.2. We varied ℓ from 32 B to 32 kB with $n = 10$ and a given real message size of 8 kB. We chose the relevant parameters for this benchmark with regard to the potential use for our proposed system in the field of cryptocurrencies. Therefore we picked sizes applicable to groups [5] and transaction sizes, validating our assumption that the performance is at its peak when ℓ is roughly equal to the size of the information to transmit. Results for varying sizes of n (not shown) lead to the same validation.

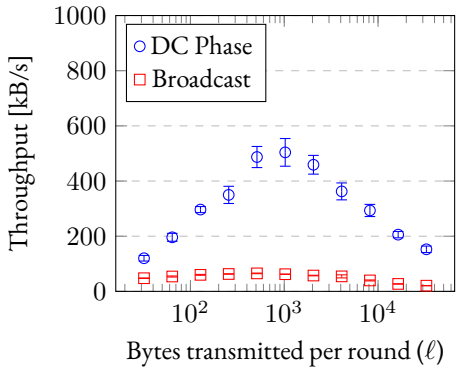


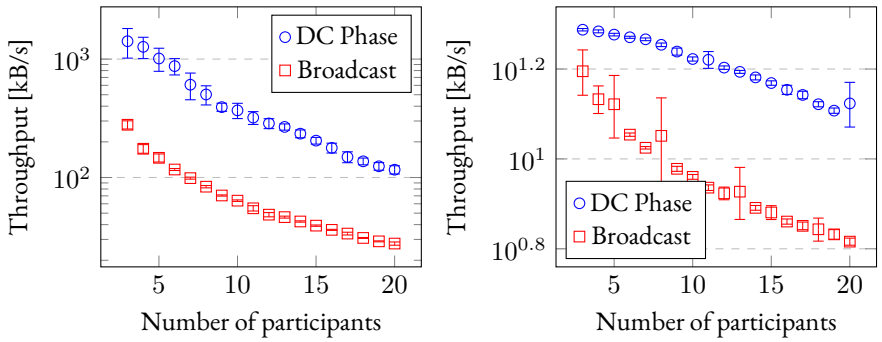
Figure 11.2: The measured throughput and its standard deviation while increasing ℓ for $n = 10$ and $k = 3$. The transmitted information has size 8 kB. Performance reaches its peak when ℓ is about the size of the transmitted information. [2]

Network Size n

While the message complexity for the core DC protocol is identical in both schemes, a round of the modified version of the DC protocol needs additional messages in the cooperation phase. When keeping k constant, the modified version of the protocol requires $\mathcal{O}(n)$ more transmissions than the original protocol. As we see in Figure 11.3a, this makes a significant difference for a low number of participants. Because both versions of the protocol are of overall complexity $\mathcal{O}(n^2)$, the linear performance penalty becomes less of a concern when n grows larger.

The increased number of sent messages is only one reason for the worse performance of our system. The time for performing one round of the protocol can be divided into two parts: time spent communicating and time spent for calculations. Our system requires a larger amount of computations compared to the classical DC protocol. In addition to performing the core DC functionality, it also splits and joins the messages to transmit using a secret-sharing-scheme, resulting in the strong performance difference in Figure 11.3a.

Network Delay



(a) Delay 0 ms

(b) Delay 100 ms

Figure 11.3: Measuring throughput in DC protocol runs over networks of various sizes. Variable n, $k = 3$, $\ell = 8\,\text{kB}$. [2]

To investigate less optimal environments, we simulated our scheme using a delay of $>$ 0ms. The gap in performance between the original DC protocol and our system is notably smaller. The results of adding a delay of 100 ms are shown in Figure 11.3b respectively, but simulations with intermediate values show similar results. We chose 100 ms as a typical representation of internet communication delay, but in real-world scenarios, it can be considerably smaller.

Note that when adding delay, our system only improves relative to the original dining-cryptographers protocol. The absolute performance of both approaches suffers under message transmission delay. We measured throughput rates of $13.58\,\text{kB/s}$ for $n = 4$ and $9.12\,\text{kB/s}$ for $n = 10$ with a delay of 100 ms.

Number of Shares k

Lastly, the value of k is the number of message shares needed to restore a message and significantly impacts the protocol. This impact is due to participants needing to compute additional methods and perform additional $k - 1$ transmissions to receive the shares. Figure 11.4 shows the results of benchmarking our system with $n = 10$, $\ell = 8\,\text{kB}$ and $k \in \{3, \ldots, 10\}$. As expected, increasing k decreases our system's performance.

11.5 Applications

As we have seen, our version of a DC protocol typically achieves throughput rates between $10\,\text{kB/s}$ and $100\,\text{kB/s}$. A real-world application for our system lies in the anonymous transmission of transaction data for blockchains, e.g., in an environment like the one proposed in [5]. Such transaction data are typically of size $< 1\,\text{kB}$, whereas group sizes are between $n = 4$ and $n = 10$ and transmission delay is around 100 ms.

Many blockchain systems produce only a few transactions per second, despite thousands of nodes participating in the network. Separating these into groups for privacy is unlikely to lead to any groups that require more than one transaction per second. Therefore, every system that can achieve speeds of > 1 transactions made per second is suitable for application

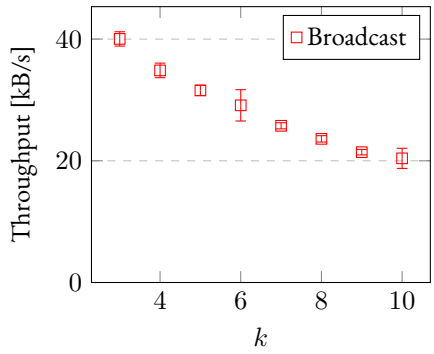


Figure 11.4: Measuring network throughput while varying parameter k. The other parameters are kept constant with $n = 10$ and ℓ as well as the size of the transmitted information as 8 kB. [2]

in a system as the one proposed in [5]. Our system is well-suited for such a task, as it can efficiently work with this load.

11.6 Conclusion

In this chapter, we introduced a novel combination of the classical dining-cryptographers protocol and Shamir's secret sharing to enforce k-anonymity during a broadcast process. This problem arises during a broadcast, as nodes that already received the message might refuse further cooperation. We showed that the protocol is computationally secure in the number of shares k, maintaining $n - |attackers|$-anonymity for at most $k - 1$ attackers.

Our system provides a first, unoptimized solution, so further work could improve the system's performance and flexibility. In our simulation, this results in throughput rates between 10 kB/s and 100 kB/s for a full broadcast simulation and over 500 kB/s with reasonable privacy settings. These performance results show our system is viable for a wide range of applications, such as blockchain-transaction dissemination in peer-to-peer networks and other peer-to-peer applications requiring high privacy guarantees.

Chapter 12

Pixy: Privacy-Increasing Group Creation

This chapter is based on a previous publication at ICNS [3].

[3] D. Mödinger, N. Fröhlich, and F. J. Hauck. "Pixy: A Privacy-Increasing Group Creation Scheme". In: *5th International Conference on Network Security (ICNS)*. 2020.

In previous chapters, we discussed network protocols applying group-based privacy mechanisms [5, 12, 32]. The privacy of these protocols is based on the indistinguishability of the originator of a message from all other group participants, i.e. the anonymity set is the group. However, if multiple participants of the group collude, they can reduce the effective privacy of this method.

The current state of the art of techniques to detect nodes pretending to be multiple identities, i.e., Sybil detection, for group based protocols performs said detection in parallel to the communication protocol. At this point privacy could have been violated already. To prevent this, testing needs to be done before any privacy requiring group communication takes place. This creates a novel combination of group-based communication techniques and Sybil detection mechanisms, as the current state of the art relies on problematic assumptions, e.g., on already established trust relationships. Those can be easily gained by the attacker through short preparation times before the attack.

We propose Pixy to tackle these problems. Pixy is a novel group creation scheme focusing on privacy for the participants. Pixy increases privacy guarantees for group creation, reducing the required group size for a given desired privacy level. Von Ahn et al. [12] and 3P3 [4] use groups for privacy and gain performance from reduced group sizes. The scheme provides a flexible component-based approach and well-suited default mechanisms.

The structure of this chapter is as follows: First, we discuss existing group creation strategies in Section 12.1. Section 12.2 gives an overview of the relevant background such as mechanisms we apply and the scenario we operate in. In Section 12.3 we describe Pixy, our group creation scheme, and provide a first evaluation of Pixy in Section 12.4.

12.1 Related Work

The literature does not have any ready-made solution for group formation under Sybil attacks or collaboration, as Sybil attack detection and prevention have to be customized to the

underlying architecture to be effective.

There are two noteworthy group creation schemes for privacy or security, which we will discuss here. The trustworthy group making algorithm [14] groups nodes based on a partially established trust graph between the nodes. The secure group agreement protocol [35] deals with a group invitation message from which the group members can be determined and verified.

Trustworthy Group Making Algorithm

In 2011 Aikebaier et al. [14] published a novel approach which builds a group of peers based on transitive trust. They apply their previous trustworthiness concept [13]: A peer is considered more trustworthy, the more messages a peer successfully forwards.

Aikebaier et al. develop a broadcast algorithm which relies on the trustworthiness of the individual peers for a formed group. Group formation is carried out via a previously established trust graph and a group initiator peer which calculates the trustworthiness of the other peers from its point of view. The trustworthiness of a node is either direct trustworthiness computed as a ratio of several successful interactions to all interactions with such a node, or as transitive trust based on direct trust multiplied by the trust of the intermediate path. For multiple paths, the maximum trust value is chosen as the trust value for this participant. If there have been no previous interactions, direct trust is undefined.

To form a group, Aikebaier et al. [14] start with one source peer which invites the neighbor peers with trustworthiness over a specific threshold. These neighbor peers then introduce their neighbor peers with a trust value over this threshold to the source peer. If a group cannot be formed with the desired threshold, the source peer reduces the threshold and restarts the group formation.

Trust information, as it is used in this protocol, plays an important role in Pixy. The protocol by Aikebaier et al. could be used to realize the trust management and initial group formation of Pixy. Beyond that, Pixy applies testing of desired properties to increase trust in group participants.

Secure Group Agreement Protocol

The approach by Corman et al. [35] describes another group building mechanism in the context of peer-to-peer video games over the Internet. In this context, a subset of peers in the network can form a verification group for occurred events of a game. Corman et al. base their approach on a distributed hash table (DHT) over a peer-to-peer network, which maps the peers and files into the key spaces using a hash function, as well as an established public key infrastructure.

A group initiator computes an invitation message for the group members and sends it to the peers of the group. Every invited peer answers with an acceptance message. If a response of a peer does not return, the group peers try to forward the invitation to the missing peer. If the answer of the absent peer is still missing, the group formation will be aborted.

An invitation message consists of a signature and hashable contents, including a group id, timestamp, purpose and an incrementing member number. Peers are chosen randomly based on the random oracle property of the hash value of the content, to prevent precomputation or groups and produce well-randomized group composition. This property of producing well-randomized groups allows for a baseline comparison. The protocol can be used in the absence of trust information but does not provide further privacy assurances beyond randomness.

Applicability

The two group formation protocols serve different purposes in the field of peer-to-peer networks and are therefore not well applicable to the given problem of increasing privacy. The one by Corman et al. [35] represents a verified group invitation message, and the other by Aikebaier et al. [14] is based on already established trust relations between peers and contains an initiator peer which starts the group formation with the most trustworthy neighbor peers.

In our situation, the verified invitation message could be used to form a group, but would not help with our problem of Sybil attacks and collaboration. The group formation by Aikebaier would support the protection of the group forming against unfamiliar nodes, but if one Sybil peer is in the trust range, it can invite other Sybil nodes and the intrusion would never be detected. So these approaches are not suitable for our problem.

12.2 Background

In this section, we will discuss the relevant background information. First, we will elaborate on our assumptions and scenario, setting important boundary conditions for viable mechanisms. Second, we will discuss some possible mechanisms and classifications of mechanisms we will apply in the scheme.

Sybil Prevention and Detection Mechanisms

When an attacker claims to have multiple identities to get an advantage, that's called a Sybil attack [41]. Sybil attacks typically occur in applications like online voting and reputation systems [60, 70]. Peer-to-peer networks with no certification authority are generally vulnerable to Sybil attacks [41] as well.

Categories of Detection Mechanisms

We provide an overview over typical mechanisms to detect Sybils based on previous survey articles [18, 60, 70].

- **Trusted Certification** makes a certification authority prevent Sybils [41].
- **Resource Testing** assumes limited hardware for participants and little variance in resources. If all participants need to prove the amount of resources they can expend, e.g., by a proof of work scheme, a Sybil attacker can only provide a fraction of the resources compared to a non-Sybil node [27, 71, 106].
- **Recurring Cost** reduces the complexity of the resource check but repeats it over time. An attacker, therefore, has to provision the required resources over longer amounts of time. Renting additional resources is therefore uneconomical [27].
- **Incentive-based Detection** provides protocol level incentives, e.g., increased trust or virtual coins, to a node reporting collusion by cryptographic proofs of messages [75].
- **Social Graph** requires an already established social trust graph, which is evaluated for connections between the area of honest nodes and dishonest node [37, 125, 126].

Most schemes try to discourage the attacker by increasing the cost of an attack over the profit of the attack. While the Sybil attack cannot be prevented in full, the probability of

a successful attack can be reduced. The effectiveness of Sybil detection or prevention technique depends on the use of a suitable techniques for the current architecture [60, 70].

We exclude Privilege Attenuation, which is not suited for peer-to-peer networks, Location and Position Verification and schemes based on Received Signal Strength Indication, as they do not work for wired internet connections. Lastly, we also do not consider Random Key Pre-Distribution as they require a static, i.e., non-dynamic, setup phase. From the remaining techniques, we will restrict ourselves to resource testing, as trusted certification either requires a trusted third party or another form of consensus. We do not consider recurring costs for now, as they could be implemented as a simple extension to our scheme. We also exclude social graph mechanisms as they require a pre-established trust graph, although they provide promising enhancements for Pixy in long-running networks.

Resource Testing

Resource testing can be realized for Sybil detection and prevention by well known computational puzzles [38, 43, 78]. In the literature, there are different designations for such puzzles.

A puzzle is used between a prover and a verifier. In general, a puzzle consists of three phases: Generation, solving and verification of the puzzle. Puzzles can be interactive or non-interactive and have a symmetric or asymmetric workload. The puzzles can further be divided by their application, which resources are used, and implementation of the verification process [15].

The considered application types are pricing puzzles, delaying puzzles, timing puzzles and AI hard puzzles, sometimes called CAPTCHAs. Delaying puzzles require a predefined amount of time and are sometimes called timelock puzzles [94] while timing puzzles have no predetermined time at construction. Pricing puzzles have a known lower-bound on resources and are used to increase the cost of an otherwise cheap operation [43].

Puzzles can further be separated by resources, most often: CPU, memory, bandwidth, network latency or ordering and human attention. Applicability depends on the use-case and setting. There are further properties of puzzles, the most important are the uniqueness of the challenge, unforgeability of solutions and non-parallelisability of the computation.

For this work, we evaluated several puzzles based on these properties, shown in Table 12.1.

	Time-lock [23, 64, 74, 94]	Non-parallelisable [59, 108]	Memory-bound [10, 42]	PoSW [33, 73]	Delay [25, 92, 120]
Resource	CPU	CPU	Memory	CPU	CPU
Simultaneous	+	+	+	+	+
Asymmetric	+	+	+	+	+
Implicit	/	±	/	/	/
Unforgeable	+	+	+	+	+
Unique Puzzle	+	+	+	+	+
Unique Solution	+	+	+	/	+
Sequential	+	±	+	+	/

Table 12.1: Overview over the characteristics of the interactive puzzles: Time-lock Puzzle, non-parallelisable puzzle, memory-bound function, proof of sequential work (PoSW) and delay function. + shows that a puzzle supports the property, while / signals no support. ± shows that support is non-homogenous over evaluated variants.

Abadi et al. Delaying Puzzle

The concept of Abadi et al. [10] for a memory-bound function is based on a tree with depth k. For now, let k be an integer. The tree is constructed from a random leaf value x_0 to the root value x_k. This construction ensures a sufficient size of the tree, around the size of $(k+1)(k+2)/2$. The depth k of the tree has to be much smaller than 2^n, with n an integer. The authors suggest that k should fulfill $k < 2^{n-5}$. The tree is computed by the function $F : [0, 2^n - 1] \to [0, 2^n - 1]$. F is a random function with no permutation.

The equation $x_{i+1} = F(x_i) \oplus i$ with the index $i \in (0, \ldots, k-1)$ calculates the nodes in the path from the leaf x_0 to the root value x_k. The calculation of the tree is a chain of repetitive applications of the function F.

The verifier has to choose a random leaf value x_0 and construct the tree path from x_0 to x_k. Over the path from x_0 to x_k, a checksum is built and sent to the prover with the root value x_k. The verifier adds the checksum of the path to x_0, the prover has to compute x_0.

To find the desired element, the prover computes the inverse function F^{-1} of F. The function F is chosen in such a way, that computing F^{-1} is less efficient than a memory access. So the most efficient solution is to construct a memory table of size 2^n for F and do reverse lookups. A depth-first search computation of the solution requires unpredictable random access to distributed locations of memory. From that, the prover constructs possible pre-image chains, which are compared to the given checksum by the verifier. If a solution is found, the verifier just needs to check if the solution matches x_0.

CPU-intensive algorithms for this puzzle solution exist, but for a good choice of the function F and the parameters n and k, these approaches do not bring any advantages. These solutions would need a longer period to solve the puzzle than to use the memory-bound one.

Abadi et al. also give relative values for the parameters. The researchers [10] suppose that the work f for the function F should be $(r/8) \leqslant f \leqslant r$ where r is the work for a memory read to prevent a fast inversion and computation. The depth k has to be chosen carefully with the assumptions that $k < 2^{n-5}$ to exceed cache lines, $k >> 4 \cdot (f/r)$ to force a high work ratio on the prover, $k \geqslant \left(2^{(n/2)+1} \cdot \sqrt{f/r} \cdot \sqrt{c}\right)$ with c as a cost factor for CPU intensive solutions to prevent a CPU-intensive search and $k \geqslant \left(2^{(n/2)+1} \cdot \sqrt{1/p} \cdot \sqrt{(f+w)/r}\right)$ so that the table can be built efficiently.

The integer p represents the number of such combined problems. A problem is considered as finding an x_0 from x_k. The costs of memory reading and writing are considered equal ($r = w$). The parameters c, r and w are integers. Abadi et al. [10] also provided an example for F and tested common cryptographic hash functions. They show that F can be constructed as a random function generated through a master function by $F(x) = MF(x) \oplus j = G(t, x)$. G consists of two pre-distributed random tables of size $2^{\frac{n}{2}}$ and discards the 16 most- and least significant bits of the multiplication of table entries. The resulting function is $F(x) = F(a_0|a_1)\,\text{middle-bits}(t_0[a_o] \cdot t_1[a_1])$.

The puzzle can be solved a little bit faster with the help of better hardware, but not as fast as it with CPU-bound puzzles. For the uniqueness of the puzzle, the function $F(x) = MF(x) \oplus j$ and x_0 has to be varied for every node. [10]

A closer mathematical and practical examination of the concept of Abadi et al. [10] in the direction of a concrete realisation would exceed the scope of this thesis. Dwork et al. [42] explored the same approach to Abadi et al., but they suggested a different random function. For simplicity, we restrict ourselves to the initial approach.

CAPTCHAs

CAPTCHAs are mechanisms to tell humans and machines apart [111, 112]. As they prevent automation, they are well suited to prevent automation-based Sybil attacks [66], e.g., by botnets. It is important to note that they can be only applied in contexts where automation is not an important goal. We argue that privacy is mostly relevant for humans, so having humans verify the initial step of a privacy-enhancing group formation is acceptable and automation is not desired.

CAPTCHAs are usually based on text, image, audio, video or games. The strength of each test depends on the development in research of AI and the realization of the technique. The most commonly used CAPTCHAs are text and image-based variants. They are easy to implement and realize [65, 83, 101, 112].

Machine learning approaches to solving classical CAPTCHAs produce success rates of up to 70% [129]. The increase in CAPTCHA solving through machine learning contributes to an arms race of CAPTCHAs and CAPTCHA solving. Researchers also produced new approaches to fight machine learning and pattern recognition using video-based gesture detection [110] and comparison-based identification [114]. These and mechanisms on adversarial examples [89, 100, 127] are promising but not developed enough to be used yet. Googles most recent version of reCAPTCHA seems to be considered secure for now.

12.3 Pixy: Group Creation Scheme

We introduced various ways of detecting Sybils. In this section, we introduce Pixy, our privacy-increasing group creation scheme, tying together the previously introduced approaches. First, we will give an overview of the scheme and then discuss the two phases of Pixy in detail. Lastly, we will show how decision making after the last phase works.

Overview

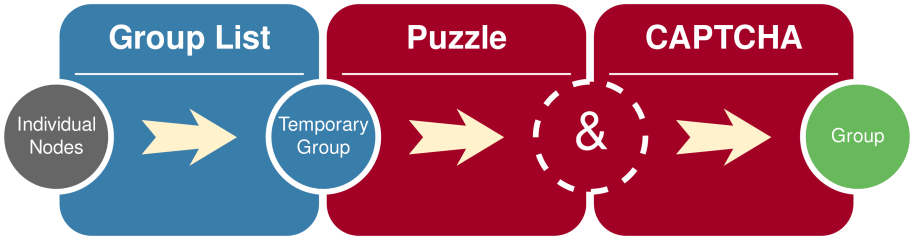


Figure 12.1: Overview of the architecture of Pixy. [3]

Pixy consists of two phases and is structured as a modular system. It is divided into two phases. The first phase is used to construct a temporary group based on group lists of untrusted groups, e.g., IP subnets. The second phase consists of aggregatable tests for participants to prove their independence. In Figure 12.1 the process of the scheme is illustrated with its two phases and its three elements in total.

The elements are building blocks and can be exchanged easily to support fast assimilation to new developments or different network architectures. This is important to create a future-proof scheme.

Phase 1: IP-based Gatekeeping

The result of the first phase is a temporary group with members that are not obviously collaborating. As we are dealing with internet messages, there is only little information we can use and will primarily use IP addresses similar to Tarzan and other systems [48, 70]. We assume the protocol can determine which peer is in which subnet, which is warranted due to the use of general internet-wide IP addresses. While we can not assume a pre-established trust graph, some participants may build a personal, hidden trust graph over time, augmenting their decision making. This trust information can be applied through the algorithms of Aikebaier et al. [13].

We use a list of groups for our scheme. The group list functions similar to a denylist, but we allow a single participant from each group. There should be a global group list for a given network, containing well known collaborating groups, e.g., Google. The collection of groups depends on the concrete threat model of the given network, e.g., a very paranoid network might restrict groups to participants from different continents by geo-IP grouping. One risk factor for group lists are VPNs and Tor like services, so depending on the attacker model, a network should include these or risk users including them in their group lists anyways. To allow for evolving group lists, distributed append-only ledgers, e.g., blockchains, might provide suitable background stability while making the lists future proof.

The check of IP addresses takes place before the actual group formation. Peers might already cancel group formation of invalid groups according to their group list If no peer cancels at this stage, the participants create connections to all participants and continue with phase 2 as a temporary group.

Phase 2: Testing

Once the temporary group is formed, participants start the testing phase. For this phase, all participants apply known Sybil detection mechanisms to test if any node in the groups are Sybils.

The second phase is modular, so it can be constructed with different puzzles. For a sensible default application, we propose two independent core modules: A delaying puzzle and a CAPTCHA.

Delaying Puzzle

The first presented module is a delaying puzzle. We recommend the delaying puzzle due to the properties we outlined in Table 12.1. As for this step, a puzzle is needed that can test the hardware of all nodes simultaneously by one verifier node. For this, the workload on the verifier side has to be low, i.e. asymmetric workloads are needed, otherwise, it can lead to a denial of service (DoS) attack, and the puzzle set per node must be unique so that no solutions can be reused.

Delay puzzles fulfill all the required criteria. As different CPUs produce a wide spread in cost, we require a memory-bound puzzle to have as little variance in devices as possible [15]. The puzzle verification should be public to reduce the work for verifier nodes which perform the current testing round. However, in this testing phase every node in the group will be a verifier node and will set the puzzle once.

The verifier node creates a unique puzzle for every node in the group, besides themselves, to prevent the reuse of solutions and checks the returned responses of the nodes. The returning time of the response will be recorded as well. The puzzles are distributed simultaneously to all nodes in the group.

We applied the puzzle by Abadi et al. [10] which fulfills our requirements (cf. Section 12.2).

CAPTCHA

In the second module of phase 2, the nodes are tested for human attention. In this puzzle, the number of verifier nodes can be reduced and thus the number of testing rounds. This step should, by design, not be automatable by the attacker or regular participants. This makes CAPTCHAs only acceptable in certain contexts, but we argue that privacy is a human-centric concept, which makes this a good compromise [66].

As discussed in Section 12.2, thus far, we lack real-world implementations of newer and open source CAPTCHA approaches [65, 83, 101, 112]. When reimplementing and evaluating the approaches is not possible, we recommend the proprietary reCAPTCHA by Google [17, 55, 113] to provide at least a concrete realisation of a CAPTCHA.

Attacks on CAPTCHAs continue to improve over time, including reCAPTCHA [29, 129]. reCAPTCHA will block some bots, and it requires the effort to develop a machine learning algorithm of the attacker to overcome this puzzle challenge. The benefit of using a CAPTCHA for a scheme like Pixy is that it prevents many kinds of attacks, while the downside is it requires frequent updates and flexibility on the CAPTCHA part.

Depending on the kind of CAPTCHA applied, the result can either be a binary decision or a 'belief of humanness', that can be used with a custom minimum threshold to accept the participant.

Decision Making

The testing of nodes in phase two requires an aggregation of scores and decision making. We recommend that each node be verifier at least once to reduce the probability of manipulation. This results in up to N, the number of total participants, testing rounds which can be different for all modules. For reduced rounds, the group should use a secure random scheme, e.g., as used by dissent [36], to select nodes. For the discussion, we will assume that N testing rounds are performed.

The approach of evaluating the puzzle results is inspired by the approach of Cárdenas-Haro and Konjevod [31]. In a testing round of the puzzle, the current verifier node sets the puzzles for every node simultaneously and uniquely. Thus, every prover node starts the computation of the solution for the puzzle at the same time and should finish at the same time. After a delaying puzzle, the results of all provers should return at the same time.

To measure the time, the time is divided into time slots of the length t. The real length t of the time slots depends on the time p for the puzzle solution as well as on the Internet delay d. The length t of the time slot should be smaller than the needed time for the puzzle computation p ($t \leq p$). This requirement prevents that a prover receives and answers the puzzle in the same time slot, therefore it would not be sufficient to measure the elapsed time. Further, it should hold that $d < t < p$, i.e., the timeslot is longer than the expected network delay. The responses to the challenges should return to the verifier in the same time slot.

The verifier creates a matrix of all response timeslots. Two peers, which are not distinct, cannot respond at the same time as the solutions have to be calculated sequentially by a Sybil node, as parallel computation would delay the response for too long. If two responses from two peers return in the same time slot, then these two peers have a high probability to be two distinct hardware devices. If the responses of two peers never return at the same time for all testing rounds, then it is very likely these two peers are Sybil nodes.

Due to network delay, a response might return a timeslot late. Therefore neighboring timeslots to the expected slot have reduced penalty for detection. The matrix values can be aggregated over all tests. Low values correspond to a high probability of Sybils being present. Suitable thresholds need careful evaluation of acceptable risk for the network though. All

results of all puzzles need to be above the desired thresholds for each puzzle for the group to be viable. If this condition is met, the group continues as a fully-formed group with the desired protocol.

12.4 Security Discussion

The literature on Sybil attack prevention and CAPTCHAs did not provide evidence on the effectiveness of the algorithms. So a full evaluation of Pixy is yet to be done. Instead, we will focus on a hypothetical exploration of the attack space. The two core goals to fulfill are:

- An attacker must not gain an advantage when Pixy is used.
- Multiple attacks must be impractical or noticeably harder by using Pixy.

Goal one is required to provide a strict improvement in privacy, while the results for goal two would indicate the effectiveness of Pixy. To conclude goal one, we evaluated possibilities to subvert the phases of our scheme.

We consider various common attackers with varying capabilities. A **single attacker node** with moderate access to resources in money and IP addresses, i.e., they can access free VPN services. Their goal is to subvert the group with little resources and preferably no cost. A **botnet**, which has many fully automated nodes with IP addresses which are well spread throughout most subnets. The goal of a botnet is to subvert the group while being economical with little expense but large scale hardware resources. A **large scale** corporation or agency of a nation-state with quite a lot of resources. The restrictions of entities of this scale are mostly political, e.g., they do not want to be noticed.

Phase 1: IP-based Gatekeeping

The first phase is primarily controlled by the node initiating the group creation, an attacker should, therefore, choose to initiate. As Pixy has no enforcement of random selection, an attacker might freely select cooperating nodes to participate, as long as they respect the global group list. This leads to the following attack vectors:

1. While **IP spoofing**, i.e., pretending to have a different IP address from the ones you control, allows to pass the first phase, usual applications of IP spoofing do not allow the attacker to control the address for subsequent phases [56]. Therefore the multi-phase setup prevents many forms of IP spoofing. IP spoofing by powerful local attackers, i.e., that can reroute traffic successfully, will still succeed.

2. Having **IP address distributed** over many unrelated subnets can be achieved with many different VPNs or comes naturally for botnets. As argued before, known VPNs should be represented sensibly in the group list to prevent cheap and inconspicuous attacks via VPNs.

Attack 2 is naturally applied for our botnet attacker, while the single attacker will have trouble depending on the setup of the group list, increasing probability of prevention. The large scale attacker can, depending on positioning, apply attack 1 without being detected, but on a large scale, the activities increase detection risk significantly.

Phase 2: Delaying Puzzle

In phase 2 of the group creation scheme and the first step for the temporary group, the participating nodes in the group are tested on their available hardware. To overcome this step, additional hardware is required, which is naturally available to botnets and large scale attackers and only slightly increases the cost to them. In contrast, single attackers will be unable to pass this phase with high probability. Possible circumventions like short term hardware rentals through cloud services introduce additional latency which increases the risk of detection significantly.

Phase 2: CAPTCHA

The last step in the scheme involves a test which requires human attention. A solution for this problem of the attacker are underground CAPTCHA solving services [83, 128]. These are services on the web which offer to solve the provided CAPTCHA by optical character recognition software or by human workers at around 1$ to 2$ per 1000 CAPTCHAs [107, 118].

Human attention for a single attacker will be natural to achieve on a small scale, but preventive in large scale systems. While it is available for large scale attackers, it is either a huge factor in cost, if done by themselves or increases the risk of detection significantly. For botnets, at last, the CAPTCHA solving services are the most viable route but increases the cost of operation.

Attack Conclusion

All presented steps are restrictive, as they restrict previously available routes. This limits the possibility of gains for an attacker by using Pixy, compared to other current systems.

While we showed that all phases can be broken, all strategies either increase cost, prevent certain kinds of attacks or increase risk of detection for large scale entities, especially through the combination of the different phases. This layered security shows our original goal of making attacks impractical or noticeably harder.

12.5 Conclusion

We developed Pixy, a flexible group creation scheme to prevent Sybil nodes in peer groups. Pixy provides increased protection from various common attacks by applying a two-phase scheme. Phase one creates a temporary group based on group lists and personal trust information to exclude Sybils and known collaborators. Phase two tests all participants of the temporary group using a memory timelock puzzle and a CAPTCHA. The results are aggregated per participant allowing every user to come to a satisfying conclusion on their own.

Our first evaluation shows that, while the system is still susceptible to powerful attackers, many common attack patterns are not viable when using Pixy. Therefore Pixy provides a clear advantage over the current state of the art.

The design of Pixy is modular so additional puzzles can be included or existing ones can be removed, e.g., the CAPTCHA if automation is required. This allows for continuous further improvements based on recent research.

Part IV

End

Chapter 13

Conclusion and Outlook

Privacy for broadcasts in blockchain networks encompasses a wide area of challenges. There are various motivations and different goals. Examples for such goals are miners of blocks, who are more focused on efficiency then privacy, while transactions are often highly sensitive. Applications might require information about the network, but participants might not want to reveal their participation or other aspects of the network.

Interactions between network protocols and consensus mechanisms can stay undetected for quite some time, leading to unexpected privacy implications through the persistence of the ledger. This interlock of systems, network layer privacy and blockchain layer privacy, will be one of the biggest challenges for lasting privacy design of future blockchain systems, now that there are strong proposals for both systems independently.

13.1 Conclusion

In this thesis, we developed various privacy-focused improvements for peer-to-peer broadcasts in public blockchain systems. From new and improved broadcast protocols in Chapters 4 to 6 and 11 to privacy friendly measuring in Chapter 10 and a privacy focused group making scheme in Chapter 12.

Chapters 4, 7 and 8 - 3P3 Flexible Strong Privacy

In Chapter 4, we describe the design of 3P3, while we analyze its properties in Chapters 7 and 8. 3P3 is a strong privacy protocol for broadcasting blockchain transactions. 3P3 can withstand a global passive observer and maliciously acting nodes jamming communication. The protocol can broadcast arbitrarily long messages with reduced overhead for empty messages.

The simulated analysis of 3P3 shows comparable performance to Dandelion and only $\approx 2.5\times$ overhead over a flood-and-prune broadcast while disregarding bandwidth overhead. The performance impact analysis of the parameters of our system provides flexible parameters for system developers using 3P3 to customize the privacy-performance decisions.

We also provided a proof of concept implementation of 3P3, validating our simulated results. The fully optimized instance only required around $0.5\text{s} \pm 0.1\text{s}$ for the tested instances. The implementation is suitable for real-world scenarios and available as open source software, but not ready for production usage.

Chapter 5 - 3P3 Phase I

Chapter 5 provides a realization of an arbitrary length extension of von Ahn et al.'s protocol. Further, the protocol is transformed from a point-to-point protocol to a group-only broadcast protocol. This allows its application in the 3P3 construction for a layered privacy protocol.

The arbitrary-length messages are made possible though introduction of a second round of dining-cryptographers communication. This creates considerable additional overhead compared to the fixed length version and requires transmission of large seed values to secure the additional round. While the optimizations improve on this discrepancy, the protocol only outperforms the fixed length protocol once a threshold of message size based on the number of participants is reached.

The largest beneficial optimization is the introduction of a less secure, but not less private, mode. This mode allows optimizations for common use-cases without compromising the privacy of the participants.

Chapter 6 - 3P3 Phase II

To provide a suitable protocol of the family of topological privacy mechanism, we transformed an established contact network protocol, adaptive diffusion, into a network protocol in Chapter 6. We extended the original results by modeling the virtual source passing probabilities in a more general way. Our model is based on the distance distribution of the underlying network. Further, we provide a privacy-friendly solution to solve the resulting equations, while smoothing out otherwise unachievable states. While the introduced indirections improve privacy for participants, as less information is available to attackers, they also reduce the reliability of the chosen parameters, as they are farther removed from reality.

To provide a real world instance of this protocol, we analyzed expected k-growing network topologies for their distance distributions. These topologies are similar to networks used by various blockchain systems. Our analysis showed approximately normally distributed shortest paths. Lastly, we performed a parameter analysis of the resulting normal distributions. This analysis showed that the normal distribution $\mathcal{N}(\mu, \sigma)$ can be approximated by a combination of logarithms and inverse exponentials. We provide and evaluate estimators based on the number of edges k and number of nodes n :

$$\begin{aligned}\mu(n,k) &\approx \frac{0.595\log(2.135n)}{\exp(0.314k)} + 0.341\log(1.626n) + \frac{0.241}{\exp(0.314k)} - 0.224\\ \sigma(n,k) &\approx 0.0345\log(0.925n) + \frac{1.222}{\exp(0.301k)} + 0.189\end{aligned}$$

These estimators allow network participants to compute required forwarding probabilities for our variant of adaptive diffusion.

Chapter 10 - Unobtrusive Monitoring

In Chapter 10, we collected extensive transaction dissemination data in the Bitcoin network. We used this data to analyze transaction dissemination latencies. 98% of the collected data is described well by a lognormal distribution, while extreme values of the distribution are better modeled using a generalized Pareto distribution. These results are independent of geographic location and stable over the measured time frame, but limited to the network of Bitcoin and cannot be easily transferred to other blockchain systems.

We constructed a latency estimator using only eight connections, the default for the Bitcoin network. This allows participants to monitor network dissemination latencies without being recognized as monitoring nodes. Known extensive monitoring solutions typically create thousands of connections, one for each known participant of the network.

We created and open sourced a proof-of-concept implementation of the monitoring construction and evaluated the monitoring tool based on the collected data. To improve the performance and accuracy of the monitoring, we construct a mechanism to determine the difference between the measurement and estimate, circumventing the unknown shift of the real distribution. The results of the provided tool show reasonably good estimations of the inherently noisy real-world data, but due to its error correction is unable to act on short term fluctuations.

Chapter 11 - Threshold Dining Cryptographers

In Chapter 11, we introduce a novel combination of a classical dining cryptographers protocol and Shamir's secret sharing technique. This combination uses the construction of the dining cryptographers protocol in an initial group phase. Every participant receives a unique share of a message split with Shamir's secret sharing during the group phase. An additional broadcast phase of the shares ensures message transmission.

Using a (n, k) Shamir's secret sharing scheme, this construction ensures that at least k group members initiate the broadcast before messages can be decrypted by general network participants. As group members require $k - 1$ additional shares to decrypt the message, they are incentivized to further participate in the protocol. In a traditional approach, participants might not cooperate, as they already received the message.

Our privacy analysis shows that the system provides privacy only based on the properties of the secret sharing scheme, not the dining-cryptographers construction. Once k or more attackers participate in the dining-cryptographers group, they are able to break the secret sharing scheme and identify the sender. We show, similar to the reliability requirement by von Ahn et al., that the scheme is secure for less than k attackers, which can be used to construct reasonably secure groups.

We provide a first, unoptimised solution, showing reasonable throughput rates for common applications in blockchain systems. In our simulation, we measured throughput rates between $10\,\mathrm{kB/s}$ and $100\,\mathrm{kB/s}$ for a full broadcast simulation and over $500\,\mathrm{kB/s}$ with reasonable privacy settings for local group dissemination.

Chapter 12 - Privacy Increasing Group Creation

Chapter 12 builds a foundation for our previously proposed group-based schemes. Pixy is a flexible group creation scheme to reduce Sybils, i.e., a single node pretending to be multiple nodes, and other collaborating nodes in peer groups. We provide a scenario based discussion of Pixy, but lack a theoretical privacy analysis.

Pixy provides increased protection from various common attacks by applying a two-phase scheme. Phase one creates a temporary group based on group lists and personal trust information to exclude Sybils and known collaborators. Phase two tests all participants of the temporary group using a memory timelock puzzle and a CAPTCHA. Attacks in Pixy are still possible, but Pixy provides an advantage over the current state of the art, improving privacy guarantees for group protocols in various common attack scenarios.

The design of Pixy is modular allowing the inclusion of additional puzzles or removal of broken or unsuited existing puzzles. This allows for continuous further improvements based on recent research.

13.2 Outlook

The results of this thesis can be further extended in various areas. We want to highlight some aspects where contributions to the field of privacy for broadcasts and network analysis can build upon this thesis.

Analysis of Network Behavior

Our analysis of the Bitcoin network should be extended to other blockchain systems, e.g., Ethereum, Monero and many more. Different systems attract different types of participants who exhibit various kinds of different behaviors. For a realistic evaluation of privacy protocols and other related research fields, correct behavior models are essential.

Secret Sharing and Dining-Cryptographers Networks

Secret sharing promises an enforced threshold of privacy by incentives beyond first cooperation. Our proposal of combining secret sharing with DC networks has shortcomings that might be overcome by alternative constructions. In a first step, a generalization of the concept should be approached. This should lead to deeper insights into the strengths and weaknesses of the mechanism.

Topological Privacy Methods

Our transformation of adaptive diffusion leads to various natural extensions for further research. While the derivation of probabilities from a given distribution is a general concept applicable to any distribution, the specific distributions and their parameters are not. The analysis of distributions should be extended to other network types, which also has applications in other fields, such as network modeling and theory.

Further, the protocol can be improved in other ways. A more direct model of the network instead of the abstract Markov chain may lead to more accurate results. Better heuristics to realize the Markov model in a real-world network and implementation of a probability to return the virtual source token, further increase the viability for various more extreme networks. Lastly, tried privacy improvements should be added and evaluated to increase the privacy guarantees of the protocol, e.g., adding noise to transmitted parameters such as the step number.

Privacy Preserving Broadcast Protocols

3P3, our presented privacy protocol, provides many opportunities for further optimization. The protocol would benefit from specialization to various alternative domains and evaluation of common behavior within these domains. One large effector of any domain is the expected size and variability of messages, i.e., are there many small messages or should one expect extremely large messages.

Privacy of Group Making Schemes

Lastly, the privacy increasing group creation scheme Pixy requires new techniques for evaluation and quantification of privacy of group making schemes. Stochastic evaluations would overvalue the propositions of such a scheme. Traditional attacker models, on the other hand, underestimate the real-world benefits of increased trust in participants.

13.3 Summary

With this thesis, we tackled various problems of privacy for broadcast protocols, with a focus on blockchain peer-to-peer networks. We published a dataset assisting future research projects as well as various models and software artifacts for modeling network properties. We constructed a novel fusion of threshold cryptography and dining-cryptographers networks, providing new insights. We designed a group creation scheme with a focus on privacy. Lastly, we built upon this group creation scheme with the design and proof-of-concept implementation of 3P3, a versatile privacy-preserving broadcasting protocol.

IEEE Copyright Notice

Bibliography

Own Publications

[1] H. Kopp, D. Mödinger, F. J. Hauck, F. Kargl, and C. Bösch. "Design of a Privacy-Preserving Decentralized File Storage with Financial Incentives". In: *Proceedings of IEEE Security & Privacy on the Blockchain (IEEE S&B) (affiliated with EUROCRYPT 2017)*. IEEE, 2017.

[2] D. Mödinger, J. Dispan, and F. J. Hauck. "Shared-Dining: Broadcasting Secret Shares using Dining-Cryptographer Groups". In: *Accepted at 21st International Conference on Distributed Applications and Interoperable Systems (DAIS)*. 2021.

[3] D. Mödinger, N. Fröhlich, and F. J. Hauck. "Pixy: A Privacy-Increasing Group Creation Scheme". In: *5th International Conference on Network Security (ICNS)*. 2020.

[4] D. Mödinger and F. J. Hauck. "3P3: Strong Flexible Privacy for Broadcasts". In: *4th International Workshop on Cyberspace Security (IWCSS 2020)*. 2020.

[5] D. Mödinger, H. Kopp, F. Kargl, and F. J. Hauck. "A Flexible Network Approach to Privacy of Blockchain Transactions". In: *Proc. of the 38th IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, 2018.

[6] D. Mödinger, J.-H. Lorenz, and F. J. Hauck. "n-Adaptive Diffusion for k-Growing Networks". In: *Submitted to Plos One*. 2021.

[7] D. Mödinger, J.-H. Lorenz, R. W. van der Heijden, and F. J. Hauck. "Unobtrusive monitoring: Statistical dissemination latency estimation in Bitcoin's peer-to-peer network". In: *PLOS ONE* 15.12 (Dec. 2020), pp. 1–21.

Own Publications (Unreviewed)

[8] D. Mödinger and F. J. Hauck. *Bitcoin Network Transaction Inv Data with Java Timestamp and Originator Id*. `https://doi.org/10.5281/zenodo.2547396`. Jan. 2019.

[9] D. Mödinger, A. Heß, and F. J. Hauck. "Arbitrary Length k-Anonymous DC Communication". In: (2021). arXiv: `2103.17091 [cs.NI]`.

Other References

[10] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. "Moderately hard, memory-bound Functions". In: *ACM Transactions on Internet Technology (TOIT)* 5.2 (2005), pp. 299–327.

[11] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. ninth Dover printing, tenth GPO printing. New York: Dover, 1964.

[12] L. von Ahn, A. Bortz, and N. J. Hopper. "K-anonymous Message Transmission". In: *Proc. of the 10th ACM Conf. on Comp. and Comm. Sec. (CCS)*. Washington D.C., USA: ACM, 2003, pp. 122–130.

[13] A. Aikebaier, T. Enokido, and M. Takizawa. "Trustworthiness among peer processes in distributed agreement protocol". In: *2010 24th IEEE Int. Conf. on Advanced Information Networking and Applications*. IEEE. 2010, pp. 565–572.

[14] A. Aikebaier, T. Enokido, and M. Takizawa. "Trustworthy Group Making Algorithm in Distributed Systems". In: *Human-centric Computing and Information Sciences* 1.1 (Nov. 2011), p. 6.

[15] I. M. Ali, M. Caprolu, and R. Di Pietro. "Foundations, Properties, and Security Applications of Puzzles: A Survey". In: *arXiv preprint arXiv:1904.10164* (2019).

[16] E. C. d. Almeida, G. Sunyé, Y. L. Traon, and P. Valduriez. "A Framework for Testing Peer-to-Peer Systems". In: *2008 19th International Symposium on Software Reliability Engineering (ISSRE)*. Nov. 2008, pp. 167–176.

[17] *Are you a robot? Introducing "No CAPTCHA reCAPTCHA"*. Google Security Blog, 2014.

[18] N. Balachandran and S. Sanyal. "A review of techniques to mitigate sybil attacks". In: *arXiv preprint arXiv:1207.2617* (2012).

[19] A. Bellet, R. Guerraoui, and H. Hendrikx. *Who started this rumor? Quantifying the natural differential privacy guarantees of gossip protocols*. 2020. arXiv: 1902.07138 [cs.DC].

[20] E. Ben Sasson et al. "Zerocash: Decentralized anonymous payments from bitcoin". In: *IEEE Symp. on Sec. and Priv. (SP)*. IEEE. 2014, pp. 459–474.

[21] A. Biryukov, D. Khovratovich, and I. Pustogarov. "Deanonymisation of Clients in Bitcoin P2P Network". In: *Proc. of the ACM SIGSAC Conf. on Comp. and Comm. Sec. (CCS)*. Scottsdale, Arizona, USA: ACM, 2014, pp. 15–29.

[22] A. Biryukov and S. Tikhomirov. "Deanonymization and linkability of cryptocurrency transactions based on network analysis". In: *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2019, pp. 172–184.

[23] N. Bitansky et al. *Time-lock puzzles from randomized encodings*. Association for Computing Machinery, 2016.

[24] S. Bojja Venkatakrishnan, G. Fanti, and P. Viswanath. "Dandelion: Redesigning the Bitcoin Network for Anonymity". In: *Proc. of the ACM Measurement and Analysis of Comp. Sys. (POMACS)* 1.1 (June 2017), 22:1–22:34.

[25] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. "Verifiable delay functions". In: *Annual International Cryptology Conference*. Springer. 2018, pp. 757–788.

[26] J. Bonneau et al. "Mixcoin: Anonymity for Bitcoin with accountable mixes". In: *Financial Cryptography and Data Sec.* Springer, 2014, pp. 486–504.

[27] N. Borisov. "Computational puzzles as Sybil defenses". In: *Sixth IEEE Int. Conf. on Peer-to-Peer Computing (P2P'06)*. IEEE. 2006, pp. 171–176.

[28] Y. Breitbart, Chee-Yong Chan, M. Garofalakis, R. Rastogi, and A. Silberschatz. "Efficiently monitoring bandwidth and latency in IP networks". In: *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*. Vol. 2. Apr. 2001, 933–942 vol.2.

[29] S. S. Brown, N. DiBari, and S. Bhatia. "I Am 'Totally' Human: Bypassing the reCaptcha". In: *2017 13th Int. Conf. on Signal-Image Technology & Internet-Based Systems (SITIS)*. IEEE. 2017, pp. 9–12.

[30] B. Butnaru et al. "P2PTester: a tool for measuring P2P platform performance". In: *2007 IEEE 23rd International Conference on Data Engineering*. Apr. 2007, pp. 1501–1502.

[31] J. A. Cárdenas-Haro and G. Konjevod. "Detecting Sybil nodes in static and dynamic networks". In: *OTM Confederated Int. Conf.s "On the Move to Meaningful Internet Systems"*. Springer. 2010, pp. 894–917.

[32] D. Chaum. "The dining cryptographers problem: Unconditional sender and recipient untraceability". In: *Journal of cryptology* 1.1 (1988), pp. 65–75.

[33] B. Cohen and K. Pietrzak. "Simple proofs of sequential work". In: *Annual Int. Conf. on the Theory and Applications of Cryptographic Techniques*. Springer. 2018, pp. 451–467.

[34] B. Conrad and F. Shirazi. "A Survey on Tor and I2P". In: *Ninth Int. Conf. on Internet Monitoring and Protection (ICIMP2014)*. 2014, pp. 22–28.

[35] A. B. Corman, P. Schachte, and V. Teague. "A Secure Group Agreement (SGA) protocol for peer-to-peer applications". In: *21st Int. Conf. on Advanced Information Networking and Applications Workshops (AINAW'07)*. Vol. 1. IEEE. 2007, pp. 24–29.

[36] H. Corrigan-Gibbs and B. Ford. "Dissent: Accountable Anonymous Group Messaging". In: *Proc of the 17th ACM Conf. on Comp. and Comm. Sec. (CCS)*. Chicago, Illinois, USA: ACM, 2010, pp. 340–350.

[37] G. Danezis and P. Mittal. "SybilInfer: Detecting Sybil Nodes using Social Networks." In: *NDSS*. San Diego, CA. 2009, pp. 1–15.

[38] D. Dean and A. Stubblefield. "Using client puzzles to protect TLS". In: *Proceedings of the 10th conference on USENIX Security Symposium-Volume 10*. USENIX Association. 2001, p. 1.

[39] C. Decker and R. Wattenhofer. "Information propagation in the Bitcoin network". In: *IEEE P2P 2013 Proceedings*. Sept. 2013, pp. 1–10.

[40] R. Dingledine, N. Mathewson, and P. Syverson. *Tor: The second-generation onion router*. Tech. rep. Naval Research Lab Washington DC, 2004.

[41] J. R. Douceur. "The Sybil Attack". In: *International workshop on peer-to-peer systems*. Ed. by P. Druschel, F. Kaashoek, and A. Rowstron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 251–260.

[42] C. Dwork, A. Goldberg, and M. Naor. "On memory-bound functions for fighting spam". In: *Annual International Cryptology Conference*. Springer. 2003, pp. 426–444.

[43] C. Dwork and M. Naor. "Pricing via processing or combatting junk mail". In: *Annual International Cryptology Conference*. Springer. 1992, pp. 139–147.

[44] Eng Keong Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. "A survey and comparison of peer-to-peer overlay network schemes". In: *IEEE Communications Surveys Tutorials* 7.2 (2005), pp. 72–93.

[45] M. Essaid, S. Park, and H.-T. Ju. "Bitcoin's dynamic peer-to-peer topology". In: *International Journal of Network Management* 30.5 (2020). e2106 nem.2106, e2106. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/nem.2106`.

[46] G. Fanti, P. Kairouz, S. Oh, and P. Viswanath. "Spy vs. Spy: Rumor Source Obfuscation". In: *SIGMETRICS Perform. Eval. Rev.* 43.1 (June 2015), pp. 271–284.

[47] G. Fanti et al. "Dandelion++: Lightweight Cryptocurrency Networking with Formal Anonymity Guarantees". In: *SIGMETRICS Perform. Eval. Rev.* 46.1 (Jan. 2019), pp. 5–7.

[48] M. J. Freedman and R. Morris. "Tarzan: A peer-to-peer anonymizing network layer". In: *Proceedings of the 9th ACM conference on Computer and communications security*. ACM. 2002, pp. 193–206.

[49] A. Fronczak, P. Fronczak, and J. A. Hołyst. "Average path length in random networks". In: *Phys. Rev. E* 70 (5 Nov. 2004), p. 056110.

[50] M. Gasca and T. Sauer. "Polynomial interpolation in several variables". In: *Advances in Computational Mathematics* 12.4 (Mar. 2000), p. 377.

[51] A. Gervais et al. "On the Security and Performance of Proof of Work Blockchains". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Vienna, Austria: ACM, 2016, pp. 3–16.

[52] P. Golle and A. Juels. "Dining Cryptographers Revisited". In: *Advances in Cryptology - EUROCRYPT 2004*. Ed. by C. Cachin and J. L. Camenisch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 456–473.

[53] Guanlin Bian, Yuehui Jin, and Tan Yang. "Delay measurement and analysis of network performance on PlanetLab". In: *2014 IEEE 3rd International Conference on Cloud Computing and Intelligence Systems*. 2014, pp. 706–711.

[54] T. Høiland-Jørgensen, B. Ahlgren, P. Hurtig, and A. Brunstrom. "Measuring Latency Variation in the Internet". In: *Proc. of the 12th Int. Conf. on Emerging Networking EXperiments and Technologies*. CoNEXT '16. Irvine, California, USA: ACM, 2016, pp. 473–480.

[55] *Introducing reCAPTCHA v3*. Google Security Blog, 2018.

[56] *IP Spoofing: An Introduction*. Symantec, Mar. 11, 2003.

[57] K. Iwase and K. Kanefuji. "Estimation for 3-parameter lognormal distribution with unknown shifted origin". In: *Statistical Papers* 35.1 (Dec. 1994), pp. 81–90.

[58] M. O. Jackson. *Social and economic networks*. Princeton university press, 2010. Chap. 4.

[59] Y. I. Jerschow and M. Mauve. "Non-parallelizable and non-interactive client puzzles from modular square roots". In: *2011 Sixth Int. Conf. on Availability, Reliability and Security*. IEEE. 2011, pp. 135–142.

[60] R. John, J. P. Cherian, and J. J. Kizhakkethottam. "A survey of techniques to prevent sybil attacks". In: *2015 Int. Conf. on Soft-Computing and Networks Security (ICSNS)*. IEEE. 2015, pp. 1–6.

[61] N. L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous univariate distributions, Vol. 2 of wiley series in probability and mathematical statistics: applied probability and statistics*. 1995.

[62] E. Jones, T. Oliphant, P. Peterson, et al. *SciPy: Open source scientific tools for Python*. 2001–.

[63] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn. "An Empirical Analysis of Anonymity in Zcash". In: *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 463–477.

[64] G. O. Karame and S. **C**apkun. "Low-cost client puzzles based on modular exponentiation". In: *European Symposium on Research in Computer Security*. Springer. 2010, pp. 679–697.

[65] K. Kaur and S. Behal. "CAPTCHA and its techniques: a review". In: *International Journal of Computer Science and Information Technologies* 5.5 (2014), pp. 6341–6344.

[66] H. Kopp, F. Kargl, C. Bösch, and A. Peter. "uMine: A Blockchain Based on Human Miners". In: *Information and Communications Security*. Cham: Springer International Publishing, 2018, pp. 20–38.

[67] P. Koshy, D. Koshy, and P. McDaniel. "An Analysis of Anonymity in Bitcoin Using P2P Network Traffic". In: *Financial Cryptography and Data Security*. Ed. by N. Christin and R. Safavi-Naini. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 469–485.

[68] B. Krishnamurthy, J. Wang, and Y. Xie. "Early measurements of a cluster-based architecture for P2P systems". In: *Internet Measurement Workshop*. 2001, pp. 105–109.

[69] S. Le Blond, D. Choffnes, W. Caldwell, P. Druschel, and N. Merritt. "Herd: A Scalable, Traffic Analysis Resistant Anonymity Network for VoIP Systems". In: *SIGCOMM Comp. Commun. Rev.* 45.4 (Aug. 2015), pp. 639–652.

[70] B. N. Levine, C. Shields, and N. B. Margolin. "A survey of solutions to the sybil attack". In: *University of Massachusetts Amherst, Amherst, MA* 7 (2006), p. 224.

[71] F. Li, P. Mittal, M. Caesar, and N. Borisov. "SybilControl: Practical Sybil defense with computational Puzzles". In: *Proceedings of the seventh ACM workshop on Scalable trusted computing*. ACM. 2012, pp. 67–78.

[72] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. "Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience". In: *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '03. Karlsruhe, Germany: Association for Computing Machinery, 2003, pp. 395–406.

[73] M. Mahmoody, T. Moran, and S. Vadhan. "Publicly Verifiable Proofs of Sequential Work". In: *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*. ITCS '13. Berkeley, California, USA: ACM, 2013, pp. 373–388.

[74] M. Mahmoody, T. Moran, and S. Vadhan. "Time-lock puzzles in the random oracle model". In: *Annual Cryptology Conference*. Springer. 2011, pp. 39–50.

[75] N. B. Margolin and B. N. Levine. "Informant: Detecting Sybils using Incentives". In: *Int. Conf. on Financial Cryptography and Data Security*. Springer. 2007, pp. 192–207.

[76] I. D. Mastan and S. Paul. *A New Approach to Deanonymization of Unreachable Bitcoin Nodes*. Cryptology ePrint Archive, Report 2018/243. 2018.

[77] S. Meiklejohn et al. "A fistful of bitcoins: characterizing payments among men with no names". In: *Proc. of the Internet Meas. Conf.* ACM. 2013, pp. 127–140.

[78] R. C. Merkle. "Secure communications over insecure channels". In: *Communications of the ACM* 21.4 (1978), pp. 294–299.

[79] I. Miers, C. Garman, M. Green, and A. D. Rubin. "Zerocoin: Anonymous distributed e-cash from bitcoin". In: *Proc. of the IEEE Symp. on Sec. and Priv. (SP)*. IEEE. 2013, pp. 397–411.

[80] A. Miller, M. Möser, K. Lee, and A. Narayanan. "An Empirical Analysis of Linkability in the Monero Blockchain". In: *CoRR* abs/1704.04299 (2017).

[81] A. Miller et al. *Discovering bitcoin's public topology and influential nodes*. `https://allquantor.at/blockchainbib/pdf/miller2015topology.pdf`. 2015.

[82] S. B. Mokhtar, G. Berthou, A. Diarra, V. Quéma, and A. Shoker. "RAC: A Freerider-Resilient, Scalable, Anonymous Communication Protocol". In: *2013 IEEE 33rd Int. Conf. on Distributed Computing Systems*. July 2013, pp. 520–529.

[83] M. Moradi and M. Keyvanpour. "CAPTCHA and its Alternatives: A Review". In: *Security and Communication Networks* 8.12 (2015), pp. 2135–2156.

[84] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. `https://bitcoin.org/bitcoin.pdf`. 2009.

[85] T. Neudecker, P. Andelfinger, and H. Hartenstein. "Timing Analysis for Inferring the Topology of the Bitcoin Peer-to-Peer Network". In: *2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. July 2016, pp. 358–367.

[86] S. Noether. *Ring Signature Confidential Transactions for Monero*. Cryptology ePrint Archive, Report 2015/1098. 2015.

[87] S. Noether and S. Noether. *Monero is Not That Mysterious*. Tech. rep. `https://lab.getmonero.org/pubs/MRL-0003.pdf`. 2014.

[88] L. Norman, S. Kotz, and N. Balakrishnan. *Continuous Univariate Distributions, Vol. 1 of wiley series in probability and mathematical statistics: applied probability and statistics*. 1994.

[89] M. Osadchy, J. Hernandez-Castro, S. Gibson, O. Dunkelman, and D. Pérez-Cabo. "No bot expects the DeepCAPTCHA! Introducing immutable adversarial examples, with applications to CAPTCHA generation". In: *IEEE Transactions on Information Forensics and Security* 12.11 (2017), pp. 2640–2653.

[90] T. P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Advances in Cryptology — CRYPTO '91*. Ed. by J. Feigenbaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 129–140.

[91] A. Pfitzmann and M. Hansen. "Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management-a consolidated proposal for terminology". In: *Version vo* 31 (2008), p. 15.

[92] K. Z. Pietrzak. "Simple verifiable delay functions". In: *10th Innovations in Theoretical Computer Science Conference*. Vol. 124. 2019.

[93] M. Ripeanu and I. Foster. "Mapping the Gnutella Network: Macroscopic Properties of Large-Scale Peer-to-Peer Systems". In: *Peer-to-Peer Systems*. Ed. by P. Druschel, F. Kaashoek, and A. Rowstron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 85–93.

[94] R. L. Rivest, A. Shamir, and D. A. Wagner. *Time-lock puzzles and timed-release crypto*. Massachusetts Institute of Technology, 1996.

[95] D. Ron and A. Shamir. "Quantitative analysis of the full bitcoin transaction graph". In: *Financial Cryptography and Data Security*. Springer, 2013, pp. 6–24.

[96] S. Roos, H. Salah, and T. Strufe. "Comprehending Kademlia Routing - A Theoretical Framework for the Hop Count Distribution". In: *CoRR* abs/1307.7000 (2013). arXiv: 1307.7000.

[97] T. Ruffing, P. Moreno-Sanchez, and A. Kate. "CoinShuffle: Practical decentralized coin mixing for Bitcoin". In: *Computer Security-ESORICS 2014*. Springer, 2014, pp. 345–364.

[98] E. Al-Shaer and Yongning Tang. "MRMON: remote multicast monitoring". In: *2004 IEEE/IFIP Network Operations and Management Symposium (IEEE Cat. No.04CH37507)*. Vol. 1. Apr. 2004, 585–598 Vol.1.

[99] A. Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613.

[100] C. Shi et al. "Adversarial CAPTCHAs". In: *arXiv e-prints* (Jan. 2019), pp. 1–16. arXiv: 1901.01107.

[101] V. P. Singh and P. Pal. "Survey of different types of CAPTCHA". In: *International Journal of Computer Science and Information Technologies* 5.2 (2014), pp. 2242–2245.

[102] W. Stadje. "The Collector's Problem with Group Drawings". In: *Advances in Applied Probability* 22.4 (1990), pp. 866–882.

[103] S. D. G. Stefan Saroiu P. Krishna Gummadi. "Measurement study of peer-to-peer file sharing systems". In: *SPIE Proc.* Vol. 4673. 2001, pp. 1–15.

[104] D. Stutzbach and R. Rejaie. "Understanding Churn in Peer-to-Peer Networks". In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. IMC '06. Rio de Janeriro, Brazil: Association for Computing Machinery, 2006, pp. 189–202.

[105] H. Subramanian. "Decentralized blockchain-based electronic marketplaces." In: *Commun. ACM* 61.1 (2018), pp. 78–84.

[106] F. Tegeler and X. Fu. "SybilConf: computational puzzles for confining Sybil attacks". In: *2010 INFOCOM IEEE Conference on Computer Communications Workshops*. IEEE. 2010, pp. 1–2.

[107] *Top 10 Captcha Solving Services Compared*. prowebscraper.

[108] S. Tritilanunt, C. Boyd, E. Foo, and J. M. G. Nieto. "Toward non-parallelizable client puzzles". In: *Int. Conf. on Cryptology and Network Security*. Springer. 2007, pp. 247–264.

[109] D. Tsoumakos and N. Roussopoulos. "Analysis and Comparison of P2P Search Methods". In: *Proceedings of the 1st International Conference on Scalable Information Systems*. InfoScale '06. Hong Kong: ACM, 2006, 25–es.

[110] E. Uzun, S. P. H. Chung, I. Essa, and W. Lee. "rtCaptcha: A Real-Time CAPTCHA Based Liveness Detection System." In: *NDSS*. 2018.

[111] L. Von Ahn, M. Blum, N. J. Hopper, and J. Langford. "CAPTCHA: Using hard AI problems for security". In: *Int. Conf. on the Theory and Applications of Cryptographic Techniques*. Springer. 2003, pp. 294–311.

[112] L. Von Ahn, M. Blum, and J. Langford. "Telling humans and computers apart automatically". In: *Communications of the ACM* 47.2 (2004), pp. 56–60.

[113] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. "reCAPTCHA: Human-based character recognition via web security measures". In: *Science* 321.5895 (2008), pp. 1465–1468.

[114] H. Wang et al. "A Captcha Design Based on Visual Reasoning". In: *2018 IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 1967–1971.

[115] P. Wang, P. Ning, and D. S. Reeves. "A k-Anonymous Communication Protocol for Overlay Networks". In: *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security*. ASIACCS '07. Singapore: Association for Computing Machinery, 2007, pp. 45–56.

[116] D. J. Watts and S. H. Strogatz. "Collective dynamics of 'small-world' networks". In: *nature* 393.6684 (1998), pp. 440–442.

[117] R. Watve, C. Mishra, and S. Sane. *Passive network latency monitoring*. US Patent 8,958,327. Feb. 2015.

[118] *Website 2CAPTCHA*. 2CAPTCHA.

[119] G. Welch, G. Bishop, et al. *An introduction to the Kalman filter*. 1995.

[120] B. Wesolowski. "Efficient verifiable delay functions". In: *Annual Int. Conf. on the Theory and Applications of Cryptographic Techniques*. Springer. 2019, pp. 379–407.

[121] S. N. Wolfson. "Bitcoin: the early market". In: *Journal of Business & Economics Research (Online)* 13.4 (2015), p. 201.

[122] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. "Dissent in Numbers: Making Strong Anonymity Scale." In: *OSDI*. 2012, pp. 179–182.

[123] A. Yeow. *Bitnodes*. `https://github.com/ayeowch/bitnodes`. Accessed: 2019-05-29. 2019.

[124] C. Yu et al. "Software-Defined Latency Monitoring in Data Center Networks". In: *Passive and Active Measurement*. Ed. by J. Mirkovic and Y. Liu. Cham: Springer International Publishing, 2015, pp. 360–372.

[125] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. "SybilLimit: A near-optimal Social Network Defense against Sybil Attacks". In: *2008 IEEE Symposium on Security and Privacy (sp 2008)*. IEEE. 2008, pp. 3–17.

[126] H. Yu, M. Kaminsky, P. B. Gibbons, and A. D. Flaxman. "SybilGuard: Defending against Sybil Attacks via Social Networks". In: *IEEE/ACM Transactions on networking* 16.3 (2008), pp. 576–589.

[127] Y. Zhang, H. Gao, G. Pei, S. Kang, and X. Zhou. "Effect of Adversarial Examples on the Robustness of CAPTCHA". In: *2018 Int. Conf. on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. IEEE. 2018, pp. 1–109.

[128] B. Zhao et al. "Towards Evaluating the Security of Real-World Deployed Image CAPTCHAs". In: *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security*. ACM. 2018, pp. 85–96.

[129] Y. Zhou, Z. Yang, C. Wang, and M. Boutell. "Breaking Google reCaptcha V2". In: *Journal of Computing Sciences in Colleges* 34.1 (2018), pp. 126–136.